Construire des bases de données – tome 2 : réalisations

Benoît Habert

20 juillet 2009

Dans la table des matières de ce deuxième tome, les titres qui comprennent $\bigoplus$ après le numéro de chapitre ou de (sous-)section constituent des compléments par rapport au premier tome.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

De manière complémentaire à la table des matières, à celle des figures, à la liste des tableaux qui viennent d'être fournies ainsi qu'au complément de glossaire et à l'index qui figurent en fin d'ouvrage, ces premières pages indiquent d'abord comment est organisé le volume (§ 1). À partir de la figure du tome 1 assurant le lien entre les grands volets du recours à un SGBD et les chapitres, sont détaillés des parcours de lecture et de travail possibles (§ 2). L'introduction aborde ensuite SQL, langue universelle des SGBD (§ 3). Elle montre les environnements qui permettent de s'en servir, graphiques ou par lignes de commande (§ 4). On trouvera au ch. XV les équivalences et différences entre :

- les dénominations informelles, strictes, propres à un logiciel donné pour les principales notions des SGBD ;
- les types d'attributs disponibles en MySQL et sous Access.

Enfin, sont explicitées les conventions qui se rajoutent à celles du tome 1 (§ 5).

1. ⊕ Organisation et utilisation

La consultation et l'utilisation des pages qui suivent supposent la lecture préalable du premier tome. Chaque chapitre comporte des exemples ou des exercices empruntés aux trois bases de données fournies. Ces exemples et exercices s'appuient souvent sur l'ensemble des notions présentées dans le tome 1. Il n'est donc pas toujours possible de lire en parallèle le tome 1 et le tome 2.

Le plan est cependant pour l'essentiel parallèle à celui de l'ouvrage-papier. Certaines sections sont vides : elles sont néanmoins conservées pour assurer le parallélisme avec l'ouvrage papier. Suivent des chapitres ou des sections supplémentaires, identifiés par le symbole ⊕ dans le titre. Dans la table des matières du tome 1, *Notions*, les titres des chapitres ou sections supplémentaires du tome 2, *Prise en main*, sont en italiques et signalés également par le symbole ▷ en lieu et place du symbole ⊕ : le numéro de page, en retrait à gauche, est alors celui du tome 2.

Au sein de chaque section parallèle, sont fournis dans l'ordre la réalisation en MySQL et en Access des opérations dont le principe, les étapes et le résultat figurent dans l'ouvrage papier, des compléments, des exemples et des exercices pour tout ou partie des trois bases de données utilisées. Une section supplémentaire en fin de chapitre donne les solutions de

tous les exercices du chapitre. Certains des exercices sont faisables avec uniquement l'ensemble des connaissances fournies depuis le début du tome 1 et de son parallèle au tome 2. D'autres supposent la maîtrise de notions développées dans la suite. Sauf mention contraire, les exercices ont deux versants, MySQL et Access, avec l'indication dans l'énoncé comme dans la correction des particularités propres à chaque environnement.

Les compléments fournis dans le tome 2 sont particulièrement importants pour ce qui concerne les quatre premiers chapitres. Ils insistent sur l'extraction et le calcul d'informations.

2. ⊕ Parcours de lecture et de travail

La figure 1 p. 17, déjà présente dans le tome 1, fournit une « carte » d'utilisation de l'ouvrage, en indiquant les chapitres et les sections correspondant aux différents moments du travail avec un SGBD.

On consultera dans tous les cas le ch XIV, consacré à l'installation du SGBD choisi et à sa mise en route, ainsi qu'à la mise en place des trois bases de données fournies comme exemples et « terrain d'entraînement ».

On se limitera, éventuellement, dans les trois études de cas fournies, à celle dont on se sent le plus proche ou qui paraît la plus immédiatement compréhensible.

Même si c'est Access qui est choisi comme SGBD, et l'interface graphique comme moyen privilégié d'interagir avec ce SGBD, on aura soin d'examiner en parallèle les requêtes en SQL (SQL Server et/ou MySQL – cf. § 3) : l'interface graphique limite les opérations envisageables (§ 4).

La pratique des bases de données est un apprentissage d'un langage comme un autre. À ce titre, elle suppose un « bain linguistique », c'est-à-dire une lecture précise et continue des notations abstraites et concrètes pour les requêtes. Les exercices, à réaliser sur machine, sont le moyen, pour le lecteur, de tester sa compréhension des notions et de leur mise en œuvre, les solutions celui de vérifer cette compréhension voire de se confronter à d'autres manières d'arriver au résultat demandé.

Les ch. III et IV consacrés à l'extraction, au tri des informations, ainsi qu'aux regroupements et aux opérations sur les colonnes sont fondamentaux. La compréhension des réalisations concrètes, sous Access, SQL Server et MySQL, des requêtes abstraites du tome 1 doit être complète. Les regroupements donnent accès à des informations de deuxième niveau souvent importantes, mais y recourir sans effort demande un peu de pratique.

L'étape suivante est celle de la maîtrise effective des jointures, à partir du ch. VII, et en s'appuyant sur les rappels du ch. VI. On s'appliquera en particulier à bien comprendre la notion souvent très précieuse de jointure externe : elle permet de saisir les décalages entre des tables.

La dernière étape, la plus ambitieuse, est celle de la modélisation. Le ch. IX fournit une présentation du problème. Les ch. XI et XII, de manière complémentaire, permettent d'une part de voir l'écart entre une première mise en base de données et un modèle du domaine abordé et d'autre part d'apprendre à utiliser un SGBD pour jauger et accroître la cohérence des informations disponibles.

Au moment jugé opportun, on créera une base de données soi-même, en utilisant les indications du ch. X. L'ajout de nouvelles tables à l'une ou l'autre des bases de données fournies, leur enrichissement et leur modification, comme y invitent d'ailleurs de nombreux exercices,

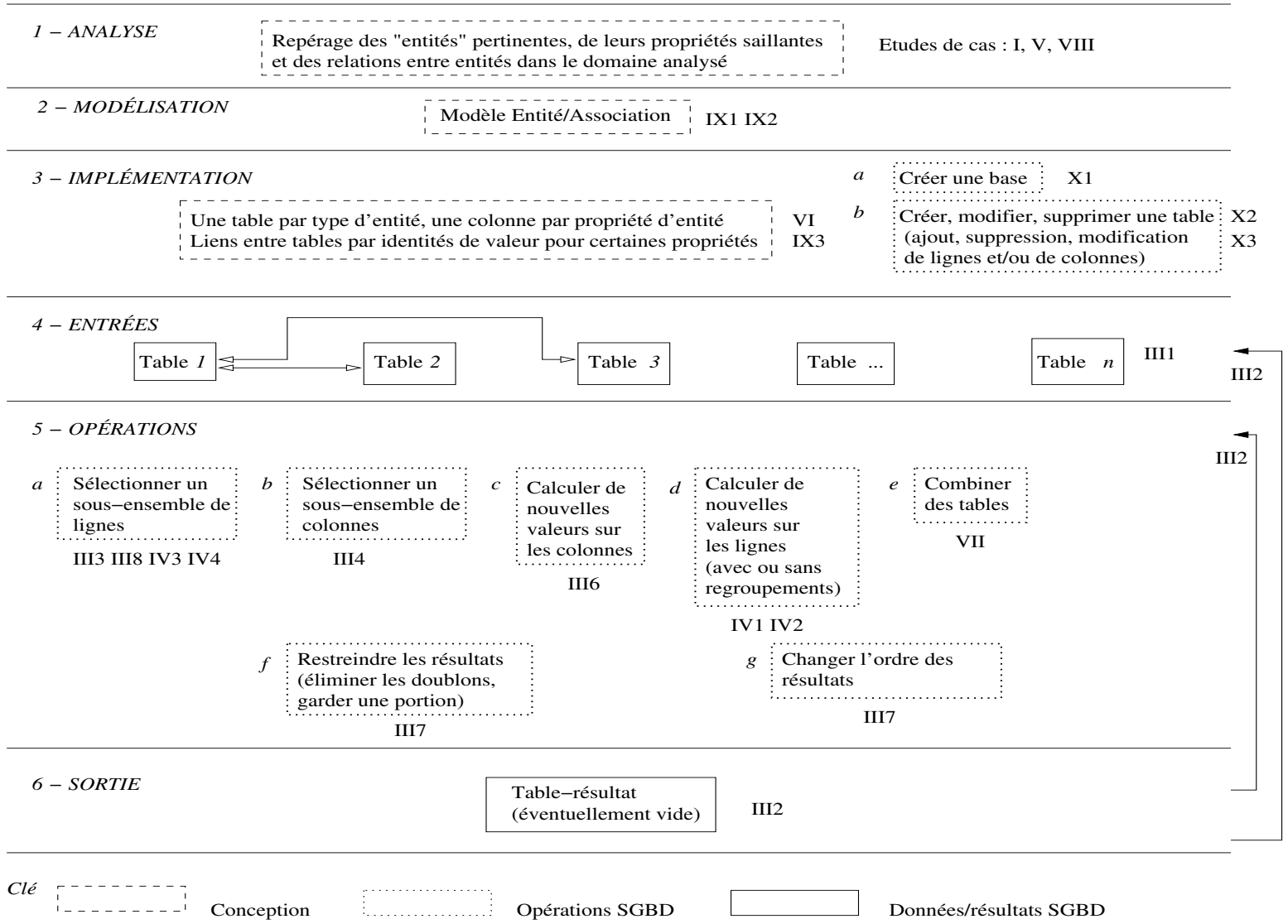


FIGURE 1 – De l'analyse d'un domaine à la base de données

constituent une étape en ce sens. On s'inspirera aussi des tables et de la modélisation de chacune des trois études de cas. Les chapitres correspondants (I pour PRÉMA, V pour PHÈDRE, VIII pour ESQUE) détaillent les tables, leur structure et leur contenu.

Les chapitres non mentionnés dans ces parcours possibles relèvent de plusieurs catégories :

- référence pour ce qui concerne par exemple les différences et convergences entre SQL, MySQL, SQL Server et Accss (ch. XIII) ;
- opérations complémentaires, comme l'import ou l'export d'informations vers ou d'un SGBD (ch. XV).

3. ⊕ La langue universelle des SGBD : SQL

Nous avons choisi de centrer la présentation des SGBD autour de la « norme » **SQL** (l'acronyme signifiait initialement *Structured Query Language*). Issu au début des années soixante-dix des travaux d'une équipe de chercheurs d'IBM, en particulier de ceux d'E. F. Codd (1970), ce langage pour SGBD a rapidement rallié les suffrages. L'étape suivante a été celle de la normalisation par l'ISO (*International Organization for Standardization*), l'institution qui fixe des **normes** internationales pour faciliter les échanges intellectuels, scientifiques, techniques et économiques entre pays. Une première version de la norme a été publiée en 1986, la suivante en 1992. La troisième date de 1999.

Les éditeurs de SGBD interprètent cependant à chaque fois la norme : ils n'intègrent pas toutes ses caractéristiques et ils rajoutent par ailleurs des traits non prévus. « . . . [L]e concept de norme est interprété avec beaucoup de sens poétique par les éditeurs de SGBD », indique joliment Hainaut (2002, p. 28). Date (2000, p. 81) précise à juste titre : « ils proposent généralement ce que l'on peut appeler "un sur-ensemble d'un sous-ensemble" », ce que symbolise la figure 2, p. 19 : ils ajoutent des fonctionnalités qui leur sont propres sans fournir toutes celles que définit SQL. Il n'en reste pas moins que SQL représente la *lingua franca*, la langue universelle, des SGBD relationnels, le **standard** *de facto*. Apprendre cette langue permet de faire face sans difficulté aux variations dialectales, un peu comme parler le français de France, de Belgique, de Suisse ou du Québec permet de se faire comprendre dans toute la francophonie.

Nous utilisons deux dialectes dans cet ouvrage, MySQL d'une part, **SQL Server** d'autre part. Access repose sur ce dernier dialecte : l'utilisation de l'interface correspond en effet à l'engendrement d'instructions en SQL Server.

Par ailleurs, au sein d'un même dialecte, il peut y avoir de légères variantes (l'équivalent des accents au sein d'une langue), en fonction des versions du logiciel. C'est le cas pour MySQL où certaines opérations avancées sont disponibles pour les dernières versions seulement. △

Pour MySQL comme pour Access et pour SQL Server, dans les mots-clés, la **casse** (l'opposition majuscules *vs.* minuscules) n'est pas pertinente. On peut aussi bien écrire le ET logique (∧) dans une condition de restriction :

- en MySQL et en SQL Server : **AND**, **and**, **And**, etc. ;
- sous l'interface d'Access : ET, Et, et, eT, etc.

Par souci de clarté, dans les instructions MySQL, les mots-clés et opérateurs sont généralement en majuscules.

Sous Access, l'initiale d'un mot-clé ou d'un opérateur est généralement majusculisée automatiquement par le logiciel dans la requête SQL Server engendrée à partir de l'interface, le reste demeurant en minuscules.

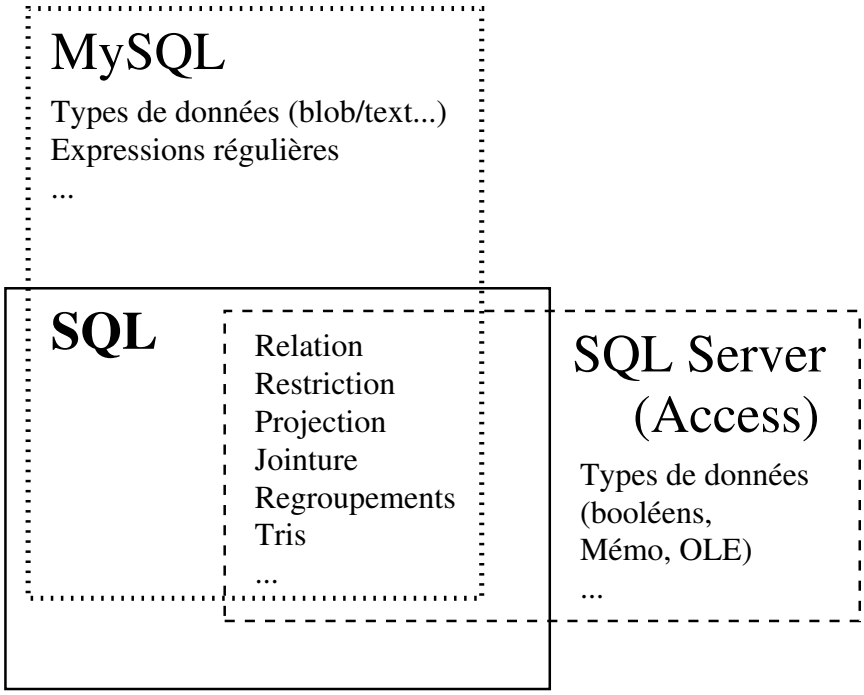


FIGURE 2 – Deux dialectes de SQL : MySQL et SLQ Server

4. ⊕ Langage textuel de requête *vs.* interface graphique

Les logiciels bureautiques (traitements de texte, tableurs, etc.) habituent à interagir avec eux via une interface graphique : des menus déroulants, des boîtes de dialogue, etc. Un tel environnement est disponible sous Access, par exemple, même s'il est également possible d'écrire directement des requêtes en SQL Server. D'autres logiciels utilisent préférentiellement un langage textuel de commandes, qui emploie un vocabulaire restreint et une syntaxe contrainte. C'est le cas de MySQL, même si plusieurs environnements graphiques (PhpMyAdmin, MySQL-Front) sont également disponibles. La figure 3 p. 21 montre la relation entre SQL et le SGBD concrètement employé. Un SGBD donné emploie un dialecte déterminé de SQL, SQL Server pour Access, et MySQL à la fois pour PhpMyAdmin et MySQL-Front. Un SGBD peut offrir une interface graphique spécifique pour formuler des requêtes et il les traduit dans le dialecte de SQL retenu. On peut en même temps directement écrire une requête dans le dialecte de SQL présent. C'est d'ailleurs généralement nécessaire pour les requêtes un peu compliquées, qui dépassent ce qu'il est possible d'exprimer via l'interface. Il est enfin possible de disposer de MySQL sans aucune interface graphique, uniquement sous forme textuelle.

Un des inconvénients des interactions par menus est qu'elles remplacent l'énoncé de contraintes par une suite d'étapes destinées à obtenir le résultat visé. L'utilisation de SQL à rebours facilite une meilleure compréhension du modèle relationnel et de la logique des opérations sous-jacentes. Par ailleurs, l'engendrement d'une requête en SQL à partir de sa formulation via une interface graphique est un processus de traduction où le résultat peut être plus ou moins proche de l'original, ne serait-ce qu'en raison du passage à l'anglais : les mots-clés de SQL sont en anglais. Ainsi ⌈Est Null⌉ dans l'interface d'Access donnera naissance à ⌈**IS NULL**⌉ dans la requête SQL Server créée, tandis que ⌈**ExtracChaine**([categorie ; 1 ; 1])⌉, qui

vise à extraire la première lettre de la colonne categorie dans la table occ_prema, débouchera en SQL Server sur ⌈**MID**([categorie], 1, 1)⌉.

Les notations employées dans le tome 1 pour les opérations présentées sont aussi proches que possible de celles utilisées en général pour les bases de données. Il s'agit de faciliter à la fois la compréhension des mécanismes abstraits en jeu, la mise en relation de ces mécanismes avec les opérations de l'algèbre relationnelle ainsi que l'articulation avec les instructions SQL apparentées et la réalisation sous Access.

Access

Il est souvent possible d'arriver à un même résultat par plusieurs chemins. Par exemple, les restrictions simples peuvent être obtenues soit par des requêtes à proprement parler, soit plus simplement par des filtres ou des formulaires (chapitre III § 3 p. 71). De la même manière, trier une table résultat peut s'effectuer lors de la formulation de la requête ou bien sur la table résultat par sélection de colonne(s) et validation d'un bouton idoine (chapitre III § 7 p. 88). Autant que possible, les alternatives sont présentées en même temps. △

Certaines opérations sont possibles avec un SGBD, pas avec un autre. C'est particulièrement le cas pour :

NOMBRE DE VALEURS DISTINCTES(<attribut>)

présentée dans le tome 1, réalisable sous MySQL via :

DISTINCT(<attribut>)

mais qui n'a pas d'équivalent immédiat avec Access ou SQL Server. Il faut alors « ruser » pour obtenir un résultat comparable.

La R_8^2 p. 49 en MySQL fait largement appel à NOMBRE DE VALEURS DISTINCTES. Elle est suivie de l'obtention sous Access de la même fonctionnalité. On se reportera à cette double réalisation pour comprendre la manière d'obtenir sous Access l'équivalent de NOMBRE DE VALEURS DISTINCTES.

L'équivalent Access/SQL Server de la requête R_{35}^2 p. 121 fournit une autre manière de combiner le calcul du nombre de valeurs distinctes d'un attribut avec d'autres résultats.

Dans les exercices, si le problème posé demande le recours à NOMBRE DE VALEURS DISTINCTES, en règle générale, seule la solution sous MySQL est fournie. Au lecteur de s'inspirer des R_8^2 p. 49 et R_{35}^2 p. 121 sous MySQL et de leurs équivalents Access/SQL Server pour parvenir à ses fins.

5. ⊕ Conventions complémentaires

Se rajoutent à celles du tome 1 les conventions suivantes :

compléments Les sections ou chapitres complémentaires sont identifiés par le symbole ⊕ avant le titre.

typographie Dans les requêtes, les mots-clés de SQL, comme **NULL**, sont généralement en majuscules grasses et les identifiants propres à l'utilisateur (noms de tables, de colonnes, etc.) en minuscules non accentuées, le caractère souligné liant éventuellement les différents composants (comme dans numero_vers). Certaines commandes engendrées par Access ont simplement une initiale à leurs mots-clés.

approfondissement, difficulté, problème Un triangle en note marginale sert de signal, comme ci-contre. △

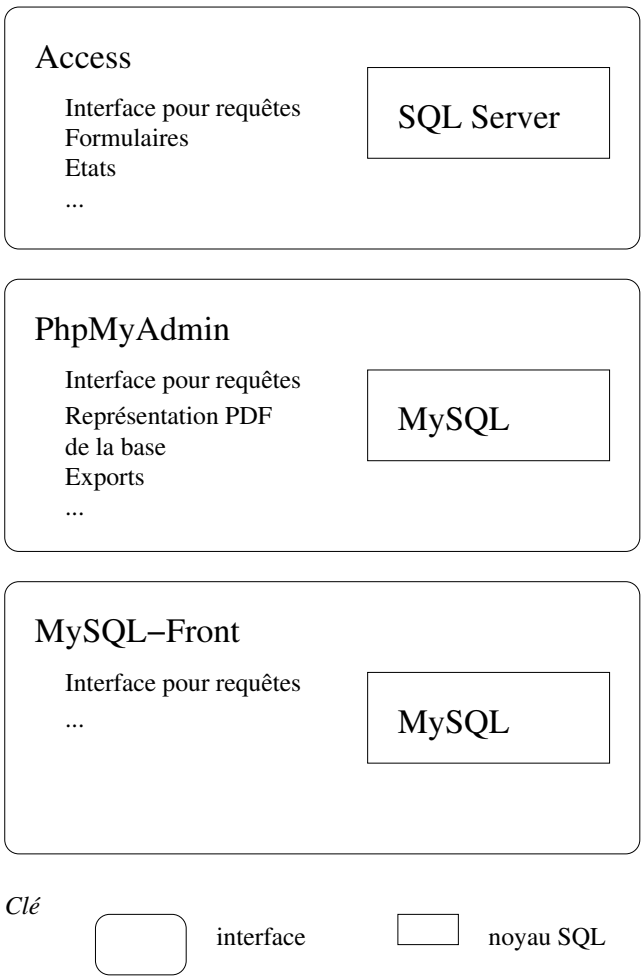


FIGURE 3 – Noyau SQL et interface(s)

glose informelle d'une requête Elle est introduite par le signe d'approximation $\simeq$. Elle est le plus souvent suivie de la notation abstraite de la requête et du renvoi à la page correspondante du tome 1.

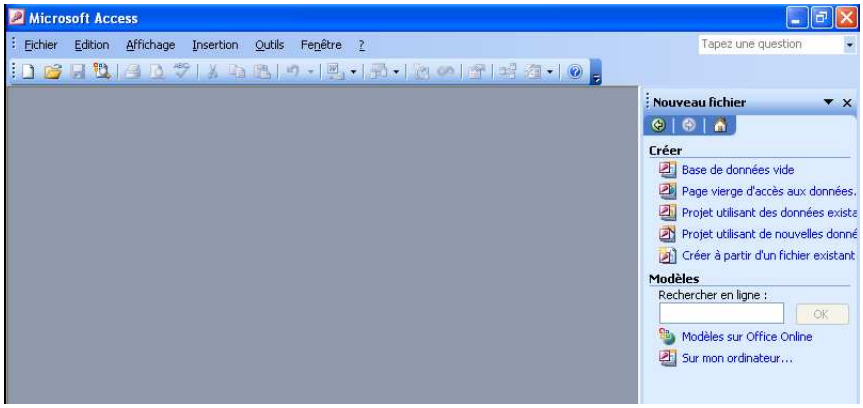
enchaînements de choix Les enchaînements de choix dans des menus, sous Access, sont séparés par la barre verticale. Ainsi ⌈Fichier | Ouvrir⌉ indique le choix du menu déroulant associé à ⌈Fichier⌉ dans la barre de menu du haut et, au sein de ce menu, de l'item ⌈Ouvrir⌉.

bases de données Les noms de certaines tables sont suffixées par _princeps, au sens d'original. Les tables ainsi suffixées sont les tables de départ, avant toute transformation. Ainsi la table fiches_originelles_princeps comporte une fiche rédigée le 2^e jour qui n'entre pas dans le cadre choisi pour l'étude et qui est donc éliminée dans la version corrigée de la table : fiches_originelles. Quand la distinction entre la base de données originale et sa version modifiée n'est pas pertinente, le nom suffixé par _princeps et le nom sans suffixe sont employés indifféremment.

captures d'écran Elles sont assorties d'un numéro encadré en note marginale (pour faciliter le repérage de ce numéro) comme dans l'exemple ci-dessous. Cet numéro encadré sert de référence. Par exemple, 1 renvoie à la capture d'écran ci-dessous.
Sous Acrobat Reader, en mode visualisation à l'écran, pour que les captures d'écran soient plus lisibles, on peut changer la résolution (passer par exemple à 125% voire 150% la taille réelle). △

renvois aux requêtes abstraites Les réalisations concrètes en MySQL ou sous Access des notations abstraites du tome 1 sont assorties d'un renvoi précis au numéro et à la page des requêtes concernées dans la partie papier. Les notations abstraites sont d'ailleurs souvent redonnées avant leur réalisation concrète pour faciliter le va-et-vient entre réalisation et abstraction. Il en va de même dans les solutions des exercices.

1



CHAPITRE I

ÉTUDE DE CAS 1 : *PRÉMATURÉS*

La structure des tables de la base PRÉMA est donnée pour MySQL. Les instructions de création sont détaillées pour la tacble infirmieres (§ 5). Les catégories employées pour les « mots » sont expliquées (§ 7) et exemplifiées (§ 12). Un certain nombre de tables supplémentaires servant à mémoriser différentes versions du texte des fiches sont présentées (§ 10). Est fourni enfin le glossaire rédigé par l'équipe médicale (§ 11).

On se reportera au ch. XV § 3 pour le détail des types de données offerts par MySQL et Access pour les attributs des tables.

1. Relier perception et situation de bébés prématurissimes

2. Organisation globale de l'enquête et des données

3. Des points d'entrée multiples

4. Les bébés

MySQL

La requête :

```
DESCRIBE bebes_princeps ;
```

donne la structure de la table bebes (tableau 1 p. 24), c'est-à-dire pour chaque attribut, sur une ligne, en colonnes, son nom et ses propriétés :

– nom (col.$_1$) ;

TABLEAU 1: PRÉMA : structure de la table bebes (MySQL)

Field$_1$	*Type*$_2$	*Null*$_3$	*Key*$_4$	*Default*$_5$	*Extra*$_6$
id	int(11)		PRI	0	
sexe	varchar(50)				
accouchement	varchar(50)				
lieu_naissance	varchar(50)				
poids_naissance	int(11)			0	
terme	decimal(4,2)			0.00	
terme_normalise	int(11)			0	
CRIB	int(11)			0	
jour_naissance	date			2000-01-01	
jour_experience	int(11)			0	
SM1	int(11)			0	
SM3	int(11)			0	
SM7	int(11)			0	
SM15	int(11)			0	
etat_J1	varchar(50)	YES			
etat_J3	varchar(50)	YES			
etat_J7	varchar(50)	YES			
etat_J15	varchar(50)	YES			

- type (col.$_2$) ;
- possibilité ou non de recevoir la marque **NULL** (col.$_3$) ;
- (partie de) clé primaire ou non (col.$_4$) ;
- valeur par défaut (col.$_5$) ;
- autres propriétés (col.$_6$).

On note que la valeur par défaut de l'attribut de type date jour_naissance est au format américain AAAA-MM-JJ (<année>-<mois>-<jour>).

Dans les descriptions de la structure des tables, la clé primaire est en gras pour faciliter son identification. C'est le cas ici de l'attribut id. △

Access

5. Les infirmières

Indexer un attribut (chapitre X, § 5 p. 322) signifie disposer de l'accès le plus rapide possible à ses valeurs. On distingue **indexation avec** et sans **doublons**. L'**indexation sans doublons** n'est possible que pour un attribut ou une combinaison d'attributs dont les valeurs sont toutes distinctes. C'est le cas par définition de la clé primaire d'une table.

On indexera systématiquement l'attribut ou la combinaison d'attributs constituant la clé primaire ainsi que les attributs qui entrent dans les clés étrangères. Clés primaires et clés étrangères sont mobilisées dans les jointures : un accès efficace à leurs valeurs est crucial.

MySQL

On comparera les instructions de création de la table concernant les infirmières avec les renseignements que fournit l'appel à **DESCRIBE** (tableau 2 p. 25).

TABLEAU 2 – PRÉMA : structure de la table infirmieres_princeps (MySQL)

Field	*Type*	*Null*	*Key*	*Default*	*Extra*
id	int(11)		PRI	0	
age	int(11)			0	
etudes	int(11)			0	
diplome	varchar(50)				
anciennete	decimal(4,2)			0.00	
service	varchar(50)	YES			

```
CREATE TABLE `infirmieres_princeps` (
  `id` int(11) NOT NULL DEFAULT '0',
  `age` int(11) NOT NULL DEFAULT '0',
  `etudes` int(11) NOT NULL DEFAULT '0',
  `diplome` varchar(50) NOT NULL DEFAULT '',
  `anciennete` decimal(4,2) NOT NULL DEFAULT '0.00',
  `service` varchar(50) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `id` (`id`) ) ;
```

La création d'une table obéit au schéma suivant :

```
CREATE TABLE <nom de la table> (
  <définition de l'attribut_1>
  ...
  <définition de l'attribut_n>
  <déclaration de la clé primaire>?
  <déclaration d'un index>*);
```

Une virgule sépare les différentes définitions et déclarations.

Une <définition d'attribut> est de la forme :

1. <nom de l'attribut>
2. <type>
3. <impossibilité d'une marque **NULL**>?
4. <existence d'une valeur par défaut>?

avec :
<impossibilité d'une marque **NULL** $\Longrightarrow$ **NOT NULL**
et
<existence d'une valeur par défaut> $\Longrightarrow$ **DEFAULT** <valeur par défaut>

Plusieurs types MySQL sont exemplifiés dans cette table :

- entier : **int**(11) ;
- nombre décimal : **decimal**(4,2) indique un nombre de 4 chiffres dont deux après le séparateur décimal ;
- **date** ;
- chaîne de caractères de longueur variable : **varchar**(50) indique que la valeur de l'attribut qui s'est vu conférer ce type peut être la chaîne vide, contenant 0 caractères (") mais peut aller jusqu'à 50 caractères. Par opposition, **char**(50) renvoie à une chaîne de longueur fixe, 50 caractères. On utilise généralement le type **varchar** pour « économiser » des caractères.

Les principaux types disponibles sous MySQL sont détaillés ch. XV § 3.

Si la combinaison de plusieurs colonnes constitue la clé primaire, leurs noms sont fournis entre parenthèses et séparés par des virgules.

Les contre-apostrophes – ‘ – qui entourent les noms de la table et des attributs ne sont pas nécessaires.

La ligne **KEY** ‘id‘ (‘id‘) indique que la table est indexée sur l’attribut id (mis entre parenthèses) et que le nom de l’index résultant est id.

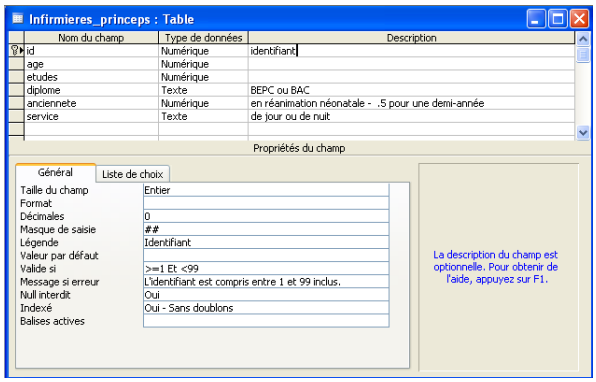
Access

En 1 figure la structure de la table infirmieres.

La clé à gauche du nom d’attribut id signale le rôle de clé primaire de l’attribut correspondant.

La colonne [Description] quand on crée ou que l’on modifie une table permet d’indiquer de manière informelle pour chaque attribut (*champ* dans la terminologie Access) son rôle, les valeurs qu’il peut prendre, etc. On saisira cette opportunité de documenter la table.

La fenêtre [Propriétés du champ] permet d’ajouter des contraintes d’attribut : valeur par défaut, **NULL** interdit ou non, restrictions sur les valeurs admissibles via [Valide si] (dans le cas présent, la valeur de l’identifiant doit être comprise entre 1 et 99 inclus). La ligne [Indexé] indique dans le cas présent que l’attribut id est indexé, ce qui accélérera les recherches et les jointures. C’est une indexation sans doublons : une clé primaire permet de distinguer les lignes d’une table, elle ne peut exister en double.



1

6. Les fiches

MySQL

La structure de la table fiches_originelles est décrite dans le tableau 3 p. 27.

Dans la description de la structure des tables, les clés étrangères sont en italiques pour faciliter leur repérage. C’est ici le cas d’id_bebe et d’id_infirmiere qui permettent des jointures avec respectivement les tables bebes et infirmieres.

Le type **MEDIUMBLOB** permet de stocker une longue chaîne de caractères, allant de 0 à 16 777 215 caractères.

Si l’attribut texte relevait du type **MEDIUMTEXT**, la requête :

TABLEAU 3 – PRÉMA : structure de la table fiches_originelles (MySQL)

Field	*Type*	*Null*	*Key*	*Default*	*Extra*
id	int(11)		PRI	0	
id_bebe	int(11)			0	
jour	int(11)			0	
heure_saisie	decimal(4,2)	YES		0.00	
poids	int(11)			0	
position	varchar(50)				
sedation	varchar(50)				
ventilation	varchar(50)				
id_infirmiere	int(11)			0	
frequence_occupation	varchar(50)				
moral_infirmiere	varchar(50)				
pronostic_infirmiere	varchar(50)				
relation_infirmiere_parents	varchar(50)				
relation_mere_bebe	varchar(50)				
frequence_visites_parents	varchar(50)				
texte	mediumblob				

TABLEAU 4 – PRÉMA : structure de la table occ_prema (MySQL)

Field	*Type*	*Null*	*Key*	*Default*	*Extra*
numero_ordre	int(11)		PRI	0	
fiche	int(11)			0	
forme_depart	varchar(50) binary				
forme_normalisee	varchar(100) binary				
lemme	varchar(100) binary				
categorie	varchar(100)				

texte **REGEXP** 'bébé'
rapatrierait aussi bien les fiches dont le texte contient la chaîne 'bébé' que celles qui contiennent 'Bébé' ou 'BéBé'. Ce type est indifférent à la casse pour les opérations de tri ou de comparaison de chaînes, à la différence de **MEDIUMBLOB**.
MySQL pour les différents types de chaînes de caractères dispose à chaque fois de deux variantes, l'une sensible à la casse dans les opérations de tri et de comparaison de chaînes, l'autre pas. Pour les types **CHAR** et **VARCHAR**, le choix d'une sensibilité à la casse s'obtient en ajoutant le mot-clé **BINARY** dans les propriétés de l'attribut. Pour les autres types, ceux suffixés par **BLOB** sont sensibles à la casse, ceux qui se terminent par **TEXT** ne le sont pas.

Access

7. Normaliser et « éclater » les textes

La structure de la table occ_prema est décrite dans le tableau 4 p. 27

7.1. ⊕ Jeu de catégories

Le jeu de catégories employé diffère de celui employé pour PHÈDRE. Il est inspiré de celui utilisé par l'étiqueteur Cordial (`http://www.synapse-fr.com`) et repose comme lui sur des dénominations anglaises. On trouvera dans (Habert, 2005, ch. I) une présentation détaillée de Cordial et de ses résultats sur *Le Dormeur du Val* d'Arthur Rimbaud.

La partie du discours, c'est-à-dire la catégorie principale, vient en première position : N, V, A, P, D, R, S, C, Y. Suivent ensuite des abréviations de traits morpho-syntaxiques et la position qui est la leur dans la catégorie complète. Ainsi, `c` ou `p` peuvent intervenir en première position après N pour distinguer un nom commun d'un nom propre. Une catégorie-balai, H, a été ajoutée pour rendre compte de deux signes notés par les infirmières dans leurs fiches papier et que la secrétaire qui a recopié les fiches n'a pas réussi à transcrire.

Les conventions utilisées dans `occ_prema` sont les suivantes :

N(oun) 1) c(ommon), p(roper), k (cardinal) 2) m(asculine), f(eminine) 3) s(ingular) p(lural) ;

V(erb) 1) m(ain), a(uxiliary) 2) i(ndicative), s(ubjonctive), m (iMperative), c(onditional), n (iNfinitive), p(articiple) 3) p(resent), i(mperfect), f(uture), a (pAst) 4) 1, 2, 3 5) m(asculine), f(eminine) 6) s(ingular) p(lural) ;

A(djective) 1) f (qualiFicative), o (Ordinal), k (cardinal), i(ndefinite), p(ossessive) 2) p(ositive) c(omparative) 4) m(asculine), f(eminine) 4) s(ingular) p(lural) ;

P(ronoun) 1) p(ersonal), d(emonstrative), i(ndefinite), s (poSsessive), t (inTerrogative), r(elative), x (refleXive), k (cardinal) 2) 1, 2, 3, 3) m(asculine), f(eminine) 4) s(ingular) p(lural) 5) Case n(ominative), a(ccusative), d(ative), o(blique) 6) Possessor s(ingular), p(lural) ;

D(eterminer) 1) a(rticle), d(emonstrative), p(ossessive), i(ndefinite), t (inTerrogative/exclamative), k (cardinal) 2) 1, 2, 3, 3) m(asculine), f(eminine) 4) s(ingular) p(lural) 5) Possessor s(ingular), p(lural) 6) Nature d(efinite), i(ndefinite) ;

R (adveRb) 1) g(eneral), p(article), x (interro-exclamative) 2) p(ositive), c(omparative), n(egative) ;

S (prepoSition) p(reposition), d(eictic) ;

C(onjunction) c(oordination), s(ubordination) ;

Y (punctuation) s(entence), w(ord).

Des exemples, où, entre crochets, chaque forme est suivie de son lemme et de sa catégorie, facilitent la compréhension de ce jeu d'étiquettes :

N(oun) [enfant enfant Nc.s], [aspirations aspiration Ncfp], [soins soins Ncmp], [Fentanyl Fentanyl Np..] ;

V(erb) [sont être Vaip3p], [a avoir Vaip3s], [environnent environner Vmip3p], [arrache arracher Vmip3s], [accepter accepter Vmn–], [avoir du mal avoir du mal Vmn–], [dérangée déranger Vmpasf], [en bougeant bouger Vmpp–] ;

A(djective) [moindre petit Afcms], [agréable agréable Afp.s], [active actif Afpfs], [agressifs agressif Afpmp] ;

P(ronoun) [l'on on Pp3.sn], [on on Pp3.sn], [la le Pp3fsa], [qui qui Pr-..a] ;

D(eterminer) [des un Da-.p-i], [les le Da-fp-d], [cette ce Dd-fs-], [ce ce Dd-ms-], [beaucoup d' beaucoup de Di-..-], [mon mon Ds1.ss], [ma mon Ds1fss] ;

R (adveRb) [+ + + + Rg], [plus plus Rgc], [que que Rgc], [non non Rgn], [pas pas Rgn], [+ + + + + + Rgp], [TRÈS très Rgp], [actuellement actuellement Rgp] ;

S (prepoSition) [après après Sp], [au moment de au moment de Sp] ;

C(onjunction) [+ + Cc], [car car Cc], [ni ni Cc], [bien que bien que Cs], [comme comme Cs] ;
Y (punctuation) [. . Yps], [: : Yps], [?? Yps], [!! Ypw], [, , Ypw], [- - Ypw], [?? Ypw].

On notera que les ponctuations sont considérés comme des « mots », puisqu'elles possèdent lemme et catégorie.

Exercice n°1 Donner la fréquence des parties du discours dans PRÉMA. On considérera que la partie du discours est la première lettre de la catégorie. On extraiera donc la première lettre de l'attribut correspondant à la catégorie.

Si nécessaire, on s'inspirera de la démarche équivalente pour PHÈDRE (chapitre V § 5 p. 207).

MySQL

La fonction
SUBSTRING(<chaîne>, <caractère de départ>, <nombre de caractères souhaités>)
extrait d'une chaîne le nombre de caractères souhaité à partir d'un caractère donné. Le premier caractère de la chaîne a comme numéro 1. Ainsi, **SUBSTRING**("Hannah", 4, 3) retourne la chaîne "nah". Dans 'H^{1}a^{2}n^3**n**$^4_1$**a**$^5_2$**h**$^6_3$', les exposants correspondent à la numérotation des caractères dans la chaîne de départ, les indices aux caractères retenus (par ailleurs en gras).

Access

ExtracChaine(<chaîne>; <caractère de départ>; <nombre de caractères souhaités>) est l'équivalent de **SUBSTRING**.

Exercice n°2 Pour chacune des différentes parties du discours des mots de PRÉMA (cf. exercice précédent), produire la liste des 10 lemmes les plus fréquents, avec leur fréquence, classés par fréquence décroissante. △

Noter que les arguments des fonctions et opérateurs sont très généralement séparés par des virgules sous MySQL et SQL Server et par des points virgules sous l'interface d'Access.

8. Individus et caractères

9. Des informations à relier

10. ⊕ Versions du texte des fiches

La table occ_prema fournit des « vues » distinctes du texte écrit par les infirmières : brut, normalisé, lemmatisé. Il est également possible d'utiliser la catégorie fournie pour distinguer par exemple *calme* adjectif de *calme* nom. Le texte est cependant éclaté en autant de lignes que de segments par fiche. Dans la mesure où les tables d'une base de données ne sont pas faites pour tenir compte de la séquentialité des informations contenues dans les lignes, on ne peut pas aisément utiliser la table occ_prema pour rechercher des cooccurrences entre mots (voir cependant *a contrario* l'exercice n°4 et sa solution). Dans le même temps, la version du texte que contient la table fiches_originelles est la version brute, riche en scories, variations inattendues, etc. On a donc reconstitué plusieurs tables (exemple dans le tableau 5 p. 30) qui contiennent chacune deux colonnes, l'identifiant de la fiche en cause et une version spécifique du texte :

TABLEAU 5 – PRÉMA : états du texte (fiche 131)

Table	*texte*
fiches_depart	BB bien vigoureux pt les soins, se debat bcp. Reste paisible en dehors des soins.
fiches_normalisées	bébé bien_R vigoureux pendant_S les soins , se_débat beaucoup . reste_V paisible en_dehors_de les soins .
fiches_lemmatisees	bébé bien_R vigoureux pendant_S le_D soin , se_débattre beaucoup . rester paisible en_dehors_de le_D soin .
fiches_en_categorie-_complete	Ncms Rgp Afpms Sp Da-__-d Ncmp Ypw V_ip3s Rgp Yps . V_ip3s Afp_s Sp Da-__-d Ncmp Yps .
fiches_en_categorie-_intermediaire	Nc Rgp A Sp Dad Nc Ypw V3s Rgp Yps . V3s A Sp Dad Nc Yps .
fiches_en_POS	N R A S D N Y V R Y . V A S D N Y .

- brut : fiches_depart ;
- normalisé : fiches_normalisees ;
- lemmatisé : fiches_lemmatisees ;
- réduit aux catégories : fiches_en_categorie_complete ;
- réduit à des catégories « simplifiées » : fiches_en_categorie_intermediaire ;
- réduit aux parties du discours ou POS (*Part of Speech*) : fiches_en_POS.

Dans les tables fiches_normalisees et fiches_lemmatisees, la partie du discours est collée en cas d'ambiguïté : on distingue ainsi *le_D*(éterminant) dans *pendant_S le_D soin* et *le_P*(ronom) dans *quand on le_P laisser tranquille*. Parallèlement, une table signaletique_fiches_princeps a été constituée, qui comprend tous les attributs de fiches_originelles sauf l'attribut texte. Une jointure sur l'identifiant de la fiche entre la table signaletique_fiches_princeps et l'une des tables mentionnées permet de disposer de la version souhaitée du texte de chaque fiche et des métadonnées correspondantes.

Exercice n°3 (MySQL)

Repérer les « mots en plusieurs mots », comme *de nouveau*, *le restant de la journé*e ou *sonde gastrique*, dans les lemmes de la table occ_prema. Pour chaque mot en plusieurs mots, identifier le nombre de mots simples correspondants. Compter enfin le nombre de mots en plusieurs mots, le nombre de mots simples correspondants, le nombre de mots simples en dehors des mots en plusieurs mots.

Le nombre de mots simples dans un mot en plusieurs mots correspond à 1 de plus que le nombre d'espaces entre les mots simples. Une manière de compter le nombre de mots simples dans un mot en plusieurs mots est alors d'éliminer les espaces en son sein, de soustraire la longueur de la chaîne résultante à celle du lemme de départ et d'ajouter 1.

On utilisera pour MySQL les fonctions :

REPLACE(<chaîne>, <chaîne à remplacer>, <chaîne de substitution>)

et

LENGTH(<chaîne>) qui retourne la longueur de la chaîne en nombre de caractères.

NB : Les équivalents SQL Server sont REPLACE et LEN.

Exercice n°4 En utilisant les techniques d'auto-jointure du chapitre VII § 4, fournir une table avec les mots qui cooccurrent à gauche de *puce* et le numéro de la fiche correspondante.

Exercice n°5 Donner le nombre de garçons et de filles qui sont qualifiés de *puce*.

Exercice n°6 Fournir une table des lemmes pour lesquels il y a plus d'une partie du discours (qui sont ambigus, donc) et le nombre de parties du discours correspondant. La partie du discours est le premier caractère de l'attribut categorie dans la table occ_prema. On voit par exemple dans le tableau 9 p. 38 que *le* apparaît à la fois dans les déterminants et dans les pronoms.

Exercice n°7 À partir du résultat de l'exercice précédent, fournir pour chacun des lemmes ayant plusieurs parties du discours, le nombre d'occurrences de chaque partie du discours. On utilisera d'abord une table créée à la volée pour mémoriser les lemmes en question. On procédera dans un deuxième temps via une requête enchâssée faisant appel à cette table.

11. ⊕ Glossaire de l'équipe médicale

algique Douloureux

antalgique Médicament ou attitude pour traiter la douleur. Equivalent à *antidouleur*, *analgésique,* parfois à *sédatif* qui introduit en plus la notion de calmer.

antidouleur Analgésique, voir *antalgique* plus haut.

aspiration Soin agressif qui consiste à descendre un tube fin dans la sonde de ventilation qui est dans la trachée pour aller retirer les sécrétions en excès.

avenir neurologique Idée qu'on se fait des chances qu'a le bébé d'évoluer sans séquelles neurologiques, c'est-à-dire avec un développement normal, des acquisitions dans les temps habituels, vision et audition normales, comportement normal.

bradycarde Fréquence cardiaque qui diminue. En général c'est le signe que le bébé oublie de respirer mais ça peut être plus grave.

bradycardies Coeur qui ralentit.

Canadou Sucre liquide qui a un effet puissament sédatif et antalgique chez le prématuré. C'est le même qui sert à faire des punchs quand on rajoute du rhum.

canal artériel Persistance d'un canal entre l'aorte et l'artère pulmonaire (2 gros vaisseaux) qui normalement se ferme à la naissance.

cartonné Modification de l'aspect de la peau qui a perdu sa souplesse naturelle. En général ce n'est pas bon, ça signe une infection grave.

coconou Dispositif, de la forme d'un boudin, fait en linges ou en mousse qui permet d'installer et de contenir l'enfant. But : confort et éviter certaines déformations.

cyanosé (Enfant bleu ou noir). Reflet cutané de la baisse du taux d'oxygène dans le sang.

désaturation Diminution de la saturation en oxygène, oublie de respirer ou a besoin de plus d'oxygène.

désature Voir *Sat.* C'est la saturation qui diminue. Les besoins en oxygène augmentent. Là aussi cela peut être dû au fait qu'il oublie de respirer.

Dextro Examen fait sur une goutte de sang pour doser la quantité de sucre. Contrôle habituel chez les diabétiques et chez les prémas.

dysmorphique Avec des malformations apparentes.

en peau à peau Bébé installé directement sur la peau de sa mère où il va passer un long moment. Pour le contact, le plaisir et la chaleur.

écho(graphie) cérébrale Visualisation du cerveau par échographie permettant de dépister un saignement, une malformation du cerveau ou d'autres complications cérébrales. Examen fait systématiquement plusieurs fois pendant le séjour.

enfant jaune Signe cutané de l'ictère (= jaunisse). Immaturité du métabolisme de la bilirubine (pigments biliaires jaunes).

entérocolite Infection intestinale grave du prématuré.

érythrosique Tout rouge, cela peut être bien parce que c'est mieux que pâle ou gris ou vert ou bleu mais ça peut être trop rouge s'il a trop de globules rouges.

ETF abréviation de « Echographie transfontanellaire ». C'est l'examen qui nous permet de regarder le cerveau en échographie pour dépister les hémorragies ou les leucomalacies.

eutrophique A le bon poids attendu pour son terme.

extubé Pour aider le bébé à respirer, on a branché un respirateur sur un tube placé dans la trachée. Quand on enlève ce tube, on extube le bébé, il est alors « extubé ».

Fentanyl Analgésique, médicament anti-douleur de la famille des opiacés, morphine.

« grimpe » dans son incubateur Arrive à se déplacer en rampant dans sa couveuse.

Hypnovel Sédatif comme du valium. Cela calme sans être vraiment anti-douloureux.

hypotrophe Trop petit pour ce qu'on attend pour cette durée de grossesse.

KTC Abréviation de cathéter central. C'est un tube très fin (0,4 mm de diamètre) et long de 15 à 20 cm qu'on insère dans une veine du bras et qu'on pousse jusque près du coeur. Cela nous permet d'apporter des calories intraveineuses tout le temps durant lequel on ne peut pas donner à manger au bébé. Parfois durant plusieurs semaines.

ictérique Jaune. Il a la jaunisse ou l'ictère c'est la même chose.

incubateur C'est la maison du préma, c'est une boite transparente à roulettes avec un chauffage pour le garder bien au chaud.

indurations Zones cutanées plus dures que le reste de la peau.

infiltré Avec des oedèmes sous la peau.

insécurisé Paraît inquiet.

intubation Pose d'un tube intratrachéal pour une ventilation artificielle.

invasif Comme en français courant. On dit qu'un geste est invasif quand il comporte une part d'invasion du corps. Par exemple la saturation n'est pas invasive car elle donne un résultat alors qu'elle n'est que collée sur la peau alors que le dextro est invasif car il nécessite une goutte de sang donc de faire une piqûre qui est invasive.

labile Instable, parfois mieux parfois moins bien.

leucomalacie Anomalie cérébrale de la substance blanche du cerveau qui signe que le cerveau a souffert et que le bébé peut en garder des séquelles.

lunettes de photothérapie lunettes en tissu qui protègent les yeux pendant les séances de lumière bleue en cas d'ictère.

méningocèle Malformation cérébrale. Le cerveau sort comme un chignon du crâne.

Narcan Antidote des médicaments anti douloureux de la famille des opiacés, les morphines. Donné quand on a des effets indésirables des morphines comme par exemple de ne plus respirer.

oedématié avec des oedèmes, gonflement, infiltration d'eau sous la peau.

oedème Gonflement sous cutané dû à une infiltration par de l'eau.

PCO2 Appareil qui, collé sur la peau, donne la teneur du sang en gaz carbonique (CO2).

photothérapie Lumière bleue qui traite l'ictère.

polypnéique Respire vite. Mal employé : on devrait dire *tachypnéique* mais on n'y arrive pas.

pose d'un KTC mise en place d'un cathéter central.

proclive ventral Un peu incliné, tête surélevée par rapport aux pieds et ventral car il est couché sur le ventre.

pronostic vital Idée qu'on se fait des chances de survie du bébé.

Raniplex Médicament anti acide donné pour prévenir un ulcère de l'estomac.

SA Semaine d'aménorrhée, c'est-à-dire délai entre le premier jour des dernières règles et la date de naissance. Par exemple 6 mois de grossesse, c'est 6 mois ou 52/2, 26 semaines plus les 2 semaines entre le premier jour des dernières règles et la conception. Donc, 6 mois de grossesse = 28 semaines d'aménorrhée = 28 SA. 24 SA, c'est à peu près 5 mois de grossesse. 29 SA c'est 6 mois et une semaine de grossesse. Il lui manque 2 mois et 3 semaines.

Sat Saturation en plus clair saturation en oxygène de l'hémoglobine ou teneur en oxygène du sang. L'appareil : le saturomètre ou oxymètre de pouls se pose sur la peau et indique un pourcentage. De 90 à 100% c'est OK chez le préma.

saturation Chiffre en pourcentage d'oxygène lié à l'hémoglobine. C'est le chiffre qui nous dit si le bébé a besoin ou non de plus d'oxygène.

saturo Saturation de la teneur du sang en oxygène.

sédation Voir *antalgique* et *Hypnovel*.

SG Abréviation de *sonde gastrique*.

soins agressifs Certains soins sont agressifs même s'ils sont nécessaires. Comme par exemple d'aspirer dans la sonde d'intubation ou de poser une perfusion intra veineuse ou de faire un dextro.

sonde gastrique Petit tube introduit par la bouche ou par le nez et poussé jusque dans l'estomac. Il va nous permettre de pousser du lait directement dans l'estomac.

sous photo Traité pour son ictère par la lumière bleue.

syndactylies Malformations au niveau des doigts de la main ou du pied. Soudures de 2 doigts entre eux.

TA Tension artérielle.

tirage intercostal Difficulté à respirer tout seul. Cela se voit car le prématuré fait des efforts importants pour respirer. On dit qu'il tire et ça se regarde sur le thorax, donc il tire au niveau des côtes (« tirage intercostal »).

trémulations Tremblements des membres. Parfois quand il y a un problème neurologique une souffrance du cerveau.

utérin De l'utérus dans lequel il était avant de naître.

ventilé Bébé aidé par un respirateur.

TABLEAU 6 – PRÉMA : fréquence des parties du discours

Catégorie	*o.*
	27126
*	641
A	4014
C	1476
D	3372
H	2
N	5131
P	2199
R	3253
S	2582
V	4412
Y	5617
^	6781

12. Solutions

Solution de l'exercice n°1 p. 29 Supposons que l'on dispose de la fonction **SOUSCHAÎNE**(<chaîne>, <caractère de départ>, <nombre de caractères souhaités>) qui extrait d'une chaîne le nombre de caractères souhaité à partir d'un caractère donné. Le premier caractère de la chaîne a comme numéro 1.

La requête abstraite est alors de la forme :

R_1^2

REGROUPER SUR(
SOUSCHAÎNE(categorie, 1, 1)
)[occ_prema]

$\hookrightarrow$

PAR GROUPE(
SOUSCHAÎNE(categorie, 1, 1) TITRE 'Catégorie',
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_1$>]

On ne garde de chaque catégorie que le premier caractère et on regroupe les occurrences sur ce premier caractère. Pour chaque regroupement, on fournit le premier caractère correspondant et le nombre d'occurrences.

Le tableau 6 p. 34 donne le résultat. On note des « catégories » inhabituelles : l'absence de catégorie, pour les espaces ; l'accent circonflexe qui indique qu'une ligne correspond à un fragment d'un « mot en plusieurs mots » et donc ne porte pas de catégorie en tant que telle. L'étoile joue un rôle un peu similaire à l'accent circonflexe. Il est utilisé en particulier pour les mots en plusieurs mots discontinus, c'est-à-dire dont les composants sont séparés dans la version de départ mais regroupés dans les autres versions, comme la négation (tableau 7 p. 35).

MySQL

Une solution possible est :

```
SELECT SUBSTRING(categorie, 1, 1) AS 'Catégorie',
    COUNT(*) AS 'o.'
FROM occ_prema
GROUP BY SUBSTRING(categorie, 1, 1) ;
```

TABLEAU 7 – PRÉMA : table occ_prema, ^ et *

numero_ordre	*fiche*	*forme_depart*	*forme_normalisee*	*lemme*	*categorie*
66346	1012	Il	il	il	Pp3ms-
66347	1012				
66348	1012	ne	ne pas	ne pas	Rpn
66349	1012				
66350	1012	se	se laisse	se laisser	V.ip3s
66351	1012		^	^	^
66352	1012	laisse	^	^	^
66353	1012				
66354	1012	pas	*	*	*
66355	1012				
66356	1012	faire	faire	faire	V.n–

Access

La requête SQL Server peut prendre différentes formes, suivant la fonction utilisée pour extraire la première lettre de la catégorie (sa « partie du discours »).

La fonction
LEFT(<chaîne>, k)
extrait les k caractères les plus à gauche de la chaîne fournie comme premier argument.

```
SELECT LEFT([categorie],1) AS Catégorie,
    Count(Occ_prema.numero_ordre) AS o
FROM Occ_prema
GROUP BY LEFT([categorie],1);
```

On peut aussi faire appel à **MID**, l'équivalent SQL Server de **SUBSTRING** pour MySQL :

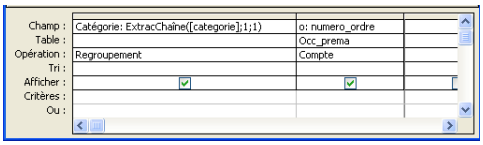
```
SELECT Mid([categorie],1,1) AS Catégorie,
    Count(Occ_prema.numero_ordre) AS o
FROM Occ_prema
GROUP BY Mid([categorie],1,1);
```

qui est engendrée par Access à partir de l'appel à EXTRACCHAÎNE : 2.

La comparaison entre l'interface et la requête SQL Server est instructive. Le nom de la fonction est « francisé » dans l'interface. Il ne correspond pas à celui de la requête SQL Server, qui par ailleurs recourt à des virgules et non à des points virgules pour séparer les arguments.

Quand le titre donné à la colonne dans la table résultat est un nom simple, sans espace, il n'est pas nécessaire de l'encadrer de crochets. △

2



Solution de l'exercice n°2 p. 29 On utilise une restriction pour se limiter aux lemmes de la catégorie choisie. On regroupe alors par lemme, et pour chaque lemme, on fournit le lemme et le nombre de lignes du regroupement correspondant. Il ne reste plus qu'à trier par nombre d'occurrences décroissant et à se limiter aux 10 premiers résultats.

R_2^2

RESTRICTION(categorie Comme 'A%')[occ_prema]
↪
REGROUPER SUR(lemme)[<résultat$_1$>]
↪
PAR GROUPE(
lemme TITRE 'Lemme',
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_2$>]
↪
TRI SUR ('o.' DÉCROISSANT)[<résultat$_3$>]
↪
LIMITÉ À(10)[<résultat$_4$>]

Les tableaux 8 et 9 p. 37 et p. 38 fournissent les résultats. Les adjectifs ne comportent qu'un vocabulaire positif. Les adverbes manifestent l'importance du degré, y compris la négation. On note une erreur dans les noms propres (*nettoyage*) et la place des prénoms, tandis que les noms de médicaments ont des fréquences faibles. Les noms manifestent l'importance des soins (le retour au contexte le montre, les prépositions fréquentes *pendant*, *lors de*, *en dehors de*, *entre* sont associées au lemme *soin*), du *contact* visuel (*oeil*), la place particulière accordée aux filles, le contact sonore (*voix*) et ce qui manifeste le tonus, le mouvement (*bras*, *jambe*). Les verbes renvoient aux « actions » typiques des bébés, mais aussi au contact visuel (le fait d'*ouvrir* les yeux est important). On remarque dans les mots dits outils la place des « mots en plusieurs mots » (*dès que*, *lors de*, *en dehors de*, *beaucoup de*). La temporalité joue un grand rôle : *quand*, *lorsque*, *dès que* dans les conjonctions. K est ici le lemme pour des déterminants numéraux (*deux*, *trois*, *quatre*...).

MySQL

La requête pour les A(djectifs) est prototypique des autres :

```
SELECT lemme AS 'Lemme',
    COUNT(*) AS 'o.'
FROM occ_prema
WHERE categorie LIKE 'A%'
GROUP BY lemme
ORDER BY 'o.' DESC
LIMIT 10 ;
```

La condition de restriction aurait aussi pu se formuler de la manière suivante :

```
WHERE categorie REGEXP 'A';
```

Access

La requête SQL Server engendrée par 3 est de la forme :

```
SELECT TOP 10
    Occ_prema.lemme AS Lemme,
    Count(Occ_prema.numero_ordre) AS o
FROM Occ_prema
WHERE (((Occ_prema.categorie) Like 'A*'))
GROUP BY Occ_prema.lemme
ORDER BY Count(Occ_prema.numero_ordre) DESC;
```

TABLEAU 8 – PREMA : premiers lemmes de mots pleins

Adjectifs

Lemme	*o.*
calme	416
petit	375
tonique	334
réactif	243
mignon	149
détendu	124
bon	99
éveillé	85
attentif	73
agréable	71

Adverbes

Lemme	*o.*
très	828
bien	443
ne pas	283
beaucoup	168
un peu	90
peu	89
assez	87
tout	72
plutôt	41
toujours	35

Noms communs

Lemme	*o.*
bébé	988
soin	919
oeil	295
contact	166
fille	118
bras	85
enfant	64
jambe	55
voix	55
tête	51

Noms propres

Lemme	*o.*
PrénomF	60
PrénomM	52
Hypnovel	5
Canadou	2
Narcan	1
F32	1
nettoyage	1
Raniplex	1
Fentanyl	1

Verbes

Lemme	*o.*
être	505
ouvrir	260
pleurer	186
c'est	179
sembler	170
avoir	138
dormir	116
se calmer	114
faire	111
réagir	108

TABLEAU 9 – PREMA : premiers lemmes de mots-outils

Conjonctions

Lemme	*o.*
et	586
mais	234
quand	179
que	123
lorsque	104
car	58
ou	43
dès que	37
donc	21
comme	14

Déterminants

Lemme	*o.*
le	1999
son	537
un	535
ce	62
K	43
peu de	30
de	29
mon	22
beaucoup de	20
tout	17

Prépositions

Lemme	*o.*
à	513
de	513
pendant	358
avec	175
lors de	131
dans	124
en dehors de	110
entre	94
pour	74
par	73

Pronoms

Lemme	*o.*
il	486
le	463
elle	329
on	295
qui	261
je	203
ce qui	34
que	18
cela	17
ce que	17

Ponctuations

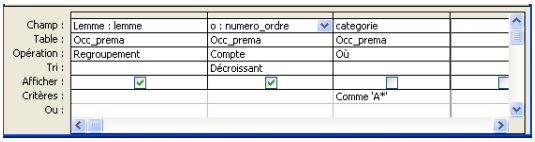
Lemme	*o.*
.	2742
,	1459
-	901
""	138
(	105
)	102
;	61
:	40
...	30
!	21

On se reportera p. 92 pour la limitation aux k premiers résultats. △

Dans l'interface d'Access, l'utilisation des regroupements change subtilement le sens à donner à certaines lignes de l'interface de requête.
Dans une requête sans regroupement, la ligne ⌈Critères⌉ permet d'édicter des conditions de restriction, c'est-à-dire qu'elle correspond à la clauses **WHERE** d'une requête SQL.
Dans une requête avec regroupement, cette ligne correspond à la clause **HAVING**. Si l'on veut utiliser une clause **WHERE**, il faut choisir ⌈Où⌉ dans le menu associé à la ligne ⌈Opération⌉. C'est ce qui est fait en 3 pour se limiter aux lemmes dont la catégorie commence par A, les adjectifs donc, dans le contexte, c'est-à-dire en aval, d'un regroupement en fonction des valeurs de l'attribut lemme (première colonne). On note la différence d'expression du motif entre MySQL et Access :

REGEXP 'A' *vs.* COMME 'A'

3



Solution de l'exercice n°3 p. 30 Isoler les lemmes de mots en plusieurs mots suppose d'abord que le lemme comporte au moins une espace et que la catégorie du lemme commence par une lettre indiquant une catégorie autre que ponctuation (Y) :

```
SELECT lemme
FROM occ_prema
WHERE categorie REGEXP '^[A-S].*'
    AND lemme REGEXP '␣' ;
```

Les résultats sont de la forme :

lemme
de façon
sonde gastrique
à les anges
le reste de le temps
pendant que

Ajouter dans la clause **SELECT** :

```
LENGTH(lemme) - LENGTH(REPLACE(lemme, '␣', '')) + 1
    AS 'nbre␣mots␣simples'
```

complète avec le nombre de mots correspondants :

lemme	*nbre mots simples*
de façon	2
sonde gastrique	2
à les anges	3
le reste de le temps	5
pendant que	2

Il est dès lors aisé d'obtenir les décomptes globaux demandés :

Mots en plusieurs mots	*mots simples correspondants*	*mots simples*
1552	3704	20477

via la requête :

```
SELECT SUM(IF(lemme REGEXP '␣', 1, 0))
          AS 'Mots␣en␣plusieurs␣mots',
       SUM(IF(lemme REGEXP '␣', LENGTH(lemme) - LENGTH(REPLACE(lemme, '␣', '')) + 1,
           0))
          AS 'mots␣simples␣correspondants',
       SUM(IF(lemme REGEXP '␣', 0, 1))
          AS 'mots␣simples'
FROM occ_prema
WHERE categorie REGEXP '^[A-S].*';
```

Solution de l'exercice n°4 p. 30 Le tableau 10 p. 41 donne le résultat. L'auto-jointure sur la table occ_prema a pour condition de rapprochement le fait que le numéro d'ordre de la deuxième instance de la table soit égal au numéro d'ordre de la première instance de la table plus 2 : il faut en effet prendre en compte l'espace qui sépare les deux mots et auquel est consacrée également un ligne de la table. Voir le tableau 7 p. 35 pour les valeurs de la colonne numero_ordre.

R_3^2

JOINTURE(
o1.numero_ordre + 2 = o2.numero_ordre
)[occ_prema ALIAS o1, occ_prema ALIAS o2]
↪
PROJECTION AVEC DOUBLONS(
o1.fiche,
o1.lemme,
o2.lemme
)[<résultat$_1$>]

△

On réfléchira au fait que l'attribut numero_ordre, qui note la séquentialité des occurrences, est strictement nécessaire à la construction de la requête souhaitée. Sans cet attribut, il ne serait pas possible de dire qu'une occurrence donnée suit une autre. On touche du doigt le fait que les SGBD relationnels ne sont pas naturellement destinés à noter des informations séquentielles.

On constate que *puce* est toujours précédé de *petite*. La seule exception, à la fiche 660, n'en est pas vraiment une : un guillemet vient en effet s'interposer entre les deux mots (*Petite "puce" toute maigre...*).

MySQL

```
SELECT o1.fiche,
       o1.lemme,
       o2.lemme
FROM occ_prema AS o1,
       occ_prema AS o2
WHERE o2.numero_ordre = o1.numero_ordre + 2
       AND o2.lemme = 'puce';
```

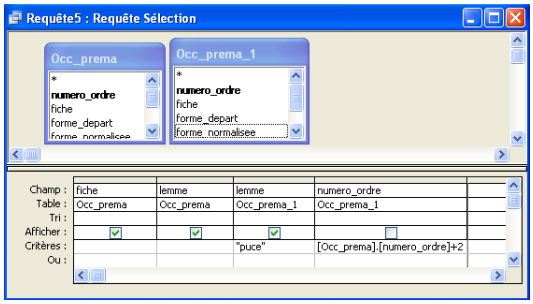
Access

La solution figure en 4.

TABLEAU 10 – PRÉMA : contextes gauches de *puce*

fiche	*lemme*	*lemme*
299	petit	puce
316	petit	puce
488	petit	puce
569	petit	puce
602	petit	puce
660		puce
676	petit	puce
694	petit	puce
706	petit	puce
788	petit	puce
809	petit	puce
853	petit	puce
857	petit	puce
930	petit	puce

4



Solution de l'exercice n°5 p. 31 Le résultat est de la forme :

Sexe	*o.*
Fille	13
Garçon	1

Le principe est simple : il s'agit d'isoler les fiches dont le texte comprend le mot *puce*, et pour les bébés correspondants (via une jointure), de les regrouper par sexe et de compter le nombre d'éléments des agrégats obtenus.

R_4^2

JOINTURE(
f.id_bebe = b.id
) [bebes ALIAS b, fiches_originelles ALIAS f)
↪
RESTRICTION(texte RESSEMBLANT À 'puce')[<résultat$_1$>]
↪
REGROUPER SUR(sexe)[<résultat$_2$>]
↪
PAR GROUPE(
sexe TITRE 'Sexe',
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_3$>]

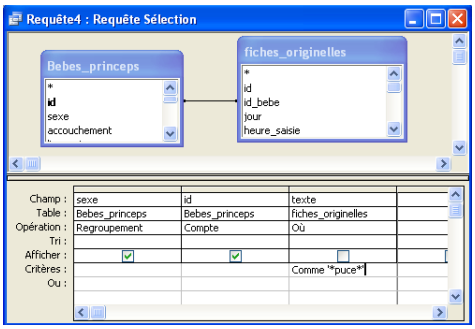
On constate que dans 13 cas sur 14, un bébé qualifié de *puce* est une fille.

MySQL

```
SELECT sexe AS 'Sexe',
       COUNT(*) AS 'o.'
FROM bebes AS b,
       fiches_originelles AS f
WHERE id_bebe = b.id
       AND texte REGEXP 'puce'
GROUP BY sexe ;
```

Access

5



Solution de l'exercice n°6 p. 31 Il y a 29 lemmes ayant 2 catégories ou plus (tableau 11 p. 43). La solution revient à regrouper par lemme et à ne garder que les regroupements pour lesquels la première lettre de l'attribut categorie varie (a plus d'une valeur).

R_5^2

REGROUPER SUR(lemme)[occ_prema]
AVEC(
NOMBRE DE VALEURS DISTINCTES(SOUSCHAÎNE(categorie,
1, 1))
)
↪
PAR GROUPE(
lemme TITRE 'Lemme',
NOMBRE DE VALEURS DISTINCTES(SOUSCHAÎNE(categorie,
1, 1)) TITRE 'catégories'
)[<résultat$_1$>]

Les ambiguïtés catégorielles répertoriées relèvent de cas classiques :

- adjectif/adverbe (*bien*) ;
- adverbe/préposition (*après*, *avant*) ;
- nom/adjectif (*calme*, *câlin*) ;
- verbe/nom (*être*, *toucher*).

MySQL

TABLEAU 11 – PRÉMA : lemmes ambigus

Lemme	*catégories*
TiretChevronFermant	3
antalgique	2
après	2
avant	2
bien	2
bleu	2
bref	2
calme	2
comme	2
câlin	2
de	2
entre	2
gauche	2
grand	2
le	2
limite	2
mal	2
même	2
pendant	3
physique	2
prématuré	2
que	3
réflexe	2
si	2
son	2
sourire	2
toucher	2
tout	3
être	2

```
SELECT lemme AS 'Lemme',
       COUNT(DISTINCT SUBSTRING(categorie, 1, 1)) AS 'catégories'
FROM occ_prema
GROUP BY lemme
       HAVING COUNT(DISTINCT SUBSTRING(categorie, 1, 1)) > 1 ;
```

Access

Solution de l'exercice n°7 p. 31 On crée tout d'abord la table des lemmes ambigus quant à leur partie du discours :

```
CREATE TABLE lemme_prema_a_k_POS
       (SELECT lemme
        FROM occ_prema
        GROUP BY lemme
             HAVING COUNT(DISTINCT SUBSTRING(categorie, 1, 1)) > 1) ;
```

Pour chacun de ces lemmes, on regroupe sur lemme et partie du discours pour obtenir le nombre d'occurrences associé :

```
SELECT lemme,
       SUBSTRING(categorie, 1, 1) AS 'POS',
       COUNT(*) AS 'o.'
FROM occ_prema
WHERE lemme IN
       (SELECT *
        FROM lemme_prema_a_k_POS)
GROUP BY lemme,
       SUBSTRING(categorie, 1, 1) ;
```

On trouve au tableau 12 p. 45 le résultat.

Tableau 12 – Préma : occurrences des lemmes ambigus

lemme	*POS*	*o.*
TiretChevronFermant	C	9
TiretChevronFermant	R	3
TiretChevronFermant	Y	6
antalgique	A	2
antalgique	N	1
après	R	3
après	S	18
avant	R	1
avant	S	6
bien	A	1
bien	R	443
bleu	A	1
bleu	N	2
bref	A	1
bref	R	1
calme	A	416
calme	N	11
comme	C	14
comme	R	2
câlin	A	2
câlin	N	31
de	D	29
de	S	513
entre	S	94
entre	V	1
gauche	A	1
gauche	N	1
grand	A	41
grand	R	19
le	D	1999
le	P	463
limite	N	1
limite	R	1
mal	A	2
mal	R	6
même	A	2
même	R	26
pendant	A	1
pendant	R	1
pendant	S	358
physique	A	2
physique	N	18
prématuré	A	4
prématuré	N	3
que	C	123
que	P	18
que	R	8
réflexe	A	1
réflexe	N	2
si	C	13
si	R	2
son	D	537
son	N	8
sourire	N	6
sourire	V	2
toucher	N	29
toucher	V	43
tout	D	17
tout	P	2
tout	R	72
être	N	6
être	V	505

CHAPITRE II

SUR QUOI TAB(U)LER ?

1. Les tableaux en texte

2. Les tableaux en cellules d'une feuille de calcul

3. Les tables dans une base de données

3.1. ⊕ Calcul d'indications globales

Âge minimum, maximum, moyen, etc. par service pour les infirmières Le regroupement des infirmières selon le service (jour *vs.* nuit) permet de calculer des indications globales sur le nombre d'entités de chaque agrégat, l'âge minimal, maximal et moyen de l'agrégat, l'écart-type de l'âge, et pareillement pour l'ancienneté. On élimine le groupe, réduit à un élément (l'infirmière 41), des infirmières dont on ignore le service en testant l'absence de la marque **NULL**. Cette requête en notation abstraite s'exprime ainsi :

TABLEAU 13 – PRÉMA : âge/ancienneté des infirmières selon le service

Service	*nombre*	*âge min.*	*âge max.*	*âge moy.*	*écart-type âge*	*anc. min*	*anc. max.*	*anc. moy.*	*écart-type anc.*
Jour	29	21	44	29.00	4.79	0.00	19.00	3.50	4.56
Nuit	12	23	40	33.08	5.22	0.00	20.00	9.29	6.53

R_6^2

RESTRICTION(service EST PAS NULL)[infirmieres]
↪
REGROUPER SUR(service)[<résultat$_1$>]
↪
PAR GROUPE(
service TITRE 'Service',
NOMBRE DE LIGNES() TITRE 'nombre',
MINIMUM(age) TITRE 'âge min.',
MAXIMUM(age) TITRE 'âge max.',
MOYENNE(age) TITRE 'âge moyen',
ÉCART-TYPE(age) TITRE 'écart-type âge',
MINIMUM(anciennete) TITRE 'anc. min.',
MAXIMUM(anciennete) TITRE 'anc. max.',
MOYENNE(anciennete) TITRE 'anc. moy.',
ÉCART-TYPE(anciennete) TITRE 'écart-type anc.'
)[<résultat$_2$>]

Le résultat figure au tableau 13 p. 47.

MySQL

FORMAT(<nombre>, k) permet de garder k décimales d'un nombre décimal.

```
SELECT service AS 'Service',
     COUNT(*) AS 'nombre',
     MIN(age) AS 'âge min.',
     MAX(age) AS 'âge max.',
     FORMAT(AVG(age),2) AS 'âge moyen',
     FORMAT(STD(age),2) AS 'écart-type âge',
     MIN(anciennete) AS 'anc. min',
     MAX(anciennete) AS 'anc. max.',
     FORMAT(AVG(anciennete),2) AS 'anc. moy.',
     FORMAT(STD(anciennete),2) AS 'écart-type anc.'
FROM infirmieres
WHERE service IS NOT NULL
GROUP BY service ;
```

Bébés traités et fiches remplies par infirmière On souhaite obtenir pour chaque infirmière son identifiant, son service, le nombre de fiches qu'elle a remplies, le nombre de bébés pour lesquels elle a rempli une fiche et la répartition de ces bébés en filles et garçons. Ces informations proviennent de 3 tables, l'identifiant et le service de la table infirmieres, l'identifiant des bébés traités de la table fiches, et enfin le sexe des bébés de la table bebes. Il faut donc effectuer une jointure entre ces trois tables en les reliant deux à deux, ce qui entraîne deux conditions de rapprochement. La requête abstraite est de la forme :

TABLEAU 14 – PRÉMA : les infirmières, les fiches et les bébés

Infirmière	*service*	*fiches*	*bébés*	*garçons*	*filles*
18	Jour	62	45	22	23
17	Jour	52	40	24	16
22	Jour	46	37	21	16
14	Nuit	44	30	16	14
19	Jour	42	31	19	12
⋮	⋮	⋮	⋮	⋮	⋮
73	Jour	10	8	4	4
62	Jour	9	8	1	7
2	Jour	7	6	3	3
44	Nuit	3	3	2	1
21	Jour	2	2	1	1

R_7^2

JOINTURE(
id_infirmiere = i.id
ET id_bebe = b.id)
[infirmieres ALIAS i,
signaletique_fiches ALIAS s,
bebes ALIAS b]
↪
REGROUPE SUR(i.id)[<résultat$_1$>]
↪
PAR GROUPE(
i.id TITRE 'Infirmière',
service,
NOMBRE DE VALEURS DISTINCTES(s.id) TITRE 'fiches',
NOMBRE DE VALEURS DISTINCTES(b.id) TITRE 'bébés',
NOMBRE DE VALEURS DISTINCTES(SI(sexe = 'Garçon', id_bebe, **NULL**) TITRE 'garçons',
NOMBRE DE VALEURS DISTINCTES(SI(sexe = 'Fille', id_bebe, **NULL**) TITRE 'filles',
)[<résultat$_2$>]
↪
TRI SUR('fiches' DÉCROISSANT, 'bébés' DÉCROISSANT)[<résultat$_3$>]

L'appel à
SI(<condition>, <résultat si condition remplie>, <résultat sinon>)
permet, pour chaque ligne concernant une infirmière donnée, si par ailleurs le bébé correspondant est un garçon, de retourner son identifiant, sinon de retourner la marque **NULL** (et réciproquement pour les filles).

Le tableau 14 p. 48 donne le résultat.

On est obligé d'employer des **noms qualifiés** pour id, nom de colonne présent dans les trois tables mais qui renvoie à des réalités distinctes. À chaque table, en SQL, est alors associé dans la clause **FROM** un **alias de table**, pour obtenir, dans les autres clauses, des noms qualifiés, c'est-à-dire non ambigus. Ainsi, dans la requête qui suit, i.id renvoie à la colonne id de la table infirmieres, dont i est l'alias, tandis que s.id renvoie à l'identifiant d'une fiche, s abrégeant signaletique_fiches. On pourrait tout aussi bien employer infirmieres.id, signaletique_fiches.id et bebes.id. Les autres colonnes ne nécessiteraient pas de nom qualifié : leurs noms ne sont

pas ambigus (age par exemple existe uniquement dans la table infirmieres). Les noms qualifiés sont en particulier nécessaires pour les auto-jointures, puisqu'un même nom d'attribut est employé par deux tables ou plus, les tables mises en relation par l'auto-jointure.

Les requêtes SQL Server engendrées par l'interface Access emploient comme noms qualifiés les noms de colonnes précédés du nom entier des tables correspondantes. En cas d'auto-jointure, un numéro est ajouté à la fin du nom de la deuxième table (et des autres tables, si l'auto-jointure relie plus de 2 tables). En MySQL, on peut, comme dans l'exemple, recourir à des abréviations, qui rendent les noms qualifiés plus aisés à écrire. La convention suivie dans ces pages est en général de prendre l'initiale de la table, quand cette initiale ne prête pas à confusion, initiale éventuellement suivie d'un numéro en cas d'auto-jointure.

MySQL

La fonction **IF** (l. 5 et l. 7) est la réalisation MySQL de la notation SI exemplifiée supra. La fonction **COUNT** ne prend en considération que les valeurs à proprement parler et ignore les marques **NULL**. L'ajout du modifieur **DISTINCT** aboutit à ne prendre en compte que les valeurs distinctes d'identifiant des bébés.

R_8^2

```
SELECT i.id AS 'Infirmière',
       service,
       COUNT(DISTINCT s.id) AS 'fiches',
       COUNT(DISTINCT id_bebe) AS 'bébés',
       COUNT(DISTINCT (IF(sexe = 'Garçon', id_bebe, NULL)))
            AS 'garçons',
       COUNT(DISTINCT (IF(sexe = 'Fille', id_bebe, NULL)))
            AS 'filles'
FROM infirmieres AS i,
       bebes AS b,
       signaletique_fiches AS s
WHERE id_infirmiere = i.id
      AND id_bebe = b.id
GROUP BY i.id
ORDER BY 'fiches' DESC,
       'bébés' DESC ;
```

La requête suivante montre « au ralenti » le fonctionnement de **IF** en MySQL ou de son équivalent **IIF** en SQL Server. Le tableau 15(haut) p. 50 juxtapose en effet la valeur de la colonne sexe et la valeur résultante en fonction des deux **IF**. Dans le premier **IF**, si la valeur est la chaîne 'Garçon', l'identifiant du bébé est gardé, il est remplacé par **NULL** dans le cas contraire. Pour le second **IF**, si la valeur est la chaîne 'Fille', l'identifiant du bébé est gardé, il est remplacé par **NULL** dans le cas contraire.

```
SELECT i.id AS 'Infirmière',
       service, s.id AS 'fiches',
       id_bebe AS 'bébés',
       sexe,
       IF(sexe = 'Garçon', id_bebe, NULL) AS 'garçons',
       IF(sexe = 'Fille', id_bebe, NULL) AS 'filles'
FROM infirmieres AS i,
       bebes AS b,
       signaletique_fiches AS s
WHERE id_infirmiere = i.id
      AND id_bebe = b.id
ORDER BY i.id LIMIT 25 ;
```

TABLEAU 15 – PRÉMA : exemple d'utilisation de **IF** ou **IIF**

Infirmière	*service*	*fiches*	*bébés*	*sexe*	*garçons*	*filles*
1	Nuit	636	77	Garçon	77	**NULL**
1	Nuit	809	98	Fille	**NULL**	98
1	Nuit	853	102	Fille	**NULL**	102
1	Nuit	708	85	Fille	**NULL**	85
1	Nuit	849	101	Fille	**NULL**	101
1	Nuit	688	83	Garçon	83	**NULL**
1	Nuit	736	90	Garçon	90	**NULL**
1	Nuit	999	120	Garçon	120	**NULL**
1	Nuit	731	90	Garçon	90	**NULL**
1	Nuit	719	88	Garçon	88	**NULL**
1	Nuit	872	105	Fille	**NULL**	105
1	Nuit	752	92	Garçon	92	**NULL**
1	Nuit	758	93	Garçon	93	**NULL**
1	Nuit	833	100	Garçon	100	**NULL**
1	Nuit	880	106	Garçon	106	**NULL**
1	Nuit	588	69	Fille	**NULL**	69
1	Nuit	641	77	Garçon	77	**NULL**
1	Nuit	686	83	Garçon	83	**NULL**
2	Jour	68	8	Garçon	8	**NULL**
2	Jour	116	12	Fille	**NULL**	12
2	Jour	91	10	Garçon	10	**NULL**
2	Jour	87	9	Garçon	9	**NULL**
2	Jour	132	14	Fille	**NULL**	14
2	Jour	212	22	Fille	**NULL**	22
2	Jour	214	22	Fille	**NULL**	22

⇓

Infirmière	*service*	*fiches*	*bébés*	*garçons*	*filles*
1	Nuit	18	15	9	6
2	Jour	7	6	3	3

On constate que l'infirmière 1 s'est occupée deux fois des bébés 77, 83 et 90 (trois garçons) et l'infirmière 2 deux fois du bébé 22 (une fille). Le tableau 15(bas) p. 50 donne pour les mêmes infirmières 1 et 2 la synthèse opérée par la première requête.

Access

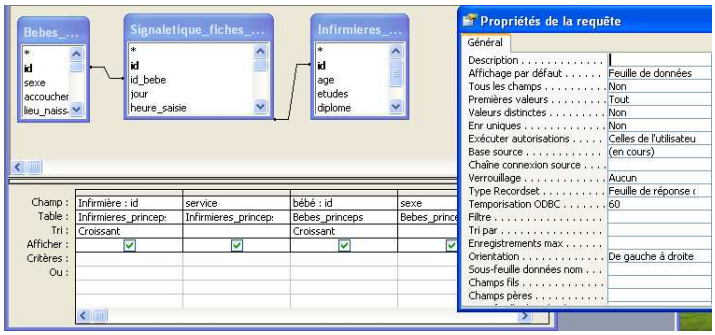
Nous allons fournir plusieurs solutions. L'une se rapproche de celle qui a été choisie pour MySQL. L'autre (le recours à une requête Analyse croisée) est spécifique à Access.

L'objectif est d'associer à chaque infirmière, représentée par son identifiant, son service, le nombre de fiches remplies, le nombre de bébés différents dont elle s'est occupée, et la répartition en garçons et filles (tableau 14 p. 48).

Le point de départ est une jointure qui permet de rassembler les informations concernant les infirmières (identifiant et service), les fiches, et via les fiches, les bébés concernés et leur sexe. C'est le <résultat$_1$> de la R_7^2 p. 48. Les informations retenues sont l'identifiant de l'infirmière, son service, l'identifiant de la fiche, l'identifiant du bébé, son sexe.

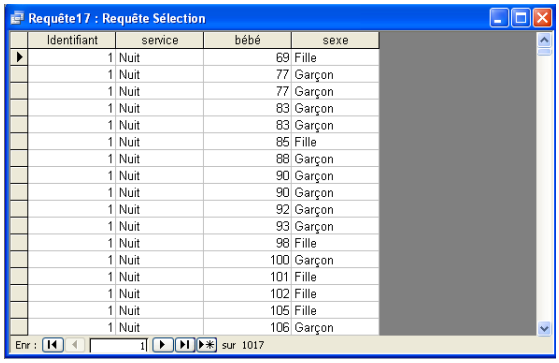
Une première requête, reposant sur cette jointure, opère le tri par infirmière et bébé.

1



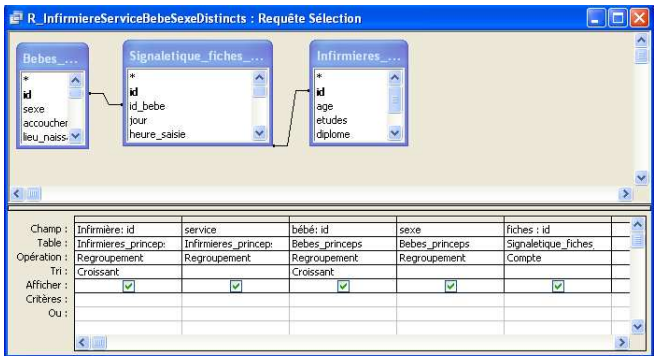
On constate ainsi en 2 que l'infirmière 1 s'est occupée une fois du bébé 69 (une fille) et 2 fois des bébés 77, 83 et 90 (des garçons). Cela correspond au choix (qui est le choix par défaut) de non pour la propriété Valeurs distinctes de la requête en 1. Cela constitue une projection avec doublons.

2



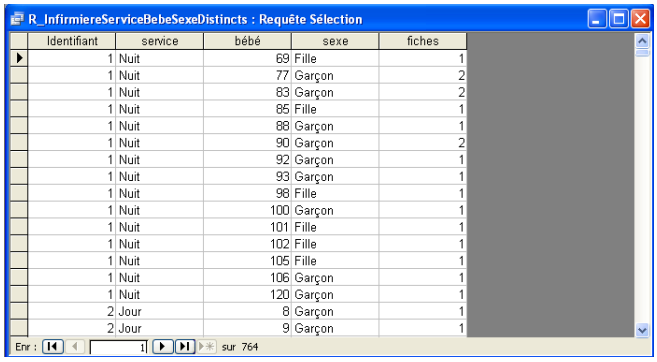
La requête 3 diffère de la requête 1 en ce qu'elle opère non plus un tri mais un regroupement, sur l'identifiant de l'infirmière, son service et l'identifiant du bébé. On constitue donc autant de groupes que de bébés traités par une infirmière donnée pendant un service donné. Ce regroupement permet de faire le compte des fiches correspondant à chacun de ces groupes.

3



On voit ainsi en 4 que le bébé 69, lorsqu'il a été traité par l'infirmière 1, de nuit, a eu une fiche seulement remplie par elle, tandis que le bébé 77 en a eu deux.

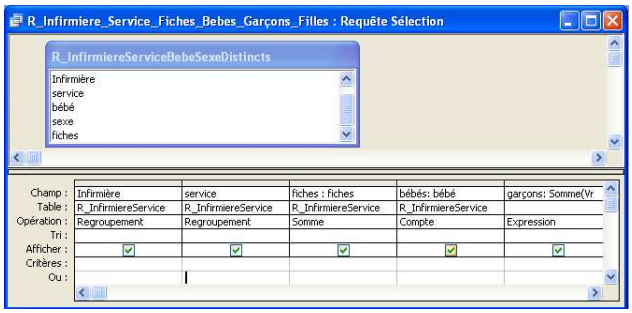
4



Cette requête est mémorisée sous le nom R_InfirmiereServiceBebeSexeDistincts. Cette requête fournit la matière première des deux solutions au problème posé. Les deux requêtes correspondantes utilisent cette requête comme « entrée ».

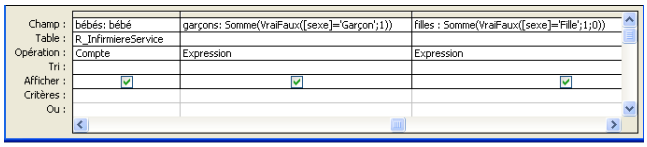
La première solution est tout à fait similaire à celle utilisée en MySQL. On opère un regroupement par infirmière et service. On peut calculer pour chaque groupe le nombre de fiches rédigées (c'est la somme des décomptes des fiches écrites pour chacun des bébés dont s'est occupée l'infirmière) et le nombre de bébés traités (c'est le décompte des bébés figurant dans le groupe). La capture 5 rend compte de ce volet de la requête.

5



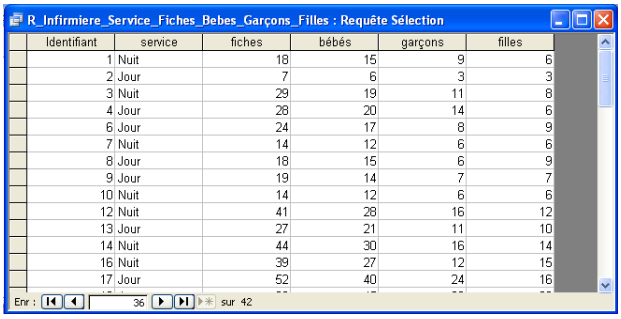
La capture 6, qui reflète le reste des colonnes lors de la mise au point de la requête, montre comment effectuer, au sein de chaque groupe de bébés traités par une infirmière, le décompte du nombre de garçons et du nombre de filles. Pour les garçons, par exemple, si la valeur de l'attribut sexe est 'Garçon', la fonction VraiFaux retourne 1, sinon, elle retourne 0. Le total, par l'appel à Somme, est bien celui des garçons présents dans les fiches rédigés par l'infirmière. La démarche est similaire pour les filles.

6



Le résultat figure en 7.

7



La requête SQL Server engendrée est :

```
SELECT R_InfirmiereServiceBebeSexeDistincts.Infirmière,
       R_InfirmiereServiceBebeSexeDistincts.service,
       Sum(R_InfirmiereServiceBebeSexeDistincts.fiches) AS fiches,
       Count(R_InfirmiereServiceBebeSexeDistincts.bébé) AS bébés,
       Sum(IIf([sexe]='Garçon',1)) AS garçons,
       Sum(IIf([sexe]='Fille',1,0)) AS filles
FROM R_InfirmiereServiceBebeSexeDistincts
GROUP BY R_InfirmiereServiceBebeSexeDistincts.Infirmière,
       R_InfirmiereServiceBebeSexeDistincts.service;
```

IIF(<condition>, <résultat si condition remplie>, <résultat sinon>) est l'équivalent SQL Server de la fonction MySQL **IF**.

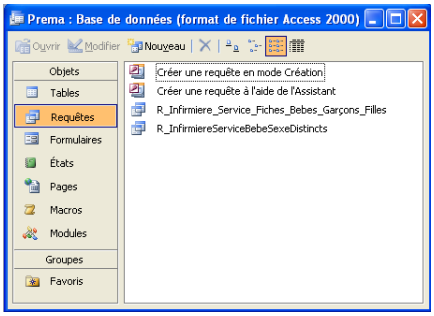
La deuxième solution fait appel à un mécanisme propre à Access, une requête Analyse croisée. Le principe général d'une requête Analyse croisée est le suivant. À partir d'une table ou d'une requête préalable, on constitue des regroupements, sur au plus trois attributs. Chaque groupe représente une ligne de la table résultat. On s'intéresse à un autre attribut, qui peut avoir plusieurs modalités ou plusieurs valeurs. On veut effectuer pour chaque groupe et chaque valeur ou chaque modalité de cet attribut, en croisant donc le groupe et les modalités/valeurs de l'attribut, un calcul : par exemple connaître le nombre de fois où ce groupe a cette modalité/cette valeur pour cet attribut.

Dans le cas présent, chaque groupe est une infirmière associée à un service donné. On veut connaître pour la valeur 'Garçon' comme pour la valeur 'Fille' de l'attribut sexe le nombre de fois où elle se rencontre pour chaque infirmière.

La démarche de création d'une requête Analyse croisée est menée pas à pas, jusqu'à des modifications qui permettent d'obtenir des résultats identiques à ceux du tableau 14 p. 48.

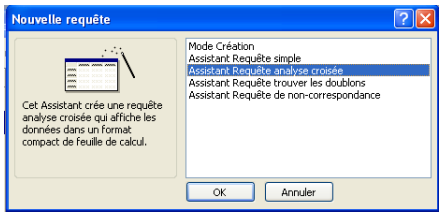
Dans l'onglet [Requêtes], on choisit [Nouveau] dans la barre de menu du haut.

8



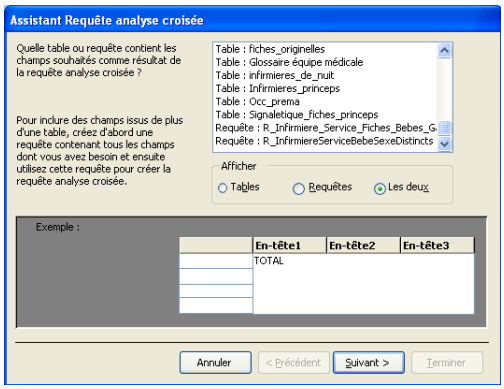
On peut alors choisir [Assistant requête analyse croisée].

9



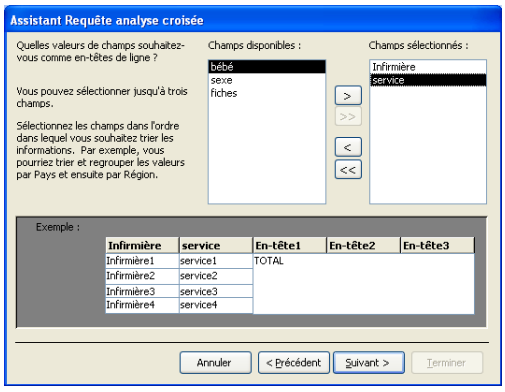
On fournit alors le nom de la table ou de la requête à partir de laquelle effectuer la requête. Il s'agit ici de R_InfirmiereServiceBebeSexeDistincts (cf. supra).

10



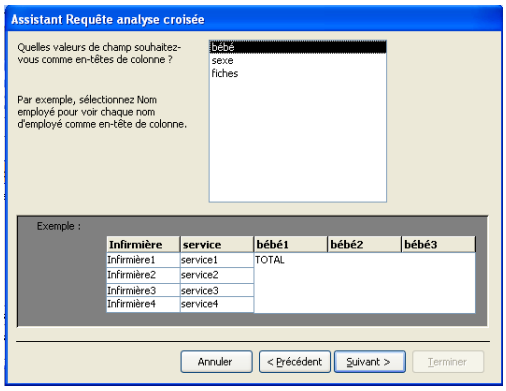
On détermine ensuite le ou les attribut(s) à partir duquel ou desquels seront constitués les groupes, ici Infirmière et service. Cette ou ces colonnes constituera ou constitueront les en-têtes de lignes de chaque groupe.

11



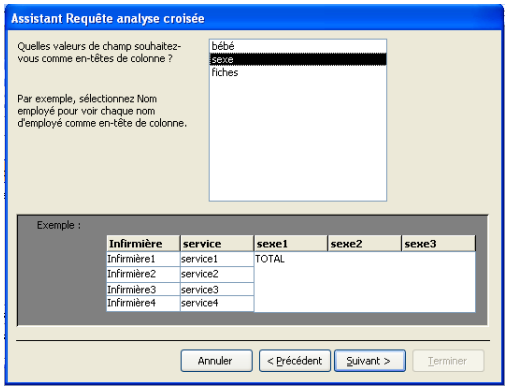
On choisit ensuite la colonne dont on veut croiser les modalités/valeurs avec les groupes constitués. Si l'on choisissait comme en 12 l'attribut bebe, on pourrait obtenir pour chaque infirmière le nombre de fois où elle s'est occupée du bébé 1, du bébé 2... et ainsi de suite jusqu'au bébé 121.

12



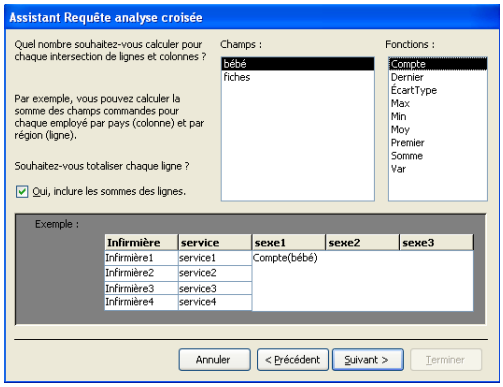
On choisit l'attribut sexe.

13



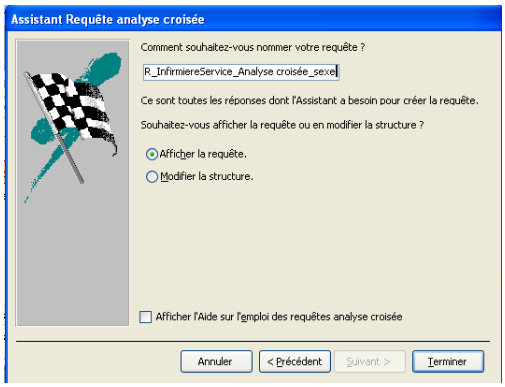
On détermine le calcul à effectuer à l'intersection d'un groupe et d'une modalité de l'attribut. Il s'agit ici d'un décompte.

14



On donne un nom à la requête résultante.

15

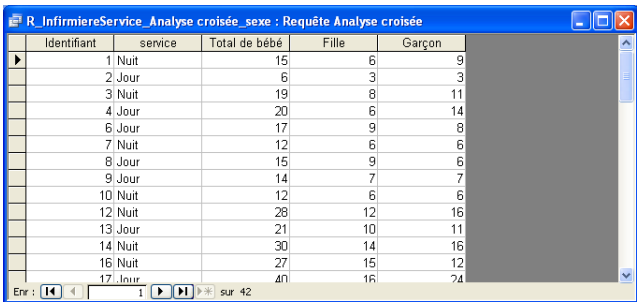


Son équivalent SQL Server est :

```
TRANSFORM Count(R_InfirmiereServiceBebeSexeDistincts.bébé) AS CompteDebébé
SELECT R_InfirmiereServiceBebeSexeDistincts.Infirmière,
       R_InfirmiereServiceBebeSexeDistincts.service,
       Count(R_InfirmiereServiceBebeSexeDistincts.bébé) AS [Total de bébé]
FROM R_InfirmiereServiceBebeSexeDistincts
GROUP BY R_InfirmiereServiceBebeSexeDistincts.Infirmière,
       R_InfirmiereServiceBebeSexeDistincts.service
PIVOT R_InfirmiereServiceBebeSexeDistincts.sexe;
```

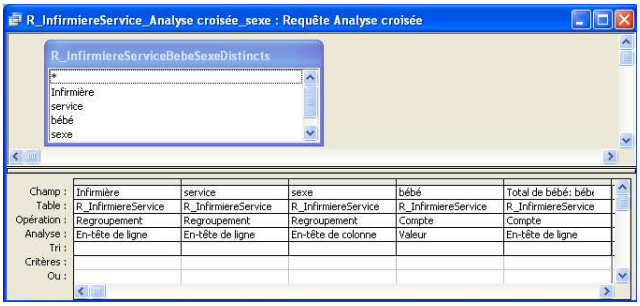
Le résultat figure en 16.

16



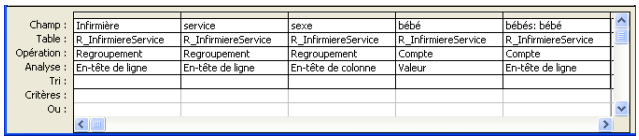
On peut modifier la requête obtenue. On voit dans la visualisation de l'interface apparaître des intitulés propres à ce type de requête : en-tête de ligne et en-tête de colonne.

17



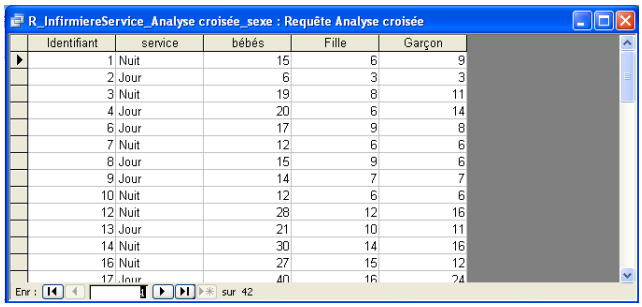
Une première modification, minime, consiste à disposer d'un titre de colonne plus parlant que Total de bébé. On donne comme titre à la colonne correspondante bébés.

18



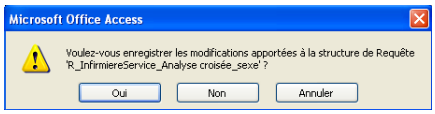
On voit le changement en 19 par rapport à 16.

19



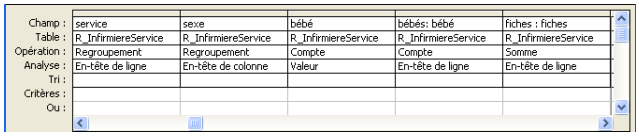
On se voit demander confirmation de la sauvegarde de la requête ainsi modifiée.

20



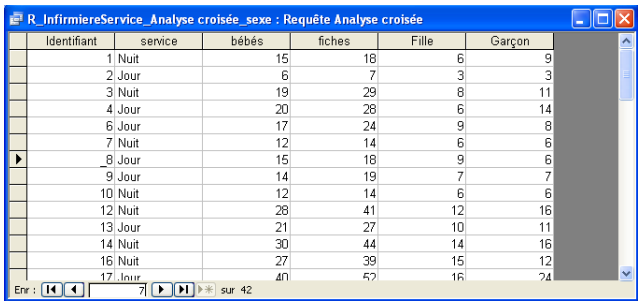
Une deuxième modification, plus subtantielle, consiste à rajouter une colonne, fiches, et à opérer la somme pour chaque infirmière, des fiches correspondant à chacun des bébés.

21



Le résultat est finalement celui qui était souhaité.

22



△

La table/requête d'entrée d'une requête Analyse croisée doit comporter au moins trois attributs. L'un sert au regroupement et fournit l'en-tête de ligne (le regroupement peut se faire sur au plus 3 attributs). Les valeurs/modalités du second attribut constituent les colonnes de la table-résultat. Ce sont les valeurs, pour un troisième attribut, du croisement d'un des regroupements avec une des valeurs/modalités de l'attribut en colonnes qui sert d'entrée au calcul dont le résultat figurera à la croisée.

4. Bases de données : aperçu

CHAPITRE III

EXTRAIRE ET TRIER LES INFORMATIONS

1. Opérations et mise en œuvre

1.1. ⊕ Démarrer/quitter, ouvrir/fermer une base de données

MySQL

Sous Unix/Linux, en mode ligne de commande, l'ouverture d'une base de données se fait par la commande :

```
mysql -u <utilisateur> -p <base de données>
```

comme dans :

```
mysql -u habert -p prema
```

qui indique que l'utilisateur (`-u`) `habert` veut utiliser la base `prema` et qu'il va pour ce faire fournir un mot de passe (`-p`).

Une invite (*prompt*) apparaît :

```
mysql>
```

Il est possible également de lancer d'abord MySQL :

```
mysql -u habert -p
```

puis d'indiquer la base que l'on souhaite ouvrir :

```
mysql>USE prema;
```

Dans tous les cas, l'utilisateur doit avoir des droits sur la base visée, soit de simple consultation, soit de consultation et modification.

Taper `quit` permet de quitter MySQL.

Le passage à la ligne – touche ⌈Entrée⌉ – après le point virgule qui marque la fin d'une requête déclenche l'exécution de cette requête.

Sous Linux/Unix, une base de données correspond à un répertoire (par exemple `/var/lib/mysql/prema`), et chaque table à un ou plusieurs fichiers dans ce répertoire. △

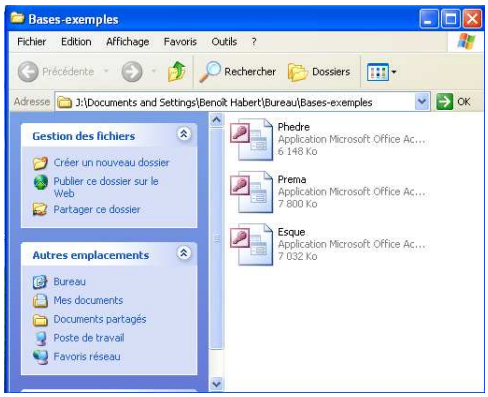
Access

Une base de données correspond à un seul fichier volumineux, se terminant par .mdb (pour Microsoft Data Base). △

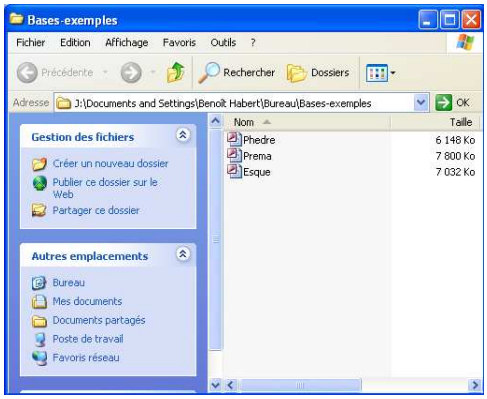
L'ouverture d'une base de données s'effectue en lançant Access et ensuite en ouvrant une base existante : [Fichier | Ouvrir].

Elle s'effectue aussi en double-cliquant sur l'icône de cette base 1, icône qui comprend une clé, ou sur son nom 2 (par exemple Prema.mdb) dans le dossier où la base est présente. Dans ce dernier cas, l'extension du fichier, c'est-à-dire le point et les caractères qui le suivent (ici .mdb), est ou non visible suivant le paramétrage de l'affichage des noms de fichiers, comme on le constate en comparant 2 et 3 (le réglage sous jacent est expliqué à la fin de cette section).

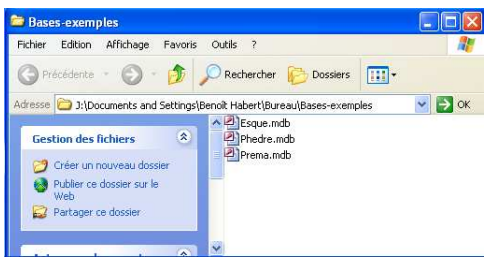
1



2

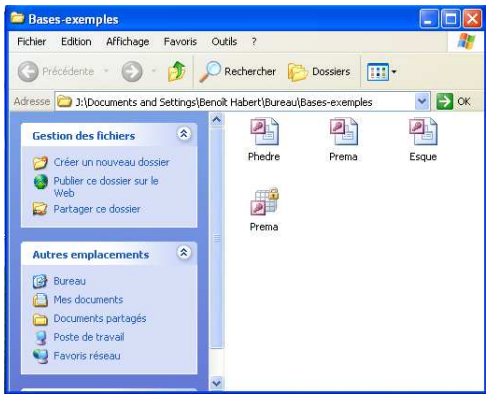


3



La base une fois ouverte, une icône supplémentaire dans le dossier, avec un cadenas 4, indique que la base est verrouillée : est autorisé un seul accès à la fois à une base donnée.

4



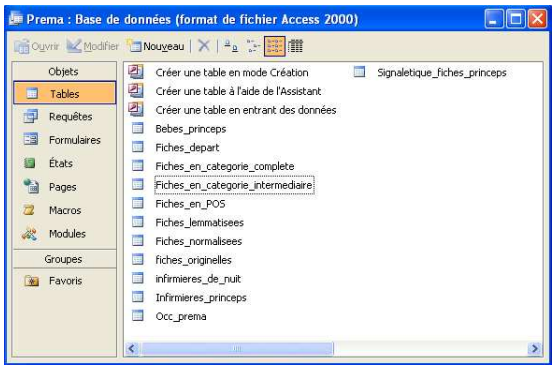
On ferme la base de données via ⌈Fichier | Fermer⌉. On peut également quitter Access (et fermer du même coup la base de données) par ⌈Fichier | Quitter⌉ ou en cliquant la croix rouge à droite de la barre de menu du haut.

Après ouverture d'une base, apparaît une fenêtre de navigation 5 6 entre les opérations pour la base en question :

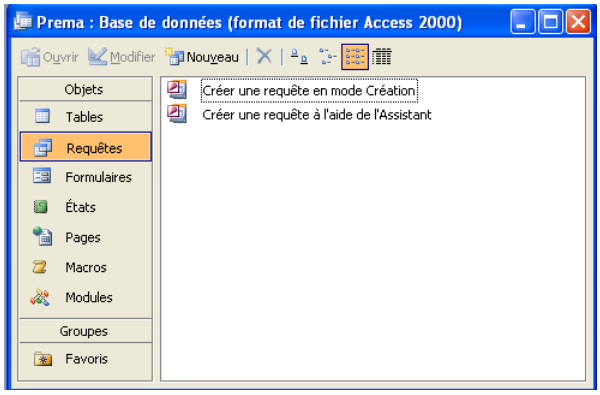
- création, modification et affichage des tables : onglet ⌈Tables⌉ 5 ;
- création, exécution, mémorisation et modification de requêtes : onglet ⌈Requêtes⌉ 6 ;
- mise en place de formulaires d'entrée d'informations dans une base : onglet ⌈Formulaires⌉ ;
- mise en place d'impressions d'informations : onglet ⌈États⌉ ;
- etc.

Ce sont les deux premiers volets qui sont privilégiés dans ces pages.

5

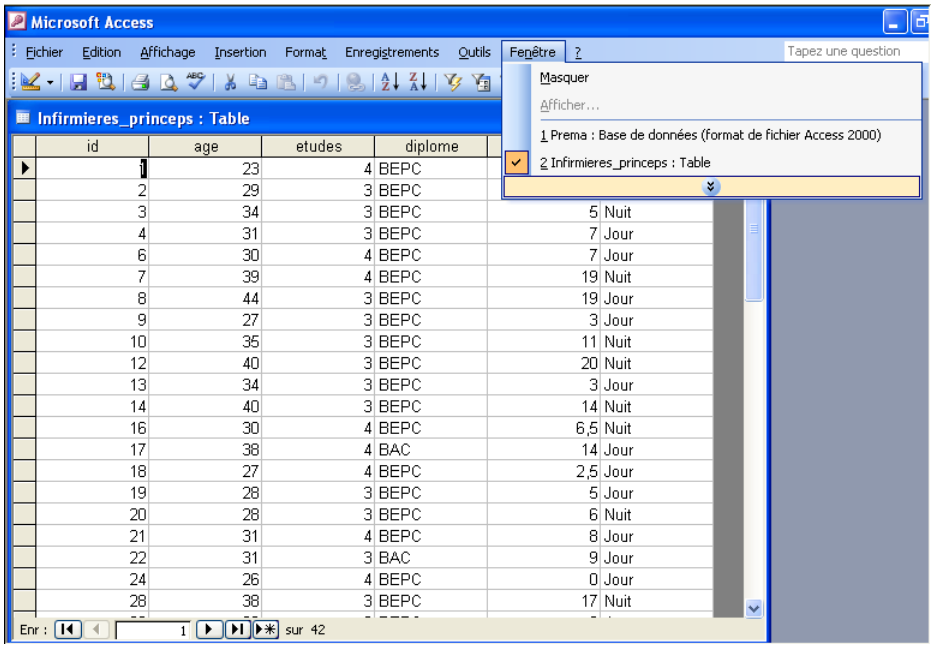


6



Lorsque plusieurs fenêtres sont ouvertes, comme en 7 où la table infirmieres est ouverte en mode feuille de données (affichage standard d'une table), on peut « retrouver » cette fenêtre de navigation à partir du menu ⌈Fenêtre⌉ de la barre du haut, en choisissant l'entrée correspondant à la base de données : ici le choix 1 ⌈Préma : Base de données (format de fichier Access 2000)⌉.

7

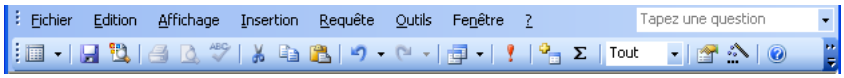


On peut également cliquer sur l'icône idoine de la barre de menu du haut : 8.

8

Activer le bouton ⌈Requêtes⌉ donne différents moyens de formuler des requêtes. Ce choix s'accompagne d'une transformation de la barre de menu du haut 9.

9



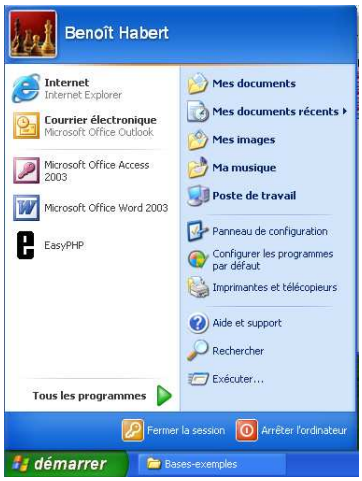
Rendre les extensions liées à une application visibles sous Windows

Un fichier sous Windows comprend un nom suivi d'une extension : trois caractères précédés d'un point. Pour rendre la lecture des noms de fichiers plus simple, les extensions sont par défaut omises dans la présentation, au sein d'un dossier, de noms de fichiers.

Windows permet d'associer à une extension donnée une application : par exemple un fichier se terminant par .pdf sera ouvert avec Acrobat Reader. Windows permet également de choisir si l'on veut ou non rendre visible l'extension des fichiers ayant une extension donnée.

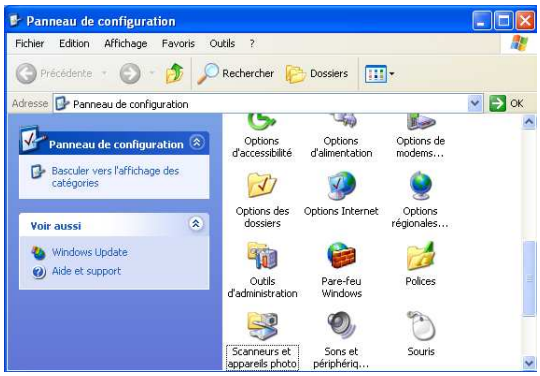
On utilise pour cela ⌈Démarrer | Panneau de configuration⌉.

10



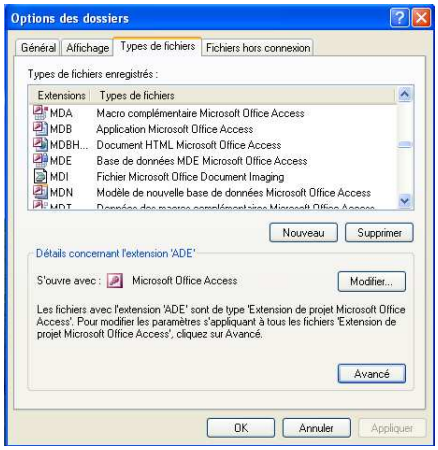
On choisit [Options des dossiers].

11

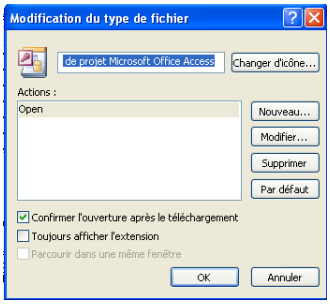


On recherche l'extension que l'on veut rendre visible avec l'ascenseur.

12



Via le choix [Avancé], on peut indiquer qu'on veut que les fichiers ayant cette extension aient cette extension directement visible :



On obtient alors un résultat du type 3.

1.2. ⊕ Forme générale d'une requête SQL

La forme générale des requêtes SQL pour extraire des données est la suivante :

```
SELECT <attribut(s) conservé(s) ou calculé(s)>
FROM <table(s) source(s)>
[WHERE <condition(s) de restriction ou de rapprochement>]
[GROUP BY <condition(s) de regroupement>
      [HAVING <condition(s) sur les agrégats >]]
[ORDER BY <attribut(s) servant au tri> [DESC|ASC]] ;
```

Les crochets encadrent une partie optionnelle.

Pour les différents composants d'une requête, on parle de **clauses** (clause **SELECT**, clause **FROM**, clause **WHERE**, etc.) :

- la clause **FROM** précise la ou les table(s) qui va/vont servir d'entrée (la combinaison de plusieurs tables est présentée au chapitre VII § 2) ;
- la clause **WHERE** édicte les critères que doivent respecter les lignes pour être prises en compte (restriction – chapitre III § 3) ; elle permet également d'indiquer des conditions de rapprochement entre tables lors de jointures (chapitre VII § 2) ;
- la clause **SELECT** détermine les colonnes retenues (projection – chapitre III § 4) et les colonnes issues d'un calcul sur les colonnes existantes (chapitre IV § 3) ;
- la clause **GROUP BY** permet les regroupements sur des combinaisons de valeurs d'attributs (chapitre IV § 2) ;
- la clause **ORDER BY** permet de trier la table résultat (chapitre III § 7).

△ L'équivalent de LIMITÉ À(k) qui permet de ne garder que k lignes dans les résultats s'exprime par **LIMIT** k en MySQL et par **TOP** k en SQL Server. Cependant cette indication se place juste après le mot-clé **SELECT** en SQL Server et en dernière position de la requête en MySQL.

Dans les exemples fournis, les clauses sont généralement isolées sur des lignes distinctes pour faciliter leur repérage et leur compréhension, même si la présence ou non d'espaces ou de passages à la ligne supplémentaires ne change pas le comportement du SGBD.

Le point-virgule signale la fin de la requête, comme dans :

```
SELECT * FROM infirmieres WHERE service = 'Nuit' ;
```

Sous Linux/Unix, une requête SQL peut d'ailleurs être entrée sur plusieurs lignes. L'invite change alors pour la 2ème ligne et les suivantes, comme dans :

```
mysql> SELECT *
–> FROM infirmieres
–> WHERE service = 'Nuit' ;
```

où mysql> représente l'invite initiale, et –> l'invite secondaire.

MySQL

≃ caractéristiques des infirmières de nuit

R_1^1 *t. 1* p. 60

RESTRICTION(service = "Nuit")[infirmieres]

La requête correspondante est :

```
SELECT id, age, etudes, diplome, anciennete, service
FROM infirmieres
WHERE service = "Nuit";
```

Elle peut aussi prendre la forme :

```
SELECT *
FROM infirmieres
WHERE service = "Nuit" ;
```

où l'étoile est une abréviation pour toutes les colonnes de la ou des tables considérées.

L'absence de restriction, c'est-à-dire la table entière, s'obtient en omettant la clause **WHERE**, ce qui donne dans l'exemple présent :

```
SELECT id, age, etudes, diplome, anciennete, service
FROM infirmieres ;
```

ou

```
SELECT *
FROM infirmieres ;
```

1.3. ⊕ Forme générale d'une requête sous Access

L'accès à une table entière s'obtient en double-cliquant sur le nom de cette table dans le volet [Tables] de la fenêtre de navigation. Cette présentation est dénommée **feuille de données**. Cliquer sur la croix en haut à droite de la table ferme la fenêtre y donnant accès. L'utilisateur peut modifier l'apparence de la table : élargissement, rétrécissement ou masquage de colonnes, tris, etc. En ce cas, au moment de la fermeture de la fenêtre donnant accès à la table, il lui est demandé s'il souhaite ou non conserver cette mise en forme. L'utilisateur peut également modifier le contenu de la table : changement de la valeur de cellules, suppression de lignes ou de colonnes, etc., voire sa structure (en supprimant des colonnes).

Pour créer une requête comme :

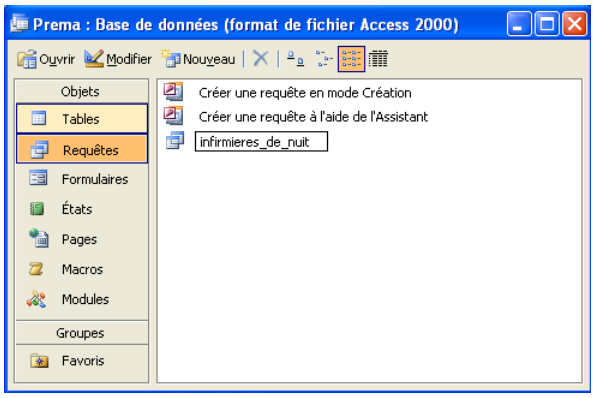
≃ caractéristiques des infirmières de nuit

R_1^1 *t. 1* p. 60

RESTRICTION(service = "Nuit")[infirmieres]

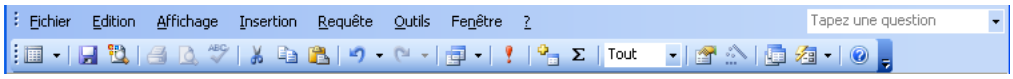
cliquer sur l'onglet [Requêtes] de la fenêtre de navigation dans la base de données, puis sur [Créer une requête en mode Création] 14.

14



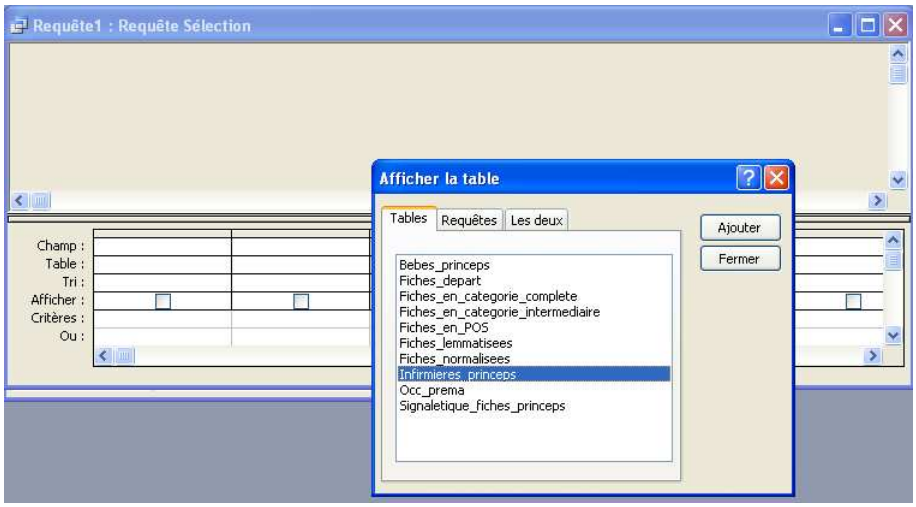
La barre de menu du haut se transforme : 15.

15

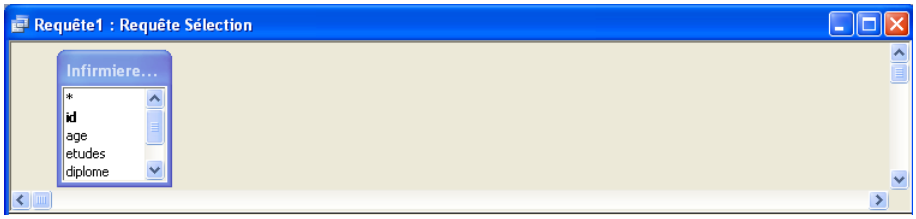


– La première étape consiste à choisir la ou les table(s) ou requête(s) concernée(s) par la nouvelle requête. Les tables et requêtes sont affichées. On ajoute celle(s) qu'on juge pertinente(s) 16. La ou les table(s) et requête(s) choisie(s) apparaissent dans la sous-fenêtre du haut 17 (éventuellement reliées par des traits marquant des **relations** au sens d'Access, c'est-à-dire des jointures).

16



17

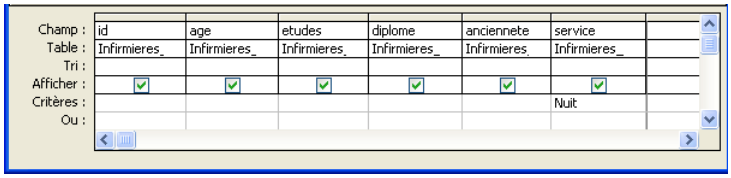


– On ajoute les attributs que l'on souhaite voir figurer dans la table résultat et/ou qui vont apparaître dans des conditions de restriction. Trois méthodes s'offrent :

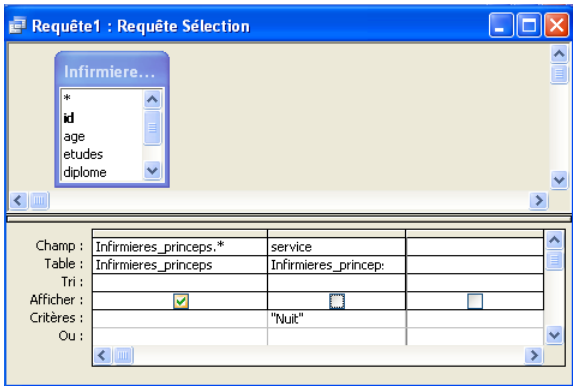
1. On double-clique sur les attributs souhaités. Ils apparaissent dans la sous-fenêtre du bas 18.
2. Ajouter un attribut dans la requête peut s'effectuer également en cliquant sur la première ligne ⌈Champ⌉ d'une colonne : un menu déroulant apparaît permettant de sélectionner l'attribut souhaité.
3. On peut également faire glisser un nom de la table affichée dans la sous-fenêtre du haut vers une colonne de la sous-fenêtre du bas.

– On peut aussi sélectionner tous les attributs en double-cliquant sur l'étoile en première place dans la liste des attributs d'une table/requête affichée dans la sous-fenêtre du haut. Si nécessaire, on double-clique également sur le ou les attribut(s) utilisés pour une restriction, mais en décochant pour eux la boîte ⌈Afficher⌉ (pour qu'ils ne le soient pas deux fois) 19.

18



19

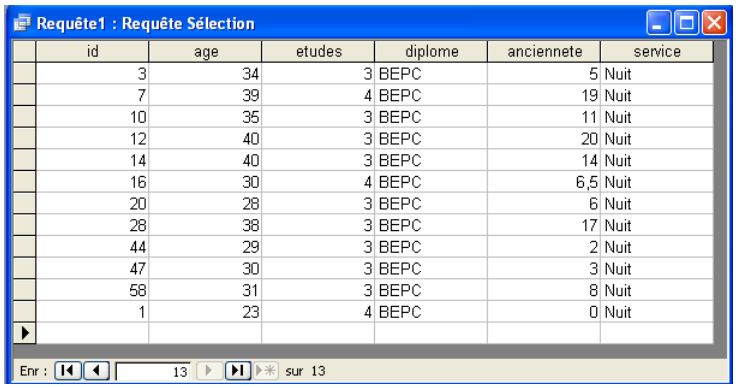


– Cocher la boîte ⌈Afficher⌉ pour une colonne donnée 18 indique que la colonne en question figurera dans la table-résultat. Dans le cas contraire, la colonne sera uniquement utilisée pour une condition de restriction mais ne figurera pas dans la table-résultat.
– La ligne ⌈Critères⌉ permet d'édicter des conditions de restriction. Dans le cas présent, indiquer Nuit dans la colonne service 18 revient à effectuer une restriction sur les infirmières travaillant de nuit. On peut entourer cette valeur textuelle de guillemets ou d'apostrophes (mais pas d'un mélange des deux). Si l'on ne le fait pas, Access rajoutera des guillemets, comme en 19.
– Cliquer sur le point d'exclamation rouge du menu du haut 20 ou passer par ⌈Requête | Exécuter⌉ exécute la requête. La table résultat apparaît en mode feuille de données 21. La ligne du bas indique le nombre de lignes de la table. Les triangles simples permettent de se déplacer d'une ligne, ceux assortis d'une barre de se déplacer en tout début ou en toute fin de table.

– Lorsqu'on clique sur la croix en haut à droite de la fenêtre contenant la table, la feuille de données, on se voit demander si l'on veut mémoriser la requête correspondante. Si l'on répond positivement et qu'on donne un nom à la nouvelle requête, ce nouveau nom apparaît dans les requêtes de la fenêtre de navigation dans la base. Double-cliquer sur ce nom « rejoue » la requête correspondante : la table résultat réapparaît.

20

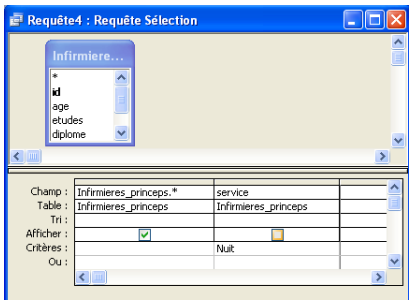
21



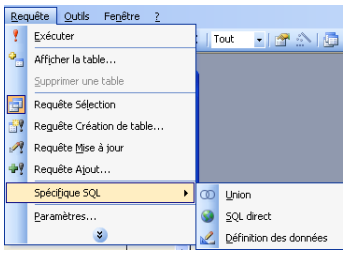
1.4. ⊕ De l'interface Access à SQL Server

Lorqu'on formule une requête via l'interface 22, on peut basculer en mode SQL Server, via ⌈Requête ⌉ Spécifique SQL ⌉ SQL direct⌉ : 23.

22



23



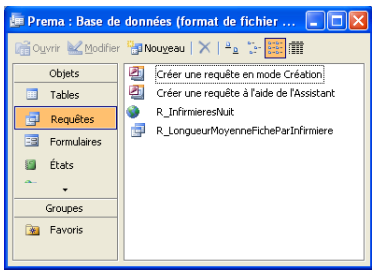
La requête automatiquement engendrée apparaît alors : 24.

24

Les noms de colonnes sont généralement qualifiés, c'est-à-dire qu'ils ont pour préfixe le nom de la table dont ces colonnes proviennent. L'engendrement conduit également à des parenthésages qui gênent un peu la lecture dans la clause **WHERE**.

On exécute la requête en cliquant sur l'icône de déclenchement d'une requête, le point d'exclamation rouge de la barre de menu du haut. Si l'on choisit de mémoriser la requête pour pouvoir la rejouer, on constate que son icône est un petit cercle, tandis qu'une requête créée via l'interface est précédée par deux petits rectangles qui se superposent : 25.

25



2. Un résultat = une table

2.1. ⊕ Accueillir dans une nouvelle table le résultat d'une requête

MySQL

On crée tout d'abord la structure de la table destinée à accueillir le résultat (chapitre X § 2) :

```
CREATE TABLE `infirmieres_nuit` (
  `id` int(11) NOT NULL default '0',
  `age` int(11) NOT NULL default '0',
  `etudes` int(11) NOT NULL default '0',
  `diplome` varchar(50) NOT NULL default '',
  `anciennete` decimal(4,2) NOT NULL default '0.00',
  `service` varchar(50) default NULL,
   PRIMARY KEY (`id`) ) ;
```

On insère ensuite dans cette table le résultat de la requête :

```
INSERT INTO infirmieres_nuit
    SELECT *
    FROM infirmieres_princeps
    WHERE service = "Nuit" ;
```

△

La requête enchâssée (**SELECT**...) doit produire une table compatible avec la table qui vient d'être créée : même nombre de colonnes, types de données compatibles colonne à colonne.

Il est également de procéder en une seule opération, de créer la table à la volée en aval :

```
CREATE TABLE infirmieres_nuit
      (SELECT *
      FROM infirmieres_princeps
      WHERE service = 'Nuit') ;
```

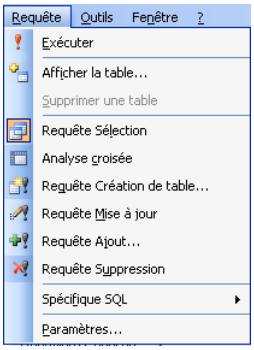
C'est un exemple de requête enchâssée, puisque la création de table utilise une sous-requête (une restriction en l'occurrence).

Le type des attributs de la table créée à la volée est celui des attributs retenus dans la sous-requête ou se déduit des calculs effectués sur les attributs utilisés (la somme des valeurs de deux attributs à valeurs numériques entières donnera naissance à un attribut de type numérique entier, par exemple).

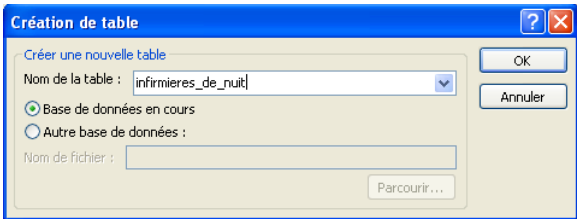
Access

Lorsqu'on est en train de composer une requête, le choix de ⌈Requête | Requête création de table⌋ 26 conduit à fournir un nom de table 27 pour accueillir le résultat de la requête.

26

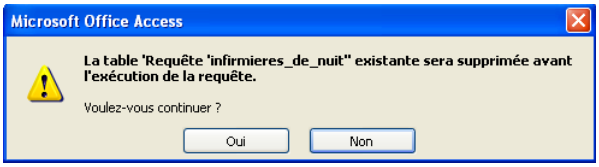


27



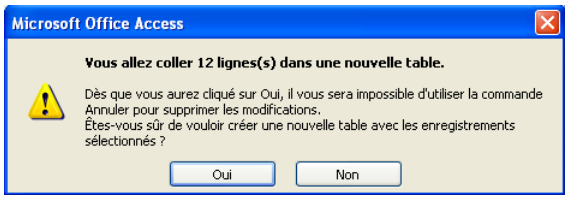
L'exécution de la requête « écrase » toute table existante de même nom si l'on confirme le choix 28.

28



La nouvelle table, dont le nombre de lignes est indiqué 29, apparaît dans les tables du volet ⌈Tables⌋ de la fenêtre de navigation 30.

29



30

2.2. ⊕ Pouvoir « rejouer » une requête

MySQL

On crée une **vue**, ce qui revient à donner un nom à une requête et donc à pouvoir s'en reservir (la rejouer), y compris en position de table d'entrée d'une autre requête :

```
CREATE VIEW vue_infirmieres_nuit
AS
    SELECT *
    FROM infirmieres_princeps
    WHERE service = "Nuit" ;
```

Cette possibilité n'est pas disponible dans toutes les versions de MySQL. Elle ne l'est en particulier pas dans la version 4.1.11 qui a été utilisée pour cet ouvrage. △

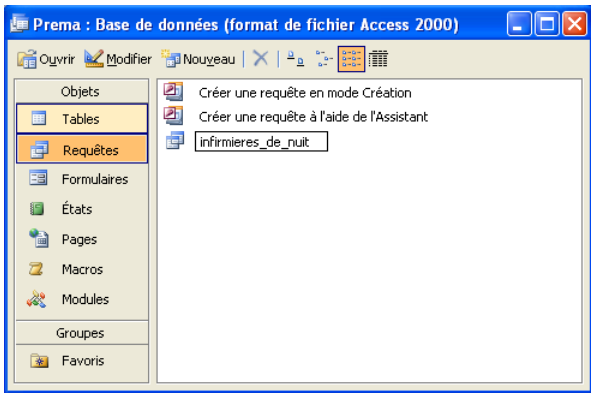
Access

Lorsqu'on crée une requête et qu'on l'exécute, au moment de fermer la feuille de données résultat, on doit choisir de conserver ou non la requête, en lui donnant un nom. Une fois sauvegardée, une requête peut être « rejouée » à volonté, en double-cliquant sur son nom.

Par ailleurs, une requête mémorisée se comporte comme une table « ordinaire », qui peut être l'objet de nouvelles requêtes ou être combinée avec d'autres. Lorsqu'on crée une requête via l'interface graphique, on peut prendre en entrée des tables, des requêtes ou toutes combinaisons des deux. Voir 16 supra.

On peut donc disposer d'un arsenal de requêtes sur une base de données pour répondre à des interrogations récurrentes.

À mémoriser ainsi des requêtes, on se retrouve rapidement disposer d'un bon nombre de requêtes : des noms explicites sont nécessaires pour pouvoir effectivement rejouer ces requêtes en sachant à quoi elles renvoient 31. △



PRÉMA

PHÈDRE

ESQUE

3. Restreindre : un sous-ensemble de lignes

3.1. ⊕ Restriction : définition relationnelle

La **restriction** est une opération qui prend comme argument une relation et qui produit une relation de même schéma comprenant uniquement les n-uplets de la relation qui vérifient la condition fournie également comme argument.

3.2. ⊕ Restriction : réalisations

MySQL

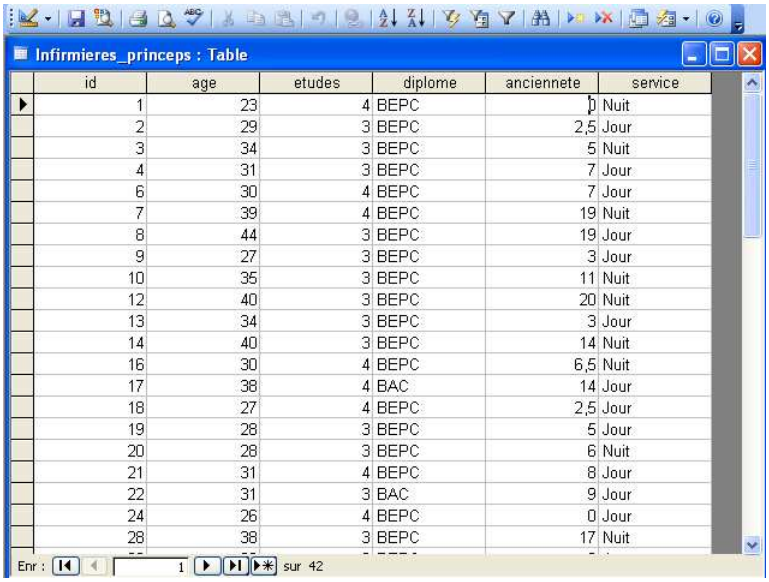
L'exemple de la restriction a servi de support supra p. 63 à la présentation de la formulation d'une requête.

Access

L'exemple de la restriction a servi de support supra p. 64 à la présentation de la formulation d'une requête.

Filtre Une deuxième manière d'arriver à une restriction aux lignes souhaitées est l'utilisation d'un **filtre**. On sélectionne la valeur souhaitée d'un attribut. Par exemple la valeur 0 pour l'ancienneté des infirmières 32.

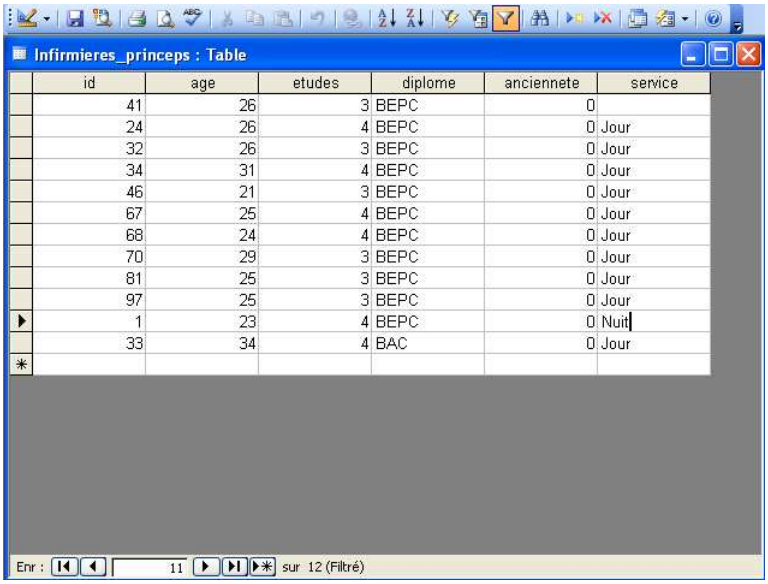
32



Cliquer sur l'icône de filtrage, la conjonction d'un entonnoir et d'un éclair dans la barre de menu du haut 33 (message-bulle [Filtrer par sélection]), aboutit à restreindre la table aux seules lignes présentant cette valeur cible 34.

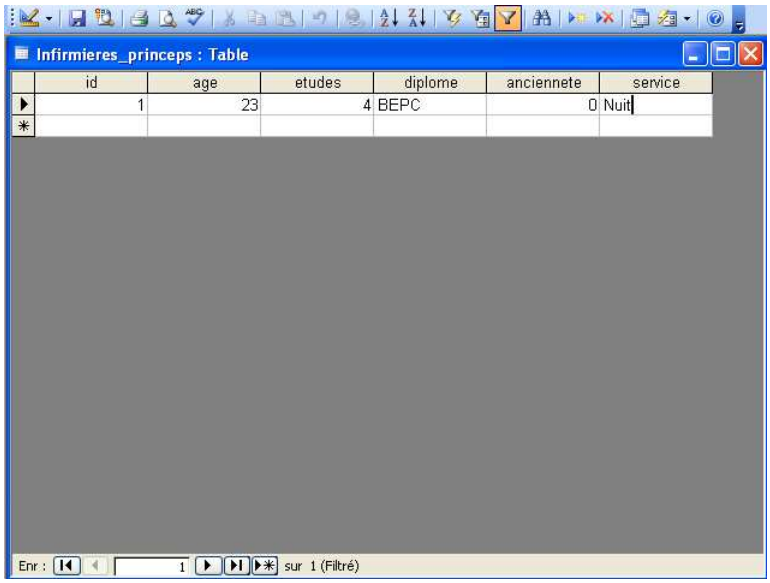
33

34



Si l'on sélectionne alors une valeur d'un autre attribut et que l'on clique sur l'icône de filtrage, on applique une deuxième restriction, en aval de la première 35. C'est l'équivalent d'un ET logique entre deux conditions de restriction. On peut continuer ainsi les restrictions sur des attributs distincts jusqu'à avoir isolé les informations jugées pertinentes. . . ou déboucher sur la table vide.

35



Pour revenir à la table de départ, sans restriction, on clique sur l'icône idoine, l'entonnoir isolé de la barre de menu du haut 36 (message-bulle ⌈Supprimer le filtre⌉).

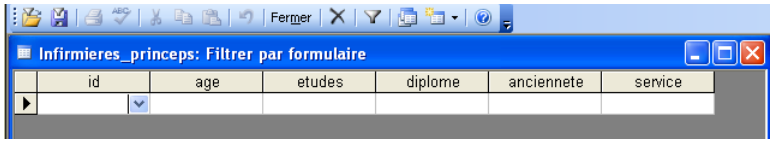
36

Formulaire Lorsqu'on examine une table en mode feuille de données, une autre solution, en utilisant l'entonnoir associé à un classeur de la barre de menu du haut 37 (message-bulle ⌈Filtrer par formulaire⌉), est l'utilisation d'un **formulaire**.

37

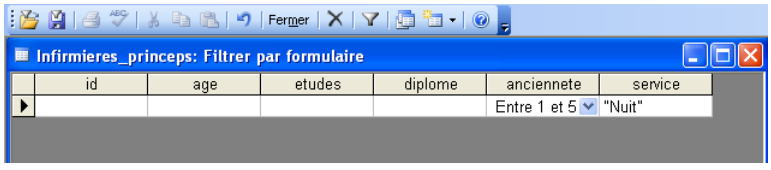
Un formulaire comprend les attributs de la table examinée 38.

38



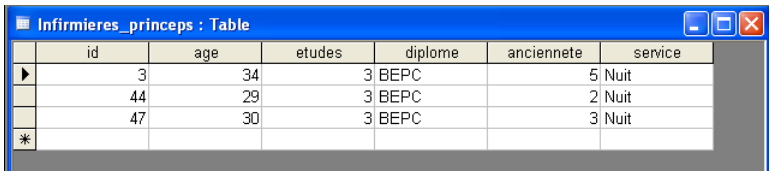
Il permet la conjonction directe de conditions de restriction 39, de la même manière qu'en mode ⌈Création de requêtes⌉.

39



Le résultat, après un clic sur l'entonnoir isolé de la barre de menu 41, est similaire à ceux de la requête correspondante 40.

40



id	age	etudes	diplome	anciennete	service
3	34	3	BEPC	5	Nuit
44	29	3	BEPC	2	Nuit
47	30	3	BEPC	3	Nuit

Cliquer à nouveau sur l'entonnoir isolé de la barre de menu 41 fait revenir à la table entière en mode feuille de données.

41

△

On notera que la condition de restriction d'un formulaire n'est pas sauvegardable : elle ne peut pas être rejouée.

△

Formulaire a donc deux sens pour Access. Il s'agit d'une part d'une manière de filtrer rapidement des lignes. C'est le sens qui vient d'être exemplifié. Il s'agit d'autre part de faciliter l'entrée par l'utilisateur de nouvelles informations dans la base de données.

3.3. ⊕ Combinaison de conditions de restrictions

MySQL

La requête :

```
SELECT *
FROM infirmieres
WHERE anciennete = 0 AND service = 'Nuit' ;
```

retient les infirmières sans ancienneté et travaillant de nuit (il n'y en a qu'une, avec 1 comme id). Elle équivaut à :

R_9^2

RESTRICTION(
anciennete = 0 ET service = "Nuit"
)[infirmieres]

On peut utiliser dans les mêmes circonstances la négation :

```
WHERE NOT(anciennete = 0 AND service = 'Nuit');
```

```
WHERE anciennete NOT BETWEEN 0 AND 5 ;
```

Avec la première condition de restriction, la requête équivaut à :

≃ caractéristiques des infirmières qui ne travaillent pas de nuit et dont l'ancienneté n'est pas égale à 0

Avec la seconde, elle revient à :

≃ caractéristiques des infirmières dont l'ancienneté n'est pas comprise entre 0 et 5 ans (bornes incluses)

Access

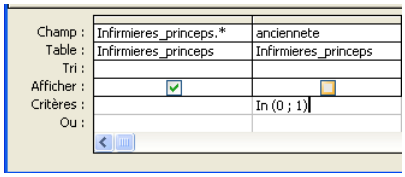
La restriction sur une colonne se place dans la ligne ⌈Critères⌉ de la colonne correspondante dans la fenêtre du bas d'expression des requêtes : 43. Le point-virgule sépare les valeurs souhaitées dans 42 comme dans 45. Les guillemets ou l'apostrophe encadrent les valeurs textuelles en 45. La négation s'obtient en préfixant de ⌈Pas⌉ la ou les valeur(s) à exclure : 44, 46.

≃ caractéristiques des infirmières ayant soit 0 soit 1 an d'ancienneté

R_6^1 *t. 1* p. 63

RESTRICTION(anciennete PARMI (0, 1))[infirmieres]

42

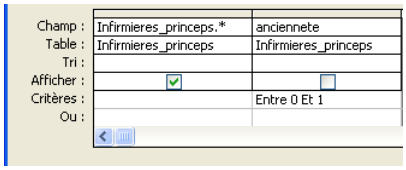


≃ caractéristiques des infirmières dont l'ancienneté est comprise entre 0 et un an (bornes incluses)

R_7^1 *t. 1* p. 63

RESTRICTION(anciennete ENTRE 0 ET 1)[infirmieres]

43

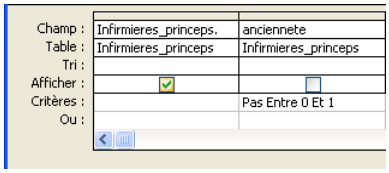


≃ caractéristiques des infirmières dont l'ancienneté n'est pas comprise entre 0 et un an (bornes incluses)

R_{10}^2

RESTRICTION(anciennete PAS ENTRE 0 ET 1)[infirmieres]

44



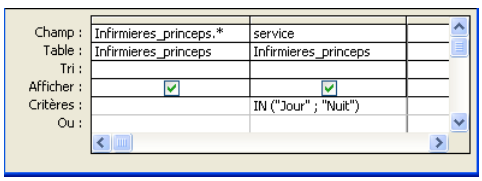
≃ caractéristiques des infirmières qui travaillent soit de jour soit de nuit.

On exclut implicitement l'infirmière dont le service n'est pas connu, lacune qui est signalée par la marque **NULL**.

R_{11}^2

RESTRICTION(service PARMI ('jour', 'nuit'))[infirmieres]

45



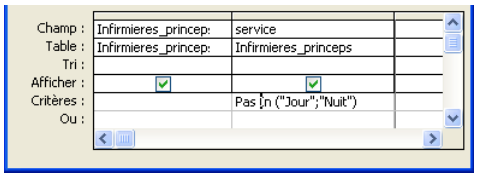
≃ caractéristiques des infirmières qui ne travaillent ni de jour ni de nuit

Cette dernière requête ne rapatrie cependant pas l'infirmière dont le service n'est pas connu. Le résultat est une table vide. La marque **NULL** n'est pas prise en compte par cette condition de restriction. Il faut employer des tests spécifiques : EST NULL ou EST PAS NULL sous Access, **IS NULL** ou **IS NOT NULL** sous SQL Server.

R^2_{12}

RESTRICTION(service PAS PARMI ('jour', 'nuit'))[infirmieres]

46



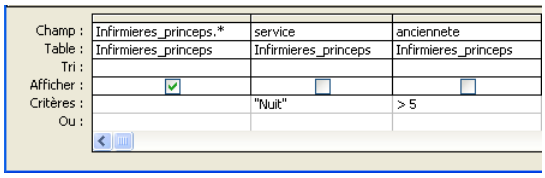
Les conditions de restrictions qui figurent sur la ligne ⌈Critères⌋ sont unies implicitement par un ET logique : la condition est l'ajout des critères édictés. La requête 47 vise une infirmière qui travaille de nuit ET dont l'ancienneté est supérieure à 5 ans. On peut toutefois utiliser un OU logique au sein d'un critère portant sur un attribut donné : 48. C'est comme si cette sous-condition était parenthésée. Si l'on veut par contre obtenir un OU logique entre des critères portant sur des attributs distincts, on utilise la ligne ⌈Ou⌋, sous la ligne ⌈Critères⌋ : 49. La requête 49 diffère donc fortement de la requête 48, puisqu'elle isole soit les infirmières travaillant de nuit et dont l'ancienneté est inférieure à 1 an soit les infirmières (quel que soit leur service) dont l'ancienneté est supérieure à 5 ans. La requête 48, elle, rapatrie des infirmières qui travaillent toutes de nuit mais dont l'ancienneté est soit inférieure à 1 an soit supérieure à 5 ans.

≃ caractéristiques des infirmières travaillant de nuit et dont l'ancienneté est supérieure à 5 ans

R^1_3 *t. 1* p. 60

RESTRICTION(
service = "Nuit" ET anciennete > 5
)[infirmieres]

47

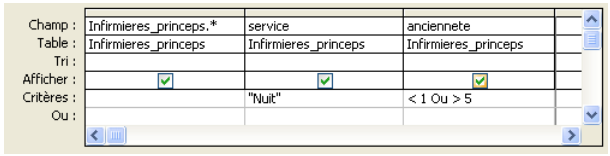


≃ caractéristiques des infirmières qui travaillent de nuit et dont soit l'ancienneté est inférieure à 1 an soit l'ancienneté est supérieure à 5 ans

R^1_4 *t. 1* p. 62

RESTRICTION(
service = "Nuit"
ET (anciennete < 1 OU anciennete > 5)
)[infirmieres]

48

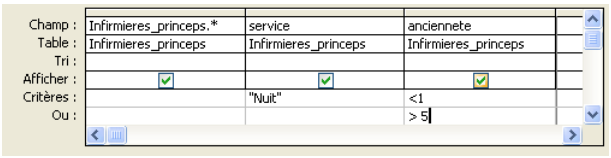


≃ caractéristiques des infirmières qui travaillent de nuit et dont l'ancienneté est inférieure à 1 an ou caractéristiques des infirmières dont l'ancienneté est supérieure à 5 ans

R^2_{13}

RESTRICTION(
(service = "Nuit"
ET (anciennete < 1)
OU anciennete > 5
)[infirmieres]

49



3.4. ⊕ Traitement de NULL

La formulation spécifique pour tester la présence de **NULL** – et sa négation : 51 – souligne le fait que **NULL** n'est pas une valeur, mais une marque.

≃ caractéristiques des infirmières dont le service n'est pas connu (c'est-à-dire comprend la marque **NULL**)

R^1_9 *t. 1* p. 63

RESTRICTION(service EST NULL)[infirmieres]

MySQL

```
SELECT *
FROM infirmieres
WHERE service IS NULL ;
```

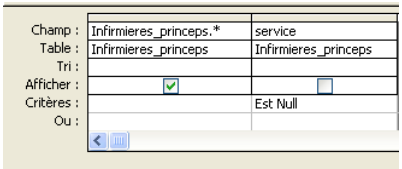
La condition inverse, sélectionner toutes les infirmières pour lesquelles le service est connu, c'est-à-dire dont l'attribut service ne comprend pas la marque **NULL**, donne naissance à la clause :

WHERE service **IS NOT NULL**

Access

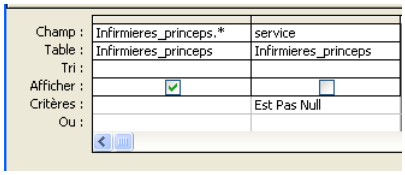
En 50, la réalisation sous Access. Sa négation figure en 51.

50



≃ caractéristiques des infirmières dont le service est connu (c'est-à-dire ne comprend pas la marque **NULL**)

51



△

Dans une table, cette marque **NULL** se « matérialise » par... le vide de la case correspondante comme le montre 52 pour l'infirmière 41, ce qui ne facilite pas exactement la différenciation avec la présence d'une chaîne vide (" ou '').

52

Infirmieres_princeps : Table

Identifiant	age	etudes	diplome	anciennete	service
36	26	4	BEPC	2	Jour
39	33	4	BEPC	3	Jour
41	26	3	BEPC	0	
43	26	4	BEPC	2	Jour
44	29	3	BEPC	2	Nuit
46	21	3	BEPC	0	Jour
47	30	3	BEPC	3	Nuit

Enr : 2 sur 42

MySQL

△

On sera attentif à l'ambiguïté du signe = en SQL. Le plus souvent, il sert à tester l'égalité. C'est le cas dans la clause **WHERE** de la requête :

```
SELECT id, age, etudes, diplome, anciennete, service
FROM infirmieres
WHERE anciennete = 0 ;
```

Cette clause se paraphrase par « dans le(s) cas où l'attribut anciennete a pour valeur 0 ». Par contre, dans la requête de mise à jour (chapitre X § 3) :

```
UPDATE fiches_originelles
SET poids = NULL
WHERE poids = 0 ;
```

associé à **SET**, le signe = sert aussi à changer effectivement la valeur d'un attribut pour les lignes sélectionnées. Les deux emplois se juxtaposent. La paraphrase est alors : « dans la table fiches_originelles, attribuer à l'attribut poids la marque **NULL** dans le(s) cas où l'attribut poids a pour valeur 0 ».

PRÉMA

Exercice n°1 L'équipe médicale distingue 3 grandes classes de poids à la naissance : inférieur à 750 g, entre 750 et 1000 g et supérieur à 1000 g.

Extraire de la table bebes et pour chaque sexe, les lignes correspondant à chaque classe (2 solutions pour la classe intermédiaire). Quelles conclusions en tirer ?

PHÈDRE

ESQUE

Exercice n°2 Extraire les lignes de la table esque qui ne correspondent pas à des adjectifs, c'est-à-dire qui n'ont pas pour valeur 'a' dans la colonne cat_derive.

4. Projeter : un sous-ensemble d'attributs

Projection : définition relationnelle

La **projection**, au sens relationnel, produit une relation définie par un sous-ensemble des attributs de la relation source et qui contient uniquement les n-uplets de la relation source réalisant des combinaisons distinctes de ce sous-ensemble d'attributs.

Projection et projection avec doublons

La projection avec doublons est l'opération primitive des SGBD. La projection au sens strict en est dérivée.

L'appellation Projection avec doublons est une dénomination informelle. Il s'agit plutôt d'une **sélection multi-ensemble**, c'est-à-dire d'une sélection qui produit un **multi-ensemble**, où un élément donné peut figurer plusieurs fois, à la différence d'un ensemble où un élément donné ne figure qu'une seule fois.

△

MySQL

La projection avec doublons s'obtient en listant uniquement les attributs souhaités dans la clause **SELECT**. L'équivalent de la requête :

≃ combinaisons, pour les infirmières, des attributs service et anciennete

c'est-à-dire :

R^1_{12} *t. 1* p. 64

PROJECTION AVEC DOUBLONS(
service, anciennete
)[infirmieres]

est :

```
SELECT service , anciennete
FROM infirmieres ;
```

Le modifieur **DISTINCT** dans la clause **SELECT** stipule d'afficher uniquement les combinaisons d'attributs qui diffèrent les unes des autres. Il permet d'obtenir la projection au sens relationnel. L'équivalent de la requête :

R^2_{14}

PROJECTION(service, anciennete)[infirmieres]

soit :

≃ combinaisons distinctes, pour les infirmières, des attributs service et anciennete

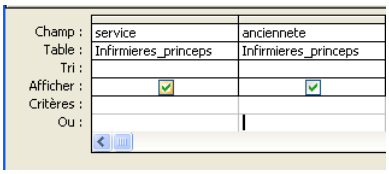
est alors :

```
SELECT DISTINCT service , anciennete
FROM infirmieres ;
```

Access

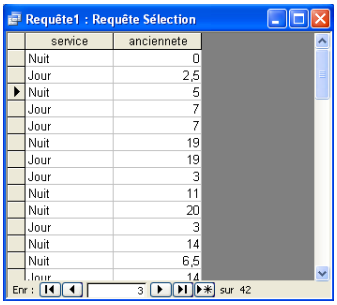
La projection avec doublons revient à sélectionner uniquement la ou le(s) colonnes visée(s) : 53.

53



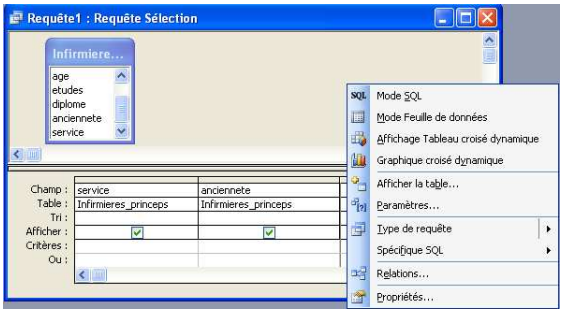
Le résultat 54 met en évidence les doublons : on trouve par exemple 2 fois la combinaison service de jour et ancienneté de 7 ans.

54



La projection sans doublons suppose d'utiliser le menu contextuel ⌈Propriétés de la requête⌋. On accède à ce menu avec un clic droit sur la sous-fenêtre du haut où se trouve affichée la table sur laquelle doit porter la requête 55.

55

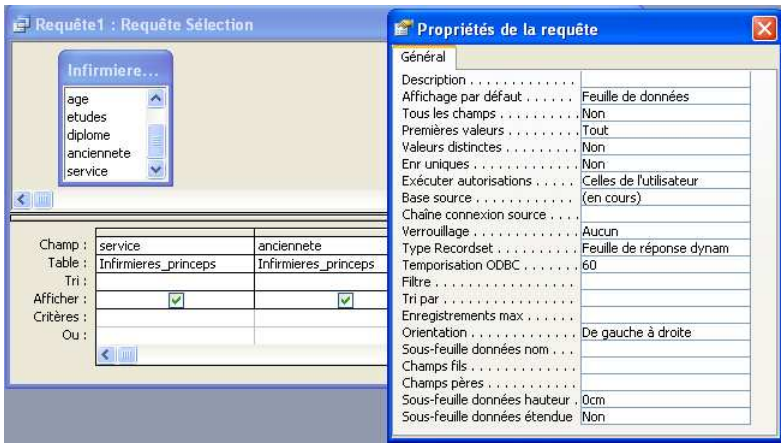


On y fait appel aussi en cliquant sur l'icône correspondante de la barre de menu 56.

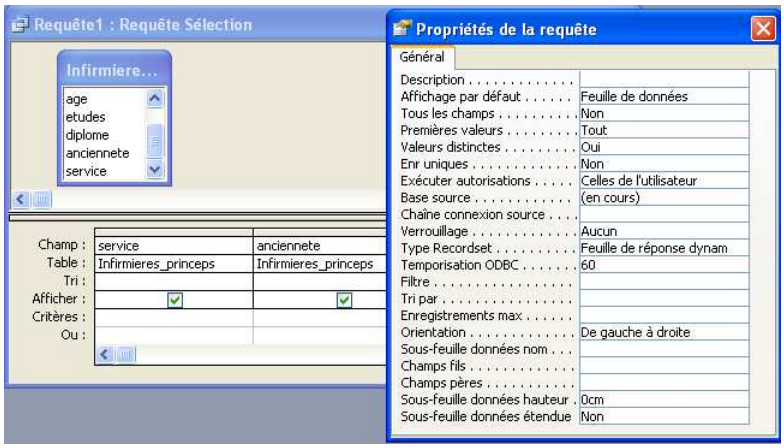
56

On choisit alors la ligne ⌈Valeurs distinctes⌋, par défaut à *Non* 57, et que l'on met à *Oui* grâce au menu déroulant 58.

57

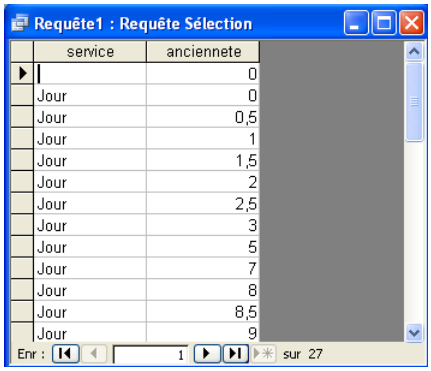


58



Une fois ce changement fait et le menu contextuel supprimé, le déclenchement de la requête aboutit effectivement à l'ensemble des valeurs distinctes 59 et au vide pour la marque **NULL**. La combinaison service de jour et ancienneté de 7 ans est désormais unique.

59



△

On remarquera que la partie principale de la requête reste identique sous l'interface et qu'il faut accéder au menu contextuel [Propriétés de la requête] pour savoir si l'on a affaire à une projection avec doublons ou à une projection sans doublons. Les requêtes SQL Server correspondant à une projection avec *vs.* sans doublons sont les suivantes :

```
SELECT Infirmieres_princeps.service,
```

TABLEAU 16 – PHÈDRE : catégories de la table occurrences

cat
Dét/Pron
Espace
Nc
V
Adj
SepPhraseOuGroupe
SepGroupe
Np
Conj
Prép
SepPhrase
Rel
Adv
SepTiradeDansVers
SepGuillemet
SepParentheseOuvrante
SepParentheseFermante

```
    Infirmieres_princeps.anciennete
FROM Infirmieres_princeps;

SELECT DISTINCT Infirmieres_princeps.service,
    Infirmieres_princeps.anciennete
FROM Infirmieres_princeps;
```

PRÉMA

Exercice n°3 Fournir les différentes modalités de la variable « relation de l'infirmière avec les parents », variable à laquelle correspond la colonne relation_infirmiere_parents de la table fiches_originelles.

PHÈDRE

La requête :

```
SELECT DISTINCT cat
FROM occurrences ;
```

permet d'obtenir la liste des 17 différentes « catégories » employées dans la table occurrences de PHÈDRE (tableau 16, p. 82). Sans ce modifieur, la table aurait. . . 28 248 lignes, soit le nombre de lignes de la table argument, et comprendrait d'innombrables doublons.

ESQUE

5. Combiner restriction et projection

≃ valeurs distinctes de l'attribut anciennete pour les seules infirmières de jour

R^2_{15}
RESTRICTION(service = 'Jour')[infirmieres]
$\hookrightarrow$
PROJECTION(anciennete)[<résultat$_1$>]

$\simeq$ valeurs de l'attribut anciennete pour les seules infirmières de jour

R^2_{16}
RESTRICTION(service = 'Jour')[infirmieres]
$\hookrightarrow$
PROJECTION AVEC DOUBLONS(anciennete)[<résultat$_1$>]

PRÉMA

MySQL

La requête suivante réalise la R^2_{15} :

```
SELECT DISTINCT anciennete
FROM infirmieres
WHERE service = 'Jour' ;
```

La projection avec doublons s'obtient par la requête :

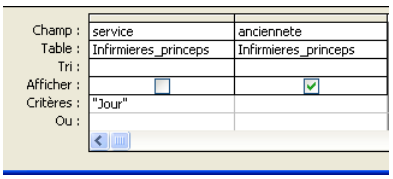
```
SELECT anciennete
FROM infirmieres_princeps
WHERE service = 'Jour' ;
```

qui produit une table avec autant de lignes que d'infirmières de jour, soit 29. Elle réalise la R^2_{16}.

Access

En 60 figure la combinaison d'une restriction (aux seules infirmières de jour) et d'une projection : une colonne seulement est fournie, anciennete. La ligne ⌈Critères⌉ contient la ou les condition(s) de restriction. La ou les colonne(s) retenue(s) dans le résultat correspond(ent) à la projection souhaitée. Par défaut, il s'agit d'une projection avec doublons, qui équivaut à la R^2_{16} p. 83 supra. Éliminer les doublons, pour obtenir l'équivalent de la R^2_{15} p. 83, suppose la démarche exemplifiée en 57 et 58 supra.

60



PHÈDRE

La requête du tableau 17 p. 84 combine restriction et projection. Portant sur la table vers de PHÈDRE, elle garde uniquement les lignes où le nombre d'interventions est égal à 3 et conserve seulement les colonnes numero_vers, personnage_s et vers :

TABLEAU 17 – PHÈDRE : vers en 3 segments

numero-_vers	*personnage_s*	*vers*
205	OENONE PHEDRE OENONE	Cet Hippolyte... / Ah, dieux! / Ce reproche vous touche.
262	PHEDRE OENONE PHEDRE	J'aime... / Qui? / Tu connais ce fils de l'amazone,
562	THERAMENE HIPPOLYTE THERAMENE	Elle vous cherche. / Moi? / J'ignore sa pensée.
763	PHEDRE OENONE PHEDRE	Quand je me meurs! / Fuyez. / Je ne le puis quitter.
1188	THESEE PHEDRE THESEE	Qu'il l'aime. / Quoi, seigneur? / Il l'a dit devant moi.

R_{17}^{2}

RESTRICTION(partage_en = 3)[vers]

$\hookrightarrow$

PROJECTION(
numero_vers,
personnage_s,
vers)
[<résultat$_1$>]

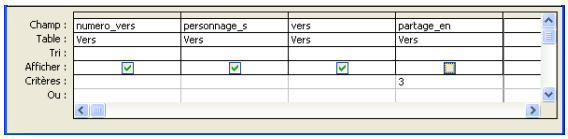
MySQL

Dans une combinaison de restriction et de projection, la restriction apparaît dans la clause **WHERE** tandis que le(s) nom(s) d'attribut(s) listé(s) dans la clause **SELECT** opère(nt) la projection :

```
SELECT DISTINCT numero_vers,
       personnage_s,
       vers
FROM vers
WHERE partage_en = 3 ;
```

Access

La restriction dans 61 se matérialise par la colonne partage_en à l'extrême-droite. Elle ne doit pas figurer dans le résultat. La boîte de la ligne [Afficher] n'est donc pas cochée. Par contre, 3 dans la ligne [Critères] indique que, pour être retenue, la ligne doit avoir cette valeur pour la colonne partage_en. Les trois premières colonnes correspondent à la projection : ce sont les colonnes conservées dans le résultat.



Exercice n°4 On donnera deux autres moyens de repérer les vers partagés en 3.

ESQUE

6. Calculer des attributs

PRÉMA

≃ combinaisons distinctes, pour les infirmières de nuit, des attributs age, anciennete et de l'âge à l'arrivée en néonatalité, c'est-à-dire de la soustraction de la valeur pour l'attribut anciennete à la valeur pour l'attribut age

R^1_{14} *t. 1* p. 71

```
RESTRICTION(service = 'Nuit')[infirmieres]
↪
PROJECTION(
age, anciennete, age - anciennete
)[<résultat1>]
```

≃ même chose, mais avec des titres de colonne plus explicites : *Âge*, *ancienneté* et *arrivée à*

R^1_{15} *t. 1* p. 71

```
RESTRICTION(service = 'Nuit')[infirmieres]
↪
PROJECTION(
age TITRE 'Âge',
anciennete TITRE 'ancienneté',
age - anciennete TITRE 'arrivée à'
)[<résultat1>]
```

MySQL

L'âge des infirmières à leur arrivée en néonatalité (âge − ancienneté dans le service) – R^1_{14} *t. 1* p. 71 – est calculé via la requête :

```
SELECT DISTINCT age,
       anciennete,
       age - anciennete
FROM infirmieres
WHERE service = 'Nuit' ;
```

L'ajout de titres aux colonnes (ou **alias**) s'obtient par le mot-clé **AS** :

```
SELECT DISTINCT
       age AS 'Âge',
       anciennete AS 'ancienneté',
       age - anciennete AS 'arrivée à'
FROM infirmieres
WHERE service = 'Nuit' ;
```

△

Le mot-clé **AS** est optionnel. On pourrait ainsi réécrire la réquête précédente :

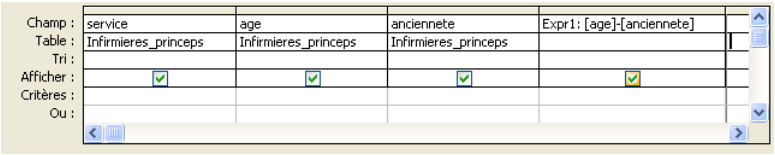
```
SELECT DISTINCT
      age 'Âge',
      anciennete 'ancienneté',
      age - anciennete 'arrivée à'
FROM infirmieres
WHERE service = 'Nuit' ;
```

Mais, par souci de clarté, nous avons choisi de fournir le mot-clé dans ces pages.

Access

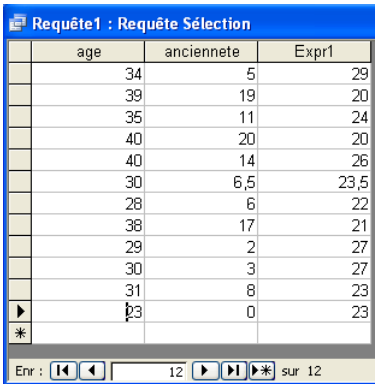
Un attribut calculé repose sur l'utilisation d'autres attributs, dont les noms sont à mettre entre crochets. Access ajoute un nom à cet attribut, de la forme Expr_n : 62.

62



C'est ce nom qui sert d'en-tête à la colonne de la table résultat : 63.

63



On peut également donner un titre plus significatif : 64, qui se retrouve dans le résultat : 65.

64

Champ :	Âge: age	ancienneté : ancienneté	arrivée à : [age]-[ancienneté]	service
Table :	Infirmieres_princeps	Infirmieres_princeps		Infirmieres_princeps
Tri :				
Afficher :	☑	☑	☑	☐
Critères :				"Nuit"
Ou :				

65

Requête1 : Requête Sélection

	Âge	ancienneté	arrivée à
	34	5	29
	39	19	20
	35	11	24
	40	20	20
	40	14	26
	30	6,5	23,5
	28	6	22
	38	17	21
	29	2	27
	30	3	27
	31	8	23
▶	23	0	23
*			

Enr : 12 sur 12

PHÈDRE

ESQUE

7. Ordonner et limiter les résultats

Tri

≃ identifiant et ancienneté des infirmières de nuit

R^1_{16} *t. 1* p. 74

RESTRICTION(service = 'Nuit')[infirmieres]
↪ PROJECTION(id, anciennete)[<résultat$_1$>]

≃ identifiant et ancienneté des infirmières de nuit, triés par ancienneté croissante

R^1_{17} *t. 1* p. 74

RESTRICTION(service = 'Nuit')[infirmieres]
↪ PROJECTION(id, anciennete)[<résultat$_1$>]
↪ TRI SUR(anciennete CROISSANT)[<résultat$_2$>]

≃ identifiant et ancienneté des infirmières de nuit, triés par ancienneté décroissante

R^1_{18} *t. 1* p. 74

RESTRICTION(service = 'Nuit')[infirmieres]
↪ PROJECTION(id, anciennete)[<résultat$_1$>]
↪ TRI SUR(anciennete DÉCROISSANT)[<résultat$_2$>]

≃ identifiant et ancienneté des infirmières de nuit, triés de manière aléatoire

R^2_{18}

RESTRICTION(service = 'Nuit')[infirmieres]
↪ PROJECTION(id, anciennete)[<résultat$_1$>]
↪ TRI ALÉATOIRE[<résultat$_2$>]

MySQL

Le tri est effectué en fonction d'une clause **ORDER BY** ajoutée en fin de requête. Elle est suivie du nom de la colonne ou des colonnes en fonction de laquelle ou desquelles on souhaite que soient ordonnés les résultats. Les entités sont classées par ordre croissant des valeurs pour chacune des colonnes en question. On peut modifier cet ordre en ajoutant **DESC**(*ending*) après le nom de la colonne.

Les requêtes suivantes produisent une table des identifiants et anciennetés des infirmières de nuit, classées :

- pour la première dans l'ordre de la table initiale (R^1_{16} *t. 1* p. 74) ;
- pour les 2 suivantes (R^1_{17} *t. 1* p. 74), qui sont des variantes, par ancienneté croissante (**ASC**, optionnel, est mis pour *ascending*) ;
- pour la quatrième (**DESC** est mis pour *descending*) par ancienneté décroissante (R^1_{18} *t. 1* p. 74) ;

TABLEAU 18 – PRÉMA : infirmières de nuit et tri aléatoire

id	*anciennete*
10	11.00
20	6.00
7	19.00
44	2.00
47	3.00
16	6.50
3	5.00
1	0.00
14	14.00
12	20.00
28	17.00
58	8.00

id	*anciennete*
3	5.00
47	3.00
20	6.00
12	20.00
16	6.50
58	8.00
14	14.00
10	11.00
1	0.00
28	17.00
44	2.00
7	19.00

id	*anciennete*
7	19.00
58	8.00
3	5.00
12	20.00
14	14.00
10	11.00
44	2.00
47	3.00
28	17.00
16	6.50
20	6.00
1	0.00

– pour la dernière de manière aléatoire – **RAND**() est mis pour *random*, le hasard (le tableau 18 p. 89 juxtapose 3 exécutions successives de cette requête, qui montrent la variation des ordres obtenus).

R^1_{16} *t. 1* p. 74

```
SELECT id, anciennete
FROM infirmieres
WHERE service = "Nuit" ;
```

R^1_{17} *t. 1* p. 74

```
SELECT id, anciennete
FROM infirmieres
WHERE service = 'Nuit'
ORDER BY anciennete ;
```

R^1_{17} *t. 1* p. 74

```
SELECT id, anciennete
FROM infirmieres
WHERE service = 'Nuit'
ORDER BY anciennete ASC ;
```

R^1_{18} *t. 1* p. 74

```
SELECT id, anciennete
FROM infirmieres
WHERE service = 'Nuit'
ORDER BY anciennete DESC ;
```

R^2_{18} p. 88

```
SELECT id, anciennete
FROM infirmieres
WHERE service = 'Nuit'
ORDER BY RAND() ;
```

La requête :

```
SELECT id, age, anciennete
FROM infirmieres
WHERE service = 'Nuit'
ORDER BY age DESC, anciennete ASC ;
```

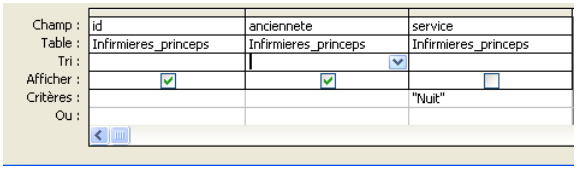
conserve, pour les infirmières travaillant de nuit (restriction), les attributs id, age et anciennete (projection). Elle trie les résultats d'abord par âge décroissant puis par ancienneté croissante (le modifieur **ASC** est optionnel). Elle combine donc deux tris successifs.

Access

La sous-fenêtre du bas de formulation de requête contient une ligne [Tri]. Laisser cette ligne à son état de départ 66 revient à produire une table-résultat dont l'ordre suit celui de la table d'entrée. Cliquer dans la case [Tri] d'une colonne fait apparaître un menu déroulant qui permet de choisir entre l'ordre croissant selon cette colonne : 67, ou bien l'ordre décroissant : 68.

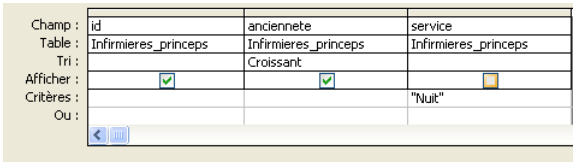
R^1_{16} *t. 1* p. 74

66



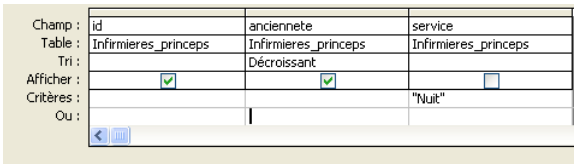
R^1_{17} *t. 1* p. 74

67



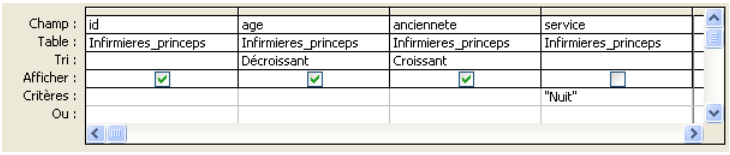
R^1_{18} *t. 1* p. 74

68



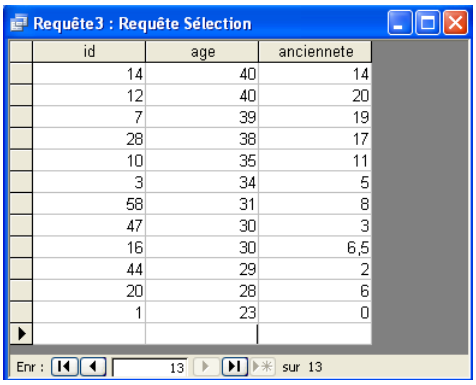
Trier sur plusieurs colonnes s'obtient en indiquant un ordre de tri pour plusieurs colonnes. Le tri s'opère d'abord sur la colonne de tri la plus à gauche, puis sur la suivante, etc. En 69, il s'effectue d'abord sur age (par valeur décroissante) puis sur anciennete (par valeur croissante), comme on le constate dans le résultat 70.

69



70

△



Il faut que les colonnes entrant dans le tri aient été placées en cohérence avec l'ordre visé lors de la formulation de la requête.

△

De la même manière qu'une colonne peut servir à une restriction sans figurer dans le résultat, une ou plusieurs colonne(s) peu(ven)t permettre des tris sans pour autant apparaître dans le résultat.

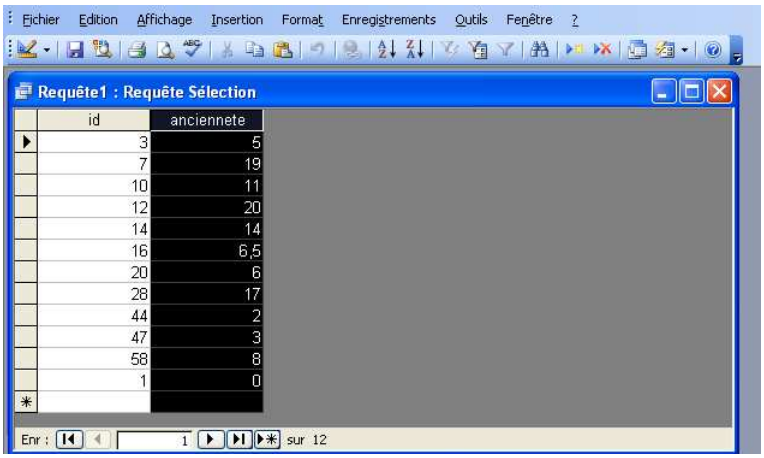
Le tri aléatoire – R^2_{18} p. 88 – s'obtient en mode SQL Server par appel à **RND**(), qui abrège *random*, le hasard :

```
SELECT Infirmieres_princeps.id,
    Infirmieres_princeps.anciennete
FROM Infirmieres_princeps
WHERE (((Infirmieres_princeps.service)="Nuit"))
ORDER BY RND() ;
```

7.1. ⊕ Tri en mode feuille de données (Access)

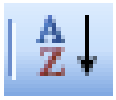
On peut également trier directement une table ouverte en mode feuille de données, sans passer par une requête. On sélectionne la colonne de tri : 71.

71



Puis on clique sur les icônes de tri de la barre de menu, en mode croissant 72 ou décroissant 73 selon ce qu'on souhaite.

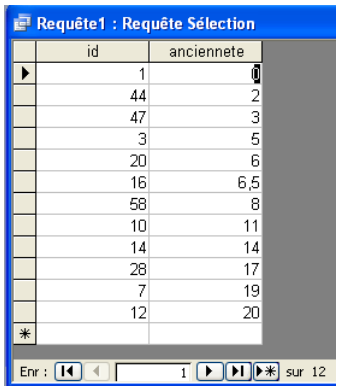
72



73

La feuille de données est triée en conséquence : 74.

74



On peut sélectionner plusieurs colonnes. Mais le tri sera alors croissant ou décroissant sur l'ensemble de ces colonnes.

Limiter les résultats

MySQL

On ajoute ⌈**LIMIT** *k*⌉ en fin de requête (après un appel éventuel à **ORDER BY**) pour se restreindre aux *k* premiers résultats. La requête :

```
SELECT id, age, anciennete
FROM infirmieres
WHERE service = 'Nuit'
ORDER BY age DESC, anciennete ASC
LIMIT 2 ;
```

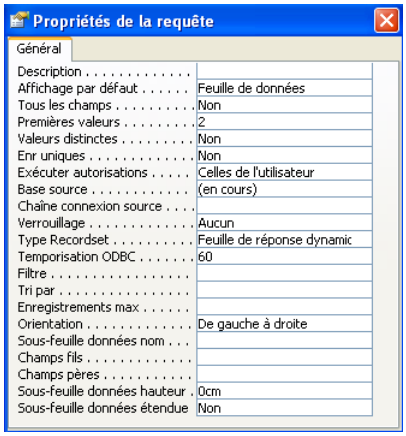
retourne la table des 2 premières lignes de la table obtenue sans le modifieur ⌈**LIMIT** *k*⌉, soit les informations sur les infirmières 14 et 12.

Access

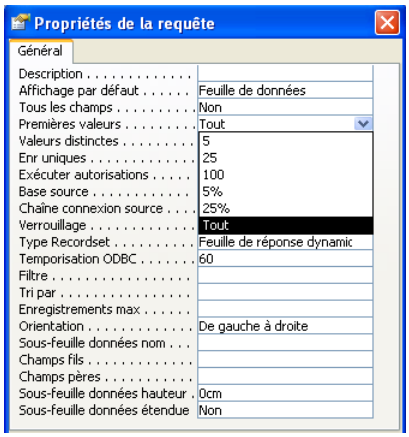
On peut régler le nombre 75 ou le pourcentage 76 de lignes souhaitées en utilisant la ligne ⌈Premières valeurs⌉ du menu contextuel ⌈Propriétés⌉, accessible par clic droit sur la

fenêtre d'affichage de la table sélectionnée pour la requête. La valeur par défaut est Tout (cf. 57).

75



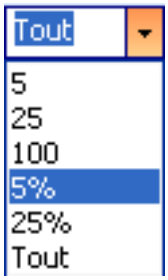
76



On peut dans ce cas taper directement le nombre de lignes souhaité.

On peut également utiliser le menu déroulant [Tout] 77 de la barre de menu du haut (à côté du sigma majuscule Σ – 78).

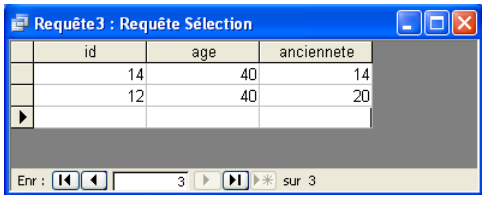
77



78

En 79 figure le résultat du choix opéré en 75.

79



En mode SQL Server, on obtient le même résultat en rajoutant juste après **SELECT** soit **TOP** k pour se limiter aux k premières lignes, soit **TOP** k PERCENT pour se limiter aux k premiers pour-cents.

PRÉMA

Exercice n°5 Tirer au hasard le texte d'une fiche de PRÉMA

PHÈDRE

Exercice n°6 Tirer au hasard un vers prononcé par Phèdre

ESQUE

8. Recherches par « air de famille »

8.1. ⊕ Filtrage grossier

MySQL

L'opérateur **LIKE** (et son contraire **NOT LIKE**) offre une première gamme, restreinte, de motifs. On fait figurer dans son contexte droit une expression comprenant des caractères spéciaux. Le caractère pour-cent remplace une suite (éventuellement vide) de caractères, tandis que le caractère souligné _ remplace un caractère quelconque. La requête suivante :

```
SELECT *
FROM fiches_originelles
WHERE texte LIKE '%bébé%' ;
```

extrait les 258 fiches qui comprennent la chaîne "bébé" à n'importe quel endroit.

La modification de la clause **WHERE** en :

⌈**WHERE** texte **LIKE** 'bébé%'⌉

extrait les 67 fiches qui commencent par la chaîne "bébé".

La modification en ⌈**WHERE** texte **LIKE** 'Bébé%'⌉ isole les 176 fiches commençant par le même mot, mais avec une majuscule à l'initiale.

On a noté, dans les fiches remplies pour le bébé 38 (tableau 5 p. 37), des variantes graphiques pour le mot *bébé* : *Bebé*, *BéBé*, *bb* et *BB*. Cette variation graphique conduit à d'autres essais, comme :

WHERE texte **LIKE** '%b_b_%'

pour rechercher les fiches dont le texte comprend 2 b séparés par un caractère quelconque et suivis par un caractère quelconque, à n'importe quel endroit dans le texte. Ce motif filtre effectivement, outre les fiches comprenant ⌈bébé⌉ celles qui incluent ⌈bebé⌉ ou ⌈bébe⌉. Mais il « ramène » également celles qui utilisent ⌈pro**ba**l**b**lement⌉ ou ⌈im**bi**l**b**ée⌉. . . On rencontre donc très rapidement les limites de l'opérateur **LIKE**. Il ne permet pas toujours d'exprimer des patrons assez contraignants : le souligné remplace un caractère quelconque, ce qui est trop « laxiste ». À l'inverse, il oblige à des combinaisons logiques complexes. Pour obtenir les cas de figure déjà rencontrés (706 fiches concernées), il faudrait en effet une clause du type :

```
WHERE texte LIKE '%bébé%'
   OR texte LIKE '%BéBé%'
   OR texte LIKE '%BB%'
   OR texte LIKE '%bb%'
   OR texte LIKE '%Bebé%' ;
```

△

On peut souhaiter exprimer un motif dans lequel figure un **caractère spécial** qui, normalement, avec l'opérateur **LIKE**, est associé à un comportement particulier. Il faut alors indiquer que l'on s'intéresse à ce caractère en tant que tel, comme **caractère littéral**. On le fait précéder d'une contre-oblique (*slash*). C'est « dé-spécialiser » ou « échapper un caractère spécial » ou le « prendre littéralement ». C'est le cas dans la requête :

```
SELECT contexte
FROM esque
WHERE contexte LIKE '%\%%'
ORDER BY RAND()
LIMIT 3 ;
```

qui isole les contextes contenant le caractère pour-cent. Le premier caractère pour-cent, comme le dernier, veut dire : 0 ou n caractères, celui du milieu, précédé d'une contre-oblique, est pris littéralement. On cherche donc un contexte tel qu'il y ait le caractère pour-cent précédé et suivi de 0 ou n caractères. Le résultat est de la forme :

contexte
27 Janvier 2000 - SPORTS #FOOTBALL. Le mercato (18 décembre-31 janvier) ne décolle pas. Faut-il revoir la période hivernale des transferts ? [...] #Le second marché annuel des transferts, clos le 31 janvier prochain, n'a pas connu le bouleversement annoncé. S'il continue d'être très critiqué, il est même rejeté par 75 **%** des entraîneurs. Mais, mis à part son nom, rien ne changera la saison prochaine. #Ça avait pourtant un joli nom, avec ce qu'il faut d'exotisme pour noyer le profane - mercato-késaco ? - sans couler le spécialiste - y a-bon-mercato-pour-les-ventes. Dans l'espace-temps football - un monde - nous vivions des périodes <mercatesques>. Nous étions au XXe siècle, l'économie était globalisée à mort, le football, son laboratoire ; le joueur, en short et panneau publicitaire en nylon, se vendait, plutôt bien d'ailleurs.
[...] je m'attarderai un peut [sic] plus sur les acteurs : servis par un trio principal haut en couleurs, Messengers nous offre par leur biais 118 mns de délire 100**%** <"BABAesque">, Yabe hiroyuki dans le rôle de Yokota nous sert un jeux trés propre comme à son habitude [...]
"$Plus près de tes sous, mon Dieu !$", tel est le long lamento de l'Eglise de France, qui ne craint qu'une seule chose : que les 56 **%** des fidèles qui, le dimanche à la messe, laissaient choir d'une main pieuse une pièce de 10 F dans la sébile ne prennent la très mauvaise habitude [...] de n'y laisser qu'une pièce de 1 euros. [...] Si l'on y ajoute les 15 % d'hypocrites qui ne laissent que 5 F [...], l'Eglise se retrouverait dans une situation <argentinesque> : la faillite !

8.2. ⊕ Filtrage fin : les expressions régulières

Les SGBD offrent souvent désormais des mécanismes de formulation de motifs plus précis et plus souples que ceux permis par **LIKE**. Ces mécanismes sont basés sur les **expressions régulières**.

MySQL

Le mot-clé introduisant une expression régulière en MySQL est **REGEXP** (son contraire étant **NOT REGEXP**). Les opérateurs post-posés déjà présentés et utilisés d'optionalité (?), d'intervention 0 ou n fois (*), d'intervention 1 ou n fois (+) ressortissent au répertoire des expressions régulières.

Les requêtes suivantes permettent d'extraire l'identifiant et le texte des fiches dont le texte répond à un motif donné.

≃ texte ne comprenant pas une des réalisations de *bébé*

R^1_{31} *t. 1* p. 80

RESTRICTION(
texte NE RESSEMBLANT PAS À
'[Bb][Eeé.]?[Bb][Eeé.]?'
)[fiches_originelles]

```
SELECT id, texte
FROM fiches_originelles
WHERE texte NOT REGEXP '[Bb][Eée.]?[Bb][Eeé.]' ;
```

≃ texte contenant *fille* en minuscules ou commençant par une majuscule

R^1_{28} *t. 1* p. 79

RESTRICTION(
texte RESSEMBLANT À
'[Ff]ille'
)[fiches_originelles]

```
SELECT id, texte
FROM fiches_originelles
WHERE texte REGEXP '[Ff]ille' ;
```

≃ texte contenant *garçon* en minuscules ou commençant par une majuscule

R^1_{29} *t. 1* p. 79

RESTRICTION(texte RESSEMBLANT À
'[Gg]arçon')[fiches_originelles]

```
SELECT id, texte
FROM fiches_originelles
WHERE texte REGEXP '[Gg]arçon' ;
```

≃ texte contenant soit une réalisation de *bébé*, soit une de *fille*, *garçon*, *enfant*, *préma*, en minuscules ou avec une majuscule initiale (et éventuellement, pour *préma*, l'absence d'accent sur le *é*)

R^2_{19}

RESTRICTION(
texte RESSEMBLANT À
'[Bb][Eeé.]?[Bb][Eeé.]?|[Gg]arçon|[Ff]ille|[Ee]nfant|↩
[Pp]r[eé]ma'
)[fiches_originelles]

```
SELECT id, texte
FROM fiches_originelles
WHERE texte
    REGEXP
     '[Bb][Eée.]?[Bb][Eeé.]?|[Gg]arçon|[Ff]ille|[Ee]nfant|[Pp]r[eé]ma' ;
```

≃ texte ne contenant ni une réalisation de *bébé*, ni une de *fille*, *garçon*, *enfant*, *préma*, en minuscules ou avec une majuscule initiale (et éventuellement, pour *préma*, l'absence d'accent sur le *é*)

R^1_{33} *t. 1* p. 81

RESTRICTION(
texte NE RESSEMBLANT PAS À
'[Bb][Eeé.]?[Bb][Eeé.]?|[Gg]arçon|[Ff]ille|[Ee]nfant|↩
[Pp]r[eé]ma'
)[fiches_originelles]

```
SELECT id, texte
FROM fiches_originelles
WHERE texte
    NOT REGEXP
     '[Bb][Eée.]?[Bb][Eeé.]?|[Gg]arçon|[Ff]ille|[Ee]nfant|[Pp]r[eé]ma' ;
```

Un ensemble de caractères peut également être désigné par ses « extrémités » dans le jeu de caractères utilisé. La clause :

```
WHERE texte REGEXP '[0-9]+' ;
```

abrège la clause :

```
WHERE texte REGEXP '[0123456789]+' ;
```

et rapatrie comme elle les fiches qui emploient un ou plusieurs chiffres (allant de 0 inclus à 9 inclus), qu'il s'agisse de mentions de semaines de grossesse (f. 991 : *malgrés ses 25 semaines*), de poids (f. 138 : *Petit bébé de 800gr*), d'heures (f. 178 : *intubation à 20h30*) ou de durée (f. 1001 : *Peau très fragile pdt 24 à 72h de Vie*), etc. On constate à cette occasion que le tiret est un **caractère spécial** au sein des crochets, puisqu'il contribue à la désignation d'un ensemble par ses bornes. Par conséquent, lorsqu'on souhaite utiliser le tiret comme **caractère littéral** au sein d'un ensemble, il doit être le seul caractère entre crochets ou le dernier, après la désignation d'un ou de plusieurs ensembles. La clause suivante :

```
WHERE texte REGEXP '[a-zéèêëàâîïôùüûç][-][a-zéèêëàâîïôùüûç]' ;
```

rapporte les 23 fiches comprenant des « mots en plusieurs mots » soudés par des traits d'union (*après-midi*) ou des pronoms post-posés (*sera-t-il*). Les caractères accentués sont ajoutés parce qu'ils ne sont pas compris dans l'ensemble borné par a et z.

La désignation d'ensembles de caractères peut se faire de manière négative, en indiquant le complémentaire d'un ensemble. L'accent circonflexe au début d'une paire de crochets indique qu'on attend à cet endroit le complémentaire de l'ensemble caractérisé par ce qui suit l'accent circonflexe. La clause :

```
WHERE texte REGEXP '[^a-zéèêëàâîïôùüûç][-][^a-zéèêëàâîïôùüûç]' ;
```

rapatrie les 347 fiches témoignant d'emplois du tiret ailleurs que comme trait d'union au sein d'un mot, puisqu'on exclut les caractères minuscules avant comme après, comme dans : *Il tète bien lors des soins de bouche - Se rendort tout de suite après les soins...* (f. 826). △

En MySQL, on peut également désigner des ensembles de caractères par des noms spécifiques, par exemple [:alpha :] pour les caractères alphabétiques, [:digit :] pour les chiffres, etc. Les équivalents de deux des motifs précédents seraient :

```
WHERE texte REGEXP '[[:digit:]]' ;
```

```
WHERE texte REGEXP '[[:alpha:]]-[[:alpha:]]' ;
```

△

Les motifs avec **LIKE** sont « ancrés » par rapport au début et à la fin de la chaîne examinée alors que ceux introduits par **REGEXP** ne le sont pas. Le motif ⌈**LIKE** 'bébé'⌉ recherche un texte qui commence et finit par *bébé* (il n'y en a d'ailleurs pas dans la table fiches_originelles) tandis que **REGEXP** 'bébé' recherche un texte où figure *bébé*, à n'importe quel endroit. Les ancrages, s'ils sont nécessaires, doivent être explicitement fournis avec **REGEXP**, par l'accent circonflexe tout au début ⌈**REGEXP** '^bébé'⌉ pour figurer le début de la chaîne examinée ou le signe dollar à la fin ⌈**REGEXP** 'bébé$'⌉ pour en figurer la fin. Dans ce contexte, dollar et accent circonflexe constituent des caractères spéciaux.

Trois requêtes et leurs résultats soulignent la différence entre **LIKE** et **REGEXP**. Les requêtes portent toutes les trois sur la table occ_prema. La première :

```
SELECT numero_ordre, forme_depart, categorie
FROM occ_prema
WHERE forme_depart LIKE 'préma' ;
```

retient (tableau 19 (a) p. 99) les lignes telles que la chaîne 'préma' débute et finit la colonne forme_depart (elle l'occupe tout entière). La seconde (tableau 19 (b) p. 99) ne diffère de la première que par le remplacement de **LIKE** par **REGEXP**. On retrouve le premier résultat si la condition de restriction s'énonce :
WHERE forme_depart **REGEXP** '^préma$'
c'est-à-dire où l'on souhaite que la chaîne 'préma' commence et termine la colonne.

Elles correspondent respectivement aux requêtes abstraites suivantes :

TABLEAU 19 – PRÉMA : **LIKE** *vs.* **REGEXP**

(*a*) R^2_{20} et R^2_{22}

numero_ordre	*forme_depart*	*categorie*
39462	préma	Afpfs
39937	préma	Afpfs
48267	préma	Ncms
49799	préma	Afpfs

(*b*) R^2_{21}

numero_ordre	*forme_depart*	*categorie*
3401	prématuré	Ncms
34306	prématuré	Afpms
39462	préma	Afpfs
39937	préma	Afpfs
48267	préma	Ncms
49799	préma	Afpfs
65770	prématuré	Ncms
65790	prématurité	Ncfs

R^2_{20}

RESTRICTION(forme_depart COMME 'préma')[occ_prema]
↪
PROJECTION AVEC DOUBLONS(
numero_ordre,
forme_depart,
categorie
)[<résultat$_1$>]

R^2_{21}

RESTRICTION(
forme_depart RESSEMBLANT À 'préma'
)[occ_prema]
↪
PROJECTION AVEC DOUBLONS(
numero_ordre,
forme_depart,
categorie
)[<résultat$_1$>]

R^2_{22}

RESTRICTION(
forme_depart RESSEMBLANT À '^préma$'
)[occ_prema]
↪
PROJECTION AVEC DOUBLONS(
numero_ordre,
forme_depart,
categorie
)[<résultat$_1$>]

On ne confondra pas l'emploi de l'accent circonflexe pour indiquer que le motif est à chercher au début du texte examiné avec l'utilisation de l'accent circonflexe au début des cro- △

chets pour indiquer le complémentaire d'un ensemble. Dans les deux cas, l'accent circonflexe constitue un caractère spécial, mais son sens change avec le contexte.

Access

Les recherches approximatives s'effectuent avec ce qui s'appelle dans la terminologie Access des **caractères génériques**. Le motif est introduit par Comme et figure entre guillemets ou entre apostrophes.

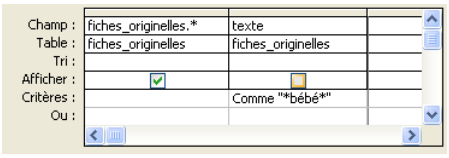
△

À la différence de la plupart des expressions régulières MySQL *supra*, l'opérateur Access * est nécessaire pour indiquer qu'il peut y avoir 0 ou n caractères quelconques avant ou après un sous-motif. Inversement, par conséquent, il n'est pas nécessaire, contrairement à MySQL de disposer d'opérateurs d'ancrage.

L'opérateur Access * équivaut au motif MySQL .* (point suivi d'étoile), où le point correspond à n'importe quel caractère et l'étoile post-posé indique que ce qui précède peut intervenir 0 ou n fois.

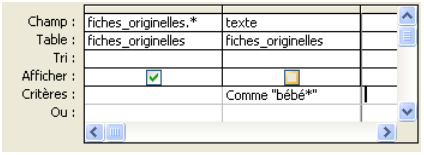
Le motif de 80 permet de chercher la chaîne "bébé" à n'importe quel endroit de l'attribut texte, grâce aux deux étoiles avant et après la chaîne visée.

80

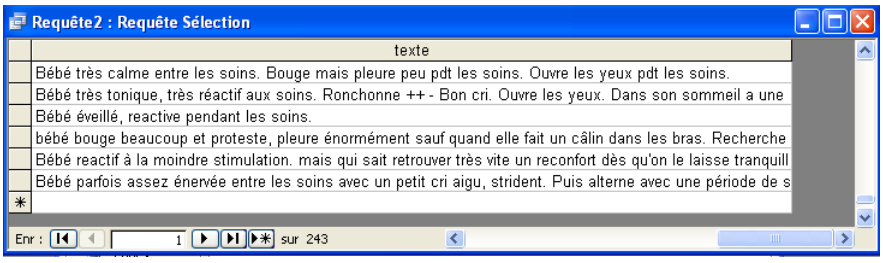


Le motif de 81 cherche par contre "bébé" tout au début de l'attribut texte et accepte un nombre indifférent de caractères quelconques après, comme le montre le résultat : 82.

81

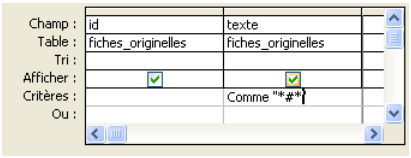


82



On constate en 82 que la recherche est indifférente à la casse, puisque sont rapatriés aussi des textes dans lesquels 'Bébé' figure tout au début.

Le dièse dans 83 remplace un chiffre quelconque. Le motif permet donc de rapatrier les fiches dont le texte contient un chiffre en n'importe quelle position : 84.



84

Requête2 : Requête Sélection

id	texte
467	- Petite fille Sage, assez tonique pendant les differents Soins. ouvre a 1/2 les yeux, très sensible à la luminosité -
578	Bébé bien eveill7, à un bon contact. avec ses parents Réagit bien aux soins
580	BB très chetive. commence à reprendre du poid. BB calme en général sauf pendant qu'on s'occupe d'elle. A 1 ictère photo continue. L'alimentation se passe pas trop mal.
804	BB jolie, qui remue bien, bouge ses 4 membres Tonique, ouvre les yeux, fait entendre sa jolie voix lorsqu'elle n'est pas contente. Apprécie les bras, surtout ceux de sa maman. Réagit à la voix de ses parents Se calme une fois réinstallée. Se positionne seule.
922	bébe très réactive au moment des soins, et exprime haut et fort son desaccord lorqu'elle est derangée. Entre les soins (faits environ* /3h), elle est calme et détendue, dort tranquillement. ouvre les yeux et regarde partout quand elle est dans les bras. au total bébé tranquille et détendue, mais bien reactive.

Enr : 63 sur 63

Le motif de 85 isole les fiches dont le texte contient un tiret : 86.

85

Champ :	id	texte
Table :	fiches_originelles	fiches_originelles
Tri :		
Afficher :	☑	☑
Critères :		Comme "*-*"
Ou :		

86

Requête2 : Requête Sélection

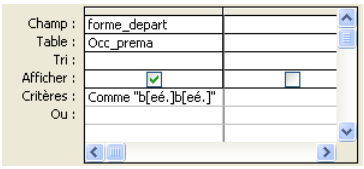
id	texte
964	Petite fille très calme - très mignonne gigote peu- bouge peu pendant les soins- tourne seule sa tête d'un côté et de l'autre.
968	Petite fille assez longue, semble très grande. calme. reagit pd les soins- pleure gigote. visage abimé : hematome de la face.
980	- Chloé ne suporte pas d'être derangée. à n'importe quel moment que ce soit - - Se calme difficilement avec la tétine - Dort entre les soins sans se reveiller -
981	bébé bouge beaucoup et proteste, pleure énormément sauf quand elle fait un câlin dans les bras. Recherche le contact peau/peau (caresses, massage...) qui la détend et dont elle semble intensement jouir. Accroche aussi par le regard. bébé très sympathique, agréable à s'occuper et ce malgré sa malformation qui inquiète et fait que je ne suis pas très à l'aise pour la manipuler -
983	petit BB chetif pleurant beaucoup -

Enr : 461 sur 462

8.3. ⊕ Pouvoir de filtrage comparé de MySQL et Access

Le tableau 151 p. 510 résume les opérateurs de filtrage disponibles sous MySQL et sous Access. Ils ne coïncident pas toujours. Une même opération peut s'exprimer différemment, comme le complémentaire d'un ensemble de caractères. Un SGBD peut offrir un opérateur absent d'un autre. Sur la table `occ_prema`, on peut sous Access isoler, grâce au motif de 87, les formes de départ qui combinent un b initial suivi soit de e soit de é puis d'un b puis, en finale, soit de e soit de é soit du point : 88, mais par contre, il ne semble pas possible de pouvoir dire que dans les deux cas, les voyelles sont optionnelles.

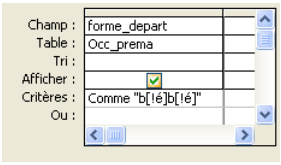
87



88

Dans le même esprit, on peut isoler en 89 celles qui commencent par un b suivi d'une lettre qui n'est pas é (le point d'exclamation joue le rôle du ^ pour MySQL), suivi de b et terminé par une lettre qui n'est pas é : 90.

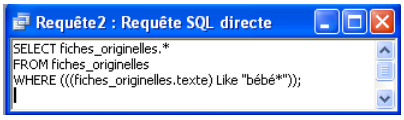
89



90

Au total, l'« arsenal » d'expressions de motifs offert par Access paraît dans l'immédiat plus pauvre que celui de MySQL. Comme le montre la requête sous-jacente à : 91, il s'agit d'ailleurs d'une légère amélioration de **LIKE**. Une version prochaine viendra peut-être enrichir les possibles.

91



Il importe de maîtriser la « boîte à outils » de filtrage de données textuelles offerte par le SGBD qu'on a choisi d'utiliser et pour cela d'explorer systématiquement la documentation. △

Exercice n°7 Comparez les résultats d'une requête apparemment identique en MySQL et Access et expliquez les différences de résultats.

MySQL

La requête :

```
SELECT lemme,
       categorie
FROM occ_prema
WHERE categorie REGEXP 'V*'
ORDER BY RAND()
LIMIT 10 ;
```

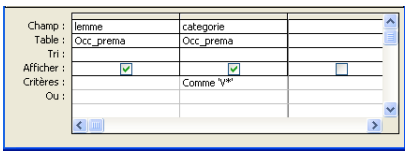
a pour résultat (entre autres résultats possibles, vu qu'il s'agit d'un tri aléatoire et de la conservation des seules 10 premières lignes) :

lemme	*categorie*
très	Rgp
et	Cc
faire	V.ip3s
soin	Ncmp
attraper	V.ip3s

Access

La requête :

92



a pour résultat :

93

PRÉMA

PHÈDRE

Décompte des associations de personnages dans les vers partagés La requête dont suivent les versions MySQL et Access isole l'attribut personnage_s de la table vers lorsqu'il comprend une espace à un endroit quelconque, c'est-à-dire au moins deux noms séparés par une espace. C'est un moyen de regrouper les vers partagés entre plusieurs personnages sans faire appel à l'attribut nombre_personnages. Le résultat est le même que celui du tableau 32, p. 144, qui, lui, fait appel à l'attribut nombre_personnages.

R^2_{23}

RESTRICTION(personnage_s COMME '% %')[vers]
↪
REGROUPER SUR(personnage_s)[<résultat$_1$>]
↪
PAR GROUPE(
personnage_s,
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_2$>]

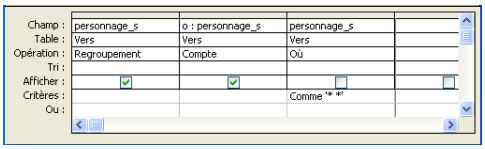
MySQL

```
SELECT personnage_s, COUNT(personnage_s) AS 'o.'
FROM vers
WHERE personnage_s LIKE '% %'
GROUP BY personnage_s ;
```

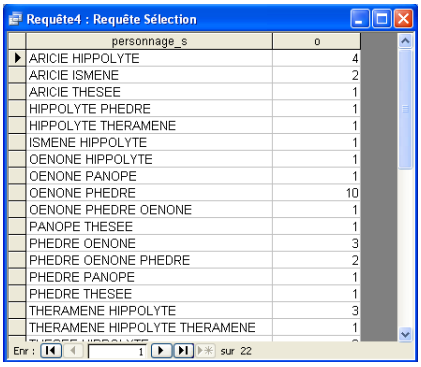
Access

La requête 94 a pour résultat 95.

94



95



Phèdre en début, milieu ou fin de vers partagé La requête pour isoler les vers où Phèdre intervient au début d'un vers partagé avec un autre personnage (résultat en haut du tableau 20, p. 107) est de la forme :

R^2_{24}

RESTRICTION(personnage_s COMME 'PHEDRE %')[vers]
↪
PROJECTION(
numero_vers TITRE 'v.',
REMPLACER(personnage_s, 'PHEDRE', '*'),
vers TITRE 'vers commencés par Phèdre'
)[<résultat$_1$>]

On suppose l'existence de la fonction :

REMPLACER(<chaîne d'entrée>, <sous-chaîne cible>, <sous-chaîne de substitution>) qui remplace dans la <chaîne d'entrée> la <sous-chaîne cible> par la <sous-chaîne de substitution>.

Cette fonction permet de substituer à la chaîne "PHEDRE" l'étoile dans personnage_s : la chaîne transformée est tout à la fois plus courte et plus lisible quant à la place où Phèdre intervient.

MySQL

L'appel à **LIKE** permet même de distinguer les cas où un personnage est le premier intervenant d'un vers partagé, le dernier, ou celui du milieu dans le cas d'un vers en 3 segments, comme le montrent les requêtes suivantes qui s'attachent au personnage Phèdre (tableau 20, p. 107) :

```
SELECT numero_vers AS 'v.',
       REPLACE(personnage_s, 'PHEDRE', '*') AS 'personnages',
       vers AS 'vers commencés par Phèdre'
FROM vers
WHERE personnage_s LIKE 'PHEDRE %' ;


SELECT numero_vers AS 'v.',
       REPLACE(personnage_s, 'PHEDRE', '*') AS 'personnages',
       vers AS 'vers où Phèdre est "encadrée"'
FROM vers
WHERE personnage_s LIKE '% PHEDRE %' ;


SELECT numero_vers AS 'v.',
       REPLACE(personnage_s, 'PHEDRE', '*') AS 'personnages',
       vers AS 'vers terminés par Phèdre'
FROM vers
WHERE personnage_s LIKE '% PHEDRE' ;
```

REPLACE est l'équivalent de la fonction REMPLACER dans la notation abstraite.

Access

La requête 96 repose sur une restriction aux vers dont la colonne personnage_s ressemble à 'PHEDRE *', c'est-à-dire PHEDRE suivi d'un espace et d'autres caractères, ce qui signale un vers partagé entre 2 personnages et où Phèdre est en première position. Le contenu de la colonne personnage_s, renommée en personnages dans le résultat, est modifié : la chaîne 'PHEDRE' est remplacée par la chaîne '*', comme le montre le résultat 97.

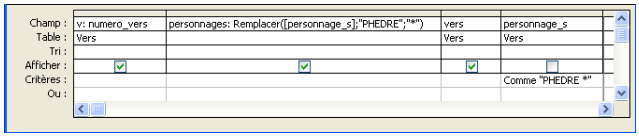
TABLEAU 20 – PHÈDRE : Phèdre dans les vers qu'elle partage

v.	*personnages*	*vers commencés par* Phèdre
157	* OENONE	Hélas ! / Dieux tout-puissants, que nos pleurs vous apaisent.
246	* OENONE	Tu le veux. lève-toi. / Parlez, je vous écoute.
262	* OENONE *	J'aime... / Qui ? / Tu connais ce fils de l'amazone,
325	* PANOPE	Ciel ! / Pour le choix d'un maître Athènes se partage.
711	* OENONE	Donne. / Que faites-vous, Madame ? Justes dieux !
763	* OENONE *	Quand je me meurs ! / Fuyez. / Je ne le puis quitter.
1619	* THESEE	Il n'était point coupable. / Ah ! père infortuné !

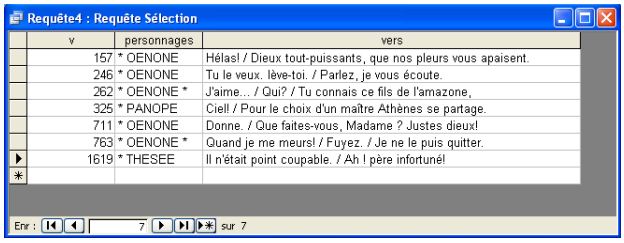
v.	*personnages*	*vers où* Phèdre *est « encadrée »*
205	OENONE * OENONE	Cet Hippolyte... / Ah, dieux ! / Ce reproche vous touche.
1188	THESEE * THESEE	Qu'il l'aime. / Quoi, seigneur ? / Il l'a dit devant moi.

v.	*personnages*	*vers terminés par* Phèdre
179	OENONE *	Quoi, Madame ? / Insensée, où suis-je ? et qu'ai-je dit ?
259	OENONE *	Aimez-vous ? / De l'amour j'ai toutes les fureurs.
260	OENONE *	Pour qui ? / Tu vas ouïr le comble des horreurs.
262	* OENONE *	J'aime... / Qui ? / Tu connais ce fils de l'amazone,
264	OENONE *	Hippolyte ? grands dieux ! / C'est toi qui l'as nommé.
670	HIPPOLYTE *	Et je vais... / Ah ! cruel, tu m'as trop entendue.
763	* OENONE *	Quand je me meurs ! / Fuyez. / Je ne le puis quitter.
835	OENONE *	Quoi ? / Je te l'ai prédit ; mais tu n'as pas voulu.
839	OENONE *	Vous mourez ? / Juste ciel ! qu'ai-je fait aujourd'hui ?
909	OENONE *	On vient ; je vois Thésée. / Ah ! je vois Hippolyte ;
914	THESEE *	Madame ; et dans vos bras met... / Arrêtez, Thésée,
1219	OENONE *	Comment ? / Hippolyte aime, et je n'en puis douter.
1225	OENONE *	Aricie ? / Ah ! douleur non encore éprouvée !
1252	OENONE *	Ils ne se verront plus. / Ils s'aimeront toujours.

96



97

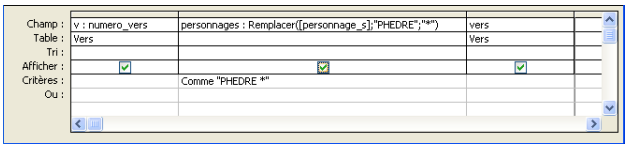


△

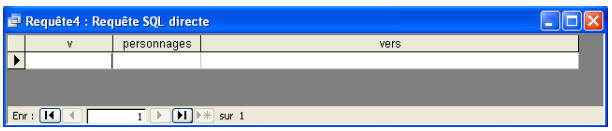
Attention : la requête 98 produit une table résultat vide 99. En effet, la condition de restriction ⌈Comme 'PHEDRE *'⌉ porte sur une colonne transformée, dans laquelle la chaîne 'PHEDRE' a en réalité disparu. On le comprend mieux en comparant les 2 requêtes SQL Server engendrées respectivement par 96 et par 98 :

```
SELECT Vers.numero_vers AS v,
       Replace([personnage_s],"PHEDRE","*") AS personnages,
       Vers.vers
FROM Vers
WHERE (((Vers.personnage_s) Like "PHEDRE *"));

SELECT Vers.numero_vers AS v,
       Replace([personnage_s],"PHEDRE","*") AS personnages,
       Vers.vers
FROM Vers
WHERE (((Replace([personnage_s],"PHEDRE","*")) Like "PHEDRE *"));
```

98



99



Vers se terminant par un phonème donné On suppose la fonction :

RENVERSER(<chaîne>)
qui retourne la chaîne de départ inversée.

TABLEAU 21 – PHÈDRE : mots à la rime se terminant par le phonème eu

occ_car	*phon. inversé*	*o.*
eux	ue	1
deux	ue d	1
feux	ue f	1
soupçonneux	ue n| o s| p uo s	1
poudreux	ue r d| uo p	1
affreux	ue r f| a	1
dangereux	ue r| @@ j| na d	1
généreux	ue r| e n| e j	2
heureux	ue r| eo	1
malheureux	ue r| eo l| a m	3
rigoureux	ue r| uo g| i r	2
amoureux	ue r| uo m| a	3
honteux	ue t| no	2
voeux	ue v	4
cheveux	ue v| @ hs	2
yeux	ue y	17
dieux	ue y d	8
adieu	ue y d| a	2

occ_car	*phon. inversé*	*o.*
lieux	ue y l	6
lieu	ue y l	2
mieux	ue y m	1
cieux	ue y s	1
dédaigneux	ue yn| ia d| e d	2
aïeux	ue y| a	2
odieux	ue| i d| o	8
prodigieux	ue| i j| i d| o r p	1
glorieux	ue| i r| o l g	2
furieux	ue| i r| u f	4
audacieux	ue| i s| a d| oo	1
ambitieux	ue| i s| i b| na	1
impétueux	ue| u t| e p| ni	1
respectueux	ue| u t| k ia p| s ia r	1
vertueux	ue| u t| r ia v	1
tortueux	ue| u t| r o t	1
incestueux	ue| u t| s ia s| ni	2

R^2_{25}

RESTRICTION(
fin_syll = 12
ET occ_phon RESSEMBLANT À 'eu$'
)[occurrences]
↪
REGROUPER SUR(occ_car)[<résultat$_1$>]
↪
PAR GROUPE(
occ_car,
RENVERSER(occ_phon) TITRE 'phon. inversé',
NOMBRE DE LIGNES() TITRE 'o.'
[<résultat$_2$>]
↪
TRI SUR(RENVERSER(occ_phon))[<résultat$_3$>]

Cette requête recherche les mots se terminant par le phonème eu et qui sont en dernière position dans le vers (tableau 21, p. 109). La syllabe sur laquelle ils finissent est la dernière : ⌈fin_syl = 12⌉. Le dernier phonème des mots sélectionnés est eu : ⌈occ_phon **REGEXP** 'eu$' ⌉. La requête trie en outre les résultats sur la transcription phonétique inversée : ⌈TRI SUR(RENVERSER(occ_phon
⌉. C'est rapprocher les mots qui pourraient rimer ensemble. On pourrait dans un deuxième temps mettre en rapport ces rimes potentielles et les rimes effectives.

MySQL

REVERSE est la réalisation MySQL de RENVERSER.

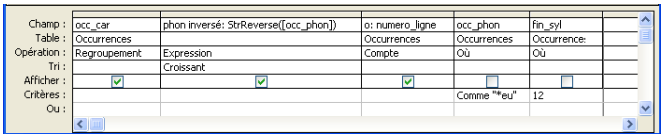
```
SELECT occ_car,
       REVERSE(occ_phon) AS 'phon. inversé',
       COUNT(*) AS 'o.'
FROM occurrences
WHERE fin_syl = 12
       AND occ_phon REGEXP 'eu$'
GROUP BY occ_car
```

```
ORDER BY REVERSE(occ_phon) ;
```

Access

La requête 100 est l'équivalent de la requête MySQL. STRREVERSE y joue le rôle de RENVERSER.

100



ESQUE

MySQL

La requête :

```
SELECT DISTINCT derive
FROM esque
WHERE derive REGEXP '[ABCDEFGHIJKLMNOPQRSTUVWXYZ]' ;
```

rapporte les dérivés tels qu'y figure, à n'importe quel endroit une quelconque des majuscules. Parmi les réponses obtenues : *James*-B*ond-iesque*, *Kim* D*ealesque*, *MOD*E*sque*, *Fight* C*lubesque*, A*rrabalesque*, D*OD**OTESQUE*, mais aussi *de* F*unesque* ou *3*D-*esque*, ou encore *le* C*orbusieresque*. Le(s) caractère(s) correspondant au motif se trouve(nt) mis en évidence.

Elle a pour représentation abstraite :

R^2_{26}

RESTRICTION(
derive RESSEMBLANT À
'[ABCDEFGHIJKLMNOPQRSTUVWXYZ]'
)[esque]
↪
PROJECTION(
Nombre de valeurs distinctes(derive)
)[<résultat$_1$>]

La même requête pourrait être abrégée en :

```
SELECT DISTINCT derive
FROM esque
WHERE derive REGEXP '[A-Z]' ;
```

On désigne donc l'ensemble des majuscules par ses « extrémités » séparés par le tiret, qui a ici, au sein des crochets, le statut d'un caractère spécial.

Le remplacement de ⌈**REGEXP** '[A-Z]'⌉ par ⌈**REGEXP** '^[A-D]'⌉ aboutit à isoler les dérivés qui commencent par une majuscule. Pour reprendre les exemples précédents, ne seront rapatriés ni *de* F*unesque* ni *3*D-*esque* ni *le* C*orbusieresque* tandis qu'A*rrabalesque* le sera.

Il est dans le même esprit aisé de savoir s'il existe ou non des dérivés en *-esque* comprenant des chiffres :

```
SELECT DISTINCT derive
FROM esque
WHERE derive REGEXP '[0-9]' ;
```

TABLEAU 22 – ESQUE : dérivés avec hiatus potentiel

derive	*derive*
tapiesque	starmaniesque
lamaesque	casanoviesque
fabliesque	victor-hugoesque
bédéesque	zappaesque
raimuesque	hurluberluesque
charabiesque	carcoesque
dariomoréniesque	avenuesque
foliesque	zidiesque
bibaesque	Uoesque
bougiesque	tooliesque
mobutuesque	dadaesque
ziziesque	ferriesque
chicagoesque	carrapichiesque
zaziesque	Le corbusiesque
incaesque	énerjiesque

Cinq dérivés comprennent un chiffre : *1°avrilesque*, *3D-esque*, *4 pattesque*, *M6-esque* et *TF1esque*.

Le remplacement de l'expression régulière par ⌈'[0-9-]'⌉ rapporte cette fois les dérivés qui soit comprennent un chiffre, comme supra, soit comprennent le tiret, comme : *idi-amindadaesque*, *private-jokesque* ou les deux, comme : *M6-esque*. L'expression régulière ⌈'-'⌉ rapatrie tous les dérivés comprenant un tiret au moins, comme :
gant-de-toilettesque, *extrême-droitesques* ou *loft-storyesque*.

L'utilisation des ensembles de caractères permet par exemple d'isoler les dérivés qui potentiellement présentent un hiatus avant le suffixe (le tableau 22, p. 111 en fournit un échantillon) :

```
SELECT DISTINCT derive
FROM esque
WHERE derive REGEXP '[aioué]esque'
      AND derive NOT REGEXP '[qg]uesque'
ORDER BY RAND()
LIMIT 30 ;
```

Ce sont les dérivés tels que le suffixe *-esque* se trouve précédé par une voyelle. On veille dans cette requête à écarter les dérivés se terminant par *-quesque* (*chiraquesque*, *diplodoquesque*, *guerniquesque*...) ou *-guesque* (*blaguesque*, *kingkonguesque*, *doudinguesque*...).

Dans la colonne `contexte` de la table `esque`, les concepteurs de la base ont choisi d'encadrer dans la plupart des cas le dérivé par les chevrons ouvrant et fermant, comme dans : « Toute cette fin de vie est d'un caractère si nettement <castelbriandesque> qu'il faut se faire violence pour ne point pasticher la manière d'écrire du poète dès que l'on veut s'en occuper. » Autrement dit, le dérivé est compris dans une chaîne commençant par un chevron ouvrant, se continuant par des caractères autres que le chevron fermant et s'achevant par un chevron fermant. On peut donc par l'expression régulière : ⌈ '<[^>]+>'⌉ isoler le dérivé « signalé » dans le contexte (le + indique ici qu'il peut y avoir 1 ou n occurrence(s) de caractères autres que le chevron fermant). On peut alors examiner le contexte gauche du dérivé. La requête :

```
SELECT contexte
FROM esque
WHERE contexte REGEXP '(très|fort|un␣peu|nettement)␣<[^>]+>' ;
```

TABLEAU 23 – ESQUE : emplois qualificatifs de dérivés (extraits)

contexte
La musique de cette chanson est **un peu** <bossanovesque> entre guillemets . . .
. . .L'agent de Keanu Reeves s'est empressé de démentir ces informations **très** <paparazziesques>" . . .
. . .Cette manière **très** <jospinesque> de ménager la chèvre et le chou . . .
Toute cette fin de vie est d'un caractère si **nettement** <castelbriandesque> . . .
Reçu un mot de Gide **fort** <gidesquement> tourné . . .

isole les contextes où le dérivé est précédé soit par *très* soit par *fort* soit par *un peu* soit par *nettement*. La présence d'un adverbe de degré avant le dérivé certifie que celui-ci n'est pas dans ce contexte un adjectif relationnel mais qu'il a un emploi qualificatif. Un adjectif relationnel est un adjectif qui remplace un nom précédé d'une préposition, il n'entre pas dans des expressions de degré : un nerf crânien est un nerf du crâne, mais on ne parle pas de nerf très crânien. On utilise un autre opérateur des expressions régulières, la barre verticale (| – parfois dénommée d'après son nom anglais *pipe*) : il combine des chaînes qui peuvent alterner à l'endroit indiqué. Le tableau 23, p. 112 donne des échantillons du résultat de cette requête.

Bruit et silence Les requêtes basées sur les expressions régulières sont confrontées au bruit, au rapatriement de lignes non pertinentes, et au silence, à l'omission de lignes pertinentes.

Exemple de bruit : la requête ci-dessus visant à obtenir les emplois qualificatifs de dérivés en *-esque* extrait également : « Le super-luxe du confort <campingesque> », où *fort* dans *confort* répond effectivement à l'expression régulière, qu'il faut donc modifier pour que le caractère avant l'un des adverbes de degré soit une espace ou une ponctuation :

```
REGEXP '[ ;,.:!?](très|fort|un peu|nettement) <[^>]+>'
```

L'ajout de cette nouvelle contrainte fait disparaître le contexte erroné.

Exemple de silence « débusqué » : lorsqu'on prend en compte la possibilité que l'adverbe de degré commence par une majuscule :

```
REGEXP '[ ;,.!?]([tT]rès|[fF]ort|[Uu]n peu|[Nn]ettement) <[^>]+>'
```

on obtient une réponse supplémentaire : « Elle retint son vagabondage mental pour revenir au livre. *U*n peu <charabiesque>. Fatal : l'art ne s'enseigne que par l'exemple. »

9. Solutions

Solution de l'exercice n°1 p. 78 Les solutions sont de la forme :

R^2_{27}

```
RESTRICTION(
sexe = 'Fille'
ET
poids_naissance ENTRE 750 ET 1000
)[bebes]
```

MySQL

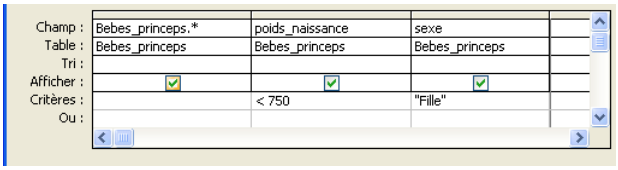
```
SELECT *
FROM bebes
WHERE poids_naissance < 750 AND sexe = "Fille" ;

SELECT *
FROM bebes
WHERE poids_naissance >= 750
  AND poids_naissance <= 1000
  AND sexe = "Garçon" ;

SELECT *
FROM bebes
WHERE poids_naissance BETWEEN 750 AND 1000
     AND sexe = "Fille" ;

SELECT *
FROM bebes
WHERE poids_naissance > 1000 AND sexe = "Garçon" ;
```

Il y a 15 filles et 15 garçons de moins de 750 g., 20 filles et 24 garçons entre 750 et 1000 g, 22 filles et 25 garçons de plus d'un kilo. Sachant qu'il y a 7 garçons de plus que de filles, la répartition paraît relativement équilibrée.

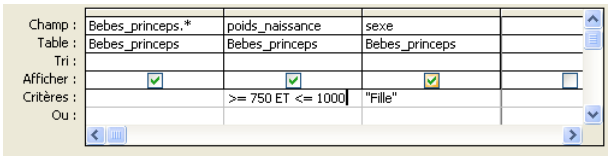
Access

Les requêtes sont fournies de 101 à 104.

101

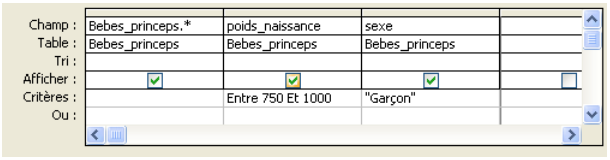


102

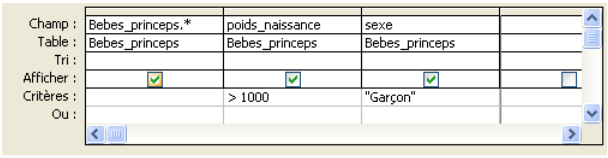


102 et 103 sont de simples variantes l'une de l'autre.

103



104



Solution de l'exercice n°2 p. 78 107 lignes ne correspondent pas à des adjectifs, des adverbes comme *badinesquement*, des noms comme *grotesque*, plus rarement des verbes comme *spongesquer* et enfin des dérivés dont la catégorie, incertaine, est marquée par un point d'interrogation : *à-la-fin-esque*. Par ailleurs, 134 autres lignes ont la marque **NULL** pour la colonne cat_derive : *plénatesque*, *moyenagesque*, *oulipesque*, etc.

R^2_{28}

RESTRICTION(
cat_derive <> 'a'
OU
cat_derive EST NULL
)[esque]

MySQL

La restriction aux lignes n'ayant pas 'a' comme valeur de cat_derive s'obtient via :

```
SELECT *
FROM esque
WHERE cat_derive <> 'a' ;
```

Elle ne restitue pas les lignes pour lesquelles figure la marque **NULL**. La condition de restriction doit être alors :

```
WHERE cat_derive <> 'a'
    OR cat_derive IS NULL ;
```

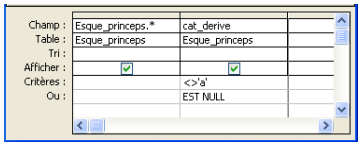
Access

Les 2 solutions diffèrent quant à la manière d'exprimer le OU logique, soit sur deux lignes, soit sur une seule ligne.

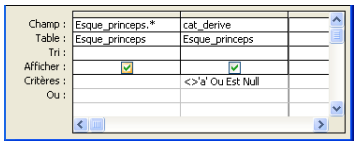
TABLEAU 24 – PRÉMA : modalités de la variable « relation infirmière-parents »

relation_infirmiere_parents
je ne connais pas les parents
moyen
bon
mauvais

105



106



Solution de l'exercice n°3 p. 82 Le résultat figure au tableau 24 p. 115. Il découle de la requête :

R^2_{29}

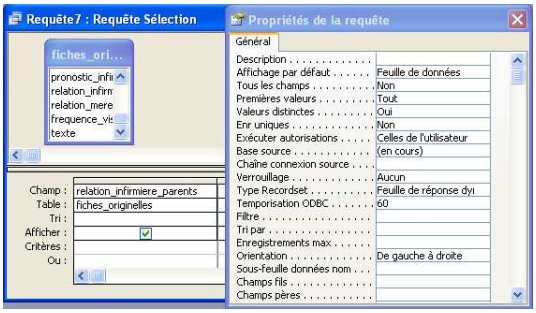
PROJECTION(
NOMBRE DE VALEURS DISTINCTES(relation_infirmiere_parents)
)[fiches_originelles]

MySQL

```
SELECT DISTINCT relation_infirmiere_parents
FROM fiches_originelles ;
```

Access

107



Solution de l'exercice n°4 p. 85 Un vers partagé en 3 se caractérise soit parce que l'attribut vers comprend 2 barres obliques séparées, soit parce que l'attribut personnage_s inclut deux espaces séparés par des majuscules.

Pour MySQL, introduit par **REGEXP**, le motif correspondant au premier cas de figure serait '/.+/' et au second '␣[A–Z]+␣'.

Pour Access, introduit par COMME, le motif correspondant au premier cas de figure serait "∗/∗/∗" et au second '∗␣∗␣∗'.

Solution de l'exercice n°5 p. 94 Tirer au hasard le texte d'une fiche de PRÉMA peut s'effectuer par la requête :

R_{30}^{2}
PROJECTION(texte)[fiches_originelles]
↪ TRI ALÉATOIRE[<résultat$_1$>]
↪ LIMITÉ À(1)[<résultat$_2$>]

MySQL

La requête MysQL correspondante :

```
SELECT texte
FROM fiches_originelles
ORDER BY RAND()
LIMIT 1 ;
```

peut retourner le texte : « Dort BB ss PHOTO Tonique pendant les soins ». Un autre appel produirait un autre texte de fiche tirée au hasard.

Access

Solution de l'exercice n°6 p. 94 Tirer au hasard un vers prononcé par Phèdre peut s'effectuer par la requête :

R_{31}^{2}
RESTRICTION(personnage_s = 'PHEDRE')[vers]
↪ PROJECTION(vers)[<résultat$_1$>]
↪ TRI ALÉATOIRE[<résultat$_2$>]
↪ LIMITÉ À(1)[<résultat$_3$>]

MySQL

La réalisation est la suivante :

```
SELECT vers
FROM vers
WHERE personnage_s = 'PHEDRE'
ORDER BY RAND()
LIMIT 1 ;
```

peut retourner la table contenant la ligne : « Où laissé-je égarer mes voeux et mon esprit? » (v. 180). Un autre appel produirait un autre vers énoncé par Phèdre et tiré au hasard parmi les 464 que prononce le personnage.

Access

Solution de l'exercice n°7 p. 103 Le motif de la requête MySQL peut se paraphraser ainsi : tel que l'attribut categorie comprenne 0 ou n V majuscule(s), à n'importe quel endroit. D'où l'apparition de lignes à l'attribut categorie ne comprenant pas du tout le caractère V majuscule.

Le motif de la requête Access a par contre pour glose : tel que l'attribut categorie commence par un V suivi de 0 ou n caractères, c'est-à-dire tel qu'il s'agisse d'un verbe.

On retrouve l'opposition entre les expressions régulières de MySQL qui ne sont pas « ancrées » et les motifs Access, qui le sont. Et l'opposition entre l'étoile qui pour Access signifie 0 ou n caractères, tandis que pour MySQL, c'est un modifieur post-posé qui signifie : 0 ou n occurrences du motif qui précède (dans cet exemple, la seule lettre V).

L'équivalent exact de la requête Access en MySQL est :

```
SELECT lemme,
     categorie
FROM occ_prema
WHERE categorie REGEXP '^V.*'
ORDER BY RAND()
LIMIT 10 ;
```

qui a pour résultat par exemple :

lemme	*categorie*
pleurer	V.ip3s
c'est	V.ip3s
ressentir	V.ip3s
aimer	V.ip3s
s'agripper	V.n–
être	V.ip3s
être	V.ip3s
être dérangé	V.n–+V.pafs
fixer	V.ip3s
se crisper	V.ip3s

CHAPITRE IV

CALCULER DE NOUVELLES INFORMATIONS

1. Indicateurs globaux

≃ nombre d'infirmières

R^1_{38} *t. 1* p. 84

```
PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'Nbre inf.'
)[infirmieres]
```

≃ nombre d'infirmières de jour

R^1_{39} *t. 1* p. 84

```
RESTRICTION(service = 'Jour')[infirmieres]
↪ PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'Nbre inf. jour'
)[<résultat1>]
```

≃ nombre d'infirmières de nuit

R^1_{40} *t. 1* p. 84

```
RESTRICTION(service = 'Nuit')[infirmieres]
↪ PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'Nbre inf. nuit'
)[<résultat1>]
```

≃ nombre d'infirmières de jour ou de nuit

R^1_{41} *t. 1* p. 84

```
RESTRICTION(service Parmi ('Nuit', 'Jour'))[infirmieres]
↪ PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'Nbre inf. jour + nuit'
)[<résultat1>]
```

$\simeq$ nombre d'infirmières ne travaillant pas de jour

R^1_{42} *t. 1* p. 84

```
RESTRICTION(service <> 'Jour')[infirmieres]
↪
PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'Nbre inf. <> jour'
)[<résultat1>]
```

MySQL

Les requêtes suivantes fournissent chacune une table d'une seule ligne, avec un ou des indicateur(s) sur un (sous-)ensemble de lignes de la table infirmieres. L'instruction **COUNT**(∗) compte le nombre de lignes de la table.

La première requête donne le nombre total de lignes de la table, les deux suivantes, celui des infirmières de jour puis de nuit, la quatrième celui des infirmières de jour ou de nuit, la cinquième celle des infirmières qui ne sont pas de jour (ce qui correspond en l'occurrence aux infirmières de nuit, n'est pas prise en compte l'infirmière dont le service n'est pas connu et est marqué par **NULL**). Les trois dernières requêtes (tableau 25(a) (b) et (c), p. 121) fournissent davantage de renseignements globaux : âge minimal (**MIN**(age)) ou maximal (**MAX**(age)), âge moyen (**AVG** abrège *average* : moyenne) et la dispersion de cette moyenne via l'écart-type (**STD** pour *standard deviation*). Les deux dernières font appel à des alias de colonnes ainsi qu'à des instructions de formatage pour la moyenne et pour l'écart-type. Les chiffres après le point décimal sont réduits à deux par l'appel à **FORMAT** qui prend comme arguments un nombre et le nombre de chiffres à conserver en partie décimale.

R^1_{38} *t. 1* p. 84

```
SELECT
    COUNT(*) AS 'Nbre inf.'
FROM infirmieres_princeps ;
```

R^1_{39} *t. 1* p. 84

```
SELECT
    COUNT(*) AS 'Nbre inf. jour'
FROM infirmieres_princeps
WHERE service = 'Jour' ;
```

R^1_{40} *t. 1* p. 84

```
SELECT
    COUNT(*) AS 'Nbre inf. nuit'
FROM infirmieres_princeps
WHERE service = 'Nuit' ;
```

R^1_{41} *t. 1* p. 84

```
SELECT
    COUNT(*) AS 'Nbre inf. jour + nuit'
FROM infirmieres_princeps
WHERE service IN ('Jour', 'Nuit') ;
```

R^1_{42} *t. 1* p. 84

```
SELECT
    COUNT(*) AS 'Nbre inf. <> jour'
FROM infirmieres_princeps
WHERE service <> 'Jour' ;
```

R^2_{32}

RESTRICTION(service = 'Jour')[infirmieres]
↪
PAR GROUPE(
NOMBRE DE LIGNES(),
MINIMUM(age),
MAXIMUM(age),
MOYENNE(age),
ÉCART-TYPE(age),
)[<résultat$_1$>]

```
SELECT
    COUNT(*),
    MIN(age),
    MAX(age),
    AVG(age),
    STD(age)
FROM infirmieres_princeps
WHERE service = 'Jour' ;
```

R^2_{33}

RESTRICTION(service = 'Jour')[infirmieres]
↪
PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'Nbre inf. jour',
MINIMUM(age) TITRE 'âge min.',
MAXIMUM(age) TITRE 'âge max.',
MOYENNE(age) TITRE 'âge moy.',
ÉCART-TYPE(age) TITRE 'écart-type âge',
)[<résultat$_1$>]

```
SELECT
    COUNT(*) AS 'Nbre inf. jour',
    MIN(age) AS 'âge min.',
    MAX(age) AS 'âge max.',
    FORMAT(AVG(age), 2) AS 'âge moy.',
    FORMAT(STD(age), 2) AS 'écart-type âge'
FROM infirmieres_princeps
WHERE service = 'Jour' ;
```

R^2_{34}

```
SELECT
    COUNT(*) AS 'Nbre inf. nuit',
    ...
FROM infirmieres_princeps
WHERE service = 'Nuit' ;
```

△

On distinguera soigneusement :
COUNT(*) qui compte le nombre de lignes et :
COUNT(**DISTINCT** <attribut>) qui compte le nombre de valeurs distinctes de l'attribut mentionné.
Le montre la requête suivante :

TABLEAU 25 – PRÉMA : indicateurs sur l'âge des infirmières

(a) Infirmières de jour – version de départ

R^2_{32} p. 120

COUNT(*)	*MIN(age)*	*MAX(age)*	*AVG(age)*	*STD(age)*
29	21	44	29.0000	4.7850

(b) Infirmières travaillant de jour – version retitrée et formatée

R^2_{33} p. 120

Nbre inf. jour	*âge min.*	*âge max.*	*âge moy.*	*écart-type âge*
29	21	44	29.00	4.79

(c) Infirmières travaillant de nuit – version retitrée et formatée

R^2_{34} p. 120

Nbre inf. nuit	*âge min.*	*âge max.*	*âge moy.*	*écart-type âge*
12	23	40	33.08	5.22

≈ nombre total d'infirmières travaillant de jour et nombre d'âges distincts pour ces infirmières

R^2_{35}

RESTRICTION(service = 'Jour')[infirmieres_princeps]
↪
PAR GROUPE(
NOMBRE DE LIGNES(),
NOMBRE DE VALEURS DISTINCTES(age)
)[<résultat$_1$>]

```
SELECT
      COUNT(*),
      COUNT(DISTINCT age)
FROM infirmieres_princeps
WHERE service = 'Jour' ;
```

qui produit comme résultat :

COUNT(*)	*COUNT(DISTINCT age)*
29	13

On obtient le nombre total d'infirmières travaillant de jour (29) et le nombre d'âges distincts de ces infirmières : 13 seulement.

On rend les titres de colonnes plus explicites :

Infirmières de jour	*âges différents*
29	13

via :

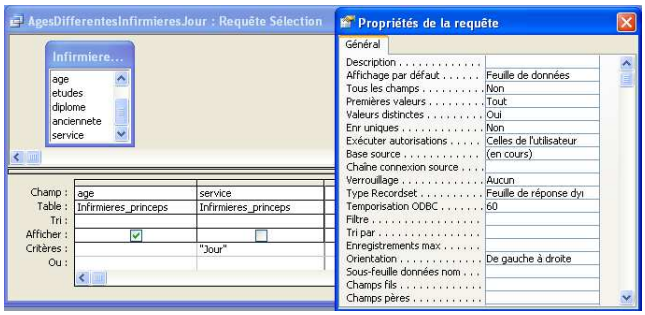
```
SELECT COUNT(*) AS 'Infirmières de jour',
      COUNT(DISTINCT age) AS 'âges différents'
FROM infirmieres
WHERE service = 'Jour' ;
```

Access

La réalisation Access de l'association du décompte du nombre total d'infirmières de jour et du nombre d'âges différents de ces infirmières s'effectue via une **requête union**. Une requête union prend deux tables de même structure (ayant les mêmes attributs) et enchaîne les lignes de l'une et les lignes de l'autre. Dans le cas présent, on cherche à obtenir 2 tables, d'une ligne chacune et ayant pour attributs Légende et nombre, qui prendront comme valeur respectivement le sens de la ligne et le nombre d'individus correspondants.

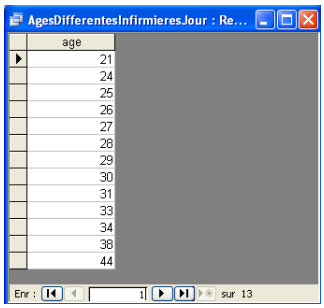
La première étape consiste à obtenir le nombre d'âges différents pour les infirmières de jour. Cela suppose d'abord de conserver uniquement les valeurs distinctes de l'attribut age pour les seules infirmières de jour.

1



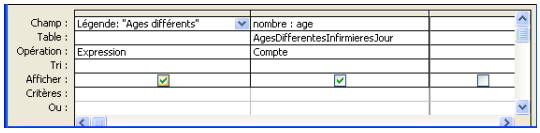
La table résultante ne garde effectivement que les 13 âges différents.

2



Une nouvelle requête prend en entrée la requête précédente. Utilisant les regroupements, elle calcule en deuxième colonne, sous le nom nombre, le décompte des lignes et en première colonne, dénommée Légende, elle place une chaîne : 'Ages différents'.

3

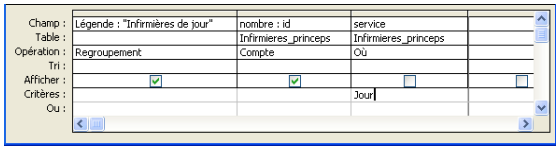


Le résultat est une table d'une seule ligne.

4

La requête suivante calcule de la même manière, mais sans avoir besoin de passer par une requête intermédiaire, le nombre total d'infirmières de jour.

5

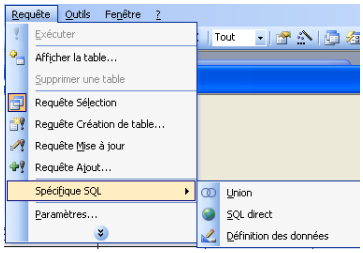


Le résultat a la même structure que 4.

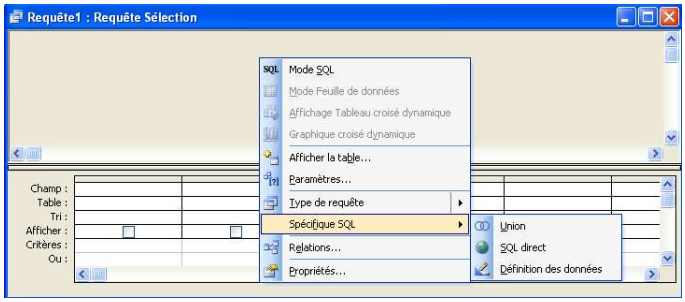
6

On crée alors une nouvelle requête et on indique qu'on veut une requête union soit via ⌈Requête ❙ Spécifique SQL ❙ Union ⌉ 7, soit, en cliquant dans la sous-fenêtre de haut de formulation interactive de la requête, via ⌈Mode SQL ❙ Union ⌉ 8.

7



8



On formule textuellement la requête union dans la fenêtre qui apparaît. Le tri par valeurs décroissantes de l'attribut Légende a pour fonction de faire apparaître les lignes dans l'ordre le plus naturel possible :

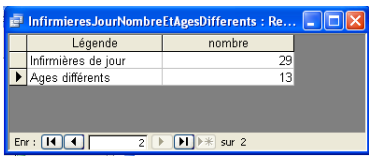
```
SELECT *
FROM NombreInfirmieresJour

UNION

SELECT *
FROM NbreAgesDifferentsInfirmieresJour;

ORDER BY [Légende] DESC;
```

Le résultat est bien conforme à ce qu'on souhaite, à ceci près que la solution MySQL s'effectue en « largeur » (le nombre total d'infirmières de jour est à gauche du nombre d'âges différents pour ces mêmes infirmières) tandis que la solution Access/SQL Server est en « hauteur ». Par ailleurs, la solution MySQL reste plus aisée et plus naturelle.

TABLEAU 26 – PRÉMA : renommer les colonnes dans une table résultat

Identifiant	*âge*	*années d'étude*	*ancienneté en néonatalité*
41	26	3	0.00
46	21	3	0.00
81	25	3	0.00
97	25	3	0.00
1	23	4	0.00
⋮	⋮	⋮	⋮

9



Le nom d'une requête union est précédé de deux anneaux.

10

Renommer les colonnes de la table résultat

MySQL

On peut associer des **alias** aux tables sources pour désambiguïser les noms d'attributs identiques dans ces tables et faciliter (raccourcir) le renvoi à ces tables. On peut opérer de la même manière pour les attributs d'une relation. L'objectif est alors d'obtenir des noms de colonnes plus parlants dans les tables résultats. C'est ce qu'exemplifie le tableau 26 p. 124.

Les alias de colonne sont des chaînes de caractères, délimitées soit par les guillemets soit par l'apostrophe. La présence d'une apostrophe interne ne pose pas de difficultés quand la chaîne est délimitée par des guillemets. Dans le cas contraire, une contre-oblique (*slash*) indique qu'il faut prendre l'apostrophe interne comme une simple apostrophe et non pas comme la fin de la chaîne choisie comme alias de colonne.

Pour la requête abstraite suivante :

≈ identifiant, âge et ancienneté des infirmières n'ayant pas d'ancienneté

R^2_{36}

RESTRICTION(anciennete = 0)[infirmieres]

↪

PROJECTION AVEC DOUBLONS(
id TITRE 'Identifiant',
age TITRE 'âge',
etudes TITRE "années d'étude"
)[<résultat$_1$>]

les deux formes suivantes de requête sont donc admissibles et engendrent le tableau 26 p. 124 :

```
SELECT id AS 'Identifiant',
       age AS 'âge',
       etudes AS "années d'étude",
```

```
FROM infirmieres
WHERE anciennete = 0 ;

SELECT ...
      etudes AS 'années␣d\'étude',
FROM␣...␣;␣
```

tandis que le remplacement de la 3^{e} ligne dans la requête par :

```
      etudes AS 'années␣d'étude'
```

produit une erreur, puisque l'interprète de commande rencontre d'abord la chaîne 'années␣d', considérée comme l'alias de la colonne etudes puis des caractères (etude) « en l'air », sans rôle, puis une nouvelle chaîne qui commence après etude et dont il attend la fin, si bien qu'une fois tapé le point-virgule de fin de requête et le passage à la ligne effectué, il ne se passe rien. L'ajout d'une apostrophe qui termine la chaîne en cours débouche sur un message d'erreur :

```
ERROR 1064 : You have an error in your SQL syntax. Check the manual that
corresponds to your MySQL server version for the right syntax to use near
'étude', anciennete AS 'ancienneté en néonatalité' FROM infirmie...
```

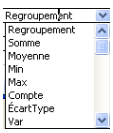
Access

La solution la plus simple sous l'interface est de faire appel aux menus liés aux regroupements. Cliquer sur le signe mathématique de somme – le sigma majuscule ∑ [11] – dans la barre du haut, fait apparaître une nouvelle ligne, ⌈Opération⌉ dans la fenêtre de définition d'une requête. Un menu déroulant dans chaque case de cette ligne permet de choisir ⌈Regroupement⌉ (valeur par défaut – cf. infra) ou une opération sur un ensemble de lignes analogue à celles manipulées sous MySQL.

[11]

En [12] et en [13] figurent les opérations globales permises : regroupement, décompte, somme, maximum, minimum, variance (var – carré de l'écart type), etc.

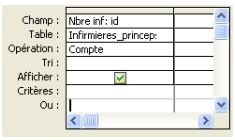
[12]



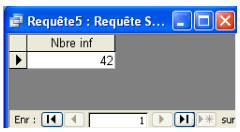
[13]

En 14, est opéré le décompte du nombre d'infirmières, par le biais de celui de leurs identifiants. Par ailleurs, 'Nbre inf' suivi de deux points avant le nom de la colonne indique le titre souhaité dans la feuille de données résultat, comme on le constate en 15.

14



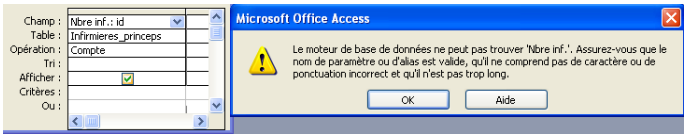
15



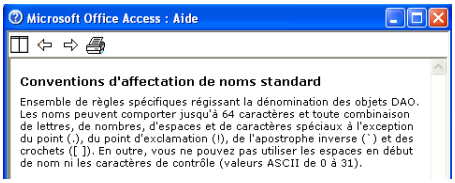
△

On notera que le titre ne peut comporter certains caractères : 16, comme le précise plus ou moins clairement l'aide : 17.

16

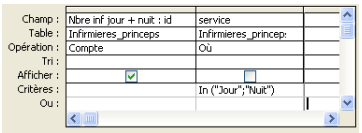


17

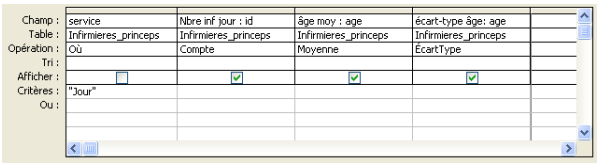


Pour opérer un décompte sur une partie seulement des lignes, on opère une restriction en choisissant 'Où' comme opération dans la ou les colonne(s) servant à la restriction. C'est l'équivalent du **WHERE** sous SQL, qui intervient en aval du regroupement (on opère d'abord la restriction et le regroupement opère sur les lignes restantes). On trouve ainsi en 18 la requête pour le nombre d'infirmières travaillant de jour ou de nuit et en 19 celle pour le nombre, l'âge moyen et l'écart type pour l'âge des infirmières de jour : 20.

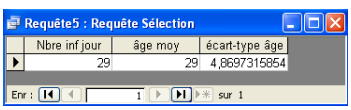
18



19



20



PRÉMA

Exercice n°1 Fournir le nombre de fiches total, puis le nombre de fiches rédigées le jour 1. Et continuer pour les jours 3, 5 et 7.

Exercice n°2 Fournir le nombre de lemmes distincts (types) et le nombre total de lemmes (occurrences) présents dans la table occ_prema et qui ont une catégorie adéquate (cf. chapitre I § 7), c'est-à-dire commençant par A(djectif), N(om), R (pour AdveRbe), S (pour PrépoSition), D(éterminant), V(erbe), C(onjonction), P(ronom).

Exercice n°3 Pour les filles puis pour les garçons, à la naissance, calculer le nombre d'individus, le poids minimal, le poids maximal, le poids moyen et l'écart-type.

PHÈDRE

Exercice n°4 Donner le nombre de vers proférés par Phèdre et où Phèdre est le seul personnage à intervenir.

Exercice n°5 Donner le nombre de vers prononcés par Oenone seule, le numéro du premier vers qu'elle profère, ainsi que le numéro du dernier. Faire de même pour Thésée.

Exercice n°6 Que faudrait-il changer aux requêtes de l'exercice 5 pour prendre en compte également à chaque fois les vers partagés avec un autre personnage (par exemple ceux qu'Oenone partage avec Phèdre) ?

ESQUE

MySQL

La requête :

```
SELECT COUNT(*) AS 'Attestations',
       COUNT(DISTINCT derive) AS 'dérivés'
FROM esque ;
```

opère le décompte de toutes les lignes de la table esque via ⌈**COUNT**(∗)⌉ et celui de toutes les valeurs distinctes de l'attribut derive. Il y a 4 384 entités différentes dans la table esque, 4 384 attestations donc, mais seulement 3 406 dérivés en *-esque* distincts. La requête fournit le résultat du tableau 27, p. 128. Son équivalent abstrait se présente de la manière suivante :

R^2_{37}
PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'Attestations',
NOMBRE DE VALEURS DISTINCTES(derive) TITRE 'dérivés'
)[esque]

Exercice n°7 Sachant que dans la table esque, la valeur 0 est utilisée pour indiquer la non-réponse pour les attributs jour, mois et annee, fournir le nombre de lignes de cette table qui sont situées dans le temps.

TABLEAU 27 – ESQUE : nombre d'attestations et de dérivés

Attestations	*dérivés*
4384	3406

2. Regroupements

≃ tri des infirmières par service

R^1_{44} *t. 1* p. 87

```
TRI SUR(service)[infirmieres]
```

≃ regroupement des infirmières par service

R^1_{45} *t. 1* p. 87

```
REGROUPER SUR(service)[infirmieres]
```

≃ par service, nombre d'infirmières, âge minimum, maximum, moyen, ancienneté minimum, maximum, moyenne

R^1_{46} *t. 1* p. 89

```
REGROUPER SUR(service)[infirmieres]
↪
PAR GROUPE(
service TITRE 'Service',
NOMBRE DE LIGNES() TITRE 'nbre',
MINIMUM(age) TITRE 'âge min.',
MAXIMUM(age) TITRE 'âge max.',
MOYENNE(age) TITRE 'âge moy.',
MINIMUM(anciennete) TITRE 'anc. min.',
MAXIMUM(anciennete) TITRE 'anc. max.',
MOYENNE(anciennete) TITRE 'anc. moy.'
)[<résultat1>]
```

≃ pour le lemme 'bébé', nombre d'occurrences de chaque forme_depart associée

R^1_{47} *t. 1* p. 90

```
RESTRICTION(lemme = 'bébé')[occ_prema]
↪
REGROUPER SUR(forme_depart)[<résultat1>]
↪
PAR GROUPE(
forme_depart,
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat2>]
```

≃ pour le lemme 'bébé', nombre d'occurrences > 1 de chaque forme_depart associée

R^1_{48} *t. 1* p. 91

```
RESTRICTION(lemme = 'bébé')[occ_prema]
↪
REGROUPER SUR(forme_depart)[<résultat1>]
↪
PAR GROUPE(
forme_depart,
NOMBRE DE LIGNES() TITRE 'o.'
)
AVEC (NOMBRE DE LIGNES() > 1)
[<résultat2>]
```

MySQL

Le tri des infirmières par service s'obtient par la requête :

R^1_{44} *t. 1* p. 87

```
SELECT *
FROM infirmieres_princeps
ORDER BY service ;
```

Le regroupement par service par :

R^1_{45} *t. 1* p. 87

```
SELECT *
FROM infirmieres_princeps
GROUP BY service ;
```

L'obtention d'indicateurs globaux pour chacun des regroupements opérés (**NULL**, Jour, Nuit) par :

R^1_{46} *t. 1* p. 89

```
SELECT
      service AS 'Service',
      COUNT(*) AS 'nbre',
      MIN(age) AS 'âge min.',
      MAX(age) AS 'âge max.',
      FORMAT(AVG(age), 2) AS 'âge moy.',
      MIN(anciennete) AS 'anc. min.',
      MAX(anciennete) AS 'anc. max.',
      FORMAT(AVG(anciennete), 2) AS 'anc. moy.'
FROM infirmieres_princeps
GROUP BY service ;
```

La restriction aux seules infirmières dont le service est effectivement connu s'obtient en ajoutant une clause **WHERE** (avant la clause **GROUP BY**), ce qui donne :

```
SELECT
      ...
FROM infirmieres_princeps
WHERE service IS NOT NULL
GROUP BY service ;
```

Pour obtenir le nombre d'occurrences de chacune des réalisations de *bébé* (R^1_{47} *t. 1* p. 90), on a recours à la requête :

```
SELECT
      forme_depart,
      COUNT(*) AS 'o.'
FROM occ_prema
WHERE lemme = 'bébé'
GROUP BY forme_depart   ;
```

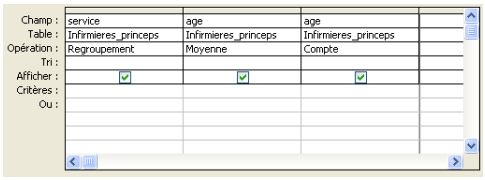
L'ajout d'une instruction [**ORDER BY** 'o.'**DESC**] après la clause [**GROUP BY**] trie le résultat par nombre d'occurrences décroissant. L'ajout de **HAVING** permet de ne garder que les réalisations qui interviennent plus d'une fois (R^1_{48} *t. 1* p. 91).

```
SELECT
      forme_depart,
      COUNT(*) AS 'o.'
FROM occ_prema
WHERE lemme = 'bébé'
GROUP BY forme_depart
      HAVING COUNT(*) > 1
ORDER BY 'o' DESC ;
```

Access

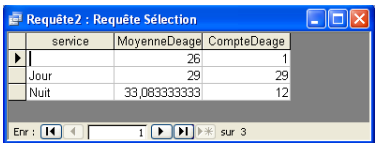
Le choix de l'opération ⌈Regroupement⌉ effectue le regroupement selon les valeurs ou les modalités de la ou des colonne(s) choisie(s). La requête 21 permet donc le calcul de l'âge moyen et du nombre pour chaque groupe d'infirmières selon le service effectué (Jour, Nuit, **NULL**).

21



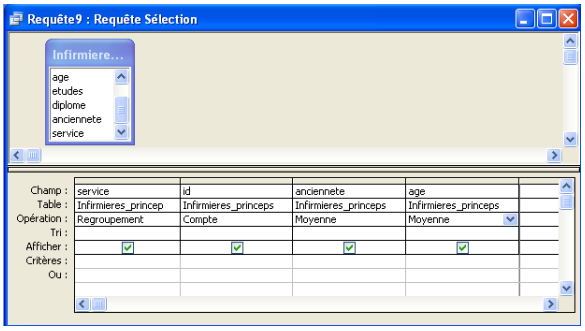
Le résultat brut 22 rend manifeste l'intérêt de donner des titres sinon plus parlants du moins plus euphoniques que ceux que fournit par défaut Access.

22

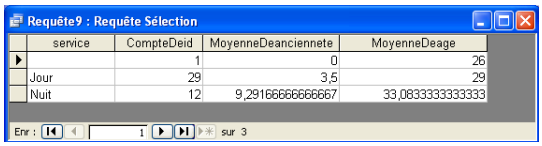


La requête 23, dans le cadre d'un regroupement par service effectué, effectue le décompte d'infirmières par groupe (appel à ⌈Compte⌉ pour la colonne id et la moyenne pour les colonnes anciennete et age : 24. On se reportera à la requête 36 infra pour la manière d'arrondir un nombre décimal.

23

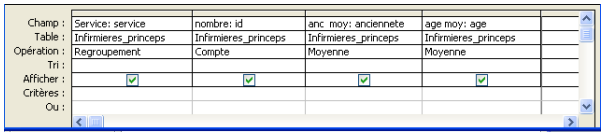


24

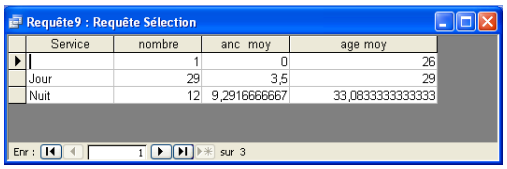


La requête 25 diffère de la précédente par des alias de colonnes : 26.

25

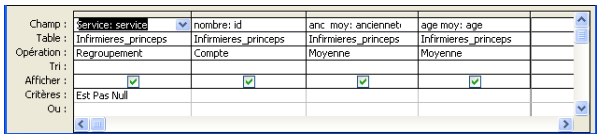


26



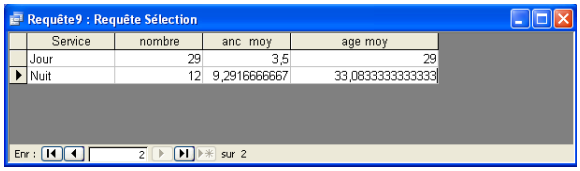
La requête 27 ajoute une contrainte sur les groupes résultants du regroupement par service : la colonne service ne doit pas contenir la marque **NULL**. C'est l'équivalent d'une clause **HAVING** en SQL.

27



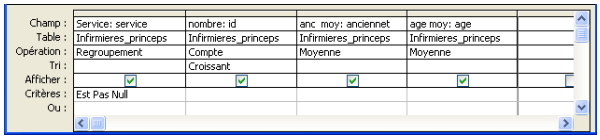
Le résultat ne comprend plus que les 2 lignes qui obéissent à cette contrainte : 28.

28

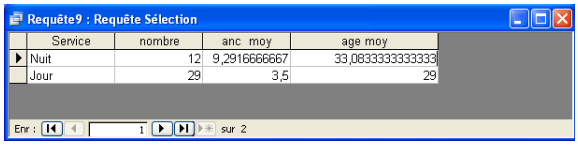


La requête 29 ajoute à la précédente le tri par ordre croissant du nombre d'infirmières par groupes : 30.

29



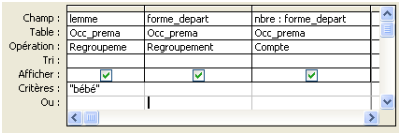
30



La requête 31 indique comment obtenir les différentes réalisations du lemme 'bébé'. Un regroupement est effectué sur deux colonnes, lemme d'abord, forme_depart ensuite. La restriction sur la colonne lemme conserve uniquement les lignes dont le lemme est 'bébé'. Pour

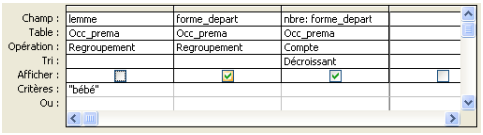
chacune des réalisations de forme_depart pour ce lemme, on demande le calcul du nombre de lignes correspondantes.

31



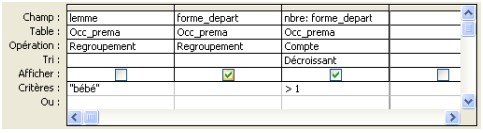
La requête 32 est une simple modification de la précédente, elle n'affiche pas la première colonne, qui ne sert qu'à la restriction au lemme visé.

32



La requête 33 ajoute une condition sur les regroupements retenus : pour figurer dans le résultat, le compte doit être supérieur à 1. C'est l'équivalent de ⌈**HAVING COUNT**(∗) > 1⌉. On élimine ainsi les hapax. Par ailleurs, on trie les lignes par nombre de lignes décroissant.

33



2.1. ⊕ Calcul de pourcentages en connaissant l'effectif de référence

≃ pour chaque regroupement des bébés par sexe, sexe, nombre d'individus et pourcentage par rapport au total

R^{1}_{49} *t. 1* p. 91

```
REGROUPER SUR(sexe)[bebes]
↪
PAR GROUPE(
sexe TITRE 'Sexe',
NOMBRE DE LIGNES() TITRE 'nbre',
NOMBRE DE LIGNES() / 121 * 100 TITRE '%'
)[<résultat1>]
```

≃ pour chaque regroupement des bébés selon la situation à J3, sexe, nombre d'individus et pourcentage par rapport au total

R^{1}_{50} *t. 1* p. 92

```
REGROUPER SUR(etat_j3)[bebes]
↪
PAR GROUPE(
etat_J3,
NOMBRE DE LIGNES() TITRE 'nbre',
NOMBRE DE LIGNES() / 121 * 100 TITRE '%'
)[<résultat1>]
```

≃ pour chaque regroupements des bébés par sexe puis par situation à J3, sexe, situation à J3, nombre d'individus et pourcentage par rapport au total

R^1_{51} *t. 1* p. 93

```
REGROUPER SUR(sexe, etat_j3)[bebes]
↪
PAR GROUPE(
sexe TITRE 'Sexe', etat_J3,
NOMBRE DE LIGNES() TITRE 'nbre',
NOMBRE DE LIGNES() / 121 * 100 TITRE '%'
)[<résultat1>]
```

$\simeq$ pour chaque regroupements des bébés par situation à J3 puis par sexe, sexe, situation à J3, nombre d'individus et pourcentage par rapport au total

R^1_{52} *t. 1* p. 93

```
REGROUPER SUR(etat_j3, sexe)[bebes]
↪
PAR GROUPE(
sexe TITRE 'Sexe', etat_J3,
NOMBRE DE LIGNES() TITRE 'nbre',
NOMBRE DE LIGNES() / 121 * 100 TITRE '%'
)[<résultat1>]
```

MySQL

On peut obtenir simplement le nombre de bébés de chaque sexe à la naissance : ⌈COUNT(*)⌉. Comme il y a au total 121 prématurés étudiés, le pourcentage occupé par chaque sexe se calcule aisément, c'est le nombre de bébés du groupe courant divisé par l'effectif total (121) et multiplié par 100 :

R^1_{49} *t. 1* p. 91

```
SELECT sexe AS 'Sexe',
    COUNT(*) AS 'nbre',
    COUNT(*) / 121 * 100 AS '%'
FROM bebes
GROUP BY sexe ;
```

L'obtention de la situation (mort, sortie de réanimation, réanimation) à J3 découle de la requête :

R^1_{50} *t. 1* p. 92

```
SELECT
    etat_J3,
    COUNT(*) AS 'nbre',
    FORMAT(COUNT(*) / 121 * 100, 2) AS '%'
FROM bebes_princeps
GROUP BY etat_j3 ;
```

Le croisement du sexe et de la situation à J3, en privilégiant d'abord le sexe et ensuite la situation, s'obtient par les deux requêtes :

R^1_{51} *t. 1* p. 93

```
SELECT
    sexe AS 'Sexe',
    etat_J3,
    COUNT(*) AS 'nbre',
    FORMAT(COUNT(*) / 121 * 100, 2) AS '%'
FROM bebes_princeps
GROUP BY sexe, etat_j3 ;
```

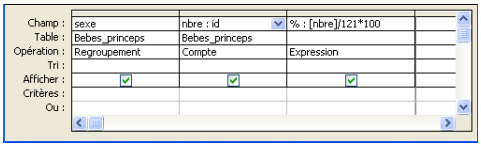
R^1_{52} *t. 1* p. 93

```
SELECT
      sexe AS 'Sexe',
      etat_J3,
      COUNT(*) AS 'nbre',
      FORMAT(COUNT(*) / 121 * 100, 2) AS '%'
FROM bebes_princeps
GROUP BY etat_j3, sexe ;
```

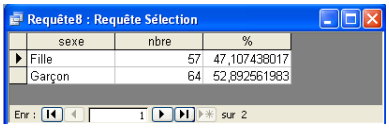
Access

34 correspond à la R^1_{49} *t. 1* p. 91. Son résultat figure en 35.

34

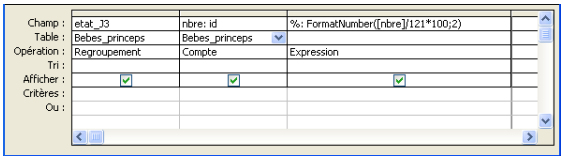


35

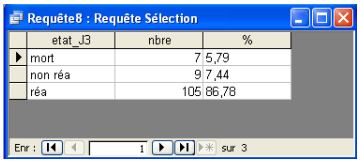


Le résultat 37 de la requête 36 (R^1_{50} *t. 1* p. 92) fait appel, pour élaguer les décimales, à la fonction **FORMATNUMBER**(<nombre décimal> ; <nombre de décimales à conserver après le séparateur décimal>).

36



37



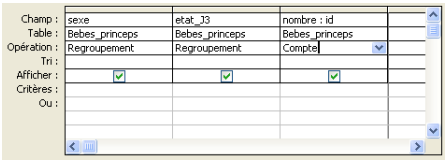
△

En comparant la requête formulée via l'interface graphique et celle engendrée en SQL Server, on note le passage du point-virgule à la virgule pour séparer les arguments de la fonction **FORMATNUMBER**.

```
SELECT Bebes_princeps.etat_J3,
      Count(Bebes_princeps.id) AS nbre,
      FormatNumber([nbre]/121*100,2) AS [%]
FROM Bebes_princeps
GROUP BY Bebes_princeps.etat_J3;
```

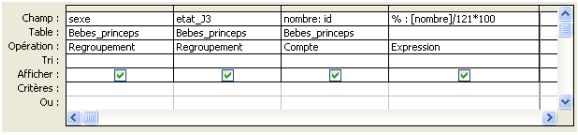
La requête 38 opère un regroupement sur le sexe puis sur l'état à J3 des bébés. Cela permet le décompte des lignes correspondant au croisement des modalités de ces deux variables : 2 pour le sexe et 3 pour l'état : en réanimation, décédé(e) ou sorti(e) du service de réanimation.

38

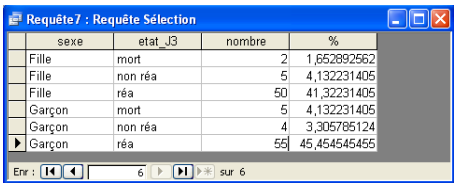


La requête 39 est une variante : elle ajoute le calcul du pourcentage pour chacun des cas de figure, comme le montre le résultat 40 (il reste à conserver un nombre raisonnable de chiffres décimaux). Elle correspond à la R^1_{51} *t. 1* p. 93. Pour obtenir ce pourcentage, on fait appel, en le mettant entre crochets, au nom nombre donné à un attribut calculé, le décompte du nombre de valeurs de l'attribut id pour un regroupement des attributs sexe et etat_J3.

39



40



2.2. ⊕ Calcul de pourcentages sans connaître l'effectif de référence

Pourcentage de fiches à J1 Le calcul de pourcentage dans cet exemple comme dans les précédents suppose de déterminer l'effectif total de référence. Cette détermination peut découler d'une requête préalable. On peut heureusement s'affranchir de ce pré-requis, comme le montre la requête suivante (tableau 28(haut) p. 136) :

R^2_{38}

```
PAR GROUPE(
'J1',
SOMME(SI(jour = 1, 1, 0)) TITRE 'fiches',
SOMME(SI(jour = 1, 1, 0)) / NOMBRE DE LIGNES() * 100
TITRE '%'
)[fiches_originelles]
```

Elle n'opère pas de regroupements, à la différence de la précédente. L'expression ⌈NOMBRE DE LIGNES()⌉ retourne donc le nombre total de fiches. L'opérateur
SI(<condition>, <résultat pour succès>, <résultat pour échec>)
appliqué à chaque fiche retourne la valeur 1 si la fiche est bien du jour 1 et 0 sinon. La somme (SOMME) des appels à cet opérateur est le nombre de fiches du jour 1. Le calcul du pourcentage des fiches du jour 1 s'effectue en utilisant ce nombre. Noter que la première colonne de la table résultat est occupée par la chaîne J1 qui est placée directement dans ce qui est retourné. La généralisation du calcul « à la volée » du total de fiches et du nombre pour un jour donné aboutit au tableau 28(bas) p. 136.

MySQL

TABLEAU 28 – PRÉMA : répartition des fiches par jour

R^2_{38} p. 135

J1	*fiches*	*%*
J1	327	32.15

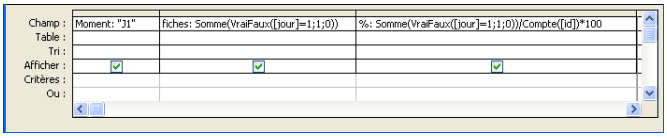
Fiches	*J1*	*% J1*	*J3*	*% J3*	*J7*	*% J7*	*J15*	*% J15*
1017	327	32.15	287	28.22	225	22.12	178	17.50

```
SELECT
      'J1',
      SUM(IF(jour = 1, 1, 0)) AS 'fiches',
      SUM(IF(jour = 1, 1, 0)) / COUNT(*) * 100 AS '%'
FROM fiches_originelles ;
```

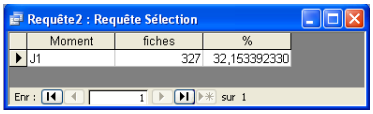
Access

Dans la requête 41, la fonction **VraiFaux**, dont les arguments sont séparés par des points-virgules, donne naissance à l'« aiguillage » **IIF** en SQL Server, tandis que **Somme** y correspond à **SUM**. La première colonne du résultat 42, renommée Moment, contient la valeur littérale "J1", ce que signale l'encadrement par les guillemets.

41



42



Access engendre la requête SQL Server suivante :

```
SELECT "J1" AS Moment,
      Sum(IIf([jour]=1,1,0)) AS fiches,
      Sum(IIf([jour]=1,1,0))/Count([id])*100 AS [%]
FROM fiches_originelles;
```

Évolution des dénominations des bébés Le tableau 29 p. 137 utilise la même démarche pour esquisser l'évolution des dénominations des bébés du jour 1 au jour 15. La requête sous-jacente, abrégée, montre la complexité des conditions qui peuvent être employées :

TABLEAU 29 – PRÉMA : les dénominations des bébés au fil du temps

R^2_{39} p. 137

Jour	*f.*	*bébé*	%	*garçon*	%	*fille*	%	*préma*	%	*enfant*	%
1	327	248	75.84	5	1.53	31	9.48	3	0.92	14	4.28
3	287	212	73.87	4	1.39	34	11.85	3	1.05	15	5.23
7	225	172	76.44	4	1.78	28	12.44	0	0.00	11	4.89
15	178	127	71.35	6	3.37	21	11.80	0	0.00	13	7.30

R^2_{39}

```
REGROUPER SUR(jour)[fiches_originelles]
↪
PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'f.',
SOMME(SI(Texte RESSEMBLANT À '[Bb][Eeé.]?[Bb][Eeé.]?', 1,
0)) TITRE 'bébé',
SOMME(SI(Texte RESSEMBLANT À '[Bb][Eeé.]?[Bb][Eeé.]?', 1,
0)) / NOMBRE DE LIGNES() * 100 Titre '%',
...
)[<résultat1>]
```

On vérifie en effet la présence d'expressions régulières dans chaque fiche. Une fiche peut d'ailleurs « contribuer » aux effectifs de plusieurs colonnes. Le tableau le montre, *préma(turé)* n'est employé qu'aux jours 1 et 3, *bébé* se tasse au fil du temps, tandis qu'une perception sexuée progresse, plus pour *garçon* que *fille*, sans que le décalage entre les deux mots soit comblé. Le lemme *Enfant* prend également plus de place.

MySQL

R^2_{39} p. 137

```
SELECT jour AS 'Jour',
     COUNT(*) AS 'f.',
     SUM(IF(texte REGEXP '[Bb][Eeé.]?[Bb][Eeé.]?', 1, 0))
          AS 'bébé',
     SUM(IF(texte REGEXP '[Bb][Eeé.]?[Bb][Eeé.]?', 1, 0))
          / COUNT(*) * 100 AS '%',
...
     SUM(IF(texte REGEXP '[Pp]réma'
          AND texte NOT REGEXP '[Pp]rématurité', 1, 0))
          AS 'préma',
     SUM(IF(texte REGEXP '[Pp]réma'
          AND texte NOT REGEXP '[Pp]rématurité', 1, 0))
          / COUNT(*) * 100 AS '%',
...
FROM fiches_originelles
GROUP BY jour ;
```

2.3. ⊕ Regroupements temporaires

MySQL

Un regroupement repose sur les différentes valeurs ou modalités d'une variable. On peut souhaiter opérer des regroupements temporaires, qui n'affectent pas les valeurs effectives de la table utilisée. L'opérateur **CASE** fonctionne comme une sorte d'aiguillage qui associe à chaque « voie » une condition et une valeur de sortie, et qui peut utiliser éventuellement une voie d'échappement, introduite par **ELSE**, pour couvrir tous les autres cas :

```
CASE
     WHEN <condition_1> THEN <valeur_1>
     ...
     WHEN <condition_k> THEN <valeur_k>
     [ELSE <valeur dans les autres cas>]
END
```

La requête :

```
SELECT DISTINCT poids_naissance
FROM bebes_princeps
LIMIT 10 ;
```

fournit les 10 premières lignes des poids de naissance des bébés :

poids_naissance
920
830
1620
1350
1045
1400
910
640
790
500

La requête

```
SELECT
     CASE
          WHEN poids_naissance < 750 THEN '< 750'
          WHEN poids_naissance > 1000 THEN '> 1000'
          ELSE '750-1000'
     END
     AS 'Classe poids naissance'
FROM bebes_princeps
LIMIT 10 ;
```

remplace la valeur du poids à la naissance par celle correspondant à la « voie » adéquate de l'aiguillage. Les deux premières lignes de la table correspondent à la troisième condition, la troisième à la deuxième condition et la dernière au premier cas de figure. On obtient alors la table :

Classe poids naissance
750-1000
750-1000
> 1000
> 1000
> 1000
750-1000
750-1000
> 1000
750-1000
< 750

Rien n'empêche dès lors d'opérer des regroupements sur de telles valeurs calculées. C'est ce qu'effectue la requête :

```
SELECT
      CASE
            WHEN poids_naissance < 750 THEN '< 750'
            WHEN poids_naissance > 1000 THEN '> 1000'
            ELSE '750-1000'
      END
            AS 'Classe poids naissance',
      COUNT(*) AS 'o.'
FROM bebes_princeps
GROUP BY
      CASE
            WHEN poids_naissance < 750 THEN '< 750'
            WHEN poids_naissance > 1000 THEN '> 1000'
            ELSE '750-1000'
      END ;
```

Elle produit la table :

Classe poids naissance	*o.*
750-1000	44
< 750	30
> 1000	47

PRÉMA

Exercice n°8 Sachant qu'il y a 121 bébés concernés par l'étude, calculer la répartition en nombre de bébés et en pourcentage entre accouchement par césarienne et par voie basse ainsi que celle entre naissance dite locale, à l'hôpital où s'effectue l'étude, et arrivée à l'hôpital après la naissance.

Exercice n°9 En utilisant les regroupements, donner le nombre et le pourcentage de fiches rédigées par jour de vie des bébés étudiés (les résultats seront proches de ceux du tableau 28(bas) p. 136), à partir du moment où l'on sait le nombre total de fiches, 1017.

Exercice n°10 Calculer en un seul tableau le nombre et le pourcentage de bébés qui sont encore dans le service de réanimation à chaque étape (J3 à J15).

Exercice n°11 Les fiches indiquent pour chaque bébé si, au moment du remplissage de la fiche, il est sous ventilation ou sous sédation. Les modalités de chacune de ces variables sont les suivantes :

Ventilation
intubé
non reponse
sonde nasale
non

Sédation
non
hypnovel ou fentanyl
canadou
non reponse

Il s'agit dans un premier temps, en utilisant les regroupements, de calculer, pour chaque variable, le nombre de fiches et le pourcentage de chaque modalité. Dans un deuxième temps, en utilisant les techniques de calcul de pourcentage présentées *supra*, fournir la répartition des modalités, en nombre et en pourcentage de fiches, au fil des jours.

Exercice n°12 (Access seulement)

Il s'agit d'utiliser l'analyse croisée fournie par Access pour examiner les interactions entre :

- sexe et sédation ;
- jour-sexe et sédation ;
- jour-sexe et ventilation.

Par jour-sexe, on entend le regroupement des informations sur la sédation ou sur la ventilation d'abord par jour et au sein d'un jour, par sexe. Cela permet d'opposer par exemple les filles et les garçons au jour 1.

On préparera, en entrée de toutes ces analyses croisées, une table (ou une requête) ayant pour attribut bébé, sexe, fiche, jour, sédation et ventilation.

Exercice n°13 (MySQL seulement) En utilisant **CASE**, regrouper les infirmières selon les classes d'ancienneté distinguées par l'équipe médicale : moins de 1 an, entre 1 et 5 ans, plus de 5 ans. Dans un deuxième temps, croiser ce regroupement avec le service des infirmières. Quelles conclusions peut-on en tirer ?

Exercice n°14 (Access seulement)

L'analyse croisée fournie par Access permet d'examiner les interactions entre service d'une part et classe d'ancienneté d'autre part, selon les classes choisies par l'équipe médicale (cf. exercice précédent).

Une première étape est d'associer à chaque ancienneté présente dans la table infirmieres la classe dont elle relève. Plutôt que de modifier la table infirmieres, une manière de faire consiste à constituer une table de toutes les valeurs de l'attribut anciennete de la table infirmieres, d'ajouter à cette table un nouvel attribut, classe_anciennete, et d'utiliser les requêtes Mise à jour d'Access pour donner à cet attribut la valeur idoine en fonction de la valeur correspondante de l'attribut anciennete.

Une deuxième étape consiste à préparer la table (ou la requête) qui servira de matière première à l'analyse croisée. Elle doit fournir l'identifiant d'une infirmière, son service, sa classe d'ancienneté.

La troisième étape est l'analyse croisée à proprement parler.

Exercice n°15 Donner les 10 lemmes les plus fréquents de la table occ_prema avec leur fréquence, par ordre de fréquence décroissante.

On se limitera dans un deuxième temps aux lemmes qui ont une catégorie « raisonnable », c'est-à-dire commençant par A(djectif), R (pour AdveRbe), S (pour PrépoSition), D(éterminant), V(erbe), C(onjonction) et P(ronom).

PHÈDRE

Regroupements par associations de personnages La requête :

R^2_{40}
RESTRICTION(partage_en > 1)[vers]
↪
PROJECTION(
...
numero_vers,
personnage_s,
partage_en
)[<résultat$_1$>]
↪
TRI SUR(personnage_s)[<résultat$_2$>]

permet de dégager les vers de *Phèdre* partagés entre personnages et de trier en fonction des personnages concernés à chaque fois. Dans le tableau 30, p. 142, l'alternance horizontale grisé/blanc dégage les groupes potentiels, la colonne grisée met en évidence l'attribut servant aux regroupements.

Le remplacement dans R^2_{40} de :
TRI SUR
par :
REGROUPER SUR
crée les groupes en fonction de l'attribut ou des attribut(s) suivant cette expression. La requête modifiée :

R^2_{41}
RESTRICTION(partage_en > 1)[vers]
↪
REGROUPER SUR(personnage_s)[<résultat$_1$>]
↪
PAR GROUPE(
...
numero_vers,
personnage_s,
partage_en,
...
)[<résultat$_2$>]

aboutit ainsi au tableau 31, p. 143.

La requête encore amendée :

TABLEAU 30 – PHÈDRE : vers partagés entre personnages

R^2_{40} p. 141

...	*numero_vers*	*personnage_s*	*partage_en*	...
...	487	ARICIE HIPPOLYTE	2	...
...	518	ARICIE HIPPOLYTE	2	...
...	524	ARICIE HIPPOLYTE	2	...
...	1339	ARICIE HIPPOLYTE	2	...
...	399	ARICIE ISMENE	2	...
...	462	ARICIE ISMENE	2	...
...	1425	ARICIE THESEE	2	...
...	670	HIPPOLYTE PHEDRE	2	...
...	725	HIPPOLYTE THERAMENE	2	...
...	463	ISMENE HIPPOLYTE	2	...
...	151	OENONE HIPPOLYTE	2	...
...	321	OENONE PANOPE	2	...
...	179	OENONE PHEDRE	2	...
...	259	OENONE PHEDRE	2	...
...	260	OENONE PHEDRE	2	...
...	264	OENONE PHEDRE	2	...
...	835	OENONE PHEDRE	2	...
...	839	OENONE PHEDRE	2	...
...	909	OENONE PHEDRE	2	...
...	1219	OENONE PHEDRE	2	...
...	1225	OENONE PHEDRE	2	...
...	1252	OENONE PHEDRE	2	...
...	205	OENONE PHEDRE OENONE	3	...
...	1645	PANOPE THESEE	2	...
...	157	PHEDRE OENONE	2	...
...	246	PHEDRE OENONE	2	...
...	711	PHEDRE OENONE	2	...
...	262	PHEDRE OENONE PHEDRE	3	...
...	763	PHEDRE OENONE PHEDRE	3	...
...	325	PHEDRE PANOPE	2	...
...	1619	PHEDRE THESEE	2	...
...	65	THERAMENE HIPPOLYTE	2	...
...	140	THERAMENE HIPPOLYTE	2	...
...	727	THERAMENE HIPPOLYTE	2	...
...	562	THERAMENE HIPPOLYTE THERAMENE	3	...
...	922	THESEE HIPPOLYTE	2	...
...	927	THESEE HIPPOLYTE	2	...
...	1469	THESEE PANOPE	2	...
...	914	THESEE PHEDRE	2	...
...	1188	THESEE PHEDRE THESEE	3	...
...	1491	THESEE THERAMENE	2	...
...	1493	THESEE THERAMENE	2	...

TABLEAU 31 – PHÈDRE : regroupements d'associations de personnages

R^2_{41} p. 141

...	*numero_vers*	*personnage_s*	*partage_en*	...
...	487	ARICIE HIPPOLYTE	2	...
...	399	ARICIE ISMENE	2	...
...	1425	ARICIE THESEE	2	...
...	670	HIPPOLYTE PHEDRE	2	...
...	725	HIPPOLYTE THERAMENE	2	...
...	463	ISMENE HIPPOLYTE	2	...
...	151	OENONE HIPPOLYTE	2	...
...	321	OENONE PANOPE	2	...
...	179	OENONE PHEDRE	2	...
...	205	OENONE PHEDRE OENONE	3	...
...	1645	PANOPE THESEE	2	...
...	157	PHEDRE OENONE	2	...
...	262	PHEDRE OENONE PHEDRE	3	...
...	325	PHEDRE PANOPE	2	...
...	1619	PHEDRE THESEE	2	...
...	65	THERAMENE HIPPOLYTE	2	...
...	562	THERAMENE HIPPOLYTE THERAMENE	3	...
...	922	THESEE HIPPOLYTE	2	...
...	1469	THESEE PANOPE	2	...
...	914	THESEE PHEDRE	2	...
...	1188	THESEE PHEDRE THESEE	3	...
...	1491	THESEE THERAMENE	2	...

R^2_{42}

RESTRICTION(partage_en > 1)[vers]
↪
REGROUPER SUR(personnage_s)[<résultat$_1$>]
↪
PAR GROUPE(
personnage_s,
COMPTE(personnage_s) TITRE 'nbre v.'
)[<résultat$_2$>]

indique le nombre de vers correspondant à chaque association de personnages (tableau 32, p. 144).

Dans la requête R^2_{43} :

R^2_{43}

RESTRICTION(partage_en > 1)[vers]
↪
REGROUPER SUR(personnage_s)[<résultat$_1$>]
AVEC (COMPTE(personnage_s) > 1)
↪
PROJECTION(
personnage_s,
COMPTE(personnage_s) TITRE 'nbre v.'
)[<résultat$_2$>]
↪
TRI SUR('nbre v.')[<résultat$_3$>]

on garde uniquement les associations de personnages dans un vers de *Phèdre* qui se produisent plus d'une fois, comme le montre le résultat du tableau 33, p. 144. Ces associations

TABLEAU 32 – PHÈDRE : nombre d'occurrences des associations de personnages

R^2_{42} p. 143

personnage_s	*nbre v.*
ARICIE HIPPOLYTE	4
ARICIE ISMENE	2
ARICIE THESEE	1
HIPPOLYTE PHEDRE	1
HIPPOLYTE THERAMENE	1
ISMENE HIPPOLYTE	1
OENONE HIPPOLYTE	1
OENONE PANOPE	1
OENONE PHEDRE	10
OENONE PHEDRE OENONE	1
PANOPE THESEE	1
PHEDRE OENONE	3
PHEDRE OENONE PHEDRE	2
PHEDRE PANOPE	1
PHEDRE THESEE	1
THERAMENE HIPPOLYTE	3
THERAMENE HIPPOLYTE THERAMENE	1
THESEE HIPPOLYTE	2
THESEE PANOPE	1
THESEE PHEDRE	1
THESEE PHEDRE THESEE	1
THESEE THERAMENE	2

sont en outre triées par nombre de vers décroissant. La paire Phèdre–Oenone occupe une place centrale (15 vers sur 28).

MySQL

La requête R^2_{40} p. 141 se concrétise en :

```
SELECT ..., numero_vers, personnage_s, partage_en, ...
FROM vers
WHERE partage_en > 1
ORDER BY personnage_s ;
```

TABLEAU 33 – PHÈDRE : associations répétées de personnages

R^2_{43} p. 143

personnage_s	*nbre v.*
OENONE PHEDRE	10
ARICIE HIPPOLYTE	4
THERAMENE HIPPOLYTE	3
PHEDRE OENONE	3
PHEDRE OENONE PHEDRE	2
ARICIE ISMENE	2
THESEE HIPPOLYTE	2
THESEE THERAMENE	2

La requête R^2_{42} p. 143 s'incarne en :

```
SELECT personnage_s,
       COUNT(personnage_s) AS 'nbre v.'
FROM vers
WHERE partage_en > 1
GROUP BY personnage_s ;
```

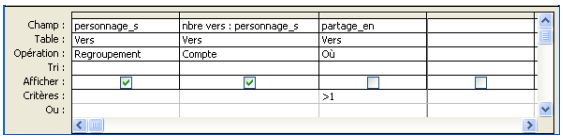
La requête R^2_{43} p. 143 donne naissance à :

```
SELECT personnage_s,
       COUNT(personnage_s) AS 'nbre v.'
FROM vers
WHERE partage_en > 1
GROUP BY personnage_s
       HAVING COUNT(personnage_s) > 1
ORDER BY 'nbre v.' DESC ;
```

Access

La requête 43 (R^2_{42} p. 143) opère un regroupement sur les valeurs distinctes de la colonne personnage_s. Entrent dans les groupes constitués uniquement les lignes pour lesquelles partage_en est plus grand que 1 : le choix de la valeur [Où] dans la colonne correspondante est ce qui permet d'édicter cette condition de restriction en amont du regroupement. Pour chaque groupe, on demande, dans la colonne du milieu, le décompte des lignes via le décompte des valeurs d'une colonne, en l'occurrence personnage_s.

43

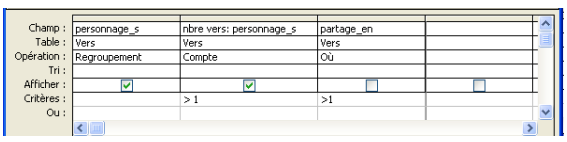


La requête SQL Server engendrée est la suivante :

```
SELECT Vers.personnage_s,
       Count(Vers.personnage_s) AS [nbre vers]
FROM Vers
WHERE (((Vers.partage_en)>1))
GROUP BY Vers.personnage_s;
```

La requête 44 (R^2_{43} p. 143) ajoute une condition de deuxième niveau, sur les groupes constitués. Ne sont préservés dans la table résultat que les groupes pour lesquels la valeur de la colonne nbre vers est supérieure à 1. C'est éliminer les hapax des associations constatées de personnages.

44



La requête SQL Server engendrée est la suivante :

```
SELECT Vers.personnage_s,
       Count(Vers.personnage_s) AS [nbre vers]
FROM Vers
WHERE (((Vers.partage_en)>1))
GROUP BY Vers.personnage_s
HAVING (((Count(Vers.personnage_s))>1));
```

Vers partagés par acte et par scène Si l'on souhaite replacer ces partages de vers dans l'ensemble de la pièce, par acte donc, ou par scène, on fera appel aux requêtes suivantes, qui décomptent d'une part le nombre de vers par acte ou par scène et d'autre part le nombre de vers partagé au sein de chaque acte ou de chaque scène :

R^2_{44}

```
REGROUPER SUR(acte)[vers]
↪
PAR GROUPE(
acte TITRE 'Acte',
NOMBRE DE LIGNES() TITRE 'nbre v.',
SOMME(SI(partage_en = 1, 1, 0)) TITRE 'v. partagés'
)[<résultat1>]
```

R^2_{45}

```
REGROUPER SUR(acte, scene)[vers]
AVEC SOMME(SI(partage_en = 1, 1, 0)) > 0
↪
PAR GROUPE(
acte TITRE 'Acte',
scene TITRE 'scène',
NOMBRE DE LIGNES() TITRE 'nbre v.',
SOMME(SI(partage_en = 1, 1, 0)) TITRE 'v. partagés'
)[<résultat1>]
```

Le résultat de ces requêtes (tableau 34, p. 147) distingue les deux premiers actes des trois autres, et bien sûr, met en évidence la place singulière de la scène 3 de l'acte I : l'aveu de Phèdre à Oenone.

MySQL

R^2_{44} p. 146

```
SELECT acte AS 'Acte',
     COUNT(*) AS 'nbre v.',
     SUM(IF(partage_en = 1, 0, 1)) AS 'v. partagés'
FROM vers
GROUP BY acte ;
```

R^2_{45} p. 146

```
SELECT acte AS 'Acte',
     scene AS 'scène',
     COUNT(*) AS 'nbre v.',
     SUM(IF(partage_en = 1, 0, 1)) AS 'v. partagés'
FROM vers GROUP BY acte, scene
HAVING SUM(IF(partage_en = 1, 0, 1)) > 0 ;
```

Access

La requête 45 (R^2_{44} p. 146) correspond à la requête SQL Server :

```
SELECT Vers.acte AS Acte,
     Count(Vers.numero_vers) AS [nbre v],
     Sum(IIf([partage_en]=1,0,1)) AS [vers partagés]
FROM Vers
GROUP BY Vers.acte;
```

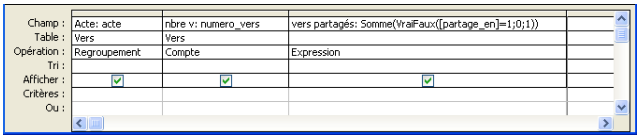
TABLEAU 34 – Phèdre : partages de vers par acte et par scène

R^2_{44} p. 146

Acte	*nbre v.*	*v. partagés*
I	366	13
II	370	11
III	264	7
IV	328	4
V	326	7

R^2_{45} p. 146

Acte	*scène*	*nbre v.*	*v. partagés*
I	I	142	2
I	II	10	1
I	III	164	8
I	IV	20	2
II	I	97	3
II	II	97	3
II	III	16	1
II	V	133	2
II	VI	23	2
III	I	76	1
III	III	88	3
III	IV	8	1
III	V	67	2
IV	IV	26	1
IV	VI	115	3
V	I	82	1
V	III	37	1
V	V	27	1
V	VI	106	2
V	VII	61	2



Exercice n°16 Sachant que *Phèdre* compte 1654 vers, en utilisant la table positions et les regroupements, fournir pour chaque syllabe le nombre de fois où elle est accentuée et le pourcentage correspondant.

Exercice n°17 Pour chacun des personnages, en utilisant les techniques de calcul de pourcentage présentées supra, déterminer le nombre total de vers dans lesquels il intervient, le nombre d'entre eux qui sont partagés avec un autre personnage, et le pourcentage que les vers partagés représentent dans ce rôle.

Exercice n°18 Pour Phèdre, Hippolyte, Thésée et Aricie, successivement, fournir un tableau avec les adjectifs, les verbes, les noms employés uniquement par ce personnage et ayant au moins deux occurrences dans *Phèdre*. On prendra soin de minusculiser systématiquement les mots, dans la mesure où les mots de la table occurrences gardent les éventuelles majuscules d'initiale de vers ou de début de phrase.

En MySQL, la minusculation s'obtient par appel à la fonction **LOWER**(<chaîne>). L'équivalent SQL Server est **LCASE**(<chaîne>) et pour Access : **MINUSCULE**(<chaîne>).

Exercice n°19 Même chose, sauf que les mots retenus doivent être en plus à la rime, en dernière position dans le vers.

ESQUE

Exercice n°20 Donner le nombre de dérivés en *-esque* pour lesquels soit on ne connaît pas la catégorie (marque **NULL** ou chaîne vide) soit il y a hésitation (marquée par un ?) sur la catégorie.

Exercice n°21 Fournir les auteurs dont plus de 5 attestations figurent dans la table esque, avec le nombre correspondant d'attestations, en triant par occurrences croissantes d'attestations.

3. Opérateurs sur les colonnes

On se reportera p. 507 pour les différences entre MySQL et SQL Server.

Opérateurs numériques

MySQL

Opérateurs numériques :

– somme (+),

TABLEAU 35 – PRÉMA : l'expérience dans le temps

R^2_{46} p. 149

Début expérience	*fin expérience*	*nombre de jours*
1999-03-17	2000-09-11	544

R^2_{47} p. 150

Début expérience	*fin expérience*	*nombre de jours*
17 03 99	11 09 2000	544

- différence (-),
- division (/),
- reste de la division entière (%),
- valeur absolue d'un nombre (**ABS**),
- arrondi vers le bas (**FLOOR**) ou vers le haut (**CEILING**),
- formatage du nombre de décimales souhaité (**FORMAT**).

Opérateurs sur une suite de données numériques (§ 2) :

- comptage (**COUNT**),
- somme (**SUM**),
- moyenne (**AVG**),
- écart-type (**STD**),
- minimum (**MIN**),
- maximum (**MAX**)...

Opérateurs sur les données temporelles

Supposons l'existence d'une fonction
JOURSPOURDATE(<date>)
qui prend en argument une date et retourne le nombre de jours entre cette date et un repère arbitraire (le jour 1 de l'an 1).

Une première requête (tableau 35, en haut, p. 149) montre comment trouver la plus ancienne date de naissance et la plus récente et comment obtenir le nombre de jours séparant les deux :

R^2_{46}

```
PAR GROUPE(
MINIMUM(jour_naissance) TITRE 'Début expérience',
MAXIMUM(jour_naissance) TITRE 'fin expérience',
JOURSPOURDATE(MAXIMUM(jour_naissance)) - JOURPOUR-
DATE(MINIMUM(jour_naissance)) TITRE 'nombre de jours'
)[bebes]
```

△

On remarque que les fonctions MINIMUM(<attribut>) et MAXIMUM(<attribut>) ont un comportement qui s'adapte au type d'attribut. Si l'attribut correspond à des valeurs numériques, MINIMUM retourne la plus petite, s'il s'agit de date, MINIMUM restitue la plus ancienne, s'il s'agit de chaînes de caractères, celle qui précède les autres dans l'ordre lexicographique (où 'a' précéde 'b' et 'a' précède 'ab', etc.).

MySQL

La fonction **TO_DAYS** est l'incarnation en MySQL de JOURSPOURDATE.

La seconde requête (tableau 35, en bas, p. 149) utilise un formatage des dates pour passer de la représentation américaine (AAAA-MM-JJ) sous laquelle est mémorisée la date à deux représentations distinctes, mais plus françaises.

R^2_{46} p. 149

```
SELECT
      MIN(jour_naissance) AS 'Début␣expérience.',
      MAX(jour_naissance) AS 'fin␣expérience',
      TO_DAYS(MAX(jour_naissance))
            - TO_DAYS(MIN(jour_naissance)) AS 'nombre␣de␣jours'
FROM bebes ;
```

R^2_{47}

```
SELECT
      DATE_FORMAT(MIN(jour_naissance), "%d %m %y")
            AS 'Début␣expérience',
      DATE_FORMAT(MAX(jour_naissance), "%d %m %Y")
            AS 'fin␣expérience',
...
```

Access

La requête 46 équivaut à la deuxième des requêtes MySQL précédentes (R^2_{47} p. 150). Noter le choix d'⌈Expression⌉ pour l'opération de la troisième colonne. On indique ainsi que la colonne correspondante de la table résultat est issue d'un calcul dont les composants figurent dans la première ligne.

46

Opérateurs sur chaînes

MySQL

Analyse de chaînes :

- longueur d'une chaîne (**LENGTH**) ;
- égalité d'une chaîne avec une chaîne cible (**STRCMP**) ;
- localisation d'une sous-chaîne au sein d'une chaîne plus vaste (**LOCATE**).

Extraction d'une sous-chaîne :

- à gauche (**LEFT**),
- à droite (**RIGHT**),
- à partir d'un point donné et pour une longueur fixée (**SUBSTRING**).

Transformation des chaînes fournies en argument :

- inversion (**REVERSE**) ;
- mise en majuscules (**UPPER**) ou en minuscules (**LOWER**) ;
- remplacement d'une sous-chaîne (**REPLACE**) ;
- concaténation (**CONCAT**).

MySQL

La fonction

LONGUEUR(<chaîne>)
retourne la longueur de la chaîne fournie en argument (0 pour la chaîne vide).

Pour obtenir le nombre de fiches remplies par chaque infirmière et leur longueur moyenne (tableau 20 p. 97), la requête R_{48}^2 opère un regroupement sur l'identifiant de l'infirmière :

R_{48}^2
REGROUPER SUR(id_infirmiere)[fiches_originelles]
↪
PAR GROUPE(
id_infirmiere TITRE 'Infirmière',
NOMBRE DE VALEURS DISTINCTES(id_bebe) TITRE 'nbre bébés',
SOMME(LONGUEUR(texte)) / NOMBRE DE LIGNES() TITRE 'nbre moy. car.'
)[<résultat$_1$>]
↪
TRI SUR('nbre moy. car.' DÉCROISSANT)[<résultat$_2$>]

Son équivalent MySQL est :

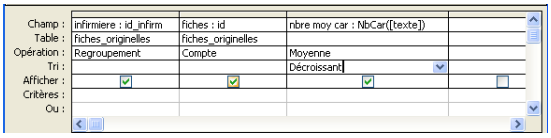
```
SELECT id_infirmiere AS 'Infirmière',
    COUNT(*) AS 'nbre fiches',
    COUNT(DISTINCT id_bebe) AS 'nbre bébés',
    SUM(LENGTH(texte)) / COUNT(*) AS 'nbre moy. car.'
FROM fiches_originelles
GROUP BY id_infirmiere
ORDER BY 'nbre moy. car.' DESC ;
```

Le modifieur **DISTINCT** permet le décompte des valeurs distinctes de l'identifiant id_bebe.

Access

La requête 47 regroupe les infirmières par identifiant, décompte le nombre de fiches correspondantes, et calcule la moyenne des longueurs des fiches. La longueur, en nombre de caractères, d'une chaîne découle de la fonction **NbCar**(<chaîne>). La feuille de données résultat affiche cette moyenne, malheureusement sans se limiter à un nombre de décimales pertinentes. Il faudrait pour cela faire appel à la fonction **FORMATNUMBER**.

47

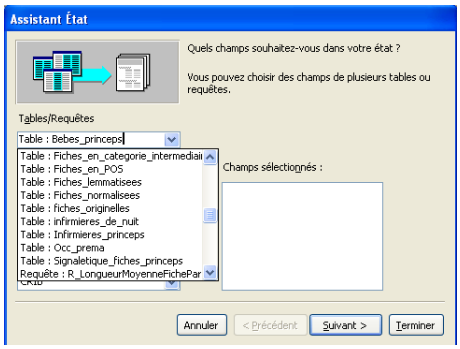


Access conduit souvent à découpler la production de résultats grossiers, comme cette moyenne à chiffres décimaux surabondants, et la présentation « lissée » de ces résultats. Le premier volet relève des opérations classiques des SGBD et en particulier des fonctions et opérateurs du dialecte de SQL employé, SLQ Server. Le second volet est délégué aux **états**. Un état est une présentation affinée d'un résultat brut. On utilise pour ce faire l'onglet de même nom dans la fenêtre de navigation dans la base de données.

Créons un état destiné à obtenir une présentation plus lisible et surtout simplement à limiter à deux chiffres après la virgule la moyenne du nombre de caractères par fiche. La

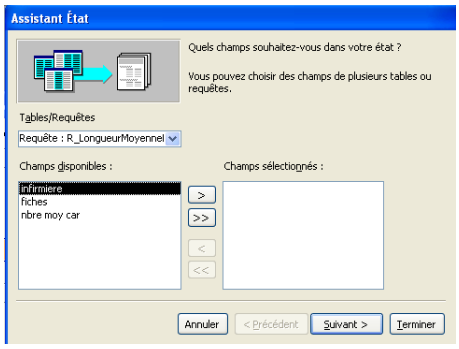
première étape 48 consiste à choisir la ou les table(s) et ou requête(s) qui vont contribuer à cette étape.

48



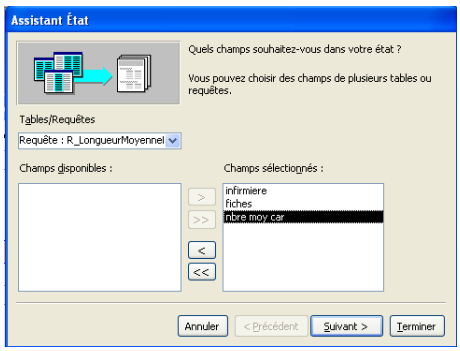
Une fois ce choix fait, dans le cas présent, la requête formulée *supra*, les attributs (champs dans la terminologie d'Access) correspondants apparaissent 49.

49



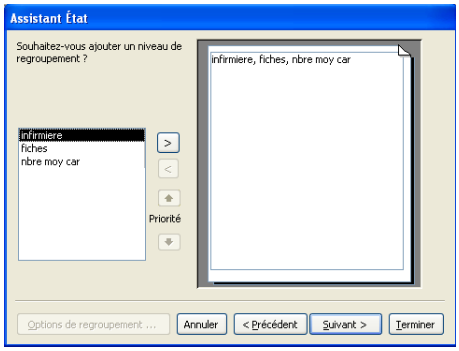
L'utilisateur détermine alors le ou les attribut(s) qu'il veut voir figurer dans l'état. Dans cet exemple, tous les attributs sont conservés 50.

50



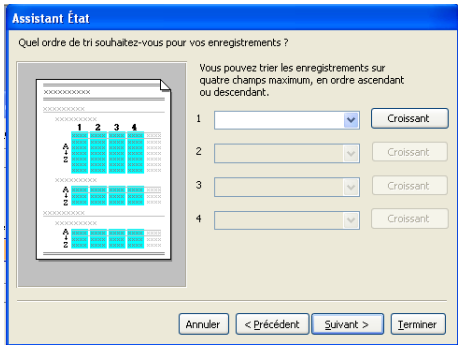
On peut opérer des regroupements au sein de l'état 51.

51



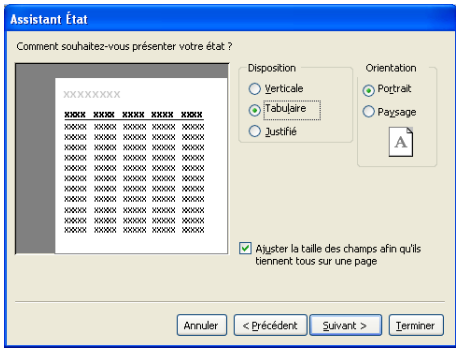
On peut trier sur au maximum quatre colonnes 52.

52

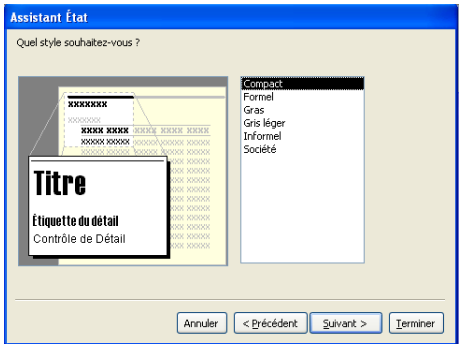


Il est possible de choisir un format de présentation 53, déterminer un style 54, donner un titre à cet état (de manière à pouvoir le réutiliser à volonté) et choisir ou non de visualiser un aperçu 55.

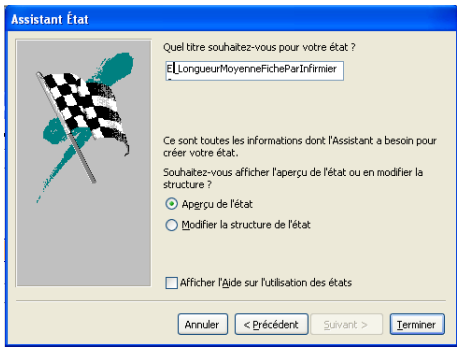
53



54



55



L'aperçu n'est d'ailleurs pas entièrement satisfaisant 56 : sur-titre dérivé du nom de sauvegarde de l'état, titres des colonnes non accentués ou cryptiques, et surtout, déluge maintenu de chiffres décimaux.

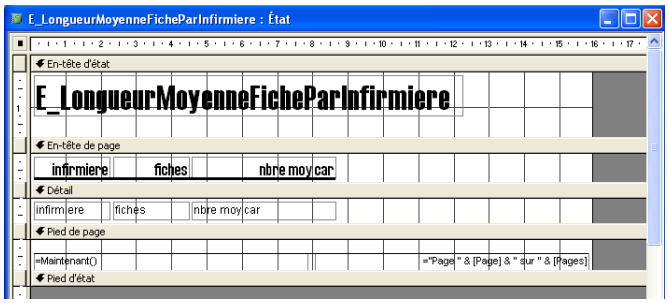
56

E_LongueurMoyenneFicheParInfirmiere

infirmiere	fiches	nbre moy car
43	12	322
24	21	276,571428571429
41	14	274,714285714286
36	23	270,565217391304
6	24	260,291666666667
47	25	245,2

Toujours dans l'onglet [États] de la fenêtre de navigation dans la base, on entreprend de modifier l'état qu'on vient de sauvegarder. On obtient une fenêtre similaire à 57.

57

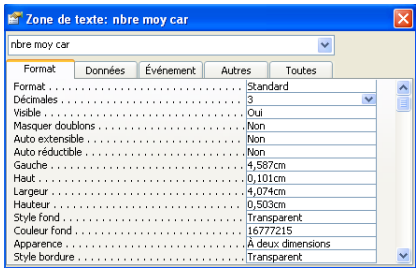


On peut modifier les sur-titre et titres en cliquant dans la zone correspondante et en effaçant, ajoutant, etc. On voit le résultat en 58 pour le sur-titre.

58

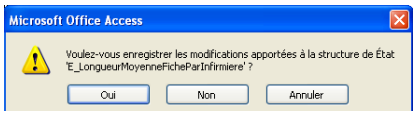
En se plaçant sur la zone d'affichage de la moyenne, on accède via le bouton droit 59 à un menu de configuration ⌈Propriétés⌋, qui permet en l'occurrence de choisir le format adéquat d'affichage de la moyenne (ici trois chiffres après la virgule).

59



On sauvegarde l'état modifié 60 .

60



Cela permet de disposer par la suite de cet état mis à jour :

Longueur moyenne des fiches par infirmière

Infirmière	fiches	nombre moyen car.
43	12	322,000
24	21	276,571
41	14	274,714
36	23	270,565
6	24	260,292
47	25	245,200
33	19	241,105
17	52	236,154
4	28	221,679
62	9	217,667

△

Sous MySQL, la clause **ORDER BY** est suivie d'un nom de colonne ou d'un alias de colonne. Lorsqu'on veut trier sur une colonne calculée, on ne peut faire suivre **ORDER BY** de la formule de calcul de cette colonne : une erreur en résulterait. Si l'on utilise dans l'exemple supra une clause ⌈**ORDER BY SUM**(**LENGTH**(texte)) / **COUNT**(∗) **DESC** ⌉ le message, peu aidant, est :

```
ERROR 1111 : Invalid use of group function
```

Access et SQL Server n'imposent pas cette contrainte sur ce qui suit **ORDER BY**.

MySQL

On peut se servir des opérateurs sur les chaînes pour extraire le contexte d'une sous-chaîne. Le tableau 36 p. 157 montre le contexte droit de *bébé* dans certaines fiches. Il provient de la requête R_{49}^2 :

```
SELECT texte,
    LOCATE('bébé', texte) AS 'pos.',
    SUBSTRING(texte, LOCATE('bébé', texte), 25)
        AS 'contexte␣droit'
FROM fiches_originelles
WHERE LOCATE('bébé', texte) >= 1
ORDER BY RAND()
LIMIT 5 ;
```

⌈**LOCATE**(<sous−chaîne>, <chaîne>)⌉ retourne le numéro du caractère où commence la sous-chaîne dans la chaîne ou 0 sinon. Le premier caractère de la chaîne est 1. Dans le premier exemple du tableau, *bébé* commence au 15^e caractère de texte. Pour les lignes qui contiennent effectivement *bébé* (clause **WHERE**), l'appel à **SUBSTRING** permet d'extraire un fragment de 25 car. de la colonne texte en commençant au caractère où débute *bébé.*

PRÉMA

PHÈDRE

MySQL

Les mots de *Phèdre* ne sont ni lemmatisés ni même normalisés. On peut trouver un même mot avec et sans une majuscule d'initiale de vers. Les deux réalisations seront comptées comme deux formes différentes. Si l'on ne prend pas en compte ce problème, un premier décompte des mots différents de *Phèdre*, par la requête :

TABLEAU 36 – PRÉMA : contextes droits de *bébé*

R^2_{49} p. 156

texte	*pos.*	*contexte droit*
C'es un petit **bébé** toujours très exitable, agité dès qu'on le touche, même pour des soins non douloureux. Il gigote dans tous les sens lève les bras les jambes se crispe. Il arrive à se calmer lorsque je lui mets une main sur la tete et une main sur le thorax, il se calme ne bouge plus.	15	bébé toujours très exitab
C'est un **bébé** qui montre bien ce qu'il aime (les massages...) et ce qu'il n'aime pas (la toilette...) Quand la maman est là on sént le bébé vraiment très détendu	10	bébé qui montre bien ce q
Jessy est un **bébé** calme en dehors des soins. Quand je m'occupe de lui, il est réactif, mais ses mouvements ne sont pas défensifs. Ne cherche pas à arracher ses sondes ou électrodes. Il s'aggripe aux doigts, mais assez mollement. Par contre, bébé éveillé, qui a très souvent les yeux ouverts, regarde, réagit bien à la voix. N'est pas très "accro" à la tétine, sauf quand elle est imbibée de canadou (à ce moment là, tète avec vigueur).	14	bébé calme en dehors des
C'est un **bébé** pas très tonique. Ouvre les yeux pendant les soins. Sa peau est abîmée avec des plaies à différents endroits du corps, ventre très ballonné, couleur grisâtre	10	bébé pas très tonique. Ou
Bébé epuisé par son infection. Sedatée, elle ne reagit pas aux soins bien que ce matin, elle ouvrait ses yeux pendant la toilette. Malgré tout, ce **bébé** semble absent et assez mal.	148	bébé semble absent et ass

R^2_{50}

RESTRICTION(cat Parmi ('Dét/Pron', 'Nc','V', 'Adj', 'Np', 'Conj', 'Prép', 'Rel', 'Adv'))[occurrences]

↪

PAR GROUPE(
NOMBRE DE VALEURS DISTINCTES(occ_car) TITRE 'Formes'
)[<résultat$_1$>]

soit :

```
SELECT COUNT(DISTINCT occ_car) AS 'formes.'
FROM occurrences
WHERE cat IN ('Dét/Pron', 'Nc','V', 'Adj', 'Np', 'Conj', 'Prép',
      'Rel', 'Adv');
```

indique un total de 3 067. Une deuxième requête, qui minusculise (**LOWER**) les mots qui ne sont pas des noms propres, qui n'ont donc pas 'Np' comme valeur de la colonne `cat`, fait baisser ce total à 2 810 :

```
SELECT COUNT(DISTINCT(IF(cat = 'Np', occ_car, LOWER(occ_car))))
  AS 'Formes'
FROM occurrences
WHERE cat IN ('Dét/Pron', 'Nc','V', 'Adj', 'Np', 'Conj', 'Prép',
      'Rel', 'Adv') ;
```

Exercice n°22 En regroupant par personnage d'abord, fournir les lemmes nominaux ayant 5 occurrences ou plus, en les triant par fréquence décroissante pour chaque personnage.

TABLEAU 37 – ESQUE : dérivé ≠ base + *-esque*

R^2_{51} p. 158

derive	*base*
sardanapalesque	Sardanapale
burtonnesque	Burton
hippopotamesque	hippopotame
livesque	live
gigolesque	gigolo
farcesque	farce
clonesque	clone
bignolesque	bignole
arizonesque	Arizona
poblanesque	poblano

derive	*base*
dahutesque	dahu
viandesque	viande
roploplesque	roploplo
smaltesque	Smalto
guerilla-esque	guerilla
abracadabresque	abracadabra
outre-tombesque	outre-tombe
clochardesque	clachard
epsilonnesque	epsilon
canulardesque	canular

ESQUE

MySQL

Quand on examine les dérivés en *-esque*, certains sont directement la concaténation de la base et du suffixe (*scoopesque* sur *scoop* ; *morpionesque* sur *morpion*) tandis que d'autres donnent lieu à un « aménagement » de la base (*zorresque* sur *Zorro* ; *colonesque* sur *colonie*). On souhaite faciliter le repérage automatique de ce deuxième cas de figure. Cela revient à chercher les dérivés qui, précisément, ne sont pas la simple concaténation de la base et du suffixe. On doit prendre trois précautions, au moins, dans l'énoncé de la requête : éliminer les « mots en plusieurs mots », comprenant un ou des espaces d'une part, mettre en minuscules la base et le dérivé (pour permettre de rapprocher *mariendbadesque* et *Mariendbad*, par exemple) d'autre part, éliminer enfin les bases inconnues (elles sont notées par un point d'interrogation). Il en résulte la requête R^2_{51} :

```
SELECT derive, base
FROM esque
WHERE base NOT LIKE '%␣%'
      AND base <> '?'
      AND CONCAT(LOWER(base), "esque") <> LOWER(derive)
ORDER BY RAND()  ;
```

Une partie des décalages constatés figure dans le tableau 37, p. 158. On met à jour ainsi :

- des erreurs de saisie : *clochardesque* sur *clachard* ;
- des chutes de la voyelle finale : *gigolesque* sur *gigolo* ;
- l'ajout de consonnes d'appui : *dahutesque* sur *dahu* et *canulardesque* sur *canular* ;
- le maintien du hiatus : *guerilla-esque* sur *guerilla* ;
- des doublements de la consonne finale : *epsilonnesque* sur *epsilon*.

4. Faire face à l'inconnu : NULL

La requête suivante permet de comparer l'évolution du poids des bébés au fil des jours selon que l'on prend en compte ou non les valeurs 0 utilisées par l'équipe médicale pour signaler l'information manquante sur ce point. Le résultat constitue le tableau 38 p. 159.

On suppose la fonction

TABLEAU 38 – Préma : poids par jour et non-réponses

Jour	*f.*	*nbre fiches poids = 0*	*moy.*	*moy. poids <> 0*	*écart-type*	*écart-type poids <> 0*	*diff. moy.*	*diff. écart-type*
1	327	0	961.21	961.21	276.70	276.70	-0.00	0.00
3	287	3	841.44	850.33	231.47	215.84	8.89	15.63
7	225	4	825.13	840.07	217.99	189.30	14.94	28.69
15	178	2	911.80	922.16	201.24	177.21	10.36	24.03

NULLSI(<valeur examinée>, <valeur à remplacer par **NULL**>)
qui examine une valeur et retourne **NULL** si la valeur examinée est égale à la valeur à remplacer par **NULL**. Dans le cas contraire, NULLSI renvoie la valeur examinée.

R_{52}^{2}

```
REGROUPER SUR(jour)[signaletique_fiches]
↪ PAR GROUPE(
jour TITRE 'Jour',
NOMBRE DE LIGNES() TITRE 'fiches',
SOMME(SI(poids = 0, 1, 0)) TITRE 'nbre fiches poids = 0',
MOYENNE(poids) TITRE 'moy.',
SOMME(poids) / SOMME(SI(poids = 0, 0, 1)) TITRE 'moy.
poids <> 0',
ECARTTYPE(poids) TITRE 'écart-type',
ECARTTYPE(NULLSI(poids, 0)) TITRE 'écart-type',
(SOMME(poids) / SOMME(SI(poids = 0, 0, 1))) -
MOYENNE(poids) TITRE 'diff. moy.',
ECARTTYPE(poids) - ECARTTYPE(NULLSI(poids, 0)) TITRE
'diff. écart-type'
)[<résultat1>]
```

MySQL

```
SELECT
    jour AS 'Jour',
    COUNT(*) AS 'fiches',
    SUM(IF(poids = 0, 1, 0)) AS 'nbre fiches poids = 0',
    FORMAT(AVG(poids),2) AS 'moy.',
    SUM(poids) / SUM(IF(poids = 0, 0, 1)) AS 'moy. poids <> 0',
    FORMAT(STD(poids), 2) AS 'écart-type',
    FORMAT(STD(NULLIF(poids, 0)),2) AS 'écart-type poids <> 0',
    SUM(poids) / SUM(IF(poids = 0, 0, 1)) -
        FORMAT(AVG(poids),2) AS 'diff. moy.',
    FORMAT(STD(poids), 2)
        - FORMAT(STD(NULLIF(poids, 0)),2)
        AS 'diff. écart-type'
FROM signaletique_fiches
GROUP BY jour ;
```

Pour chaque regroupement (un agrégat = les fiches pour un jour donné), l'expression :

```
SUM(IF(poids = 0, 1, 0)) AS 'nbre fiches poids = 0',
```

renvoie la valeur 1 si le poids de la fiche est égal à 0 et 0 sinon. La somme de tous les 1 d'un regroupement donne le nombre de fiches pour lesquelles le poids vaut 0. Le calcul inverse, plus bas :

```
SUM(poids)
    / SUM(IF(poids = 0, 0, 1)) AS 'moy. poids <> 0
```

permet d'obtenir le nombre de fiches dont le poids est différent de 0, nombre qui est alors utilisé pour calculer la moyenne des poids de l'agrégat sans les fiches dont le poids vaut 0. L'appel à **NULLIF**, par exemple dans :

```
FORMAT(STD(NULLIF(poids, 0)),2)
    AS 'écart-type poids <> 0'
```

renvoie **NULL**, quand son premier argument, poids, a la même valeur que son deuxième argument, ici 0. Il y a donc remplacement des 0 par la marque **NULL**. On se souvient que les marques **NULL** sont ignorées par des opérateurs comme **COUNT**, **STD**, etc.

PRÉMA

PHÈDRE

ESQUE

5. Solutions

Solution de l'exercice n°1 p. 127 On obtient 1017 fiches au total, 327 fiches pour le jour 1, 286 pour le jour 3, 225 pour le jour 7, 178 pour le jour 15.

Une première solution correspond à une restriction aux lignes pour lesquelles jour vaut 1 et demande le calcul explicite du nombre de lignes qui restent.

R^2_{53}

```
RESTRICTION(jour = 1)[fiches_originelles]
↪
PAR GROUPE(
NOMBRE DE LIGNES()
)[<résultat1>]
```

Une deuxième solution opère un regroupement sur l'attribut jour, ne garde que ce qui concerne le jour 1, et demande le décompte des lignes présentes.

R^2_{54}

```
REGROUPER SUR(jour)[fiches_originelles]
AVEC(jour =1)
↪
PAR GROUPE(
NOMBRE DE LIGNES()
)[<résultat1>]
```

MySQL

Pour J1, la requête R^2_{53} p. 160 est de la forme :

```
SELECT COUNT(*)
FROM fiches_originelles
WHERE jour = 1 ;
```

△

On notera que les requêtes :

```
SELECT COUNT(*)
FROM fiches_originelles
GROUP BY jour
     HAVING jour = 1 ;

SELECT COUNT(jour)
FROM fiches_originelles
GROUP BY jour
     HAVING jour = 1 ;
```

équivalentes, semble-t-il de la R^2_{54} p. 160, qui reposent sur un regroupement et édictent une contrainte de deuxième niveau, en restreignant à un des groupes issus de ce regroupement, sont rejetées avec le message d'erreur :

```
ERROR 1054 (42S22) : Unknown column 'jour' in 'having clause'
```

Tout se passe comme si **HAVING** ne pouvait faire référence qu'à une colonne effectivement employée dans la clause **SELECT**.

Par contre, ne pose pas de problème la requête proche :

```
SELECT jour,
    COUNT(*)
FROM fiches_originelles
GROUP BY jour
     HAVING jour = 1 ;
```

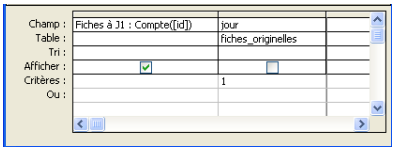
Une autre solution est :

```
SELECT SUM(IF(jour = 1, 1, 0))
FROM fiches_originelles ;
```

Access

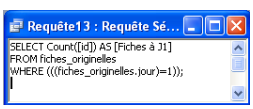
On trouve la réalisation de la R^2_{53} p. 160 en 62.

62



Son équivalent SQL Server figure en 63.

63



En 64 le résultat.

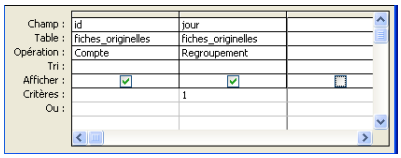
TABLEAU 39 – PRÉMA : nombre de types et d'occurrences de lemmes

Types	*o.*
1801	32058

64

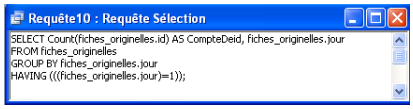
L'instanciation de la R_{54}^{2} p. 160 figure en 65.

65



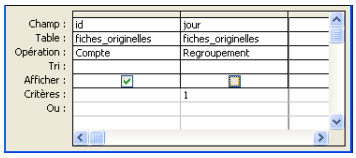
On trouve en 66 l'équivalent SQL Server.

66

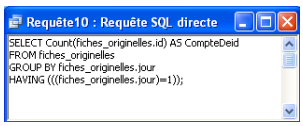


On notera qu'Access n'impose pas les mêmes restrictions que MySQL sur **HAVING**. On peut en effet ne pas faire figurer dans les résultats l'attribut utilisé comme condition de deuxième niveau : 67, comme le montre également l'équivalent SQL Server : 68. Le résultat figure en 69.

67



68



69

Solution de l'exercice n°2 p. 127 Le résultat figure au tableau 39 p. 162.

TABLEAU 40 – Préma : poids à la naissance par sexe

Filles

nbre	*poids min.*	*poids max.*	*poids moy.*	*écart-type*
57	530	1680	957.81	270.30

Garçons

nbre	*poids min.*	*poids max.*	*poids moy.*	*écart-type*
64	480	1660	954.53	285.51

MySQL

```
SELECT
      COUNT(DISTINCT lemme) AS 'Types',
      COUNT(*) AS 'o.'
FROM occ_prema
WHERE categorie REGEXP '[ANRSDVCP]';
```

qui incarne :

R^2_{55}

RESTRICTION(
categorie RESSEMBLANT À '[ANRSDVCP]'
)[occ_prema]
↪
PAR GROUPE(
NOMBRE DE VALEURS DISTINCTES(lemme) TITRE 'Types',
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_1$>

La condition de restriction pourrait être exprimée plus précisément de la manière suivante :

WHERE categorie **REGEXP** '^[ANRSDVCP]';

c'est-à-dire en spécifiant que la majuscule correspondant aux parties du discours attendues doit figurer en début de catégorie. Le résultat est bien sûr le même, puisqu'il n'y a de majuscule qu'à l'initiale dans les valeurs de l'attribut categorie.

Solution de l'exercice n°3 p. 127 Le résultat pour les filles figure au tableau 40 p. 163 en haut (en bas, modulo un changement de la condition de restriction, le résultat pour les garçons). Il provient de la requête :

R^2_{56}

RESTRICTION(sexe = 'Fille')[bebes]
↪
PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'nbre',
MINIMUM(poids_naissance) TITRE 'poids min.',
MAXIMUM(poids_naissance) TITRE 'poids max.',
MOYENNE(poids_naissance) TITRE 'poids moy.',
ECARTTYPE(poids_naissance) TITRE 'écart-type'
)[<résultat$_1$>]

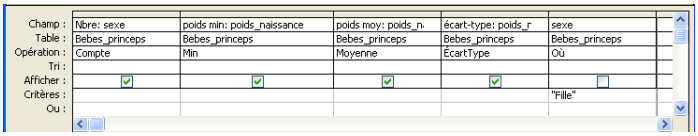
MySQL

```
SELECT COUNT(sexe) AS 'nbre',
       MIN(poids_naissance) AS 'poids␣min.',
       MAX(poids_naissance) AS 'poids␣max.',
       FORMAT(AVG(poids_naissance), 2) AS 'poids␣moy.',
       FORMAT(STD(poids_naissance), 2) AS 'écart-type'
FROM bebes
WHERE sexe = 'Fille' ;
```

Access

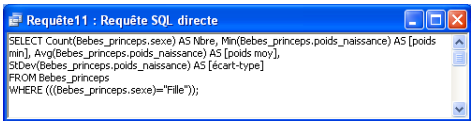
La requête 70 est l'équivalent de la requête MySQL supra. Après avoir rajouté une colonne ⌈Opération⌉ en cliquant sur le Σ de la barre de menu du haut, et en aval d'une restriction, qui figure dans la dernière colonne, sont opérés le décompte (1^{e}col.), la recherche du minimum (2^{e}col.), le calcul de la moyenne (3^{e} col.) et celui de l'écart-type (4^{e} col.).

70



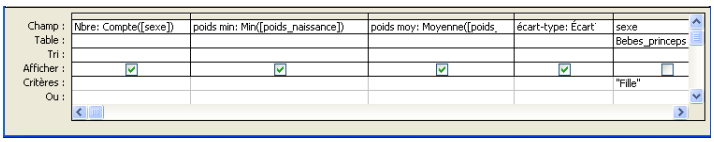
En 71 la requête SQL Server engendrée.

71

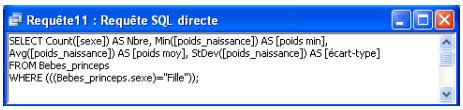


On pourrait également, au lieu de rajouter une colonne ⌈Opération⌉ en cliquant sur le Σ de la barre de menu du haut, directement faire appel à ces opérations dans l'en-tête de chaque colonne, comme en 72 (SQL Server engendré en 73). Le résultat sera le même : 74.

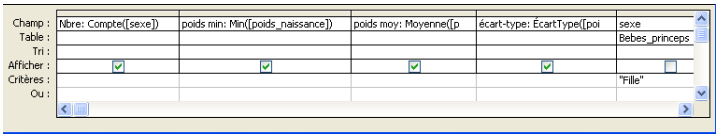
72



73



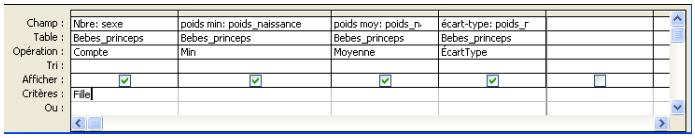
74



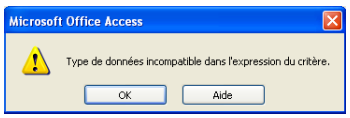
On remarquera que la requête SQL Server engendrée en 71 et en 73 est la même.

La requête 75 provoque par contre une erreur : 76.

75



76

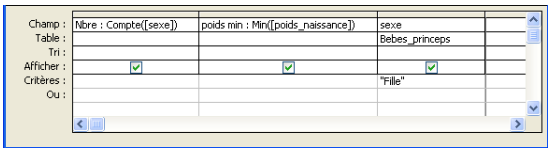


Cette erreur découle du mauvais usage de la ligne ⌈Critères⌉ en aval d'un regroupement. Dans la 1^{e} col., on a opéré un compte, donc on obtient une valeur numérique, qui n'est pas comparable avec la chaîne de caractères 'Fille', comme le montre la fin de la requête engendrée en SQL Server :

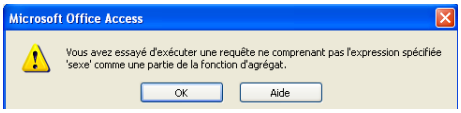
```
HAVING (((Count(Bebes_princeps.sexe))="Fille"));
```

La requête 77 déclenche une erreur : 78. Pourtant, à part des colonnes omises, elle ne diffère de la requête 72 que par la dernière colonne, cochée dans le premier cas, non coché dans le deuxième cas.

77

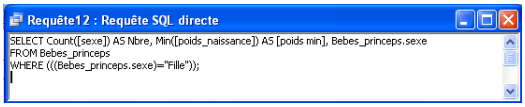


78



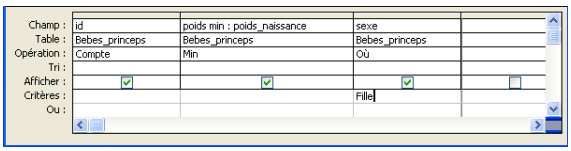
La requête SQL Server engendrée figure en 79 . Tout se passe comme si lorsqu'un attribut destiné à figurer dans la table résultat découle d'une opération sur les agrégats, tous les autres attributs de la table résultat doivent également relever d'une opération sur les agrégats.

79



De la même manière, la requête 80, parallèle à 70, à ceci près que doit figurer dans le résultat la colonne où figure l'opération Où, provoque une erreur : 81.

80



81

Solution de l'exercice n°4 p. 127 La requête abstraite est de la forme :

R^2_{57}

RESTRICTION(personnage_s = 'PHEDRE')[vers]
↪
PAR GROUPE(
NOMBRE DE LIGNES()
)[<résultat$_1$>]

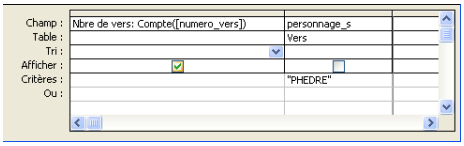
MySQL

```
SELECT COUNT(*)
FROM vers
WHERE personnage_s = 'PHEDRE' ;
```

Access

La solution 82 consiste à opérer une restriction sur la colonne `personnage_s` : la ligne pour être retenue doit avoir pour cet attribut la valeur 'PHEDRE'. On appelle alors explicitement la fonction **Compte** sur la colonne `numero_vers` et on donne à cet attribut calculé un titre plus parlant (à gauche des deux points).

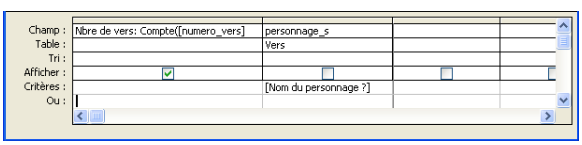
82



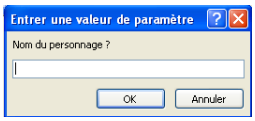
Une généralisation de cette solution, pour ne pas écrire autant de requêtes que de personnages de *Phèdre*, est le recours aux **requêtes paramétrées**. Une requête paramétrée est une requête qui, lorsqu'elle est déclenchée, demande à l'utilisateur de compléter la requête en fournissant la valeur d'un paramètre. Dans le cas présent, le paramètre est le nom du personnage dont on veut connaître le nombre de vers.

On voit en 83 comme formuler une requête paramétrée. Ce qui est mis entre crochets (ces crochets sont nécessaires) dans la colonne `personnage_s` est le message qui, lors de l'exécution de la requête, est affiché pour l'utilisateur dans la fenêtre de dialogue lui demandant de fournir une valeur : 84 .

83



84



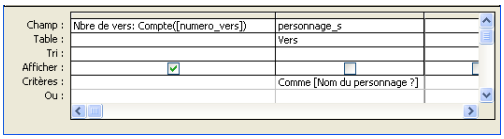
Après qu'une valeur a été fournie en 85 , le reste de la requête est exécuté et son résultat fourni : 86 . Réexécuter la requête permet de demander le nombre de vers pour un autre personnage.

85

86

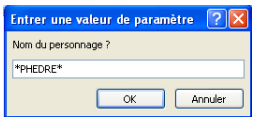
On peut utiliser un air de famille dans une requête paramétrée. La requête 87 emploie l'opérateur d'approximation [Comme] pour comparer la valeur demandée à ou fournie par l'utilisateur et le contenu de la colonne personnage_s.

87

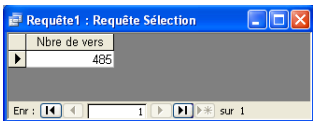


Il est alors possible de recourir aux opérateurs d'approximation, comme en 88 : est visée une restriction aux lignes pour lesquelles la chaîne 'PHEDRE' figure en n'importe quelle position. La requête précédente n'opérait que sur les lignes dans lesquelles cette chaîne occupait l'entièreté de la colonne personnage_s. On constate d'ailleurs en 89 que le nombre de vers décomptés est plus important qu'en 86 puisqu'il englobe également les vers partagés où intervient Phèdre.

88



89



Solution de l'exercice n°5 p. 127 La requête abstraite est de la forme :

TABLEAU 41 – PHÈDRE : place d'Oenone et de Thésée

COUNT(numero_vers)	*MIN(numero_vers)*	*MAX(numero_vers)*
202	143	1328

COUNT(numero_vers)	*MIN(numero_vers)*	*MAX(numero_vers)*
224	913	1654

R^2_{58}

RESTRICTION(personnage_s = OENONE)[vers]

↪

PAR GROUPE(
NOMBRE DE LIGNES(),
MINIMUM(numero_vers),
MAXIMUM(numero_ligne)
)[<résultat$_1$>]

Le tableau 41 p. 168 en haut donne le nombre de vers prononcés par Oenone seule, le numéro du premier vers qu'elle profère, ainsi que le numéro du dernier. Le tableau en bas en fait autant pour Thésée.

MySQL

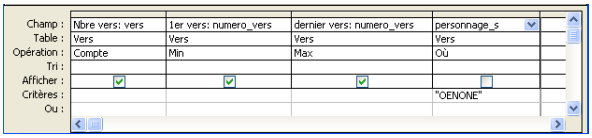
La requête concernant Oenone s'énonce ainsi :

```
SELECT COUNT(numero_vers),
       MIN(numero_vers),
       MAX(numero_vers)
FROM vers
WHERE personnage_s = 'OENONE';
```

Access

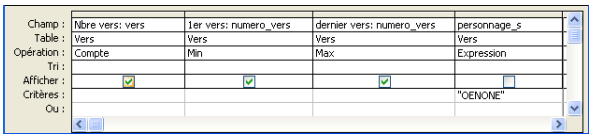
La requête :

90



diffère de la requête :

91



uniquement par le type d'opération employé : ⌈Où⌋ dans le premier cas, ⌈Expression⌋ dans le second. Le premier donne naissance à un **WHERE** :

```
SELECT Count(Vers.vers) AS [Nbre vers],
     Min(Vers.numero_vers) AS [1er vers],
     Max(Vers.numero_vers) AS [dernier vers]
FROM Vers
WHERE (((Vers.personnage_s)="OENONE"));
```

tandis que le second engendre un **HAVING** (d'ailleurs sans **GROUP BY** correspondant) :

```
SELECT Count(Vers.vers) AS [Nbre vers],
     Min(Vers.numero_vers) AS [1er vers],
     Max(Vers.numero_vers) AS [dernier vers]
FROM Vers
HAVING (((Vers.personnage_s)="OENONE"));
```

Solution de l'exercice n°6 p. 127 La requête abstraite est de la forme :

R^2_{59}

RESTRICTION(
personnage_s RESSEMBLANT À 'OENONE'
)[vers]
↪
PAR GROUPE(
NOMBRE DE LIGNES(),
MINIMUM(numero_vers),
MAXIMUM(numero_ligne)
)[<résultat$_1$>]

Le tableau 41 p. 168 en haut donne le nombre de vers prononcés par Oenone seule, le numéro du premier vers qu'elle profère, ainsi que le numéro du dernier. Le tableau en bas en fait autant pour Thésée.

MySQL

Par rappport à la solution de l'exercice précédent, il suffit de remplacer

```
... WHERE personnage_s = 'OENONE' ;
```

par :

```
... WHERE personnage_s REGEXP 'OENONE' ;
```

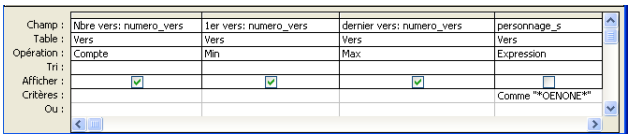
L'expression régulière permet de chercher le nom du personnage à n'importe quel endroit au sein de l'attribut `personnage_s`. On constate alors qu'Oenone intervient dans 220 vers (seule dans 202), tandis que Thésée intervient dans 234 vers (seul dans 224). Le premier vers et le dernier vers de ces deux personnages sont les mêmes qu'on utilise ou non une expression régulière : Oenone comme Thésée commencent et terminent leur présence sur scène par un vers plein.

On pourrait également formuler la restriction en faisant appel à des mécanismes plus frustes :

```
... WHERE personnage_s LIKE '%OENONE%' ;
```

Access

92



Solution de l'exercice n°7 p. 127 Une date complète (jour, mois, année) est telle que chacune des colonnes correspondantes a une valeur supérieure à 0, qui sert à noter l'absence de réponse. On notera qu'avec cette convention, la présence d'un zéro pour le jour et le mois, mais pas l'année, indique une attestation dont seule l'année est spécifiée. Il en va de même pour la présence d'un zéro seulement pour le jour : mois et année sont seuls indiqués.

R^2_{60}

RESTRICTION(
annee > 0
ET jour > 0
ET mois > 0
)[esque]
↪
PAR GROUPE(
NOMBRE DE LIGNES()
)[<résultat$_1$>]

MySQL

La requête :

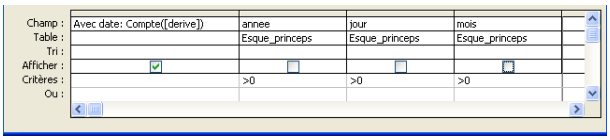
```
SELECT COUNT(*)
FROM esque
WHERE annee > 0 AND jour > 0 and mois > 0 ;
```

indique que 2 901 attestations de *-esque* sont assorties d'une date apparemment complète.

Access

La première solution en 93 crée une première colonne qui est un attribut calculé par appel à la fonction **Compte**. Les autres colonnes représentent la condition de restriction.

93



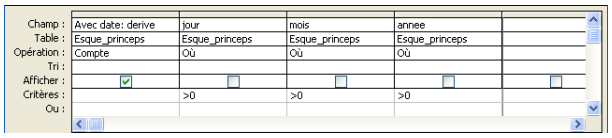
Dans la deuxième solution 94, l'appel à la fonction **Compte** est effectué sur la ligne [Opération]. Dans les deux cas, le passage au mode SQL Server en 95 montre clairement l'alliance de l'appel à une fonction de décompte et de la restriction. Les conditions de restriction sur les colonnes jour, mois et annee sont implicitement reliées par un ET logique dans l'interface, ce que la version SQL explicite.

TABLEAU 42 – PRÉMA : accouchement et lieu de naissance

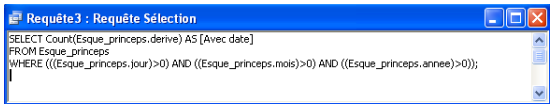
Accouchement	*nbre*	*%*
césarienne	54	44.63
voie basse	67	55.37

Naissance	*nbre*	*%*
locale	99	81.82
non locale	22	18.18

94



95



Solution de l'exercice n°8 p. 139 La répartition en effectifs et en pourcentage se trouve dans le tableau 42 p. 171. Les requêtes sont :

R^2_{61}

REGROUPER SUR(accouchement)[bebes]
↪
PAR GROUPE(
accouchement TITRE 'Accouchement',
Nombre de lignes() TITRE 'nbre',
Nombre de lignes() / 121 * 100 TITRE '%'
)[<résultat$_1$>]

R^2_{62}

REGROUPER SUR(lieu_naissance)[bebes]
↪
PAR GROUPE(
lieu_naissance TITRE 'Naissance',
Nombre de lignes() TITRE 'nbre',
Nombre de lignes() / 121 * 100 TITRE '%'
)[<résultat$_1$>]

MySQL

Les requêtes sont les suivantes :

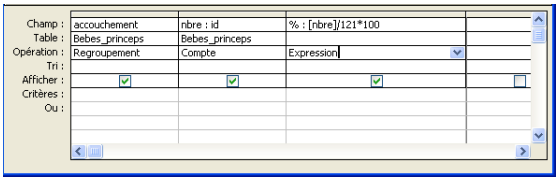
```
SELECT
      accouchement AS 'Accouchement',
      COUNT(*) AS 'nbre',
      COUNT(*) / 121 * 100 AS '%'
FROM bebes
GROUP BY accouchement ;
```

```
SELECT
      lieu_naissance AS 'Naissance',
      COUNT(*) AS 'nbre',
      COUNT(*) / 121 * 100 AS '%'
FROM bebes
GROUP BY lieu_naissance ;
```

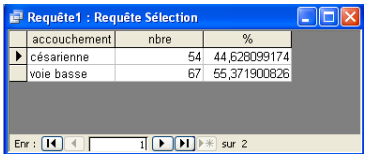
Access

La formule de calcul du pourcentage pour chaque type d'accouchement en 96 fait appel à '[nbre]', le nom donné à l'attribut calculé par décompte du nombre de valeurs dans la colonne id.

96



97



La requête SQL engendrée est :

```
SELECT Bebes_princeps.accouchement,
      Count(Bebes_princeps.id) AS nbre,
      [nbre]/121*100 AS [%]
FROM Bebes_princeps
GROUP BY Bebes_princeps.accouchement;
```

△

La requête MySQL équivalente :

```
SELECT accouchement,
      COUNT(*) AS 'nbre',
      'nbre' / 121 * 100 AS '%'
FROM bebes
GROUP BY accouchement ;
```

n'a pas le même résultat :

accouchement	*nbre*	%
césarienne	54	0
voie basse	67	0

Tout se passe comme si MySQL, à la différence d'Access et de SQL Server, ne considérait pas le nom donné dans la table-résultat comme un nom d'attribut comme les autres et ne pouvait donc pas l'utiliser dans des calculs.

TABLEAU 43 – PRÉMA : répartition des fiches par jour

Jour	*fiches*	%
1	327	32.15
3	287	28.22
7	225	22.12
15	178	17.50

Solution de l'exercice n°9 p. 139 Les résultats de la requête R^2_{63} figurent dans le tableau 43 p. 173.

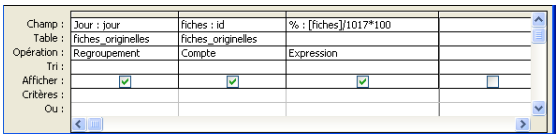
R^2_{63}

```
REGROUPER SUR(jour)[fiches_originelles]
↪
PAR GROUPE(
jour TITRE 'Jour',
NOMBRE DE LIGNES() TITRE 'fiches',
NOMBRE DE LIGNES() / 1017 * 100 TITRE '%'
)[<résultat1>]
```

MySQL

```
SELECT jour AS 'Jour',
    COUNT(*) AS 'fiches',
    COUNT(*) / 1017 * 100 AS '%'
FROM fiches_originelles
GROUP BY jour ;
```

Access



98

Solution de l'exercice n°10 p. 139 Le tableau 44 p. 174 indique pour chaque jour le nombre de bébés concernés par l'étude et le pourcentage correspondant. Il résulte de la requête R^2_{64}. À chaque étape, certains bébés vont suffisamment mieux pour quitter le service de réanimation néonatale voire l'hôpital. D'autres décèdent. Au jour 15, il ne reste plus que 60.33% de l'effectif de départ.

TABLEAU 44 – PRÉMA : nombre de bébés par jour

bébés J1	*J3*	*%*	*J7*	*%*	*J15*	*%*
121	105	86.78	85	70.25	73	60.33

R^2_{64}

```
PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'bébés J1',
SOMME(SI(etat_J3 = 'réa', 1, 0)) TITRE 'J3',
SOMME(SI(etat_J3 = 'réa', 1, 0)) NOMBRE DE LIGNES() * 100
TITRE '%',
SOMME(SI(etat_J7 = 'réa', 1, 0)) TITRE 'J7',
SOMME(SI(etat_J7 = 'réa', 1, 0)) NOMBRE DE LIGNES() * 100
TITRE '%',
SOMME(SI(etat_J15 = 'réa', 1, 0)) TITRE 'J15',
SOMME(SI(etat_J15 = 'réa', 1, 0)) NOMBRE DE LIGNES() *
100 TITRE '%'
)[bebes]
```

MySQL

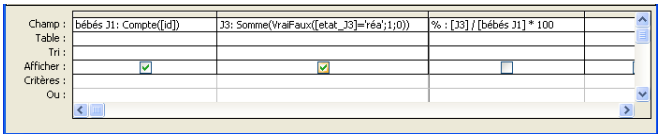
La requête correspondante est :

```
SELECT
    COUNT(*) AS 'bébés J1',
    SUM(IF(etat_J3 = 'réa', 1, 0)) AS 'J3',
    SUM(IF(etat_J3 = 'réa', 1, 0)) / COUNT(*) * 100 AS '%',
    SUM(IF(etat_J7 = 'réa', 1, 0)) AS 'J7',
    SUM(IF(etatJ7 = 'réa', 1, 0)) / COUNT(*) * 100 AS '%',
    SUM(IF(etat_J15 = 'réa', 1, 0)) AS 'J15',
    SUM(IF(etat_J15 = 'réa', 1, 0)) / COUNT(*) * 100 AS '%'
FROM bebes ;
```

Access

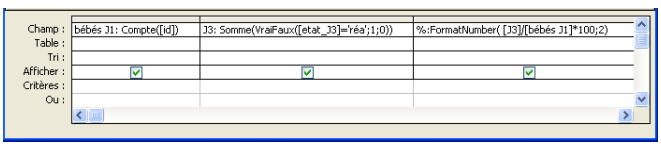
99 fournit l'esprit de la solution.

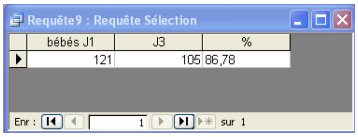
99



100 y ajoute le formatage du pourcentage, comme en témoigne le résultat 101.

100





Solution de l'exercice n°11 p. 139 Les tableaux 45 p. 176 et 46 p. 176 combinent pour la sédation et la ventilation des bébés à chaque fois les regroupements pour l'ensemble des fiches, en haut, et, en bas, la répartition des modalités observées (effectif et pourcentage), en aval d'un regroupement par jour. L'absence de sédation varie peu en proportion, sauf au jour 7, où elle augmente. La ventilation diminue du fil du temps, surtout à partir du jour 3. Ce recul touche particulièrement l'intubation. Sont fournies les requêtes R^2_{65} et R^2_{66} pour la sédation. Les requêtes pour la ventilation sont similaires.

R^2_{65}

```
REGROUPER SUR(sedation)[fiches_originelles]
↪
PAR GROUPE(
sedation TITRE 'Sédation',
NOMBRE DE LIGNES() TITRE 'fiches',
NOMBRE DE LIGNES() / 1017 * 100 TITRE '%'
)[<résultat1>]
↪
TRI SUR('%')[<résultat2>]
```

R^2_{66}

```
REGROUPER SUR(jour)[fiches_originelles]
↪
PAR JOUR(
NOMBRE DE LIGNES() TITRE 'fiches',
SOMME(SI(sedation = 'non', 1, 0)) TITRE 'Non',
SOMME(SI(sedation = 'non', 1, 0)) / NOMBRE DE LIGNES() *
100 TITRE '%',
SOMME(SI(sedation = 'hypnovel ou fentanyl', 1, 0)) TITRE
'Hypno. Fent.',
SOMME(SI(sedation = 'hypnovel ou fentanyl', 1, 0)) / NOM-
BRE DE LIGNES() * 100 TITRE '%',
SOMME(SI(sedation = 'canadou', 1, 0)) TITRE 'canadou',
SOMME(SI(sedation = 'canadou', 1, 0)) / NOMBRE DE
LIGNES() * 100 TITRE '%'
)[<résultat1>]
```

MySQL

```
SELECT sedation AS 'Sédation',
     COUNT(*) AS 'fiches',
     COUNT(*) / 1017 * 100 AS '%'
FROM fiches_originelles
GROUP by sedation
ORDER BY '%' ;

SELECT jour AS 'Jour',
     COUNT(*) AS 'fiches',
     SUM(IF(sedation = 'non', 1, 0)) AS 'Non',
```

TABLEAU 45 – PRÉMA : sédation globalement et par jour

Sédation	*fiches*	%
non reponse	3	0.29
canadou	11	1.08
hypnovel ou fentanyl	77	7.57
non	926	91.05

Jour	*fiches*	*Non*	%	*Hypno. Fent.*	%	*canadou*	%
1	327	298	91.13	27	8.26	1	0.31
3	287	258	89.90	23	8.01	5	1.74
7	225	209	92.89	13	5.78	3	1.33
15	178	161	90.45	14	7.87	2	1.12

TABLEAU 46 – PRÉMA : ventilation globalement et par jour

Ventilation	*fiches*	%
non reponse	1	0.10
sonde nasale	272	26.75
non	305	29.99
intubé	439	43.17

Jour	*fiches*	*non*	%	*sonde nasale*	%	*intubé*	%
1	327	75	22.94	62	18.96	190	58.10
3	287	94	32.75	92	32.06	101	35.19
7	225	71	31.56	73	32.44	81	36.00
15	178	65	36.52	45	25.28	67	37.64

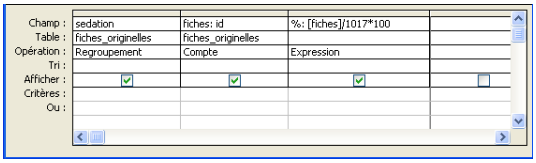
```
      SUM(IF(sedation = 'non', 1, 0))
            / COUNT(*) * 100 AS '%',
      SUM(IF(sedation = 'hypnovel ou fentanyl', 1, 0)) AS 'Hypno. Fent.',
      SUM(IF(sedation = 'hypnovel ou fentanyl', 1, 0))
            / COUNT(*) * 100 AS '%',
      SUM(IF(sedation = 'canadou', 1, 0))AS 'canadou',
      SUM(IF(sedation = 'canadou', 1, 0))
            / COUNT(*) * 100 AS '%'
FROM fiches_originelles
GROUP BY jour ;

SELECT ventilation AS 'Ventilation',
      COUNT(*) AS 'fiches',
      COUNT(*) / 1017 * 100 AS '%'
FROM fiches_originelles
GROUP by ventilation
ORDER BY '%' ;

SELECT
      SUM(IF(ventilation = 'sonde nasale' AND jour = 1, 1, 0)) AS 'J1',
      SUM(IF(ventilation = 'sonde nasale' AND jour = 1, 1, 0))
            / SUM(IF(jour = 1, 1, 0)) *100 AS '%',
      SUM(IF(ventilation = 'sonde nasale' AND jour = 3, 1, 0)) AS 'J3',
      SUM(IF(ventilation = 'sonde nasale' AND jour = 3, 1, 0))
            / SUM(IF(jour = 3, 1, 0)) * 100 AS '%',
      SUM(IF(ventilation = 'sonde nasale' AND jour = 7, 1, 0)) AS 'J7',
```

```
      SUM(IF(ventilation = 'sonde nasale' AND jour = 7, 1, 0))
            / SUM(IF(jour = 7, 1, 0)) * 100 AS '%',
      SUM(IF(ventilation = 'sonde nasale' AND jour = 15, 1, 0)) AS 'J15',
      SUM(IF(ventilation = 'sonde nasale' AND jour = 15, 1, 0))
            / SUM(IF(jour = 15, 1, 0)) * 100 AS '%'
FROM fiches_originelles ;
```

Access

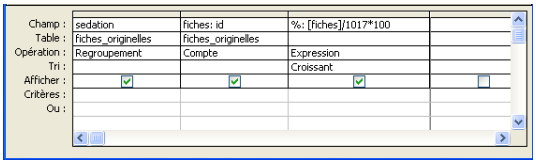
L'équivalent de la R_{65}^2 p. 175 est :

102



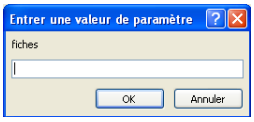
au tri près par pourcentage croissant. Curieusement, quand on ajoute ce tri :

103



le nom fiches qui faisait référence au nombre de fiches du groupe courant est considéré comme un paramètre :

104



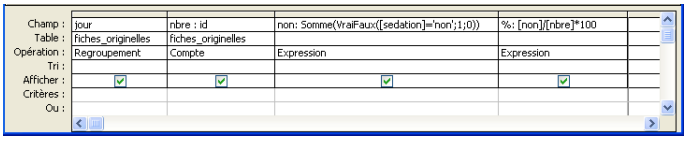
L'examen de la requête SQL Server engendrée :

```
SELECT fiches_originelles.sedation,
      Count(fiches_originelles.id) AS fiches,
      [fiches]/1017*100 AS [%]
FROM fiches_originelles
GROUP BY fiches_originelles.sedation
ORDER BY [fiches]/1017*100;
```

montre que c'est la présence de fiches entre crochets dans la partie **ORDER BY** qui est interprétée comme un paramètre pour la requête.

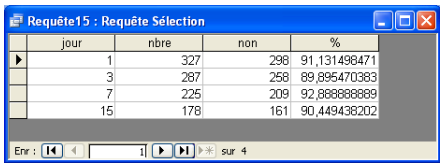
Le principe de l'équivalent de la requête R_{66}^2 p. 175 est :

105



Son résultat figure en 106.

106

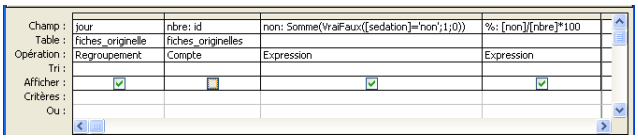


La requête SQL Server engendrée est la suivante :

```
SELECT fiches_originelles.jour,
       Count(fiches_originelles.id) AS nbre,
       Sum(IIf([sedation]='non',1,0)) AS non,
       [non]/[nbre]*100 AS [%]
FROM fiches_originelles
GROUP BY fiches_originelles.jour;
```

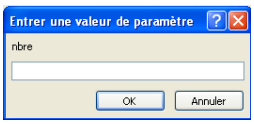
La requête 107 ne diffère de la requête 105 que par le fait que la colonne non, qui totalise le nombre de lignes du groupe courant, n'est pas censée figurer dans le résultat (la case Afficher n'est pas cochée).

107



Le comportement de la requête n'est plus le même. Elle devient une requête paramétrée et l'utilisateur se voit demander la valeur à donner au paramètre non :

108

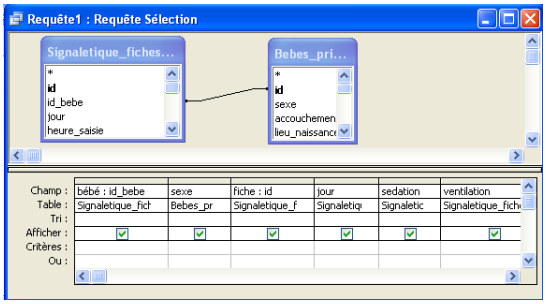


Quand on examine la requête SQL Server engendrée, on constate qu'effectivement, le calcul du nombre de lignes du groupe courant a disparu :

```
SELECT fiches_originelles.jour,
       Sum(IIf([sedation]='non',1,0)) AS non,
       [non]/[nbre]*100 AS [%]
FROM fiches_originelles
GROUP BY fiches_originelles.jour;
```

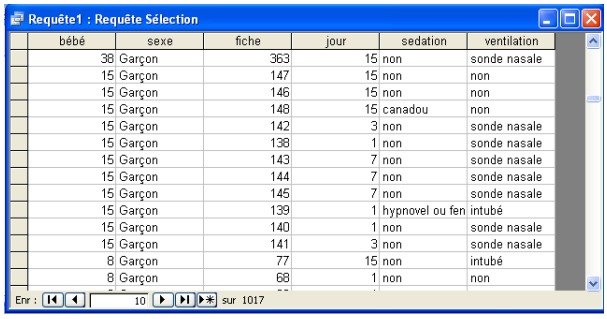
Solution de l'exercice n°12 p. 140 Une première étape utilise la jointure entre les tables signaletique_fiches et bebes (sur l'identifiant du bébé) pour conserver les attributs correspondant à l'identifiant du bébé, à son sexe, à l'identifiant de la fiche, à la sédation et à la ventilation.

109



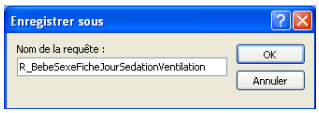
Le résultat figure en 110.

110



111 correspond à la sauvegarde de la requête sous le nom R_BebeSexeFicheJourSedationVentilation.

111



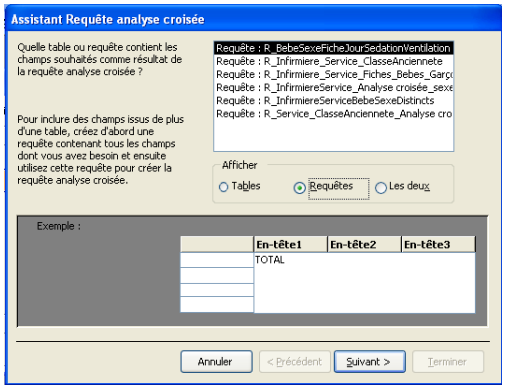
Les étapes des quatre analyses croisées (dont seuls les résultats sont fournis pour les trois dernières) sont les suivantes :

1. choix de la table/requête d'entrée (ici R_BebeSexeFicheJourSedationVentilation) ;
2. choix de l'attribut ou des attributs (au maximum 3) servant d'en-têtes de lignes, c'est-à-dire des regroupements par rapport auxquels le croisement avec les valeurs/modalités d'un autre attribut sera opéré ;
3. choix de l'attribut dont les valeurs/modalités vont servir d'en-tête de colonne, c'est-à-dire de l'attribut « croisé » avec les regroupements opérés à l'étape précédente ;
4. type de calcul opéré à l'intersection (décompte, minimum, maximum, etc) et attribut (différent des attributs sélectionnés aux étapes précédentes) utilisé pour ce calcul ;
5. sauvegarde de la requête Analyse croisée en lui fournissant un nom ;
6. exécution et résultat.

Ces étapes vont permettre dans un premier temps de faire le croisement entre sexe et ventilation.

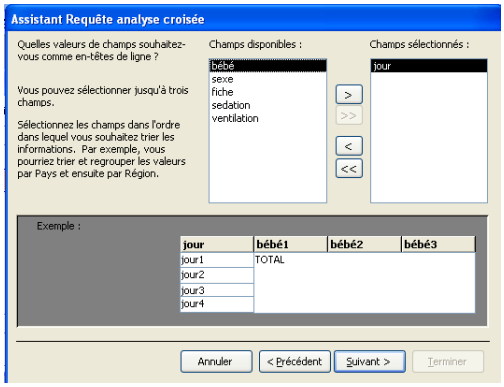
On choisit la requête d'entrée constituée supra : R_BebeSexeFicheJourSedationVentilation.

112



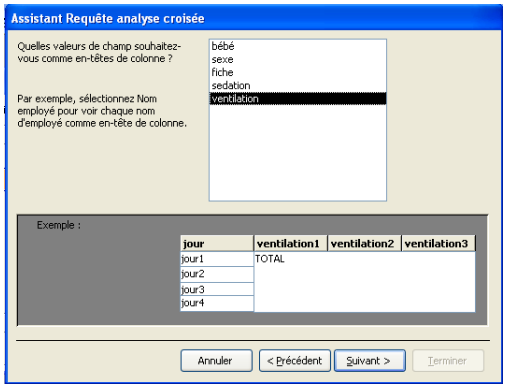
L'attribut jour est choisi comme en-tête de ligne. C'est par jour que se fera le regroupement des lignes de la requête de départ

113



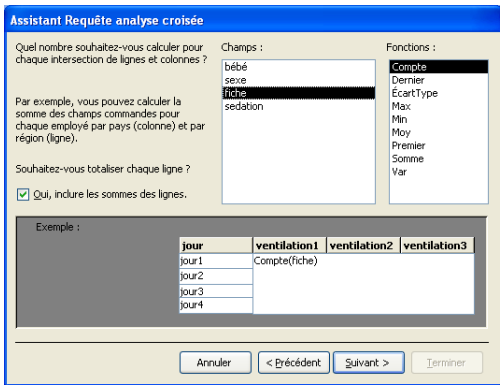
L'attribut ventilation est retenu comme en-tête de colonne. Ses modalités vont donner lieu à autant de colonnes dans l'analyse croisée.

114



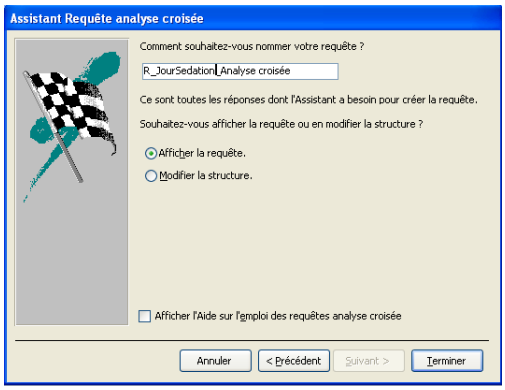
Le résultat du calcul qui va figurer à l'intersection entre un jour donné et une modalité de l'attribut ventilation est le décompte du nombre de fiches correspondant, c'est-à-dire l'application de l'opérateur COMPTE à l'attribut fiche, qui a pour valeur l'identifiant de la fiche.

115



La requête est sauvegardée.

116



Son résultat en 117 est identique à celui du tableau 45 p. 176, pourcentages en moins.

117

R_JourSedation_Analyse croisée : Requête Analyse croisée

jour	Total de fiche	intubé	non	non reponse	sonde nasale
1	327	190	75		62
3	287	101	94		92
7	225	81	71		73
15	178	67	65	1	45

Enr : 1 sur 4

En 118, le croisement sexe et sédation.

118

R_SexeSedation_Analyse croisée : Requête Analyse croisée

sexe	Total de fiche	canadou	hypnovel ou fen	non	non reponse
Fille	480	6	31	441	2
Garçon	537	5	46	485	1

Enr : 2 sur 2

En 119, le croisement jour-sexe et sédation.

119

R_JourSexeSedation_Analyse croisée : Requête Analyse croisée

jour	sexe	Total de fiche	canadou	hypnovel ou fen	non	non reponse
1	Fille	151	1	11	139	
1	Garçon	176		16	159	1
3	Fille	142	3	11	127	1
3	Garçon	145	2	12	131	
7	Fille	104	2	1	101	
7	Garçon	121	1	12	108	
15	Fille	83		8	74	1
15	Garçon	95	2	6	87	

Enr : 4 sur 8

En 120, le croisement jour-sexe et ventilation.

120

R_JourSexeVentilation_Analyse croisée : Requête Analyse croisée

jour	sexe	Total de fiche	intubé	non	non reponse	sonde nasale
1	Fille	151	83	44		24
1	Garçon	176	107	31		38
3	Fille	142	42	57		43
3	Garçon	145	59	37		49
7	Fille	104	25	48		31
7	Garçon	121	56	23		42
15	Fille	83	30	38		15
15	Garçon	95	37	27	1	30

Enr : 1 sur 8

Solution de l'exercice n° 13 p. 140

MySQL

La première requête pour regrouper les infirmières selon les classes d'ancienneté distinguées par l'équipe médicale est la suivante :

```
SELECT
      CASE
            WHEN anciennete < 1 THEN '< 1'
            WHEN anciennete > 5 THEN '> 5'
            ELSE '1-5'
      END
      AS 'Classe ancienneté',
      COUNT(*) AS 'o.'
FROM infirmieres_princeps
GROUP BY
      CASE
            WHEN anciennete < 1 THEN '< 1'
            WHEN anciennete > 5 THEN '> 5'
            ELSE '1-5'
       END ;
```

Le résultat est le suivant :

Classe ancienneté	*o.*
1-5	14
< 1	13
> 5	15

Le croisement entre service et classes d'ancienneté s'obtient par :

```
SELECT
      service AS 'Service',
      CASE
            WHEN anciennete < 1 THEN '< 1'
            WHEN anciennete > 5 THEN '> 5'
            ELSE '1-5'
      END
      AS 'Classe ancienneté',
      COUNT(*) AS 'o.'
FROM infirmieres_princeps
GROUP BY
      service,
      CASE
            WHEN anciennete < 1 THEN '< 1'
            WHEN anciennete > 5 THEN '> 5'
            ELSE '1-5'
      END ;
```

On obtient :

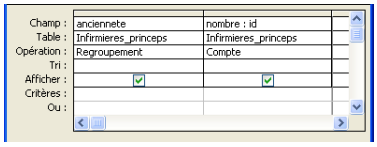
Service	*Classe ancienneté*	*o.*
NULL	< 1	1
Jour	1-5	11
Jour	< 1	11
Jour	> 5	7
Nuit	1-5	3
Nuit	< 1	1
Nuit	> 5	8

Si les 3 classes sont à peu près équilibrées lorqu'on considère toutes les infirmières, on constate que cet équilibre résulte de la conjonction de deux déséquilibres : les classes moins de 1 an et entre 1 et 5 ans sont favorisées par les infirmières de jour, tandis que les infirmières de plus de 5 ans dominent nettement parmi les infirmières de nuit.

Solution de l'exercice n°14 p. 140 Une première étape consiste à créer une table associant à une valeur donnée de l'attribut anciennete la classe d'ancienneté correspondante.

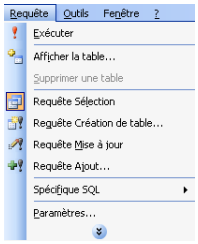
On commence par une requête opérant un regroupement de la table infirmieres sur l'attribut anciennete, conservant cet attribut et le nombre d'occurrences de chaque groupe.

121



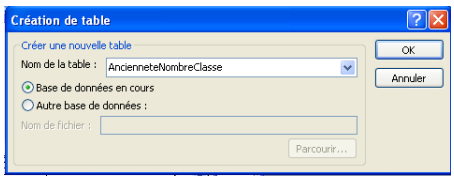
On fait appel à l'onglet [Requête] de la barre de menu du haut pour choisir une requête Création de table.

122



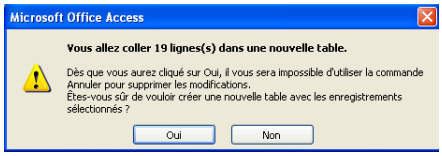
On se voit demander le nom de la table à créer.

123



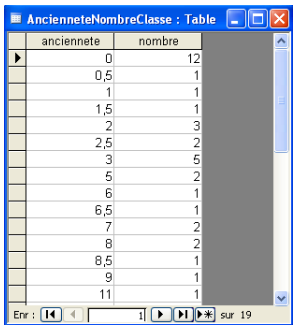
Un avertissement est émis, sur le nombre de lignes qui vont être ajoutées à cette nouvelle table.

124



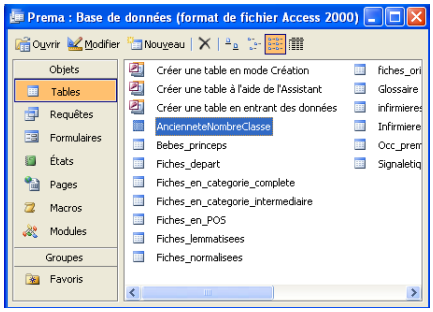
En 125, un extrait de la table résultante.

125



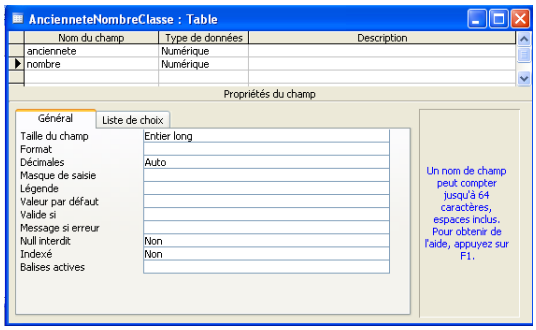
À partir de l'onglet [Tables] de la fenêtre de navigation dans la base de données, on peut modifier la structure de la table AncienneteNombreClasse.

126



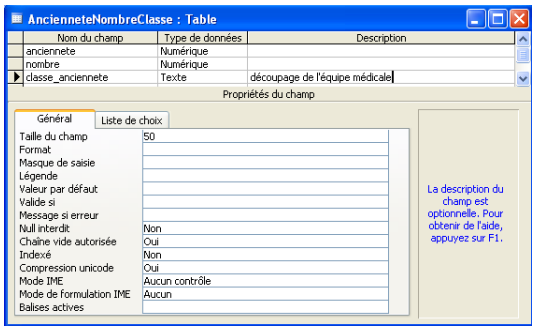
On trouve en 127 la structure de la table avant modification.

127



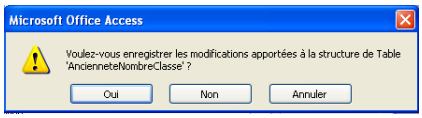
On ajoute un attribut classe_anciennete, de type Texte.

128



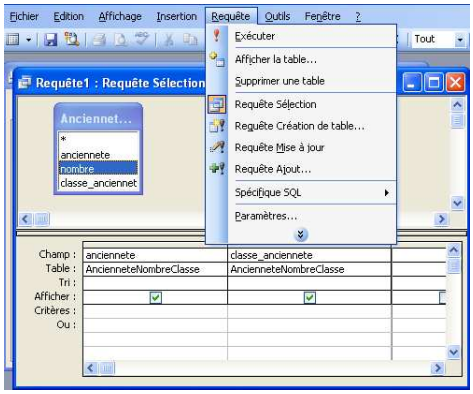
On confirme la modification de la structure de la table.

129



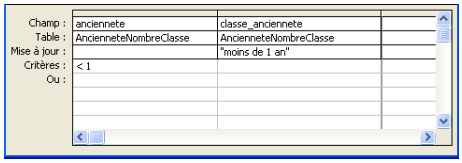
On crée alors une requête qui porte sur les attributs anciennete et classe_anciennete.

130



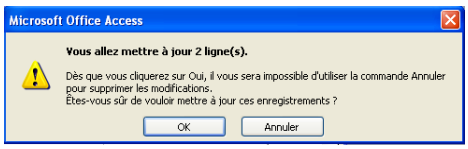
Le choix de [Requête Mise à jour] dans l'onglet [Requête] 122 de la barre de menu du haut aboutit à l'ajout d'une ligne [Mise à jour] dans la requête. On peut indiquer que lorsque l'attribut anciennete est inférieur à 1, classe_anciennete prend pour valeur 'moins de 1 an'.

131



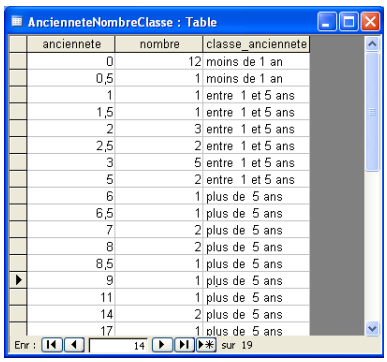
L'exécution de cette requête déclenche une demande de confirmation.

132



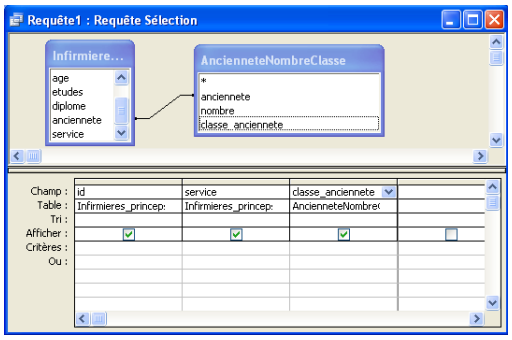
On agit de la même manière pour les deux autres valeurs de classe_anciennete et on obtient un résultat du type :

133



La deuxième étape consiste à créer la table ou la requête qui servira de matière première à l'analyse croisée. La requête résultante R_Infirmiere_Service_ClasseAnciennete est issue d'une jointure entre la table infirmieres et la table AncienneteNombreClasse sur l'identité de valeur des attributs anciennete. Elle fournit pour chaque identifiant d'infirmière le service et la classe d'ancienneté correspondants.

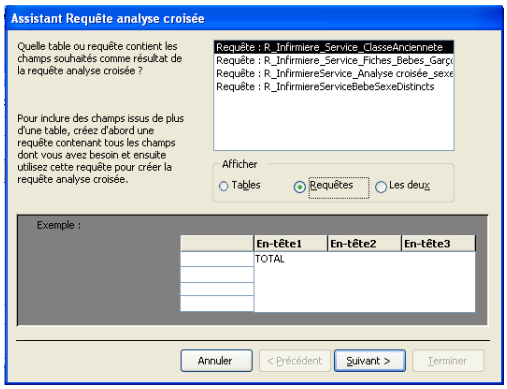
134



La troisième étape est l'analyse croisée à proprement parlée, via ce choix dans l'onglet [Nouveau] du volet [Requêtes] de la fenêtre de navigation dans la base de données.

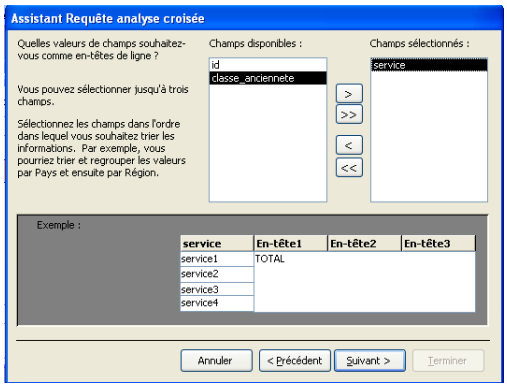
On commence par choisir la table/requête d'entrée, ici la requête R_Infirmiere_Service_Classe-Anciennete.

135



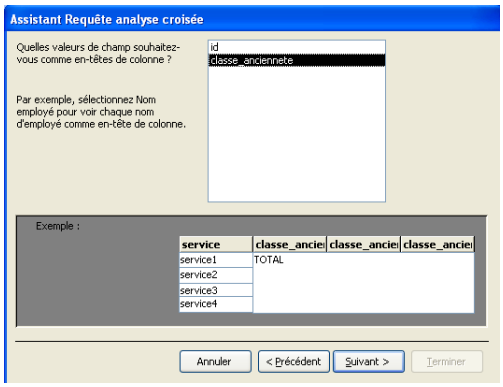
On choisit l'attribut à partir duquel opérer les regroupements et qui sera l'en-tête des lignes : service.

136



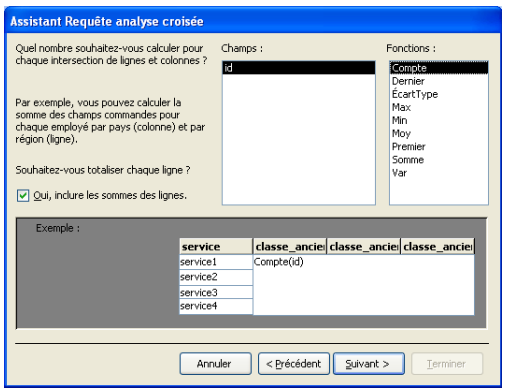
On choisit ensuite l'attribut dont les valeurs/modalités constitueront les en-têtes des autres colonnes du résultat : classe_anciennete.

137



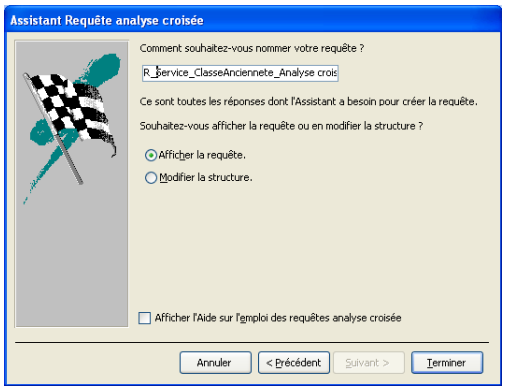
On choisit enfin l'attribut et le calcul à lui appliquer pour l'intersection service | anciennete. On compte ici le nombre d'identifiants d'infirmières, donc le nombre d'infirmières qui rentre dans la classe en question pour une valeur donnée de service.

138



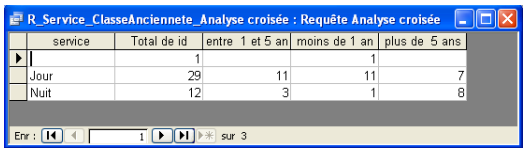
On sauvegarde la requête Analyse croisée.

139



Le résultat permet de saisir aisément les dépendances entre les modalités des deux variables en question.

140



Solution de l'exercice n°15 p. 141 Les résultats sont les suivants :

lemme	*o.*
	27126
CIRC	6781
.	2742
le	2462
,	1459
bébé	988
soin	919
-	901
très	828
*	641

lemme	*o.*
le	2462
bébé	988
soin	919
très	828
et	586
son	545
de	542
un	535
à	513
être	511

Leur correspondent les requêtes suivantes :

R^2_{67}

```
REGROUPER SUR(lemme)[occ_prema]
↪
PAR GROUPE(
lemme,
NOMBRE DE LIGNES() Titre 'o.'
)[<résultat1>]
↪
TRI SUR('o.' DÉCROISSANT)[<résultat2>]
↪
LIMITÉ À(10)[<résultat3>]
```

R^2_{68}

```
RESTRICTION(
categorie RESSEMBLANT À '^[ANRSDVCP]'
)[occ_prema]
↪
REGROUPER SUR(lemme)[<résultat1>]
↪
PAR GROUPE(
lemme,
NOMBRE DE LIGNES() Titre 'o.'
)[<résultat2>]
↪
TRI SUR('o.' DÉCROISSANT)[<résultat3>]
↪
LIMITÉ À(10)[<résultat4>]
```

MySQL

La première requête est de la forme :

```
SELECT
    lemme,
    COUNT(*) AS 'o.'
FROM occ_prema
GROUP BY lemme
ORDER BY 'o.' DESC
LIMIT 10 ;
```

La seconde en diffère uniquement par l'expression régulière utilisée pour la restriction qui a été ajoutée :

TABLEAU 47 – Phèdre : pourcentage d'accents par position

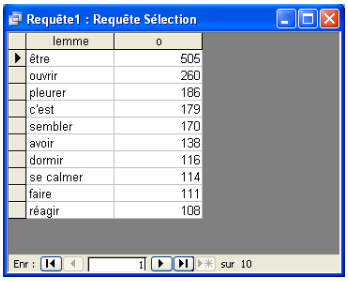
Syllabe	*nbre accents*	*%*
1	221	13.36
2	620	37.48
3	698	42.20
4	484	29.26
5	107	6.47
6	1625	98.25
7	161	9.73
8	453	27.39
9	873	52.78
10	345	20.86
11	40	2.42
12	1650	99.76


```
... WHERE categorie REGEXP '^[ANRSDVCP]' ...
```

Access

141

142



Solution de l'exercice n°16 p. 148 Le résultat de la R^2_{69} figure au tableau 47 p. 189.

R^2_{69}

```
REGROUPER SUR(position)[positions]
↪
PAR GROUPE(
position TITRE 'Syllabe',
SOMME(accent) TITRE 'nbre accents',
SOMME(accent) / 1654 * 100 TITRE '%'
)[<résultat1>]
```

TABLEAU 48 – PHÈDRE : proportion de vers partagés par personnage

Aricie : vers	*vers partagés*	%
140	7	5.00

Hippolyte : vers	*vers partagés*	%
362	14	3.87

Ismène : vers	*vers partagés*	%
34	3	8.82

Oenone : vers	*vers partagés*	%
220	18	8.18

Panope : vers	*vers partagés*	%
38	4	10.53

Phèdre : vers	*vers partagés*	%
485	21	4.33

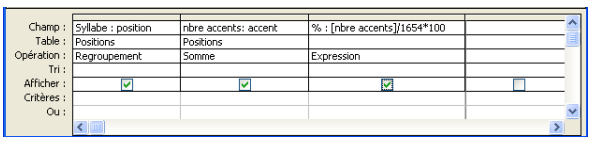
Théramène : vers	*vers partagés*	%
183	7	3.83

Thésée : vers	*vers partagés*	%
234	10	4.27

MySQL

```
SELECT
      position AS 'Syllabe',
      SUM(accent) AS 'nbre_accents',
      FORMAT(SUM(accent) / 1654 * 100, 2) AS '%'
FROM positions
GROUP BY position ;
```

Access



Solution de l'exercice n°17 p. 148 On commence par opérer une restriction sur les vers où intervient le personnage. On compte un vers partagé quand l'attribut `partage_en` est supérieur à 1. Le résultat figure au tableau 48 p. 190.

R^2_{70}

```
RESTRICTION(
personnage COMME '%ARICIE%'
)[vers]
↪
PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'Aricie : vers',
SOMME(SI(partage_en >1, 1, 0)) TITRE 'vers partagés',
SOMME(SI(partage_en >1, 1, 0)) / NOMBRE DE LIGNES() *
100 TITRE '%'
)[<résultat1>]
```

MySQL

```
SELECT
    COUNT(*) AS 'Aricie : vers',
    SUM(IF(partage_en > 1, 1, 0)) AS 'vers partagés',
    SUM(IF(partage_en > 1, 1, 0)) / COUNT(*) * 100 AS '%'
FROM vers
WHERE personnage_s LIKE '%ARICIE%' ;
```

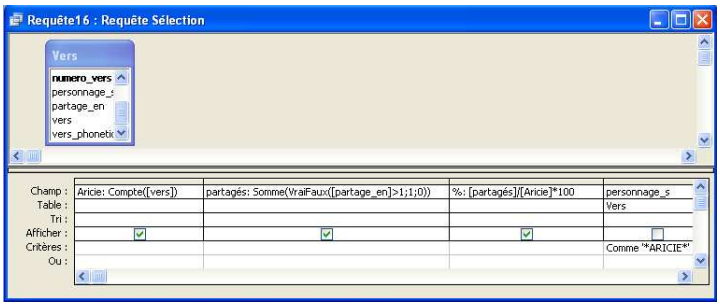
L'opérateur **LIKE** pourrait d'ailleurs être remplacé par **REGEXP** :

```
SELECT
    ...
FROM vers
WHERE personnage_s REGEXP 'ARICIE' ;
```

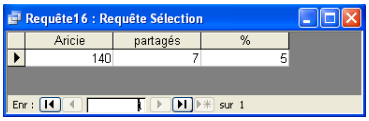
Access

144 fournit une première solution, dont le résultat figure en 145.

144



145

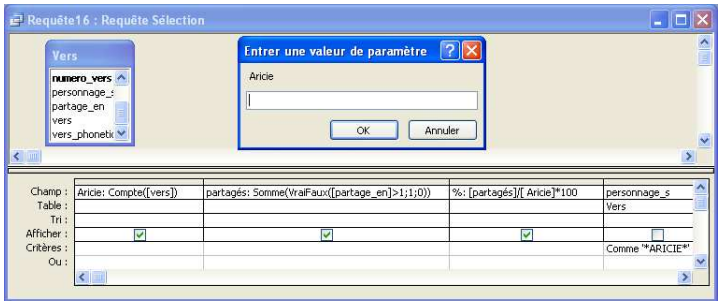


△

146, apparemment identique à 144, en diffère cependant pour la formulation du calcul dans la 3ecolonne : [Aricie] remplace [Aricie], une espace figure après le crochet ouvrant. Dans ces conditions, pour Access, ce qui figure entre crochets n'est pas le nom d'une colonne

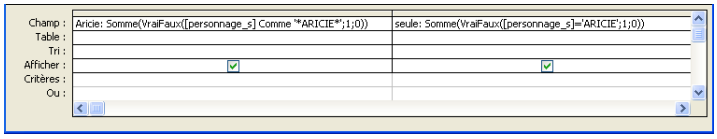
préexistante dont la valeur peut être utilisée, mais un paramètre. On obtient donc, sans le désirer, une requête paramétrée, ce que montre le haut de l'écran.

146



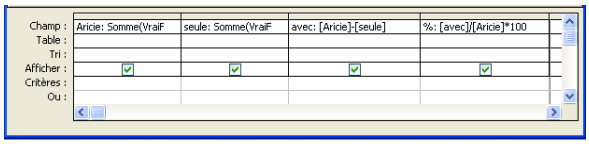
La solution 147 (partie gauche de la requête) est très proche de la précédente, à ceci près qu'elle n'opère pas de restriction mais effectue le filtrage dans le calcul des colonnes à fournir en résultat. On calcule le nombre de vers où Aricie est seule, c'est-à-dire où la chaîne 'ARICIE' occupe à elle seule la colonne personnage_s.

147



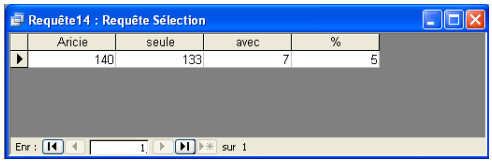
On peut alors en 148 (partie droite de la requête) calculer le nombre de vers prononcés avec un autre personnage : la valeur de la colonne Aricie moins la valeur de la colonne seule.

148



En 149 se trouve le résultat.

149



△

Noter en 150 que si l'on décoche la colonne seule, Access considère qu'on a défini une requête paramétrée dont le paramètre, demandé à l'utilisateur, est la valeur de seule.

La requête SQL Server engendrée est en effet de la forme :

```
SELECT
    Sum(IIf([personnage_s] Like '*ARICIE*',1,0)) AS Aricie,
    [Aricie]-[seule] AS avec,
    [avec]/[Aricie]*100 AS [%]
FROM Vers;
```

où la valeur de la colonne seule n'est pas définie et donc doit être demandée à l'utilisateur, tandis qu'est engendrée pour 147 et 148 la requête :

```
SELECT
    Sum(IIf([personnage_s] Like '*ARICIE*',1,0)) AS Aricie,
    Sum(IIf([personnage_s]='ARICIE',1,0)) AS seule,
    [Aricie]-[seule] AS avec,
    [avec]/[Aricie]*100 AS [%]
FROM Vers;
```

Solution de l'exercice n°18 p. 148 Les résultats figurent du tableau 49 au tableau 52, de la p. 194 à la p. 196.

Chacune des requêtes est construite sur le même modèle :

R^2_{71}

RESTRICTION(
cat PARMI ('Adj', 'Nc', 'Np', 'V')
)[occurrences]
↪
REGROUPER SUR(occ_car)[<résultat$_1$>]
AVEC(
NOMBRE DE VALEURS DISTINCTES(personnage) = 1
ET personnage = 'PHEDRE'
ET NOMBRE DE LIGNES() > 1
)[<résultat$_1$>]
↪
PAR GROUPE(
personnage,
occ_car,
cat,
NOMBRE DE LIGNES() Titre 'o.'
)[<résultat$_2$>
↪
TRI SUR(cat, 'o.' DÉCROISSANT)[<résultat$_3$>]

On commence par se restreindre aux mots dits pleins (adjectifs, noms communs, noms propres et verbes) et par regrouper les occurrences d'un même mot (REGROUPER SUR). On

TABLEAU 49 – PHÈDRE : mots (hors hapax) prononcés uniquement par Phèdre

personnage	*occ_car*	*cat*	*o.*
PHEDRE	descendue	Adj	3
PHEDRE	sensible	Adj	3
PHEDRE	vains	Adj	2
PHEDRE	odieuse	Adj	2
PHEDRE	épouvantée	Adj	2
PHEDRE	suivi	Adj	2
PHEDRE	prompt	Adj	2
PHEDRE	perdue	Adj	2
PHEDRE	vus	Adj	2
PHEDRE	trouvé	Adj	2
PHEDRE	flatteurs	Adj	2
PHEDRE	fermé	Adj	2
PHEDRE	osé	Adj	2
PHEDRE	divers	Adj	2
PHEDRE	Malheureuse	Adj	2
PHEDRE	Insensée	Adj	2
PHEDRE	conseils	Nc	5
PHEDRE	aveu	Nc	4
PHEDRE	détours	Nc	3
PHEDRE	urne	Nc	2
PHEDRE	tourments	Nc	2
PHEDRE	penchant	Nc	2
PHEDRE	oreille	Nc	2

personnage	*occ_car*	*cat*	*o.*
PHEDRE	tourment	Nc	2
PHEDRE	déesse	Nc	2
PHEDRE	traits	Nc	2
PHEDRE	fil	Nc	2
PHEDRE	comble	Nc	2
PHEDRE	Labyrinthe	Nc	2
PHEDRE	Soleil	Nc	2
PHEDRE	tremble	V	4
PHEDRE	meurs	V	3
PHEDRE	rougis	V	3
PHEDRE	souffrir	V	3
PHEDRE	pouvais	V	3
PHEDRE	osai	V	2
PHEDRE	implorer	V	2
PHEDRE	aimer	V	2
PHEDRE	ranimer	V	2
PHEDRE	fût	V	2
PHEDRE	laissais	V	2
PHEDRE	soutiens	V	2
PHEDRE	séduire	V	2
PHEDRE	évitais	V	2
PHEDRE	vis	V	2
PHEDRE	nuire	V	2
PHEDRE	Délivre	V	2
PHEDRE	mesure	V	2
PHEDRE	fuyais	V	2
PHEDRE	mourrai	V	2
PHEDRE	voyaient	V	2
PHEDRE	prier	V	2
PHEDRE	brûle	V	2

conserve uniquement les groupes, c'est-à-dire les mots pleins, qui ne proviennent que d'un seul personnage : ⌈NOMBRE DE VALEURS DISTINCTES(personnage) = 1⌉.

Dans ces groupes, on retient ceux prononcés par le personnage choisi et dont le nombre d'occurrences dépasse 1 : personnage = 'PHEDRE' ET NOMBRE DE LIGNES() > 1.

Le tri est opéré d'abord par catégorie morpho-syntaxique, puis par fréquence décroissante.

MySQL

```
SELECT
      personnage,
      occ_car,
      cat,
      COUNT(occ_car) as 'o.'
FROM occurrences
WHERE cat IN ('Adj', 'Nc', 'Np', 'V')
GROUP BY occ_car
HAVING COUNT(DISTINCT personnage) = 1
      AND personnage = 'PHEDRE'
      AND COUNT(occ_car) > 1
ORDER BY cat, 'o.' DESC ;
```

TABLEAU 50 – PHÈDRE : mots (hors hapax) prononcés uniquement par Hipollyte

personnage	*occ_car*	*cat*	*o.*
HIPPOLYTE	adoucie	Adj	2
HIPPOLYTE	communs	Adj	2
HIPPOLYTE	sacrés	Adj	2
HIPPOLYTE	dû	Adj	2
HIPPOLYTE	longue	Adj	2
HIPPOLYTE	sincère	Adj	2
HIPPOLYTE	auguste	Adj	2
HIPPOLYTE	droits	Nc	4
HIPPOLYTE	récit	Nc	3
HIPPOLYTE	tutelle	Nc	2
HIPPOLYTE	obstacle	Nc	2
HIPPOLYTE	outrages	Nc	2
HIPPOLYTE	jeunesse	Nc	2
HIPPOLYTE	mensonge	Nc	2
HIPPOLYTE	histoire	Nc	2
HIPPOLYTE	entretien	Nc	2
HIPPOLYTE	querelle	Nc	2
HIPPOLYTE	successeur	Nc	2
HIPPOLYTE	sentiments	Nc	2
HIPPOLYTE	adresse	Nc	2
HIPPOLYTE	serment	Nc	2
HIPPOLYTE	exemple	Nc	2
HIPPOLYTE	intérêt	Nc	2
HIPPOLYTE	dépouille	Nc	2
HIPPOLYTE	noms	Nc	2
HIPPOLYTE	oubli	Nc	2
HIPPOLYTE	sceptre	Nc	2
HIPPOLYTE	Pitthée	Np	2
HIPPOLYTE	Sparte	Np	2
HIPPOLYTE	pars	V	4
HIPPOLYTE	peuvent	V	3
HIPPOLYTE	Confier	V	3
HIPPOLYTE	vouloir	V	3
HIPPOLYTE	troubler	V	3
HIPPOLYTE	sortez	V	2
HIPPOLYTE	balancer	V	2
HIPPOLYTE	réprouve	V	2
HIPPOLYTE	pourrait	V	2
HIPPOLYTE	émouvoir	V	2
HIPPOLYTE	aviez	V	2
HIPPOLYTE	écoutant	V	2
HIPPOLYTE	oubliez	V	2
HIPPOLYTE	trouve	V	2

TABLEAU 51 – PHÈDRE : mots (hors hapax) prononcés uniquement par Aricie

personnage	*occ_car*	*cat*	*o.*
ARICIE	Présents	Nc	3
ARICIE	adieu	Nc	2
ARICIE	songe	Nc	2
ARICIE	Partez	V	3
ARICIE	honorer	V	2

TABLEAU 52 – PHÈDRE : mots (hors hapax) prononcés uniquement par Thésée

personnage	*occ_car*	*cat*	*o.*
THESEE	certains	Adj	2
THESEE	immortelle	Adj	2
THESEE	purgé	Adj	2
THESEE	outragé	Adj	2
THESEE	traître	Nc	5
THESEE	bonté	Nc	2
THESEE	prison	Nc	2
THESEE	accueil	Nc	2
THESEE	embrassements	Nc	2
THESEE	scélérats	Nc	2
THESEE	vienne	V	4
THESEE	jurait	V	2
THESEE	condamner	V	2
THESEE	fuir	V	2
THESEE	soient	V	2

Solution de l'exercice n°19 p. 148 Il suffit d'ajouter aux requêtes précédentes la contrainte que la fin du mot soit en 12^{e} syllabe (résultat tableau 53 p. 197).

R^{2}_{72}

RESTRICTION(
cat PARMI ('Adj', 'Nc', 'Np', 'V')
ET fin_syl = 12
)[occurrences]
↪
REGROUPER SUR(occ_car)[<résultat$_1$>]
AVEC(
NOMBRE DE VALEURS DISTINCTES(personnage) = 1
ET personnage = 'PHEDRE'
ET NOMBRE DE LIGNES() > 1
)[<résultat$_1$>]
↪
PAR GROUPE(
personnage,
occ_car,
cat,
NOMBRE DE LIGNES() Titre 'o.'
)[<résultat$_2$>
↪
TRI SUR(cat, 'o.' DÉCROISSANT)[<résultat$_3$>]

MySQL

```
SELECT
      personnage,
      occ_car,
      cat,
     COUNT(occ_car) as 'o.'
FROM occurrences
WHERE cat IN ('Adj', 'Nc', 'Np', 'V') AND fin_syl = 12
GROUP BY LOWER(occ_car)
```

TABLEAU 53 – PHÈDRE : mots à la rime propres à 1 personnage

personnage	*occ_car*	*cat*	*o.*
PHEDRE	descendue	Adj	3
PHEDRE	perdue	Adj	2
PHEDRE	épouvantée	Adj	2
PHEDRE	divers	Adj	2
PHEDRE	vainqueur	Nc	2
PHEDRE	cheveux	Nc	2
PHEDRE	détours	Nc	2
PHEDRE	enfers	Nc	2
PHEDRE	outrage	V	3
PHEDRE	souffrir	V	2
PHEDRE	implorer	V	2
PHEDRE	nuire	V	2


```
    HAVING
        COUNT(DISTINCT personnage) = 1
        AND personnage = 'PHEDRE'
        AND COUNT(occ_car) > 1
ORDER BY cat, 'o.' DESC ;
```

Solution de l'exercice n°20 p. 148

R^2_{73}

RESTRICTION(
cat_derive EST NULL
OU cat_derive COMME '%?%'
)[esque]
↪
PAR GROUPE(
NOMBRE DE LIGNES()
)[<résultat$_1$>]

MySQL

Il y a 139 dérivés en *-esque* dont la catégorie est inconnue ou flottante, indiquent les 3 requêtes suivantes, qui diffèrent uniquement par la manière d'exprimer le motif :

```
SELECT COUNT(*)
FROM esque
WHERE cat_derive IS NULL
    OR cat_derive LIKE '%?%' ;
```

△

On notera que le point d'interrogation est un caractère spécial pour les expressions régulières. On ne peut donc l'employer tel quel. Des formulations comme :

```
... WHERE cat_derive IS NULL OR cat_derive REGEXP '?' ;
... WHERE cat_derive IS NULL OR cat_derive REGEXP '\?' ;
```

déclenchent en effet un message d'erreur :

```
Got error 'repetition-operator operand invalid' from regexp
```

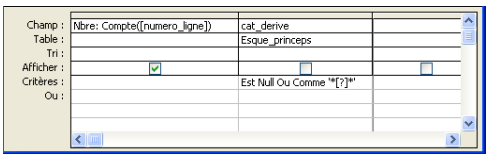
dans la mesure où le point d'interrogation intervient normalement comme opérateur post-posé de répétition optionnelle. Une double contre-oblique ou encore l'utilisation d'un ensemble de caractères réduit au singleton ? signale l'emploi littéral du point d'interrogation :

```
... WHERE cat_derive IS NULL OR cat_derive REGEXP '\\?' ;

... WHERE cat_derive IS NULL OR cat_derive REGEXP '[?]' ;
```

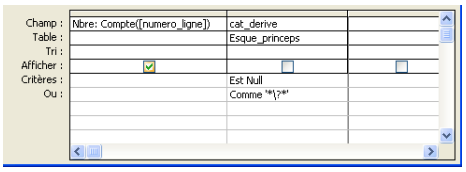
Access

Le point d'interrogation ayant aussi un rôle spécifique pour Access pendant les recherches approximatives (il remplace un caractère quelconque), il ne peut être employé comme tel et doit figurer dans un contexte permettant de voir qu'il est pris littéralement et non comme un méta-caractère. C'est le cas entre crochets 151 ou précédé d'une contre-oblique 152. Dans ce dernier cas, cependant, le nombre d'occurrences n'est mystérieusement pas tout à fait le même en 153 : 134 au lieu de 139.

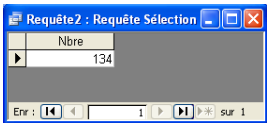
151



152



153



Solution de l'exercice n°21 p. 148 On constate, au travers du tableau 54 p. 199, que la requête :

R^2_{74}

REGROUPER SUR(auteur)[esque]
AVEC(NOMBRE DE LIGNES() > 5)
↪
PAR GROUPE(
auteur
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_1$>]

met en évidence la place singulière de San Antonio dans les attestations que rassemble la table esque. On note également qu'une petite différence dans le rendu de la coordination (*et* ou &) aboutit à séparer en 2 lignes les contextes dus aux deux Goncourt.

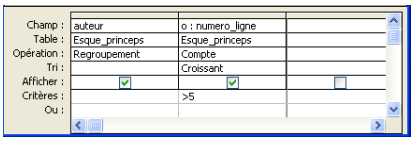
MySQL

TABLEAU 54 – ESQUE : les auteurs de plus de 5 dérivés en *-esque*

auteur	*o.*
Riou (A.)	6
Frappat (B.)	6
Quélin (J.-P.)	6
Baudou (J.)	6
Thibaudet (A.)	6
Legrand (D.)	6
Goncourt (E. & J. de)	6
Faurisson (R.)	6
Mortaigne (V.)	6
Heymann (D.)	7
Godard (C.)	7
Coljon (T.)	7
Rabelais (F.)	7
Fallet (R.)	7
Benaim (L.)	7
Guérin (P.)	7
Lonchampt (J.)	7
Daudet (L.)	8
Simon (C.)	8
Salmi (M., trad. par J. Chuzeville)	8
Zand (N.)	8
Van Vaerenbergh (O.)	9
Goncourt (E. de)	9
Bradfer (F.)	11
Montaigne (M. de)	11
Giraud (J.)	14
Goncourt (E. et J. de)	14
Balzac (H. de)	15
Marcelle (P.)	16
Georges (P.)	17
Dagen (P.)	17
Honorez (L.)	17
Verlaine (P.)	20
Boudard (A.)	25
San-Antonio	449

```
SELECT
      auteur,
      COUNT(auteur) AS 'o.'
FROM esque
GROUP BY auteur
      HAVING COUNT(auteur) > 5
ORDER BY 'o.' ;
```

Access



Solution de l'exercice n°22 p. 157

R^2_{75}

```
    RESTRICTION(cat = 'Nc')[occurrences]
↪   REGROUPER SUR(
    personnage,
    MINUSCULE(occ_car)
    )[<résultat1>]
    AVEC(NOMBRE DE LIGNES() >= 5)
↪
    PAR GROUPE(
    personnage TITRE 'Personnage',
    MINUSCULE(occ_car) TITRE 'mot',
    NOMBRE DE LIGNES() TITRE 'o.'
    )[<résultat2>]
↪
    TRI SUR(
    personnage,
    'o.' DÉCROISSANT
    )[<résultat3>]
```

Le résultat figure au tableau 55 p. 201. Les mots sont minusculisés pour éliminer les majuscules artificielles de début de vers.

MySQL

```
SELECT
      personnage AS 'Personnage',
      LOWER(occ_car) AS 'mot',
      COUNT(*) AS 'o.'
FROM occurrences
WHERE cat = 'Nc'
GROUP BY
      personnage,
      LOWER(occ_car)
  HAVING COUNT(*) >= 5
ORDER BY personnage, 'o.' DESC ;
```

TABLEAU 55 – PHÈDRE : noms communs > 5 o. par personnage

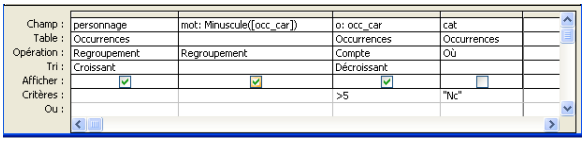
Personnage	*mot*	*o.*
ARICIE	seigneur	13
ARICIE	yeux	7
ARICIE	père	5
HIPPOLYTE	père	14
HIPPOLYTE	dieux	11
HIPPOLYTE	coeur	10
HIPPOLYTE	madame	10
HIPPOLYTE	amour	10
HIPPOLYTE	fils	7
HIPPOLYTE	seigneur	7
HIPPOLYTE	sang	6
HIPPOLYTE	jour	5
HIPPOLYTE	voeux	5
HIPPOLYTE	âme	5
HIPPOLYTE	lieux	5
HIPPOLYTE	vertu	5
OENONE	madame	12
OENONE	dieux	10
OENONE	yeux	8
OENONE	sang	7
OENONE	fils	7
OENONE	jour	5
OENONE	amour	5

Personnage	*mot*	*o.*
PHEDRE	yeux	20
PHEDRE	coeur	20
PHEDRE	fils	13
PHEDRE	amour	12
PHEDRE	ciel	11
PHEDRE	sang	11
PHEDRE	dieux	10
PHEDRE	père	9
PHEDRE	époux	9
PHEDRE	soin	8
PHEDRE	jour	8
PHEDRE	main	8
PHEDRE	seigneur	8
PHEDRE	crime	7
PHEDRE	front	7
PHEDRE	mère	6
PHEDRE	hélas	6
PHEDRE	horreur	6
PHEDRE	remords	6
PHEDRE	voeux	6
PHEDRE	haine	6
PHEDRE	nom	6
PHEDRE	mort	5
PHEDRE	fois	5

Personnage	*mot*	*o.*
PHEDRE	monstre	5
PHEDRE	oenone	5
PHEDRE	conseils	5
PHEDRE	larmes	5
PHEDRE	fureur	5
PHEDRE	moment	5
PHEDRE	raison	5
THERAMENE	seigneur	16
THERAMENE	héros	5
THERAMENE	yeux	5
THERAMENE	voix	5
THESEE	fils	14
THESEE	dieux	10
THESEE	yeux	7
THESEE	père	7
THESEE	lieux	6
THESEE	coeur	5
THESEE	sang	5
THESEE	fois	5
THESEE	voeux	5

Access

La requête :



155

diffère très légèrement de la version MySQL (sont retenus les mots dont le nombre d'occurrences est strictement supérieur à 5). L'équivalent SQL Server est :

```
SELECT Occurrences.personnage,
       LCase([occ_car]) AS mot,
       Count(Occurrences.occ_car) AS o
FROM Occurrences
WHERE (((Occurrences.cat)="Nc"))
GROUP BY Occurrences.personnage,
       LCase([occ_car])
  HAVING (((Count(Occurrences.occ_car))>5))
ORDER BY Occurrences.personnage,
       Count(Occurrences.occ_car) DESC;
```

CHAPITRE V

ÉTUDE DE CAS 2 : *PHÈDRE*

1. Vers, syntaxe, drame : vues en tension

1.1. ⊕ Répartition des fins de phrases par personnages et positions métriques

Le calcul de la répartition des fins de phrases selon les personnages et les principales positions métriques s'effectue par des requêtes proches sous MySQL et sous Access. Les fonctions MySQL **FORMAT** et **IF** sont remplacées respectivement sous SQL Server par les fonctions **FORMATNUMBER** et **IIF**. Elles correspondent à la requête abstraite :

R^2_{76}

```
    RESTRICTION(cat = 'SepPhrase')[occurrences]
↪
    REGROUPER SUR(personnage)[<résultat1>]
↪
    PAR GROUPE(
    NOMBRE DE LIGNES() TITRE 'fin de phrases',
    SOMME(SI(debut_syl = 6, 1, 0)) TITRE '6',
    SOMME(SI(debut_syl = 6, 1, 0)) / NOMBRE DE LIGNES() *
    100 TITRE '%',
    . . .
    )[<résultat2>]
↪
    TRI SUR('fin de phrases')[<résultat3>]
```

MySQL

```
SELECT personnage AS 'Personnage',
       COUNT(*) AS 'fin de phrases',
       SUM(IF (debut_syl = 6, 1, 0)) AS '6',
       SUM(IF (debut_syl = 6, 1, 0)) / COUNT(*) * 100 AS '%',
       SUM(IF (debut_syl = 12, 1, 0)) AS '12',
       SUM(IF (debut_syl = 12, 1, 0)) / COUNT(*) * 100 AS '%',
       SUM(IF (debut_syl NOT IN (6, 12), 1, 0)) AS 'autre',
       SUM(IF (debut_syl NOT IN (6, 12), 1, 0))
            / COUNT(*) * 100 AS '%'
FROM occurrences
WHERE cat = 'SepPhrase'
GROUP BY personnage
ORDER BY 'fin de phrases' ;
```

Access

```
SELECT Occurrences.personnage,
       COUNT(Occurrences.numero_vers) AS [fin de phrases],
       SUM(IIF([debut_syl]=6, 1, 0)) AS [syll 6],
       FORMATNUMBER(SUM(IIF([debut_syl]=6, 1, 0)) /
            COUNT(*) * 100, 2) AS [% 6],
       SUM(IIF([debut_syl]=12, 1, 0)) AS [syll 12],
       FORMATNUMBER(SUM(IIF([debut_syl]=12, 1, 0)) /
            COUNT(*) * 100, 2) AS [% 12],
       SUM(IIF([debut_syl] NOT IN (6,12), 1, 0)) AS [syll autre],
       FORMATNUMBER(SUM(IIF([debut_syl] NOT IN (6, 12), 1, 0)) /
       COUNT(*) * 100, 2) AS [% autre]
FROM Occurrences
WHERE (((Occurrences.cat)="SepPhrase"))
GROUP BY Occurrences.personnage
ORDER BY '[fin de phrases]';
```

La clause ⌈**ORDER BY** '[fin de phrases]'⌉ peut s'écrire également ⌈**ORDER BY** 'fin de phrases'⌉. Le résultat figure en 1.

1

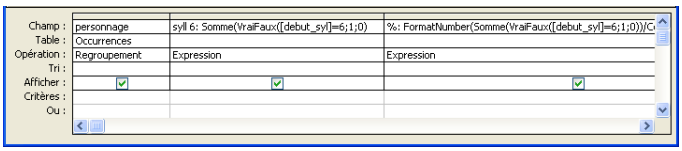
PersonnageFinPhrasesPositions : Requête Sélection

personnage	fin de phrases	syll 6	% 6	syll 12	% 12	syll autre	% autre
ISMENE	15	0	0,00	14	93,33	1	6,67
PANOPE	18	1	5,56	17	94,44	0	0,00
ARICIE	80	6	7,50	70	87,50	4	5,00
THERAMENE	94	5	5,32	85	90,43	4	4,26
OENONE	147	8	5,44	120	81,63	19	12,93
THESEE	150	7	4,67	124	82,67	19	12,67
HIPPOLYTE	187	12	6,42	168	89,84	7	3,74
PHEDRE	318	18	5,66	268	84,28	32	10,06

Enr : 2 sur 8

Faute de place, une partie seulement de la requête formulée via l'interface graphique est fournie en 2 .

2



On remarque que **SUM** et **IIF** sous SQL Server correspondent sous Access respectivement à ⌈Somme⌉ et à ⌈VraiFaux⌉. Les arguments de cette dernière fonction sont séparés par des

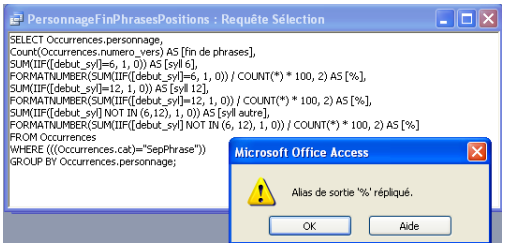
points-virgules et non par des virgules. Par contre, la fonction **FORMATNUMBER** reste inchangée. Le calcul du pourcentage de syllabes en position 6 qui correspondent à une fin de phrase s'effectue par l'expression :

```
% 6 : FormatNumber(Somme(VraiFaux([debut_syll] = 6 ; 1 ; 0)) / Compte(*) * 100, 2)
```

△

On note en 3 qu'Access, contrairement à MySQL, ne semble pas accepter que des alias de colonne portent le même nom.

3



Exercice n°1 Pour chaque personnage de *Phèdre*, calculer le nombre de mots, de phrases, de tirades, ainsi que le nombre de mots en moyenne par phrase et par tirade.

2. Une base pour articuler les points de vue

On se reportera au ch. XV § 3 pour le détail des types de données offerts par MySQL et Access pour les attributs des tables.

3. Les vers

3.1. ⊕ Conventions phonétiques du métromètre

Les conventions phonétiques de (Beaudouin, 2002) sont présentées dans le tableau 56 p. 205.

MySQL

La structure de la table vers figure au tableau 57 p. 205.

Access

La structure de la table vers figure en 4 .

TABLEAU 56 – PHÈDRE : conventions phonétiques du métromètre

Voyelles	
Métromètre	*exemple*
@	l**e**, qu**e**
@@	flamm**e**
ai	p**è**re, m**ê**me
an	puiss**an**ce, am**an**t
e	hym**é**née, parl**er**
eu	y**eu**x, l**ieu**
i	v**i**e, emp**i**re
in	r**ien**, m**ain**
o	m**o**rt, pers**o**nne
oe	c**oe**ur, pl**eu**rs
on	rais**on**, h**on**te
oo	**au**tre, ch**o**se
ou	v**ou**s, am**ou**r
u	v**ue**, pl**u**s
un	comm**un**s, import**un**s
ywi	l**ui**, r**ui**ne
wa	m**oi**, gl**oi**re
win	p**oin**t, m**oin**s

Consonnes	
Métromètre	*exemple*
p	pas, point
t	tourment, fortune
k	pourquoi, conduire
b	troubler, bien
d	douleur, dieu
g	guerre, garantir
f	fatal, fille
s	superbe, audace
sh	châtiment, charmer
v	révéler, victime
z	visage, jalousie
j	jour, âgé, manger
l	autel, calme
r	rive, séjour
m	madame, chemin
n	nuage, fortune
ny	seigneur, poignard

Semi-voyelles	
Métromètre	*exemple*
y	**y**eux, l**i**eu
yw	lui, fuir
w	oui, ouest

Consonnes de liaison	
Métromètre	*exemples*
ln	u**n e**mpereur
lr	respecte**r u**n amour
lt	tou**t à** coup
lz	le**s e**nnemis

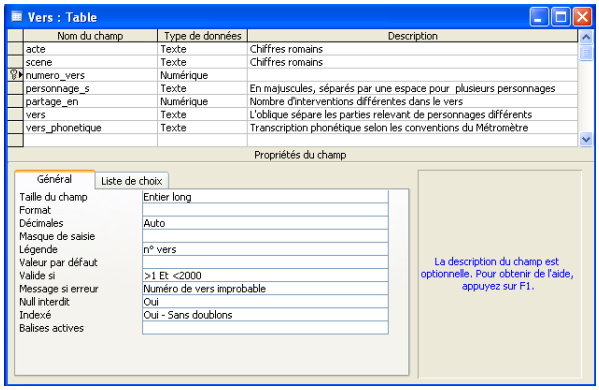
TABLEAU 57 – PHÈDRE : structure de la table vers (MySQL)

Field	*Type*	*Null*	*Key*	*Default*	*Extra*
acte	varchar(5) binary				
scene	varchar(5) binary				
numero_vers	int(11)		PRI	1	
personnage_s	varchar(255) binary				
partage_en	int(11)			1	
vers	varchar(200) binary				
vers_phonetique	varchar(250) binary				

TABLEAU 58 – PHÈDRE : structure de la table positions (MySQL)

Field	*Type*	*Null*	*Key*	*Default*	*Extra*
numero_vers	int(11)			1	
position	int(11)			1	
syll	varchar(15) binary				
voy	varchar(5) binary				
cat	varchar(30) binary				
accent	int(11)			0	
fin_mot	int(11)			0	
numero_ligne	int(11)		PRI		auto_increment

4



4. Les positions métriques

MySQL

La structure de la table positions figure au tableau 58 p. 206.

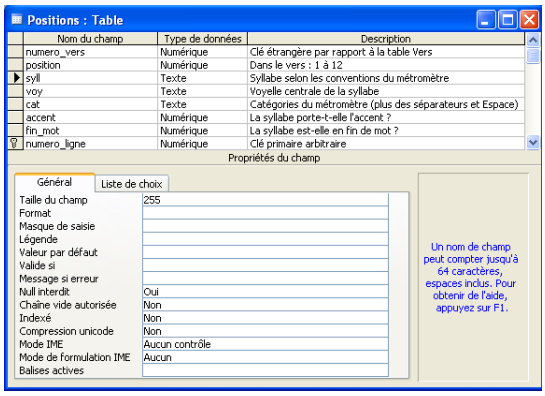
Cette table comprend un attribut à valeur auto-incrémentée (cf. auto_increment dans la colonne Extra). C'est le moyen de donner automatiquement à chacune des lignes d'une table un numéro d'ordre arbitraire tel qu'une nouvelle ligne ait un numéro d'ordre plus grand que tous les numéros d'ordre déjà existants. Chaque nouvelle ligne ajoutée comprenant un tel attribut voit cet attribut prendre la valeur suivante de la dernière valeur donnée automatiquement.

Access

La structure de la table occurrences figure en 5 .

Tableau 59 – Phèdre : structure de la table occurrences (MySQL)

Field	*Type*	*Null*	*Key*	*Default*	*Extra*
numero_vers	int(11)			1	
personnage	varchar(100) binary				
numero_tirade	int(11)			0	
numero_phrase	int(11)			0	
occ_car	varchar(100) binary				
occ_phon	varchar(100) binary				
cat	varchar(100) binary	YES			
type_liaison	char(2)			ss	
liaison	varchar(5) binary	YES			
debut_car	int(11)			0	
fin_car	int(11)			0	
debut_phon	int(11)			0	
fin_phon	int(11)			0	
debut_syl	decimal(4,1)			0.0	
fin_syl	decimal(4,1)			0.0	
syllabe_accentuee	int(11)			0	
numero_ligne	int(11)		PRI		auto_increment



5. Les « mots »

MySQL

La structure de la table occurrences figure au tableau 59 p. 207.

Un certain nombre de requêtes complémentaires permettent de cerner le contenu de cette table. La première :

```
SELECT cat AS 'Catégorie',
     COUNT(*) AS 'o.'
FROM occurrences
GROUP BY cat ;
```

dont l'équivalent abstrait est :

R^2_{77}
REGROUPER SUR(cat)[occurrences]
↪ PAR GROUPE(
cat TITRE 'Catégorie',
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_1$>]

fournit le nombre d'occurrences de chaque catégorie (tableau 60 p. 209). On note que les espaces sont catégorisées comme telles, ainsi que les différents types de séparateurs. La virgule constitue un séparateur de groupe. Certaines ponctuations, comme le point d'exclamation ou le point d'interrogation, peuvent représenter tantôt des séparateurs de groupe, tantôt des séparateurs de phrase.

La seconde requête :

```
SELECT LOWER(occ_car) AS 'Mot',
    COUNT(*) AS 'o.'
FROM occurrences
WHERE cat = 'Nc'
GROUP BY LOWER(occ_car)
ORDER BY 'o.' DESC
LIMIT 10 ;
```

dont l'équivalent abstrait est :

R^2_{78}
RESTRICTION(cat = 'Nc')[occurrences]
↪ REGROUPER SUR(
MINUSCULE(occ_car)
)[<résultat$_1$>]
↪
PAR GROUPE(
MINUSCULE(occ_car) TITRE 'Mot',
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_2$>]
↪
TRI SUR('o.' DÉCROISSANT)[<résultat$_3$>]
↪
LIMITÉ À(10)[<résultat$_4$>]

rapatrie les 10 premiers noms par nombre d'occurrences décroissant. Il suffit de changer la condition de restriction pour obtenir le résultat pour une autre catégorie. On a pris la précaution de minusculiser ces mots pour que les majuscules d'initiale de vers ou de phrase ne viennent pas brouiller les rapprochements. Les tableaux 61 et 62 p. 210 et p. 211 donnent les résultats pour les catégories des mots dits pleins et pour les mots dits outils respectivement. Les requêtes pour les autres catégories que Nc diffèrent par la catégorie cible. Pour les noms propres (Np), on ne minusculise pas la valeur de l'attribut occ_car. Examiner ces tableaux permet de comprendre les choix d'étiquetage, de catégorisation qui ont été faits et donc d'en tenir compte, de composer avec, lors des requêtes. On constate ainsi que la catégorie Adj rassemble également les participes passés. On remarque des erreurs d'étiquetage (*il* nom propre). On voit la place des variations morphologiques (*je* et *j'* ; *de*, *d'*, *du*, *des*).

Exercice n°2 En utilisant Access et son interface (et non le mode SQL direct), formulez la requête permettant de connaître la fréquence de chaque catégorie (hors espaces et séparateurs) de la table occurrences.

TABLEAU 60 – PHÈDRE : fréquence des catégories de la table occurrences

Catégorie	*o.*
Adj	1403
Adv	855
Conj	412
Dét/Pron	4206
Espace	11574
Nc	2582
Np	279
Prép	1427
Rel	586
SepGroupe	1117
SepGuillemet	2
SepParentheseFermante	1
SepParentheseOuvrante	1
SepPhrase	1009
SepPhraseOuGroupe	283
SepTiradeDansVers	47
V	2464

Exercice n°3 En utilisant Access et son interface (et non le mode SQL direct), donnez les 10 adjectifs les plus fréquents de la table occurrences. Le résultat sera identique à celui du tableau 61 p. 210. Veillez à minusculiser les mots, en utilisant la fonction SQL Server **LCASE**(<chaîne à transformer>) ou bien son équivalent Access : **MINUSCULE**(<chaîne à transformer>).

Exercice n°4 En utilisant Access et son interface (et non le mode SQL direct), formulez la requête paramétrée permettant de connaître les 10 occurrences les plus fréquentes d'une catégorie donnée de la table occurrences. La requête paramétrée demande à l'utilisateur la catégorie dont on veut les 10 occurrences les plus fréquentes.

6. Solutions

Solution de l'exercice n°1 p. 204 Le tableau 63 p. 212 montre le résultat. Il faut prendre garde à éliminer les occurrences correspondant aux espaces et obliques (*slash*) entre tirades au sein d'un vers et qui sont attribuées à ENTRE_PERSONNAGES, pour ne garder que les personnages effectifs. On ne compte par ailleurs que les occurrences qui ne commencent pas par E ou par S, c'est-à-dire qu'on élimine les espaces ou les séparateurs (cf. tableau 60 p. 209).

Tableau 61 – Phèdre : 10 premiers Adj, Adv, Nc, Np, V

Adj

Mot	*o.*
pu	19
vu	18
fait	16
funeste	14
juste	13
triste	13
coupable	11
vain	10
seule	10
odieux	10

Adv

Mot	*o.*
n’	104
plus	88
point	63
trop	44
si	43
-même	41
pas	39
déjà	23
jamais	21
peut-être	19

Nc

Mot	*o.*
yeux	53
fils	52
seigneur	51
dieux	45
père	42
coeur	41
sang	36
amour	35
madame	32
ciel	25

Np

Mot	*o.*
Hippolyte	39
Phèdre	36
Thésée	32
Aricie	17
Oenone	14
Athènes	12
Trézène	10
Neptune	9
il	8
Ismène	7

V

Mot	*o.*
est	140
a	112
ai	89
ont	36
suis	27
peut	22
sont	19
faut	19
aime	19
être	18

TABLEAU 62 – PHÈDRE : 10 premiers Conj, Dét/Pron, Rel, Prép

Conj

Mot	*o.*
et	301
mais	63
ou	16
jusqu'	11
donc	8
ni	6
lorsqu'	3
puisqu'	2
quoiqu'	1
car	1

Dét/Pron

Mot	*o.*
je	278
vous	236
le	235
un	225
l'	223
la	207
les	150
ne	136
j'	129
mon	120

Rel

Mot	*o.*
que	205
qu'	120
qui	95
si	35
où	29
dont	26
quand	19
pourquoi	14
quoi	13
comme	12

Prép

Mot	*o.*
de	353
à	209
d'	189
dans	90
pour	84
en	63
des	61
au	53
par	48
du	39

R^2_{79}

RESTRICTION(
personnage <> 'ENTRE_PERSONNAGES'
)[occurrences]
↪
REGROUPER SUR(personnage)[<résultat$_1$>]
↪
PAR GROUPE(
NOMBRE DE VALEURS DISTINCTES(numero_phrase) TITRE 'phrases',
SOMME(SI(cat NE RESSEMBLANT PAS À '^[ES]', 1, 0))
/ NOMBRE DE VALEURS DISTINCTES(numero_phrase) TITRE 'nombre moy. mots / phrase',
...
)[<résultat$_2$>]
↪
TRI SUR ('nombre moy. mots / phrase')[<résultat$_3$>

On constate des inégalités en longueur moyenne de phrase entre des personnages dont l'intervention dans la pièce est importante : Oenone, Phèdre et Thésée font des phrases plus courtes qu'Hippolyte et Théramène. Phèdre, au sein des personnages aux phrases les plus courtes, est responsable par contre de tirades en moyenne nettement plus longues.

MySQL

```
SELECT personnage,
    COUNT(DISTINCT numero_phrase) AS 'phrases',
    SUM(IF(cat NOT REGEXP '^[ES]', 1, 0)) /
```

TABLEAU 63 – PHÈDRE : personnages et nombre moyen de mots par phrase et tirade

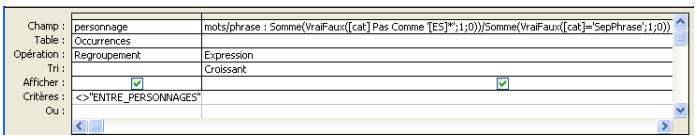
personnage	*phrases*	*nbre moy. mots / phrase*	*tirades*	*nbre moy. mots / tirade*
OENONE	147	12.04	47	37.66
PHEDRE	318	13.14	57	73.28
THESEE	150	13.15	35	56.34
ARICIE	80	14.68	22	53.36
HIPPOLYTE	187	16.07	42	71.57
THERAMENE	94	16.14	17	89.24
PANOPE	18	17.61	6	52.83
ISMENE	15	18.73	7	40.14

```
        COUNT(DISTINCT numero_phrase) AS 'nbre moy. mots / phrase',
    COUNT(DISTINCT numero_tirade) AS 'tirades',
    SUM(IF(cat NOT REGEXP '^[ES]', 1, 0)) /
        COUNT(DISTINCT numero_tirade) AS 'nbre moy. mots / tirade'
FROM occurrences
WHERE personnage <> 'ENTRE_PERSONNAGES'
GROUP BY personnage
ORDER BY 'nbre moy. mots / phrase' ;
```

Access

La solution diffère à dessein de celle pour MySQL sur deux points en 6. En premier lieu, l'élimination des occurrences attribuées à ENTRE_PERSONNAGES ne s'effectue pas en amont du regroupement, dans une clause **WHERE** mais en aval, dans une clause **HAVING**. En second lieu, sont décomptées explicitement les occurrences qui ont pour catégorie SepPhrase, c'est-à-dire séparateur de phrase, tandis que la requête MySQL utilise le nombre de valeurs distinctes pour le personnage de l'attribut numero_phrase.

6



Solution de l'exercice n°2 p. 208 Plusieurs solutions sont fournies, pour pousser à l'imagination lors de l'écriture de requêtes : il y a souvent plus d'une manière possible d'obtenir une table résultat donnée. Ce sont des variantes (sur la condition de restriction de deuxième niveau, en aval du regroupement) de la requête :

R^2_{80}

```
    REGROUPER SUR(cat)[occurrences]
    AVEC(<condition de restriction de 2e niveau>)
↪   PAR GROUPE(
    cat TITRE 'Catégorie',
    NOMBRE DE LIGNES() TITRE 'o'
    )[<résultat1>]
```

Les conditions successives de restriction de deuxième niveau, qui s'appuient sur les motifs possibles pour les catégories jugées pertinentes (tableau 60 p. 209), sont :

1. cat PAS COMME '^[ES].*'
2. cat PARMI ('Adj', 'Adv', 'Conj', 'Dét/Pron', 'Nc', 'Np', 'Prép', 'Rel', 'V')

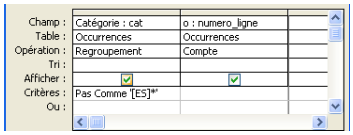
3. cat <> 'Espace' ET cat PAS COMME '^Sep.*'

△

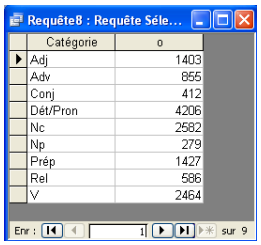
Les différences entre ces notations abstraites et celles qui sont fournies supra tiennent à des manières différentes d'exprimer les motifs des recherches approximatives.

Un premier motif en 7 élimine les occurrences dont la catégorie commence par E(space) ou S, c'est-à-dire toute la liste des séparateurs, comme le montre le résultat 8.

7

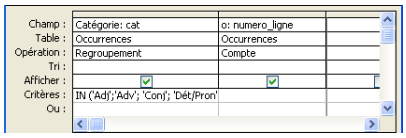


8



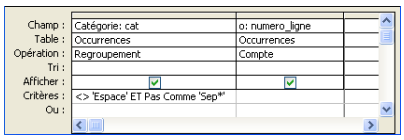
Une seconde requête 9, plus lourde à écrire, énumère explicitement tous les cas licites.

9



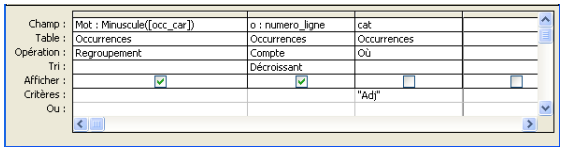
Une troisième requête 10 est la conjonction logique du refus d'une valeur donnée (Espace) et d'un motif négatif : la catégorie ne doit pas commencer par Sep, ce qui écarte tous les séparateurs.

10



Solution de l'exercice n°3 p. 208 Modulo le changement de la condition de restriction (le remplacement de Nc par Adj), c'est l'équivalent de la R^2_{78} p. 208. Si l'on veut obtenir les 10 premiers résultats, il faut en 11 mettre la valeur 10 dans la case permettant sous Access de contrôler le nombre ou le pourcentage de lignes produites (dans la barre de menu du haut) ou bien utiliser le menu ⌈Propriétés de la requête⌉ que l'on obtient avec le bouton droit sur la sous-fenêtre du haut de formulation des requêtes (cf. solution de l'exercice suivant).

11

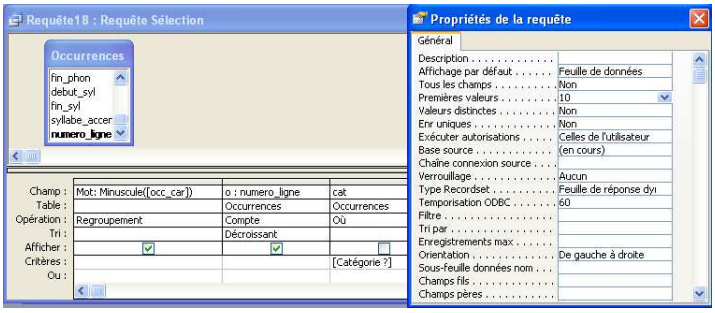


Cette manipulation faite, Access engendre la requête SQL Server :

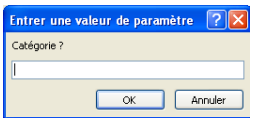
```
SELECT TOP 10
      LCase([occ_car]) AS Mot,
      Count(Occurrences.numero_ligne) AS o
FROM Occurrences
WHERE (((Occurrences.cat)="Adj"))
GROUP BY LCase([occ_car]), Occurrences.cat
ORDER BY Count(Occurrences.numero_ligne) DESC;
```

Solution de l'exercice n°4 p. 209 La requête figure en 12. Le regroupement est opéré sur la version minusculisée de l'attribut `occ_car` (première colonne). La restriction aux occurrences de la catégorie visée est effectuée par le paramétrage du critère (3^e colonne) : la mise entre crochets de ⌈ Catégorie ?⌉ signale que les valeurs de l'attribut `cat` seront à comparer à la valeur que fournira l'utilisateur lorsque le message entre crochets sera affiché 13.

12

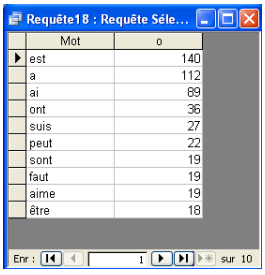


13



En 14, le résultat quand l'utilisateur a fourni V comme valeur cible.

14



CHAPITRE VI

RETOUR AUX BASES. . .

1. Une ligne de table = une entité unique

1.1. ⊕ Une ligne de table = une conjonction d'assertions sur une même entité

PRÉMA

Pour les infirmières 97 et 81, une écriture des caractéristiques des infirmières sous forme de conjonction de prédicats donnerait le résultat suivant :

$\exists x$ APourIdentifiant(x, 97)
$\bigwedge$ Âge(x, 25)
$\bigwedge$ AnnéesEtudes(x, 3)
$\bigwedge$ Diplôme(x, BEPC)
$\bigwedge$ AncienneténNéonatalité(x, 0.00)
$\bigwedge$ ServiceDe(x, Jour)

$\exists y$ APourIdentifiant(y, 81)
$\bigwedge$ Âge(y, 25)
$\bigwedge$ AnnéesEtudes(y, 3)
$\bigwedge$ Diplôme(y, BEPC)
$\bigwedge$ AncienneténNéonatalité(y, 0.00)
$\bigwedge$ ServiceDe(y, Jour)

Les infirmières 97 et 81 se différencient uniquement par leur identifiant. Elles présentent les mêmes valeurs pour toutes les autres propriétés.

1.2. ⊕ Clés primaires, secondaires, étrangères

Conventions typographiques Dans les ouvrages de présentation des bases de données, le nom de la clé primaire est souvent souligné dans les tables fournies en exemple. Nous avons plutôt recours au gras. C'est d'ailleurs également la convention retenue par Access. Le soulignement présente l'inconvénient de couper les jambages des lettres. Il réduit la lisibilité des chaînes de caractères ainsi distinguées.

PRÉMA

Table	*clé primaire*	*clé secondaire*	*clé étrangère*
bebes	id		
infirmieres	id		
fiches_originelles	id		id_bebe, id_infirmiere
occ_prema	numero_ordre		fiche
signaletique_fiches	id		id_bebe, id_infirmiere
fiches_...	id		id

La colonne id, un identifiant arbitraire, fait fonction de clé primaire dans toutes les tables, sauf dans occ_prema où la colonne numero_ordre joue ce rôle. Dans les tables fiches_originelles et signaletique_fiches (qui reprend toutes les colonnes de fiches_originelles, sauf texte), les colonnes id_bebe et id_infirmiere servent de clé étrangère avec les tables bebes et infirmieres respectivement. Dans les tables fiches_..., la colonne id est tout à la fois clé primaire et clé étrangère avec la table signaletique_fiches.

PHÈDRE

Table	*clé primaire*	*clé(s) secondaire(s)*	*clé étrangère*
vers	numero_vers	vers ou vers_phonetique	
positions	numero_ligne	numero_vers + position	numero_vers
occurrences	numero_ligne	numero_vers + debut_car	numero_vers

Pour les tables positions et occurrences, on utilise comme clé primaire un numéro d'ordre, numero_ligne. On pourrait pour la table positions utiliser la combinaison, toujours unique, du numéro de vers et de la position de la syllabe métrique dans le vers. Pour la table occurrences, la combinaison numero_vers, occ_car et debut_car identifie également une ligne unique à chaque fois. C'est donc également une clé secondaire. Il en irait de même des combinaisons de numero_vers, de la colonne occ_phon avec une indication de localisation (debut_car, fin_car, etc.).

Les colonnes numero_vers des tables positions et occurrences sont des clés étrangères : elles permettent une jointure avec la colonne numero_vers de la table vers.

Exercice n°1 Pour la syllabe 'il', de catégorie 'Dét/pron', fournir le nombre d'occurrences selon la position métrique.

Exercice n°2 Donner par personnage le nombre d'occurrences des mots contenant la chaîne 'monstr' en minuscules ou commençant par une capitale.

Fournir également les vers contenant 'monstr' avec, pour chaque vers, le personnage qui prononce le mot et en triant par personnage.

ESQUE

Table	*clé primaire*	*clé secondaire*	*clé étrangère*
esque	numero_ligne		

Les SGBD reposent sur l'**hypothèse du monde fermé**. Y figurent toutes les assertions qui définissent le domaine d'application. Dans cette optique, si l'on ne trouve pas dans la table esque le dérivé *xmlesque*, on en déduira que, sous réserve d'inventaire, ce dérivé n'existe pas.

On pourrait imaginer que les contextes sont tous distincts et que donc l'attribut contexte fournit une clé secondaire. La requête :

R^2_{81}
PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'Lignes',
NOMBRE DE VALEURS DISTINCTS(contexte) TITRE 'contextes distincts'
)[esque]

dont l'incarnation MySQL est :

```
SELECT COUNT(*) AS 'Lignes',
    COUNT(DISTINCT contexte) AS 'contextes distincts'
FROM esque_princeps ;
```

montre que ce n'est pas le cas :

Lignes	*contextes distincts*
4383	4012

On peut isoler les contextes qui interviennent plus d'une fois par la requête :

R^2_{82}
REGROUPER SUR(contexte)[esque]
AVEC(NOMBRE DE LIGNES() > 1)
↪ PAR GROUPE(
contexte,
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_1$>]

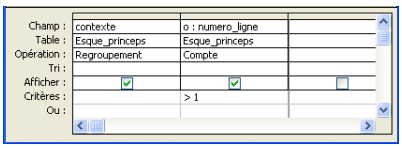
La réalisation MySQL est :

```
SELECT contexte,
    COUNT(*) AS 'o.'
FROM esque_princeps
GROUP BY contexte
    HAVING COUNT(*) > 1 ;
```

TABLEAU 64 – Esque : contextes doublons

Contexte	*o.*
NULL	362
Avez-vous remarqué la prolifération des restaurants rapides dans les rues de notre belle cité ? [...] Le mal est insidieux. Il ne s'agit pas là d'alerter vos mornes estomacs contre une indigestion graisseuse, une asphyxie <graillonnesque>. Bien pire : Paris commence à souffrir du «syndrome fast-food».	2
C'est en Belgique que le mouvement surréaliste officiel a été le plus chouette. Parce qu'alors qu'en France et dans d'autres pays il était très stalinisé, il était inféodé au parti, en Belgique, c'était le cas aussi, mais c'était avec une marge de manoeuvre très grande et très farfelue par rapport à l'orthodoxie <André Bretonesque>.	2
Dîner <gionesque> [...]	2
Il faudra souvent, cependant, leur signaler ce fait par courrier <'escargoesque'> (snail mail), c'est à dire par le courrier postal classique, considéré comme ultra-lent par rapport au courrier électronique où l'on peut pratiquement discuter en direct.	2
Le parler que j'ayme, c'est un parler simple et naïf [...], non pédantesque, non fratesque, non <pleideresque>, mais plutôt soldatesque, comme Suetone appelle celuy de Julius Caesar [...]	2
Les gardiens de prison poursuivent leur mouvement de grève pour réclamer des embauches, des augmentations de salaires et une amélioration globale de leur condition de travail. La F.A. tient évidemment à assurer de son total soutien... les prisonniers encore une fois pris en otage par la gente <matonesque>.	2
Les lieux : Agrégat pouilleux juché aux arbres tels des cages à poules haut-perchées, le village <elfesque> donne l'impression d'une ménagerie malsaine où des Elfes à long bras, voyageant de branche en branche tels des chimpanzés difformes, surplombent une nuée d'esclaves avilis oeuvrant avec peine sur des fermes puantes.	2
On aurait également pu arrêter notre choix sur la couleur orange, une couleur joyeuse, chaleureuse et invitante qui irait quand même assez bien avec le comportement et la personnalité de notre animateur, journaliste et fier représentant vedette Garry Björnsön, mais qui irait plutôt mal avec le fait que l'on ne produit pas notre émission <«bingoesque»> le matin ou le midi, où le soleil brille de tous feux [...]	2
Puis son webmaster, dont nous tairons le nom de peur de nous attirer les foudres de Belial (he he he...) s'est révélé être l'être le plus <tachesque> de la terre.	2

et son équivalent Access :



1

Le tableau 64 p. 218 montre qu'au delà des 362 lignes qui ont **NULL** pour contexte et qui correspondent probablement à des dérivés issus du dépouillement de dictionnaires, il existe 9 contextes en 2 exemplaires chacun. Il resterait à vérifier si les doublons en question le sont également pour les autres attributs.

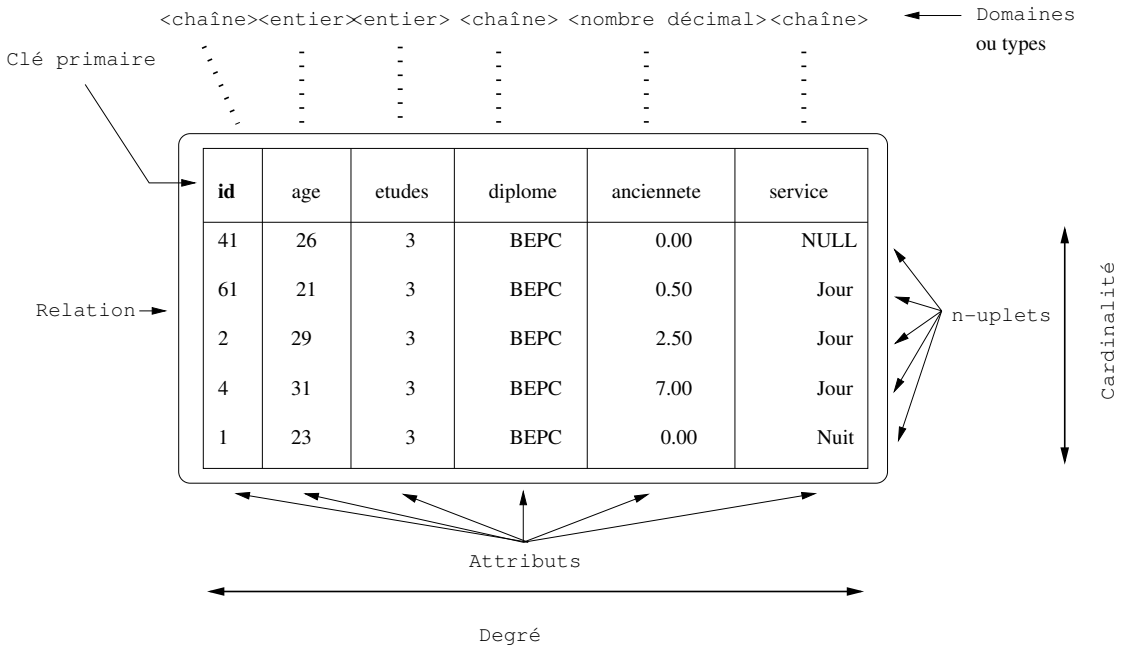


FIGURE 4 – PRÉMA : la terminologie relationnelle à l'œuvre

2. Une table = une relation

PRÉMA

L'ordre de présentation des lignes et des colonnes de la table infirmieres ne change pas « l'état du monde » représenté par cette table, les faits qui y sont notés. Il en va de même pour la table bebes.

Par contre, les fiches sont ordonnées dans le temps, pour un bébé donné, d'où l'attribut jour, et globalement, la datation est alors reconstituable via les dates de naissance des bébés. Au sein d'une fiche, les occurrences et les informations attenantes de la table occ_prema se suivent.

La figure 4, adaptée de Date (2000, p. 110), « décore » la table concernant les infirmières de la terminologie relationnelle pertinente.

PHÈDRE

À la différence de la base ESQUE, pour PHÈDRE, les réalités à modéliser – les vers, les positions métriques, les « mots » – sont séquentielles. Il est donc nécessaire de disposer d'attributs permettant de situer la place, dans le déroulement de la pièce, de chaque instance de vers, de position métrique, de mot, par rapport aux autres instances du même type. C'est le rôle de numero_vers dans la table vers, de numero_ligne ou de combinaisons d'attributs dans les tables positions (numero_vers + position) et occurrences (numero_vers + debut_car).

ESQUE

Pour esque, l'ordre dans lequel sont fournies les attestations n'a pas d'importance.

3. Les facettes d'une entité

3.1. ⊕ Les types disponibles varient selon les SGBD

Certains types sont présents dans certains SGBD et pas dans d'autres. C'est le cas par exemple du type booléen, qui attend les valeurs vrai ou faux (ou bien oui ou non). Ce type existe sous Access sous le nom Oui/Non mais il doit être « mimé » sous MySQL par exemple par un attribut numérique à valeur entière prenant comme valeur 1 (pour vrai) et 0 (pour faux). C'est ainsi que sont utilisés les attributs accent et fin_mot dans la table positions. Access met à disposition un type permettant de renvoyer à des « objets » informatiques comme des fichiers-sons ou des fichiers-images. À l'inverse, MySQL fournit une palette plus riche qu'Access pour les données temporelles. Access distingue les chaînes courtes (de 255 caractères au plus) et les documents « longs » (champs Mémo, jusqu'à 64 000 caractères). MySQL offre un éventail beaucoup plus large et qui permet de mémoriser des volumes beaucoup plus importants.

On se reportera au ch. XV § 3 pour le détail des types de données offerts par MySQL et Access pour les attributs des tables.

On retiendra de ces divergences « dialectales » un principe de précaution : dans la mesure du possible, choisir, pour une base de données réalisées avec un SGBD déterminé, les types facilitant les transferts vers d'autres SGBD, en particulier ceux qui relèvent de la norme SQL. △

3.2. ⊕ Cacher la réalisation *vs.* rester économe

Les SGBD ont pour objectif explicite global de « cacher », d'épargner à l'utilisateur le détail de la réalisation du stockage et de l'interrogation des informations présentes dans une base de données. Le choix des types des attributs d'une relation est un des endroits où cet objectif se trouve partiellement battu en brèche. Justement parce que les SGBD sont conçus pour pouvoir stocker des tables sans limite de taille *a priori*, ils proposent à l'utilisateur, pour un même domaine général (valeurs textuelles, numériques, temporelles), plusieurs types spécifiques, de taille croissante, en fonction de la précision des informations souhaitée. À l'utilisateur d'utiliser à bon escient cet éventail pour être aussi « économe » que possible. Par exemple, dans la table occurrences, la colonne type_liaison est une chaîne de 2 caractères, parce qu'on sait que les différentes modalités de cette variable ne dépasseront pas ce cadre (tableau 56 p. 205). △

Avant de diminuer la taille choisie pour un attribut, il importe donc de calculer la taille maximale des valeurs contenues (cf. exercice suivant) et d'estimer la taille maximale escomptable.

Exercice n°3 Calculer la longueur maximale des valeurs des attributs occ_phon et occ_car de la table occurrences de PHÈDRE. Utiliser pour la version MySQL la fonction **LENGTH**(<chaîne>), pour la version SQL Server, la fonction **LEN**(<chaîne>) et pour la version Access la fonction **NbCar**(<chaîne>).

PRÉMA

Le domaine d'un attribut contraint les opérations qui peuvent être effectuées sur lui, opérations détaillées aux chapitre III et IV. Par exemple, à partir de l'attribut age, il est possible de calculer l'âge moyen des infirmières, ou encore l'âge minimal et l'âge maximal des infirmières représentées dans la table. Par contre, il n'y aurait pas de sens à vouloir appliquer à l'âge des infirmières la fonction LONGUEUR, qui, appliquée à une chaîne, en retourne le nombre de caractères.

MySQL

MySQL dispose de deux fonctions qui permettent de transformer, quand c'est possible, le type d'une valeur en un autre type :

CAST(<valeur> **AS** <type souhaité>)

CONVERT(<valeur>, <type souhaité>)

Le tableau 65 p. 222 est le tri de la table infirmieres par ordre croissant des identifiants, dont on a choisi, dans la base fournie, de les représenter comme des nombres entiers, pour accélerer les comparaisons et jointures. Il résulte de la requête :

```
SELECT *
FROM infirmieres_princeps
ORDER BY id ;
```

Le tableau 66 p. 223 est le tri de la même table, mais cette fois-ci par ordre croissant des chaînes de caractères correspondant aux identifiants. L'instruction de tri est transformée en :

```
ORDER BY CAST(id AS CHAR) ;
```

ou en :

```
ORDER BY CONVERT(id , CHAR) ;
```

On le constate, alors que le 2 suit directement le 1 quand il s'agit de nombres, s'interposent entre eux 10, 11. . . 19 quand il s'agit de chaînes de caractères. Il faut donc distinguer clairement l'entier 41 et la chaîne de caractères "41". Bien que leur visualisation soit identique, il s'agit de deux objets différents, relevant d'opérations distinctes. △

Dans la base de données PRÉMA, on a choisi de représenter par un nombre entier les identifiants des infirmières, par souci d'efficacité. Pour autant, ces nombres ne représentent pas des quantités. Il n'y aurait pas de sens à vouloir calculer l'« identifiant moyen » des infirmières.

PHÈDRE

ESQUE

4. Solutions

Solution de l'exercice n°1 p. 216 On vérifie que la catégorie de la syllabe est Dét/Pron pour ne pas inclure des fragments de mots (des syllabes) ayant pour prononciation 'i l'.

R^2_{83}

RESTRICTION(
syll = 'i l'
ET cat = 'Dét/Pron'
)[phedre]
↪ REGROUPER SUR(position)[<résultat$_1$>]
↪
PAR GROUPE(
position TITRE 'Position',
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_2$>]

La table résultat se trouve au tableau 67 p. 223.

TABLEAU 65 – PRÉMA : table infirmieres – tri par identifiant (numérique)

id	*age*	*etudes*	*diplome*	*anciennete*	*service*
1	23	4	BEPC	0.00	Nuit
2	29	3	BEPC	2.50	Jour
3	34	3	BEPC	5.00	Nuit
4	31	3	BEPC	7.00	Jour
6	30	4	BEPC	7.00	Jour
7	39	4	BEPC	19.00	Nuit
8	44	3	BEPC	19.00	Jour
9	27	3	BEPC	3.00	Jour
10	35	3	BEPC	11.00	Nuit
12	40	3	BEPC	20.00	Nuit
13	34	3	BEPC	3.00	Jour
14	40	3	BEPC	14.00	Nuit
16	30	4	BEPC	6.50	Nuit
17	38	4	BAC	14.00	Jour
18	27	4	BEPC	2.50	Jour
19	28	3	BEPC	5.00	Jour
20	28	3	BEPC	6.00	Nuit
21	31	4	BEPC	8.00	Jour
22	31	3	BAC	9.00	Jour
24	26	4	BEPC	0.00	Jour
28	38	3	BEPC	17.00	Nuit
32	26	3	BEPC	0.00	Jour
33	34	4	BAC	0.00	Jour
34	31	4	BEPC	0.00	Jour
36	26	4	BEPC	2.00	Jour
39	33	4	BEPC	3.00	Jour
41	26	3	BEPC	0.00	**NULL**
43	26	4	BEPC	2.00	Jour
44	29	3	BEPC	2.00	Nuit
46	21	3	BEPC	0.00	Jour
47	30	3	BEPC	3.00	Nuit
58	31	3	BEPC	8.00	Nuit
61	26	4	BEPC	3.00	Jour
62	33	3	BEPC	1.00	Jour
65	21	3	BEPC	0.50	Jour
67	25	4	BEPC	0.00	Jour
68	24	4	BEPC	0.00	Jour
70	29	3	BEPC	0.00	Jour
73	29	3	BAC	1.50	Jour
81	25	3	BEPC	0.00	Jour
97	25	3	BEPC	0.00	Jour
99	31	4	BEPC	8.50	Jour

TABLEAU 66 – PRÉMA : table infirmieres – tri par identifiant (caractères)

id	*age*	*etudes*	*diplome*	*anciennete*	*service*
1	23	4	BEPC	0.00	Nuit
10	35	3	BEPC	11.00	Nuit
12	40	3	BEPC	20.00	Nuit
13	34	3	BEPC	3.00	Jour
14	40	3	BEPC	14.00	Nuit
16	30	4	BEPC	6.50	Nuit
17	38	4	BAC	14.00	Jour
18	27	4	BEPC	2.50	Jour
19	28	3	BEPC	5.00	Jour
2	29	3	BEPC	2.50	Jour
20	28	3	BEPC	6.00	Nuit
21	31	4	BEPC	8.00	Jour
22	31	3	BAC	9.00	Jour
24	26	4	BEPC	0.00	Jour
28	38	3	BEPC	17.00	Nuit
3	34	3	BEPC	5.00	Nuit
32	26	3	BEPC	0.00	Jour
33	34	4	BAC	0.00	Jour
34	31	4	BEPC	0.00	Jour
36	26	4	BEPC	2.00	Jour
39	33	4	BEPC	3.00	Jour
4	31	3	BEPC	7.00	Jour
41	26	3	BEPC	0.00	**NULL**
43	26	4	BEPC	2.00	Jour
44	29	3	BEPC	2.00	Nuit
46	21	3	BEPC	0.00	Jour
47	30	3	BEPC	3.00	Nuit
58	31	3	BEPC	8.00	Nuit
6	30	4	BEPC	7.00	Jour
61	26	4	BEPC	3.00	Jour
62	33	3	BEPC	1.00	Jour
65	21	3	BEPC	0.50	Jour
67	25	4	BEPC	0.00	Jour
68	24	4	BEPC	0.00	Jour
7	39	4	BEPC	19.00	Nuit
70	29	3	BEPC	0.00	Jour
73	29	3	BAC	1.50	Jour
8	44	3	BEPC	19.00	Jour
81	25	3	BEPC	0.00	Jour
9	27	3	BEPC	3.00	Jour
97	25	3	BEPC	0.00	Jour
99	31	4	BEPC	8.50	Jour

TABLEAU 67 – PHÈDRE : positions du pronom *il*

Position	*o.*
1	45
2	2
3	9
4	6
5	1
7	18
9	1

MySQL

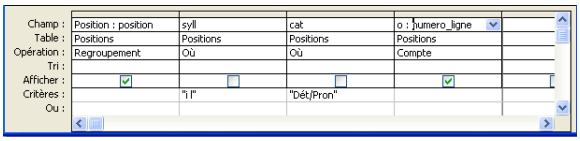
La requête est de la forme :

```
SELECT position AS 'Position',
    COUNT(*) AS 'o.'
FROM positions
WHERE syll = 'i␣l'
    AND cat = 'Dét/Pron'
GROUP BY position ;
```

Access

La requête figure en 2 .

2



Solution de l'exercice n°2 p. 216 Le tableau 68 p. 225 fournit les décomptes par personnages. Il est issu de la requête :

R^2_{84}

RESTRICTION(
occ_car RESSEMBLANT À '[Mm]onstr'
)[occurrences]
↪ REGROUPER SUR(personnage)[<résultat$_1$>]
↪
PAR GROUPE(
personnage TITRE 'Personnage',
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_2$>]

On trouve dans le tableau 69 p. 225 les vers contenant le lemme *monstre* par personnage. On perçoit immédiatement les différences d'emploi (littéral *vs.* figuré), en particulier de Phèdre par rapport aux autres personnages. La requête correspondante s'énonce ainsi :

R^2_{85}

JOINTURE(
o.numero_car = v.numero_vers
)[occurrences ALIAS o, vers ALIAS v]
↪ RESTRICTION(
occ_car RESSEMBLANT À '[Mm]onstr'
)[<résultat$_1$>]
↪
PROJECTION
o.numero_vers,
personnage,
vers
)[<résultat$_2$>]
↪
TRI SUR(personnage)[<résultat$_3$>]

TABLEAU 68 – PHÈDRE : *monstr* occurrences par personnage

Personnage	*o.*
ARICIE	1
HIPPOLYTE	5
PHEDRE	5
THERAMENE	4
THESEE	3

TABLEAU 69 – PHÈDRE : *monstr* par personnage

numero_vers	*personnage*	*vers*
1444	ARICIE	Ont de monstres sans nombre affranchi les humains ;
79	HIPPOLYTE	Les monstres étouffés et les brigands punis,
99	HIPPOLYTE	Qu'aucuns monstres par moi domptés jusqu'aujourd'hui
520	HIPPOLYTE	Croit-on que dans ses flancs un monstre m'ait porté ?
938	HIPPOLYTE	Déjà plus d'un tyran, plus d'un monstre farouche
948	HIPPOLYTE	Souffrez, si quelque monstre a pu vous échapper,
649	PHEDRE	Par vous aurait péri le monstre de la Crète,
701	PHEDRE	Délivre l'univers d'un monstre qui t'irrite.
703	PHEDRE	Crois-moi, ce monstre affreux ne doit point t'échapper.
884	PHEDRE	Je le vois comme un monstre effroyable à mes yeux.
1317	PHEDRE	Je ne t'écoute plus. va-t'en, monstre exécrable :
1516	THERAMENE	Parmi des flots d'écume, un monstre furieux.
1522	THERAMENE	Le ciel avec horreur voit ce monstre sauvage ;
1529	THERAMENE	Pousse au monstre, et d'un dard lancé d'une main sûre,
1531	THERAMENE	De rage et de douleur le monstre bondissant
963	THESEE	Livré par ce barbare à des monstres cruels
970	THESEE	A ses monstres lui-même a servi de pâture ;
1045	THESEE	Monstre, qu'a trop longtemps épargné le tonnerre,

MySQL

La requête correspondant au tableau 68 p. 225 est la suivante :

```
SELECT personnage AS 'Personnage',
       COUNT(*) AS 'o.'
FROM occurrences
WHERE occ_car REGEXP '[Mm]onstr'
GROUP BY personnage ;
```

La requête aboutissant au tableau 69 p. 225 est la suivante :

```
SELECT o.numero_vers,
       personnage,
       vers
FROM occurrences AS o,
       vers AS v
WHERE o.numero_vers = v.numero_vers
      AND occ_car REGEXP '[Mm]onstr'
ORDER BY personnage ;
```

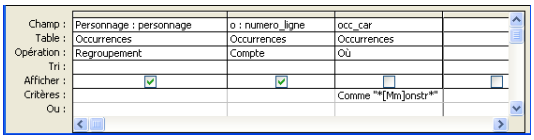
On peut d'ailleurs obtenir le même résultat directement à partir de la table vers, sans jointure :

```
SELECT numero_vers,
       personnage_s,
       vers
FROM vers
WHERE vers REGEXP '[Mm]onstr'
ORDER BY personnage_s ;
```

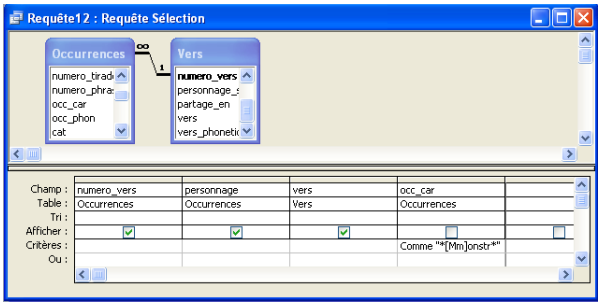
Access

On trouve en 3 , la requête source du tableau 68 p. 225, en 4 , celle du tableau 69 p. 225.

3



4



Solution de l'exercice n°3 p. 220

R^2_{86}

PAR GROUPE(
MAXIMUM(LONGUEUR(occ_phon)) TITRE 'Max. occ. phon.',
MAXIMUM(LONGUEUR(occ_car)) TITRE 'Max. occ. car.
)[occurrences]

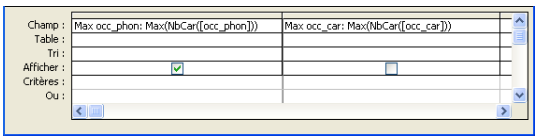
MySQL

```
SELECT MAX(LENGTH(occ_phon)) AS 'Max. occ. phon.',
       MAX(LENGTH(occ_car)) AS 'Max. occ. car.'
FROM occurrences ;
```

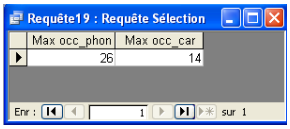
Access

Le résultat de 5 figure en 6.

5



6



La requête SQL Server engendrée est :

```
SELECT Max(Len([occ_phon])) AS [Max occ_phon],
       Max(Len([occ_car])) AS [Max occ_car]
FROM Occurrences;
```

CHAPITRE VII

METTRE EN RELATION LES INFORMATIONS

1. Combiner les informations : le produit relationnel

1.1. ⊕ Préparer les tables à mettre en relation

Le produit relationnel est une opération n-aire, c'est-à-dire que cette opération peut prendre n arguments. Il est de la forme :

PRODUIT[<table$_1$>, <table$_2$>... <table$_n$>]

Dans les présentations des SGBD, on trouve souvent la dénomination **produit cartésien**. △
Le produit cartésien de deux ensembles est un ensemble de couples d'éléments du premier et du deuxième ensemble, un **couple** étant ordonné, à la différence d'une **paire**. La dénomination « produit cartésien » n'est pas entièrement appropriée : le résultat du produit relationnel n'est pas vraiment un ensemble de couples de lignes. En effet l'ordre des colonnes n'est pas significatif dans les SGBD, même si la réalisation concrète du produit relationnel place les colonnes dans un certain ordre : les colonnes de la première table entrant dans le produit sont suivies par celles de la deuxième table, et ainsi de suite.

MySQL

On crée les tables actes et personnages et on les peuple via des projections ou restrictions sur la table vers.

```
CREATE TABLE actes (
      acte VARCHAR(5) BINARY NOT NULL DEFAULT "",
      PRIMARY KEY (acte)) ;
```

```
INSERT INTO actes
      SELECT DISTINCT acte
      FROM vers ;

CREATE TABLE personnages (
      personnage VARCHAR(100) NOT NULL DEFAULT "",
      PRIMARY KEY (personnage)) ;

INSERT INTO personnages
      SELECT DISTINCT personnage_s
      FROM vers
      WHERE partage_en = 1 ;
```

La clause [**WHERE** partage_en = 1] est nécessaire pour conserver uniquement les mentions d'un personnage seul et non les associations de personnages. L'ajout de **DISTINCT** dans les clauses **SELECT** aboutit à conserver un seul exemplaire des valeurs de la colonne visée. C'est donc une projection et non une projection avec doublons qui est mise à contribution.

La dernière requête pourrait se baser sur la table occurrences et non sur la table vers, à condition d'éliminer la valeur ENTRE_PERSONNAGES :

```
INSERT INTO personnages
      SELECT DISTINCT personnage
      FROM occurrences
      WHERE personnage <> 'ENTRE_PERSONNAGES' ;
```

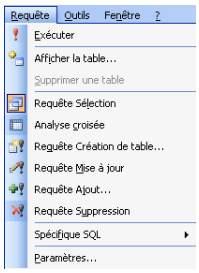
Access

Requête création de table La requête 1 porte sur la table vers. Elle combine restriction et projection. Elle retient en effet les lignes pour lesquelles l'attribut partage_en a pour valeur 1, c'est-à-dire les vers où n'intervient qu'un seul personnage : c'est le rôle du critère. Elle ne garde alors que les valeurs distinctes de l'attribut personnage_s. Les valeurs distinctes sont obtenues en faisant apparaître le formulaire [Propriétés de la requête], via un clic droit dans la fenêtre du haut de la requête, et en positionnant à [oui] la ligne [Valeurs distinctes].

1

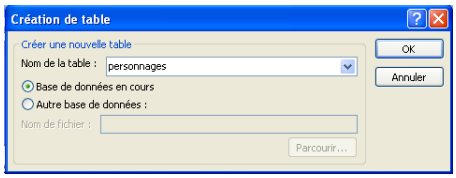
Le choix de l'onglet [Requête] 2 de la barre du haut permet de sélectionner [Requête Création de table].

2

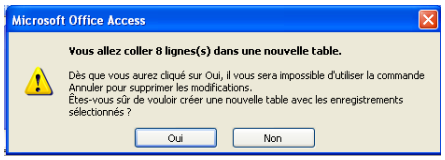


S'ensuit la demande du nom de la nouvelle table qui va être créée et dans laquelle les lignes issues de la requête iront se placer 3 ainsi qu'un message d'avertissement 4.

3

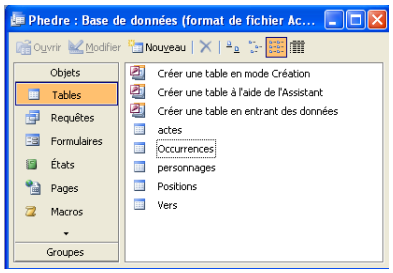


4



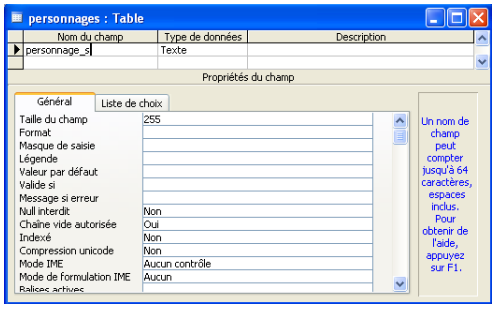
On note en 5 l'apparition effective de la table personnages dans le volet [Tables] de la fenêtre de navigation dans la base.

5



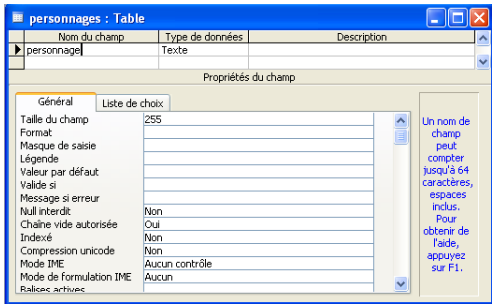
Lorsqu'on examine la structure de cette table (bouton [Modifier]de la fenêtre de navigation), on constate que le nom de la colonne unique est personnage_s 6.

6



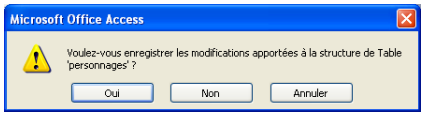
Par souci de cohérence, on enlève le souligné et le s à la fin 7.

7



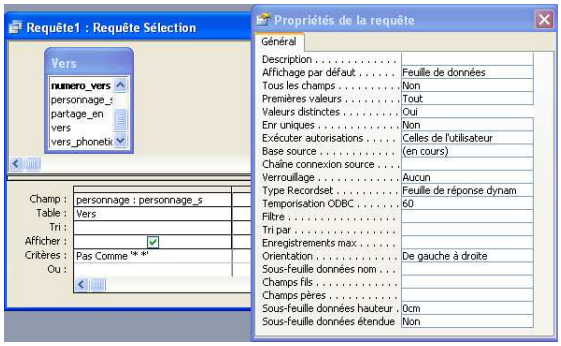
La modification doit être confirmée lorsqu'on ferme cette fenêtre 8.

8



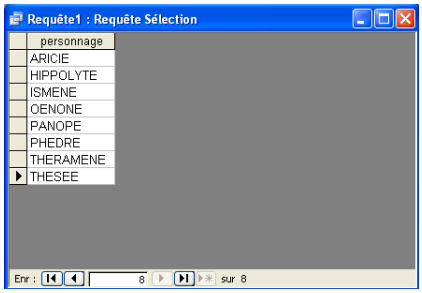
Une autre manière pour créer la table personnages consiste à donner directement à la colonne résultant le titre souhaité 9. Pour diversifier les angles d'attaque, cette variante repose par ailleurs sur une restriction différente : on retient les lignes dont l'attribut personnage_s ne contient pas de blanc (d'espace). C'est le rôle du motif dans 9.

9



La feuille de données 10 qui en est issue montre qu'on obtient bien le résultat souhaité.

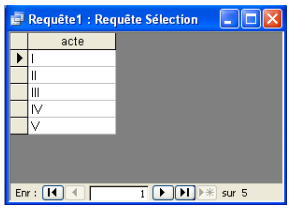
10



On procède de manière similaire en 11 pour obtenir une table actes 12.

11

12



1.2. ⊕ Obtenir le produit relationnel

≃ produit relationnel des tables actes et personnages

R^{1}_{57} *t. 1* p. 120

PRODUIT[actes, personnages]

MySQL

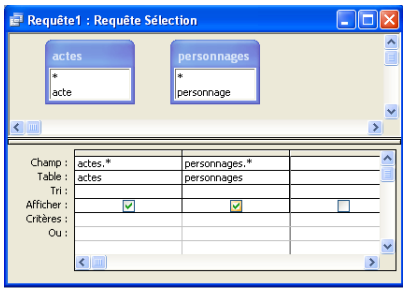
```
SELECT *
FROM actes,
     personnages ;
```

Le produit relationnel s'obtient en listant simplement dans la clause **FROM** les tables à croiser. L'étoile dans la clause **SELECT** entraîne le listage dans l'ordre des colonnes des tables croisées.

Access

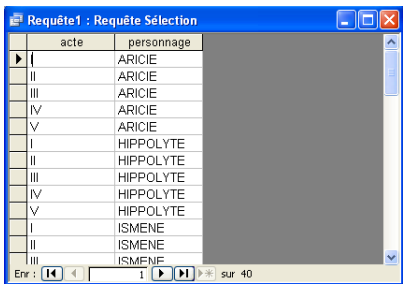
Le produit relationnel est une requête ordinaire 13. Les deux tables actes et personnages sont ajoutées dans la fenêtre du haut. La fenêtre du bas montre que l'on souhaite les attributs de chacune des deux tables.

13



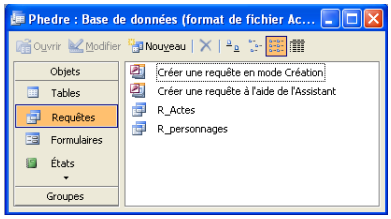
La feuille de données 14 donne un aperçu des 40 lignes du résultat.

14



Une requête = une table Le produit relationnel peut être obtenu en combinant les lignes de deux tables crées « à la volée », comme on vient de le faire. Il est également possible d'utiliser des requêtes en lieu et place de tables. Cette fois, on mémorise les requêtes 15 et non plus les tables issues des requêtes.

15

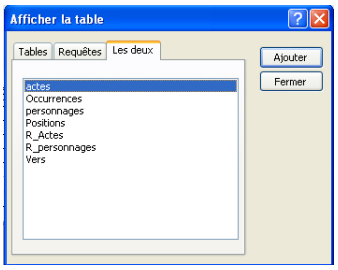


Lorsqu'on veut obtenir le produit, au lieu d'ajouter des tables à la requête correspondante, on ajoute les requêtes R_Actes et R_Personnages qui ont été mémorisées 16, en utilisant l'onglet [Requêtes].

16

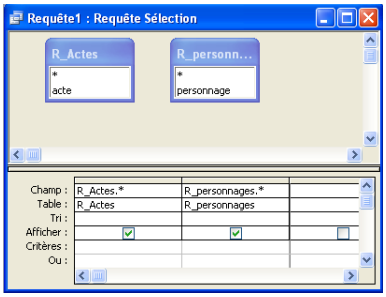
L'onglet [les deux] 17 montre bien que pour Access requêtes et tables sont sur le même plan, que les requêtes sont des tables « à la demande ».

17



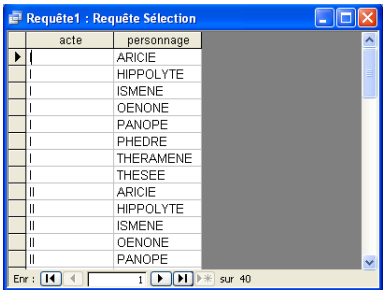
Une fois ces deux requêtes mises en entrée, le produit se formule de la même manière que précédemment 18.

18



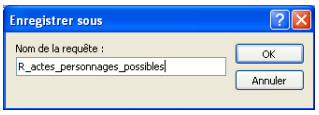
Le résultat est identique 19.

19



Pour rester dans cette logique de mémorisation de traitements plus que de résultats, la requête calculant le produit relationnel est elle aussi mémorisée en tant que telle 20.

20



Combinaisons réalisées actes × personnages

R_{58}^{1} *t. 1* p. 120

↪ RESTRICTION(partage_en = 1)[vers]

PROJECTION(acte, personnage_s)[<résultat$_1$>]

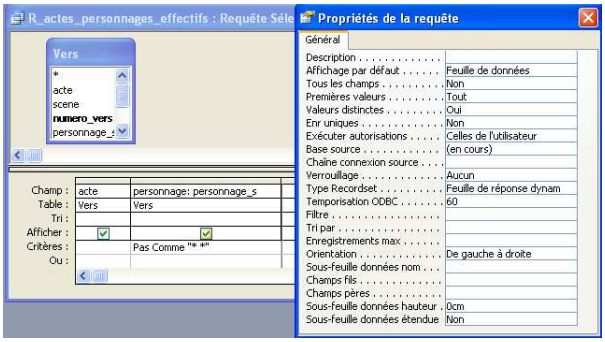
MySQL

```
SELECT DISTINCT acte,
       personnage_s
FROM vers
WHERE partage_en = 1 ;
```

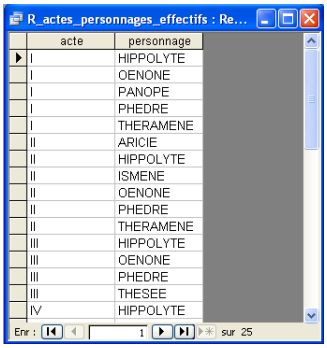
Access

La requête mémorisée R_actes_personnages_effectifs 21, quand on la « rejoue » en double-cliquant sur son nom dans le volet [Requêtes] de la fenêtre de navigation dans la base, fournit les 25 combinaisons de numéro d'acte et de nom de personnage 22. Cette requête calcule toutes les combinaisons distinctes de ces deux colonnes, avec une restriction qui constitue une variante par rapport à la requête MySQL qui précède : la valeur de l'attribut personnage_s ne doit pas comprendre de blanc (d'espace).

21



22



PRÉMA

Dans le tableau 70, p. 236, les deux tables d'entrée, ventilations (a) et sedations (b) contiennent respectivement toutes les valeurs observées dans la table signaletique_fiches pour l'attribut ventilation et celles de l'attribut sedation. La table (c) est le résultat du produit relationnel des tables ventilations et sedations. Elle résulte de la requête :

R^2_{87} PRODUIT[ventilations, sedations]

La table (c) contient 16 lignes (les 4 lignes de ventilations $\times$ les 4 lignes de sedations).

TABLEAU 70 – PRÉMA : produit relationnel ventilations × sedations

(a) `ventilations`

ventilation
intubé
non reponse
sonde nasale
non

(b) `sedations`

sedation
non
hypnovel ou fentanyl
canadou
non reponse

(c) = `ventilations` × `sedations`

ventilation	*sedation*
intubé	non
non reponse	non
sonde nasale	non
non	non
intubé	hypnovel ou fentanyl
non reponse	hypnovel ou fentanyl
sonde nasale	hypnovel ou fentanyl
non	hypnovel ou fentanyl
intubé	canadou
non reponse	canadou
sonde nasale	canadou
non	canadou
intubé	non reponse
non reponse	non reponse
sonde nasale	non reponse
non	non reponse

Les 3 requêtes qui suivent retournent respectivement le nombre d'occurrences de chacune des valeurs possibles des attributs ventilation et sedation dans la table signaletique_fiches, ainsi que le nombre d'occurrences de chacune des combinaisons réalisées (11 sur les 16 potentielles) :

R^2_{88}

```
REGROUPER SUR(ventilation)[signaletique_fiches]
↪ PAR GROUPE(
ventilation,
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat1>]
```

R^2_{89}

```
REGROUPER SUR(sedation)[signaletique_fiches]
↪ PAR GROUPE(
sedation,
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat1>]
```

R^2_{90}

```
REGROUPER SUR(
ventilation, sedation
)[signaletique_fiches]
↪ PAR GROUPE(
ventilation,
sedation
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat1>]
```

TABLEAU 71 – PRÉMA : combinaisons réalisées ventilation × sédation

(a) R^2_{88} p. 236

ventilation	*o.*
intubé	439
non	305
non reponse	1
sonde nasale	272

(b) R^2_{89} p. 236

sedation	*o.*
canadou	11
hypnovel ou fentanyl	77
non	926
non reponse	3

(c) R^2_{90} p. 236

ventilation	*sedation*	*o.*
intubé	hypnovel ou fentanyl	73
intubé	non	366
non	canadou	8
non	hypnovel ou fentanyl	2
non	non	293
non	non reponse	2
non reponse	non	1
sonde nasale	canadou	3
sonde nasale	hypnovel ou fentanyl	2
sonde nasale	non	266
sonde nasale	non reponse	1

On peut alors comparer les combinaisons possibles (tableau 70 (c), p. 236) aux combinaisons réalisées (tableau 71 (c), p. 237). Dans le tableau 70 (c), p. 236, sont grisées les combinaisons non réalisées.

MySQL

La réalisation MySQL de la R^2_{87} p. 235 est :

```
SELECT *
FROM ventilations,
     sedations ;
```

R^2_{88} p. 236

```
SELECT ventilation,
     COUNT(*) AS 'o.'
FROM signaletique_fiches
GROUP BY ventilation ;
```

R^2_{89} p. 236

```
SELECT sedation,
     COUNT(*) AS 'o.'
FROM signaletique_fiches
GROUP BY sedation ;
```

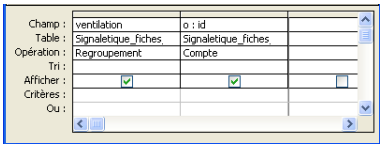
R^2_{90} p. 236

```
SELECT ventilation,
     sedation,
     COUNT(*) AS 'o.'
FROM signaletique_fiches
GROUP BY ventilation, sedation ;
```

Access

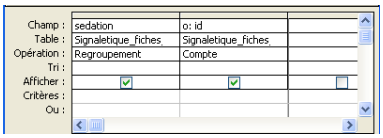
R^2_{88} p. 236

23



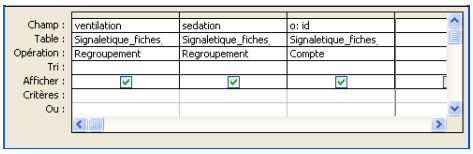
R^2_{89} p. 236

24



R^2_{90} p. 236

25



PHÈDRE

ESQUE

2. Combiner et restreindre : la jointure

2.1. ⊕ Jointures à la demande (MySQL) *vs.* « stables » (Access)

R^1_{59} *t. 1* p. 123

JOINTURE(
vers.numero_vers = occurrences.numero_vers
)[vers, occurrences)

R^1_{60} *t. 1* p. 123

JOINTURE(
v.numero_vers = o.numero_vers
)[vers ALIAS v, occurrences ALIAS o)

R^1_{61} *t. 1* p. 123

JOINTURE NATURELLE[vers, occurrences)

En MySQL, la jointure entre deux ou n tables n'est pas attachée aux tables ainsi mises en relation. Rien ne dit lorsqu'on examine la structure d'une table qu'elle entre régulièrement dans des jointures avec telle ou telle autre table. On peut néanmoins déclarer qu'un attribut joue le rôle de clé étrangère dans une table et le nom de la table « mère ».

Sous Access, par contraste, on « pose » une fois pour toutes les jointures qui définissent la base de données. Elles sont toujours présentes (on peut néanmoins les supprimer - cf. infra). On peut cependant rajouter dans une requête une condition de rapprochement entre tables qui est valide uniquement pendant la durée de cette requête. C'est le cas par exemple dans les auto-jointures.

MySQL

La R^1_{59} *t. 1* p. 123 se formule de la manière suivante :

```
SELECT *
FROM vers,
     occurrences
WHERE vers.numero_vers = occurrences.numero_vers ;
```

Noter le recours à un nom qualifié : le nom d'attribut numero_vers seul est ambigu, puisqu'il peut renvoyer à une colonne de la table vers comme à une colonne de la table occurrences. Préfixé du nom de la table concernée, il est univoque. Dans l'équivalent de la R^1_{60} *t. 1* p. 123, par souci de concision, le nom complet de chaque table est remplacé par une abréviation :

```
SELECT *
FROM vers AS v,
     occurrences AS o
WHERE v.numero_vers = o.numero_vers ;
```

Dans tous les cas, l'alias de table est introduit par **AS**, comme les titres de colonnes dans les tables-résultats. Cet **AS** est optionnel, mais nous l'utilisons systématiquement, par souci de lisibilité.

Ce type de jointure s'appelle une **jointure interne** (par opposition aux jointures externes, destinées à mettre en évidence les décalages entre tables). On peut alors écrire la requête aussi sous la forme :

```
SELECT *
FROM vers AS v
INNER JOIN occurrences AS o
ON v.numero_vers = o.numero_vers ;
```

La condition de rapprochement est introduite par **ON** et non par **WHERE**.

L'équivalent de la R^1_{61} *t. 1* p. 123 indique explicitement l'appel à une **jointure naturelle**, où la condition de rapprochement est l'identité de valeur entre la ou les colonne(s) de même nom dans les tables ainsi rapprochées :

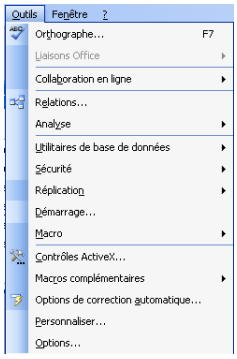
```
SELECT *
FROM vers
NATURAL JOIN occurrences ;
```

Un seul exemplaire du ou des attribut(s) de même nom est conservé dans le résultat d'une jointure naturelle alors que ce n'est pas le cas pour une jointure interne autre. △

Access

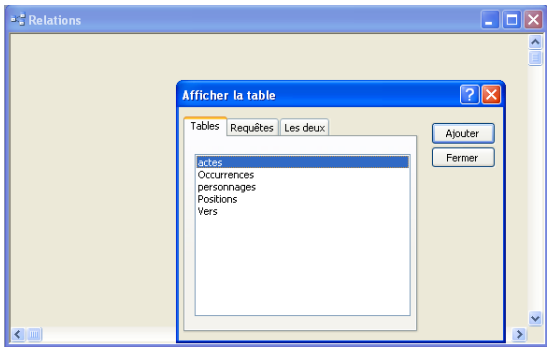
Établir les jointures « stables » Le menu ⌈Outils⌉ donne accès au choix ⌈Relations⌉ 26.

26



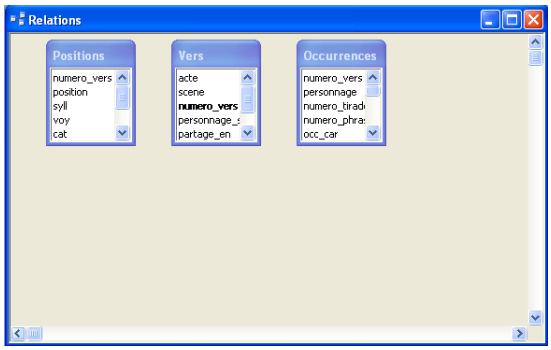
Il fait apparaître une fenêtre spécifique 27 où formuler les conditions de rapprochement permanentes entre tables. Dans cette fenêtre, comme dans l'établissement d'une requête, on choisit les tables à rapprocher.

27



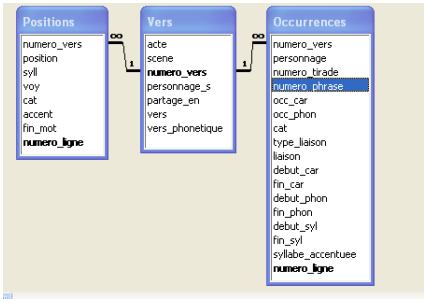
En 28 figurent les 3 tables initiales de la base PHÈDRE. Établir une jointure se réalise en faisant glisser le nom de l'attribut qui sert de clé étrangère dans la table « fille » sur l'attribut servant de clé primaire dans la table « mère ».

28



On trouve en 29 le résultat de cette opération : les attributs numero_vers dans les tables Positions et Occurrences ont le rôle de clé étrangère par rapport à l'attribut de même nom dans la table Vers. Les liens entre les tables manifestent ces rôles. Noter qu'Access ajoute sur ces liens la cardinalité des relations. À une ligne de la table Vers correspondent n lignes des tables Positions et Occurrences. Le signe de l'infini ∞ symbolise cette correspondance multiple.

29

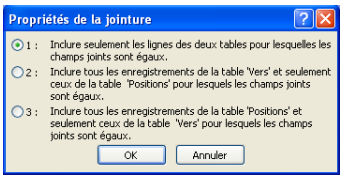


Paramétrer les jointures « stables » Lorsqu'on clique sur un des liens symbolisant les jointures « stables » entre tables, on obtient la boîte de dialogue 30 qui permet de régler les règles d'utilisation de la jointure. Sont ici cochés l'application de l'intégrité référentielle et le choix d'effacer en cascade les lignes de la table « fille » qui perdent leur correspondant dans la table « mère » par suite de suppression d'une ligne dans cette dernière table.

30

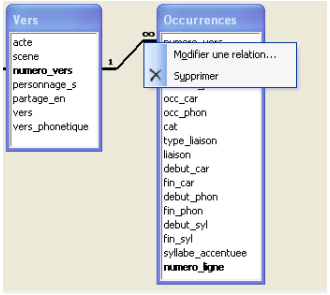
Cliquer sur le bouton ⌈Type jointure⌋ fait apparaître le choix 31 entre, en haut, la jointure la plus courante, celle qui retient uniquement les lignes des deux tables qui partagent la même valeur pour les colonnes rapprochées et, au milieu et en bas, les jointures externes.

31



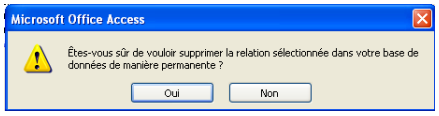
Modifier ou supprimer une jointure « stable » À tout moment, cliquer sur le lien entre deux tables qui rappelle la jointure « stable » qui les rapproche permet de modifier la jointure voire de la supprimer 32.

32



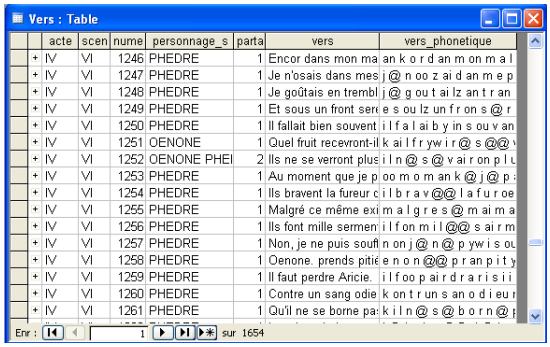
Une confirmation est alors demandée dans ce second cas de figure 33. La suppression d'une jointure « stable » peut en effet modifier le comportement de certaines requêtes mémorisées qui y faisaient appel.

33



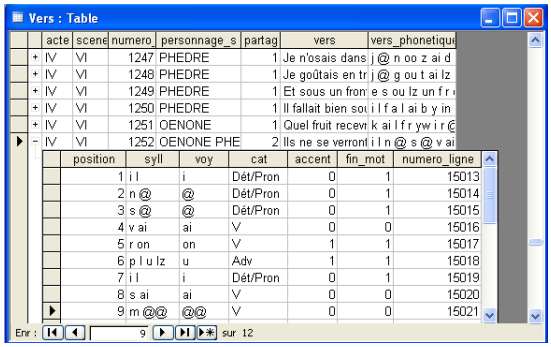
Matérialisation des jointures en mode feuille de données Lorsqu'on affiche en mode feuille de données une table entretenant une jointure stable avec une autre, cette jointure est rappelée par un signe + en début de ligne. C'est ce qu'on observe en 34 pour la table Vers. Cliquer sur ce signe + aboutit à visualiser dans une « sous-fenêtre » la ou les ligne(s) liées de l'autre table.

34



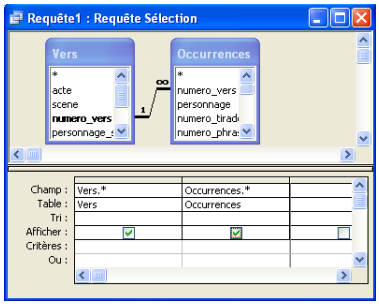
On constate effectivement en 35 qu'à la ligne qui décrit le vers 1252 correspondent plusieurs lignes de la table Positions (12, puisqu'il s'agit d'alexandrins).

35



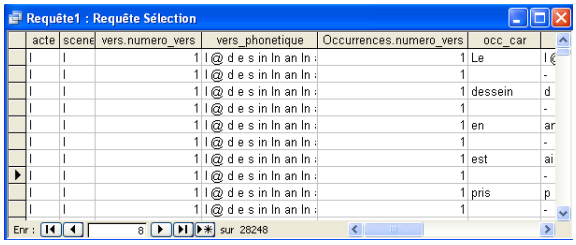
Utilisation des jointures « stables » dans des requêtes Les jointures stables apparaissent lors de la création d'une requête. En 36, les deux tables Vers et Occurrences ont été ajoutées dans la fenêtre du haut. Le lien qui les rapproche est rappelé en même temps. La fenêtre du bas stipule de fournir les colonnes des deux tables.

36



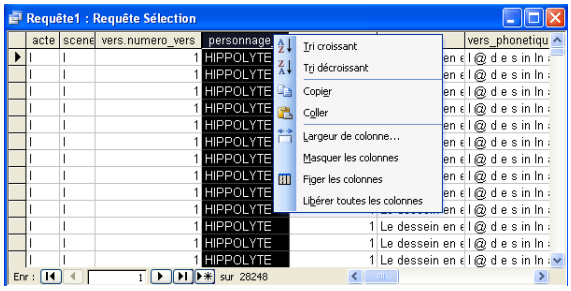
La feuille de données résultante 37 montre qu'Access laisse tels quels les noms de colonnes qui ne donnent pas naissance à des équivoques (c'est le cas pour acte ou scene dans la table Vers et pour occ_car dans la table Occurrences). Par contre, les colonnes numero_vers sont automatiquement préfixés par le nom de la table origine pour lever toute ambiguïté.

37



Pour rendre visible dans la même feuille de données les deux colonnes servant à la jointure, on a été amené à masquer une partie de colonnes provenant de la table Vers. Pour masquer une colonne, il faut la sélectionner : on place le pointeur de la souris sur le titre de la colonne jusqu'à ce qu'il se transforme en une flèche grasse vers le bas ; on clique alors le bouton droit, la colonne passe en inverse vidéo (blanc sur noir). Le clic droit donne alors accès à un menu déroulant 38 où l'on choisit l'entrée [Masquer les colonnes].

38



PRÉMA

Le produit relationnel des tables signaletique_fiches et fiches_normalisees aurait 1 034 289 lignes ($1\,017^2$). Il découle de la requête :

R^2_{91}

```
Produit[
signaletique_fiches,
fiches_normalisees]
```

TABLEAU 72 – PRÉMA : jointure signaletique_fiches-fiches_normalisees

id	*id_-bebe*	*jour*	*heure_-saisie*	*poids*	...	*id*	*texte*
931	111	7	21.50	650	...	931	bébé très tonique , n__pas aime être_dérangé -_Y râle .
352	36	15	6.00	1060	...	352	bébé bien_R énergique . gigote beaucoup même_R pendant_S les périodes . entre_S les soins . grosse colère quand on l__P embête .

Nous nous intéressons en fait au sous-ensemble de ces lignes où le texte de fiches_normalisees correspond à la signalétique remplie au même moment par l'infirmière. La condition de rapprochement est l'identité entre la valeur de l'attribut id dans la table signaletique_fiches et celle de l'attribut id dans la table fiches_normalisees. Ce sous-ensemble peut provenir d'une jointure avec cette condition de rapprochement :

R^2_{92}
JOINTURE(
s.id = f.id
)[signaletique_fiches ALIAS s,
fiches_normalisees ALIAS f]

Ce sous-ensemble peut provenir aussi d'une jointure naturelle qui repose sur l'identité de valeurs entre les attributs de même nom dans les tables concernées (dans le cas présent, id) :

R^2_{93}
JOINTURE NATURELLE[
signaletique_fiches,
fiches_normalisees]

La table produite par de telles jointures a bien évidemment seulement 1 017 lignes.

Le tableau 72 p. 244 présente un sous-ensemble des résultats. On note la répétition, soulignée par le grisé, de l'attribut id, qui figure dans les deux tables.

MySQL

Sont fournies trois instanciations de R^2_{92} p. 244. Pour id, on peut recourir à un **nom qualifié** (deuxième instanciation). On lui préfixe alors le nom de sa table (ou bien l'alias du nom de cette table, si on lui en a associé un via **AS**), suivi d'un point, comme le montrent les requêtes.

Dans la mesure où l'attribut permettant la jointure porte le même nom dans les deux tables, on peut également recourir à la **jointure naturelle** où la condition de rapprochement est l'identité de valeur pour le(s) attribut(s) de même nom. C'est la quatrième requête (**NATURAL JOIN**).

△

On assortit généralement la jointure d'une projection, en particulier pour ne garder qu'un exemplaire de l'attribut ou des attribut(s) servant au rapprochement. La jointure naturelle ne garde qu'un exemplaire du ou des attribut(s) permettant le rapprochement.

R^2_{92} p. 244

```
SELECT *
FROM signaletique_fiches,
     fiches_normalisees
WHERE signaletique_fiches.id = fiches_normalisees.id ;

SELECT *
FROM signaletique_fiches AS s,
     fiches_normalisees AS f
WHERE s.id = f.id ;

SELECT *
FROM signaletique_fiches
INNER JOIN fiches_normalisees
ON signaletique_fiches.id = fiches_normalisees.id ;
```

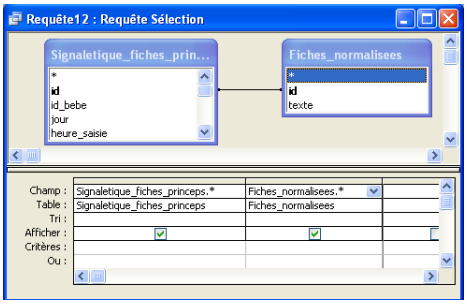
R^2_{93} p. 244

```
SELECT *
FROM signaletique_fiches
NATURAL JOIN fiches_normalisees ;
```

Access

En 39, la réalisation de la R^2_{92} p. 244 :

39



Son équivalent SQL Server est :

```
SELECT Signaletique_fiches_princeps.*,
     Fiches_normalisees.*
FROM Signaletique_fiches_princeps
INNER JOIN Fiches_normalisees
     ON Signaletique_fiches_princeps.id = Fiches_normalisees.id;
```

Exercice n°1 Grâce à la table créée et peuplée pour l'exercice 3, en utilisant également la table signaletique_fiches, fournissez, par jour, les minima, maxima, moyennes pour les fiches en termes d'occurrences comme de types de lemmes.

Exercice n°2 Fournir pour chaque jour le nombre de fiches correspondant à des filles et à des garçons, ainsi que, dans un deuxième temps, les pourcentages qui en découlent.

PHÈDRE

ESQUE

3. Combiner jointure et regroupement

≃ nombre de vers dont un personnage de *Phèdre* est entièrement ou partiellement le locuteur

R^1_{62} *t. 1* p. 126

```
RESTRICTION(
personnage <> 'ENTRE_PERSONNAGES'
)
[occurrences]
↪
REGROUPER SUR(personnage)[<résultat1>]
↪
PAR GROUPE(
personnage,
NOMBRE DE VALEURS DISTINCTES(numero_vers) TITRE 'n-
bre vers'
)
[<résultat2>]
```

≃ nombre de vers par acte et par personnage dans *Phèdre*

R^1_{64} *t. 1* p. 128

```
JOINTURE NATURELLE[occurrences, vers]
↪
REGROUPER SUR(personnage, acte)[<résultat1>]
↪
PAR GROUPE(
personnage,
acte,
NOMBRE DE VALEURS DISTINCTES(numero_vers) TITRE 'n-
bre vers'
)
AVEC (personnage <> 'ENTRE_PERSONNAGES')
[<résultat2>]
```

△

On ne pourrait pas utiliser comme point de départ la table vers. En effet, dans cette table, l'attribut personnage_s combine en une seule chaîne les noms des personnages dans le cas de vers partagés (partage_en a une valeur supérieur à 1). Un regroupement sur les valeurs de l'attribut personnage_s aboutirait à des groupes correspondant aux vers portés par un personnage seul et à d'autres correspondant aux vers relevant de toutes les associations de personnages. Les vers où intervient un personnage donné ne seraient pas tous rassemblés.

MySQL

R^1_{62} *t. 1* p. 126

À partir de la table occurrences, on souhaite connaître le nombre de vers dont un personnage de *Phèdre* est entièrement ou partiellement le locuteur. Les fragments – espaces, séparateur de tirades interne au vers symbolisé par l'oblique : / – ne relevant pas d'un personnage sont

attribués à ENTRE_PERSONNAGES et sont à enlever des décomptes, d'où la clause **WHERE** dans la requête qui suit. Par ailleurs, comme, pour un vers donné, dans cette table, on compte autant de numéros de vers que d'occurrences, on n'en garde à chaque fois qu'un exemplaire, ce qui explique l'appel à **DISTINCT** (le mot-clé **DISTINCT** permet d'obtenir des lignes toutes différentes, mais aussi des valeurs distinctes pour une colonne donnée) :

```
SELECT personnage,
     COUNT(DISTINCT numero_vers) AS 'nbre␣v.'
FROM occurrences
WHERE personnage <> 'ENTRE_PERSONNAGES'
GROUP BY personnage ;
```

R^1_{64} *t. 1* p. 128

Pour obtenir le nombre de vers par acte et par personnage, on combine les informations des tables vers et occurrences.

```
SELECT personnage,
     acte,
     COUNT(DISTINCT o.numero_vers) AS 'nbre␣vers'
FROM occurrences AS o
NATURAL JOIN vers
GROUP BY personnage, acte
   HAVING personnage <> 'ENTRE_PERSONNAGES' ;
```

Une autre formulation possible est :

```
SELECT personnage,
     acte,
     COUNT(DISTINCT o.numero_vers) AS 'nbre␣vers'
FROM occurrences AS o, vers AS v
WHERE o.numero_vers = v.numero_vers
GROUP BY personnage, acte
   HAVING personnage <> 'ENTRE_PERSONNAGES' ;
```

Une autre encore est :

```
SELECT personnage,
     acte,
     COUNT(DISTINCT o.numero_vers) AS 'nbre␣vers'
FROM occurrences AS o, vers AS v
WHERE o.numero_vers = v.numero_vers
     AND personnage <> 'ENTRE_PERSONNAGES'
GROUP BY personnage, acte
```

La différence entre cette dernière requête et la précédente est qu'ici l'élimination des lignes où personnage vaut 'ENTRE_PERSONNAGES' s'opère en amont du regroupement alors que dans la pénultième requête, elle s'opère en aval : c'est une restriction de deuxième niveau qui prend en compte les caractéristiques des groupes constitués.

On aimerait savoir la proportion de ces vers qui correspond à des vers partagés. L'information sur le partage figure dans la table vers. Par contre, cette table ne permet pas directement d'attribuer un vers partagé aux deux personnages qui le profèrent : la colonne personnage_s contient la suite des noms des personnages et ne donne pas accès à chacun d'eux. La jointure entre les 2 tables :

```
SELECT personnage_s,
     partage_en,
     personnage,
```

TABLEAU 73 – PHÈDRE : jointure et vers partagés

personnage_s	*partage_en*	*personnage*	*numero_vers*	*occ_car*
⋮	⋮	⋮	⋮	⋮
PHEDRE OENONE PHEDRE	3	PHEDRE	262	J'
PHEDRE OENONE PHEDRE	3	PHEDRE	262	aime
PHEDRE OENONE PHEDRE	3	PHEDRE	262	...
PHEDRE OENONE PHEDRE	3	ENTRE_PERSONNAGES	262	
PHEDRE OENONE PHEDRE	3	ENTRE_PERSONNAGES	262	/
PHEDRE OENONE PHEDRE	3	ENTRE_PERSONNAGES	262	
PHEDRE OENONE PHEDRE	3	OENONE	262	Qui
PHEDRE OENONE PHEDRE	3	OENONE	262	?
PHEDRE OENONE PHEDRE	3	ENTRE_PERSONNAGES	262	
PHEDRE OENONE PHEDRE	3	ENTRE_PERSONNAGES	262	/
PHEDRE OENONE PHEDRE	3	ENTRE_PERSONNAGES	262	
PHEDRE OENONE PHEDRE	3	PHEDRE	262	Tu
PHEDRE OENONE PHEDRE	3	PHEDRE	262	
PHEDRE OENONE PHEDRE	3	PHEDRE	262	connais
PHEDRE OENONE PHEDRE	3	PHEDRE	262	
PHEDRE OENONE PHEDRE	3	PHEDRE	262	ce
PHEDRE OENONE PHEDRE	3	PHEDRE	262	
PHEDRE OENONE PHEDRE	3	PHEDRE	262	fils
PHEDRE OENONE PHEDRE	3	PHEDRE	262	
PHEDRE OENONE PHEDRE	3	PHEDRE	262	de
PHEDRE OENONE PHEDRE	3	PHEDRE	262	
PHEDRE OENONE PHEDRE	3	PHEDRE	262	l'
PHEDRE OENONE PHEDRE	3	PHEDRE	262	amazone
PHEDRE OENONE PHEDRE	3	PHEDRE	262	,
⋮	⋮	⋮	⋮	⋮

```
      o.numero_vers,
      occ_car
FROM occurrences AS o, vers AS v
WHERE o.numero_vers = v.numero_vers ;
```

aboutit à une table combinant les informations nécessaires (tableau 73 p. 248). Une jointure naturelle aboutit de manière plus concise au même résultat :

```
SELECT personnage_s,
      partage_en,
      personnage,
      vers.numero_vers,
      occ_car
FROM occurrences
NATURAL JOIN vers ;
```

On note qu'il faut, dans cette requête, qualifier le nom d'attribut `numero_vers` sous peine de déclencher une erreur :

```
ERROR 1052 (23000) : Column 'numero_vers' in field list is ambiguous
```

Pour chacune des lignes correspondant à un vers donné, si ce vers est partagé, on garde le numéro de vers. L'appel à **DISTINCT** permet a posteriori de n'en conserver qu'un exemplaire. Si le vers n'est pas partagé, la marque **NULL** est renvoyée par **IF** : elle disparaît du décompte dans la mesure où **COUNT**, comme d'autres opérateurs numériques sur les regroupements,

ignore la marque **NULL** (chapitre IV § 4). La requête combine les informations des deux tables, en associant jointure et regroupement. Elle calcule également le pourcentage de vers partagés par personnage et le nombre de vers en 3 parties (tableau 74, p. 250). On note le décalage entre les personnages nobles, où les vers partagés représentent 5% au plus, et les suivantes, où la proportion dépasse 8%. Le rôle de faire-valoir des suivantes est clair. Phèdre est par ailleurs le personnage de premier plan qui profère le plus de vers en 3 parties, en particulier dans sa relation avec Oenone.

```
SELECT personnage,
  COUNT(DISTINCT o.numero_vers) AS 'nbre v.',
  COUNT(DISTINCT
    (IF(partage_en > 1, o.numero_vers, NULL)))
       AS 'v.partagés',
  COUNT(DISTINCT
    (IF(partage_en > 1, o.numero_vers, NULL)))
       / COUNT(DISTINCT o.numero_vers) * 100 AS '%',
  COUNT(DISTINCT
    (IF(partage_en = 3, o.numero_vers, NULL)))
       AS 'dont v. 3 parties'
FROM occurrences AS o,
     vers AS v
WHERE o.numero_vers = v.numero_vers
      AND personnage <> 'ENTRE_PERSONNAGES'
GROUP BY personnage
ORDER BY '%' ;
```

Elle instancie la requête :

R^2_{94}

JOINTURE(
o.numero_vers = v.numero_vers)
[occurrences ALIAS o,
vers ALIAS v]
↪ RESTRICTION(
personnage <> 'ENTRE_PERSONNAGES'
)[<résultat$_1$>]
↪
REGROUPER SUR(personnage)[<résultat$_2$>]
↪
PAR GROUPE(
personnage,
NOMBRE DE VALEURS DISTINCTES(o.numero_vers) TITRE 'nbre v.',
NOMBRE DE VALEURS DISTINCTES(SI(partage_en > 1, o.numero_vers, NULL)) TITRE 'v. partagés',
NOMBRE DE VALEURS DISTINCTES(SI(partage_en > 1, o.numero_vers, NULL)) / NOMBRE DE VALEURS DISTINCTES(o.numero_vers) * 100 TITRE '%',
NOMBRE DE VALEURS DISTINCTES(SI(partage_en = 3, o.numero_vers, NULL)) TITRE 'dont v. en 3 parties'
)[<résultat$_3$>]
↪
Tri sur('%')[<résultat$_4$>]

TABLEAU 74 – PHÈDRE : vers partagés par personnage

personnage	*nbre v.*	*v. partagés*	*%*	*dont v. 3 parties*
THERAMENE	183	7	3.83	1
HIPPOLYTE	362	14	3.87	1
THESEE	234	10	4.27	1
PHEDRE	485	21	4.33	4
ARICIE	140	7	5.00	0
OENONE	220	18	8.18	3
ISMENE	34	3	8.82	0
PANOPE	38	4	10.53	0

4. Jointures

4.1. ⊕ Auto-jointure

Rapprocher les vers en relation de rime

R^{1}_{65} *t. 1* p. 130

```
JOINTURE(
v2.numero_vers % 2 = 0
ET v2.numero_vers = v1.numero_vers + 1
)
[vers ALIAS v1, vers ALIAS v2]
↪
RESTRICTION(
v1.numero_vers TITRE 'n° 1',
v2.numero_vers TITRE 'n° 2',
v1.vers TITRE 'vers1',
v2.vers TITRE 'vers2
)
[<résultat1>]
```

MySQL

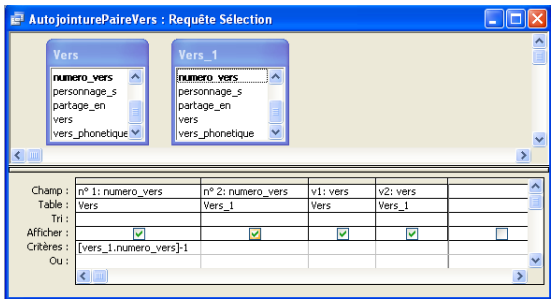
R^{1}_{65} *t. 1* p. 130

```
SELECT v1.numero_vers AS 'n° 1',
       v2.numero_vers AS 'n° 2',
       v1.vers AS 'vers 1',
       v2.vers AS 'vers 2 '
FROM vers AS v1, vers AS v2
WHERE v2.numero_vers % 2 = 0
       AND v2.numero_vers = v1.numero_vers + 1 ;
```

Access

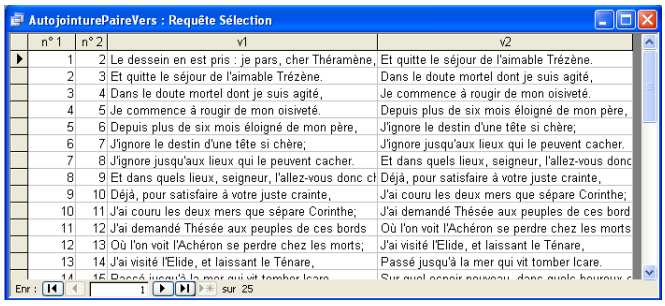
La requête 40, qui instancie la R^{1}_{65} *t. 1* p. 130, n'édicte qu'une condition : que le numéro du premier vers soit égal au numéro du deuxième vers moins 1. On note en 40 l'engendrement automatique par Access d'un nom qualifié – Vers_1 – pour la deuxième instance de la table Vers qui entre dans cette auto-jointure.

40



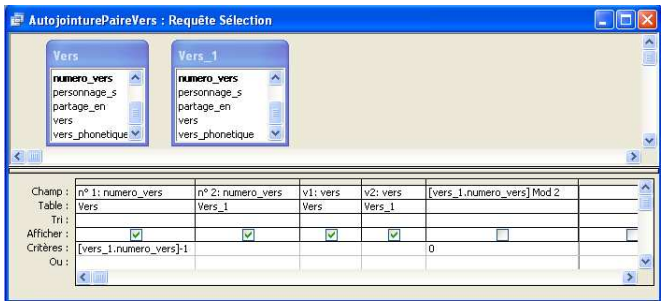
Le résultat en 41 le montre, chaque vers successif de *Phèdre* est mis en relation avec son successeur, si bien que les vers ne sont pas forcément en relation de rime : le vers 2 ne rime pas avec le vers 3, puisque la pièce constitue une suite de paires de rimes plates.

41

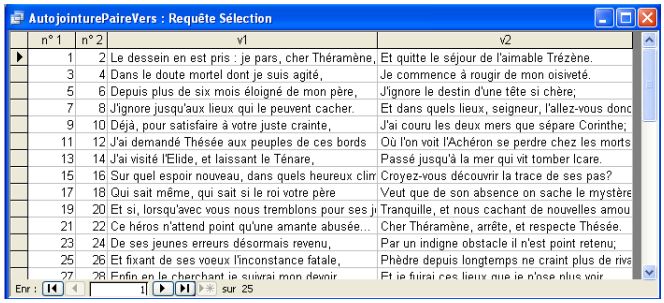


La requête 42 ajoute une contrainte : le numéro du deuxième vers doit être pair (le reste de la division par 2 de ce numéro, qui s'obtient par appel à la fonction MOD, abréviation de modulo, est égal à 0), ce qui produit 43.

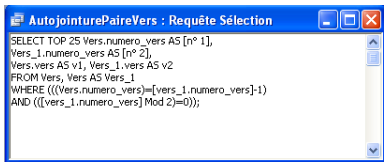
42



43



L'équivalent SQL Server de la requête interactive est fournie en 44.



Rapprocher les mots en relation de rime

R^{1}_{66} *t. 1* p. 131

```
JOINTURE(
o1.numero_vers = o2.numero_vers - 1
)
[occurrences ALIAS o1, occurrences ALIAS o2]
↪ RESTRICTION(
o2.numero_vers % 2 = 0
ET o1.fin_syl = 12
ET o1.cat PARMI ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv',
'Adj', 'Nc', 'Np', 'V')
ET o2.fin_syl = 12
ET o2.cat PARMI ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv',
'Adj', 'Nc', 'Np', 'V')
)[<résultat1>]
↪ PROJECTION(
o1.numero_vers TITRE 'n° 1',
o2.numero_vers TITRE 'n° 2',
o1.occ_car TITRE 'mot1'
o2.occ_car TITRE 'mot2'
)[<résultat2>]
```

MySQL

R^{1}_{66} *t. 1* p. 131

```
SELECT o1.numero_vers AS 'n° 1',
       o2.numero_vers AS 'n° 2',
       o1.occ_car AS 'mot1',
       o2.occ_car AS 'mot2'
FROM occurrences AS o1,
       occurrences AS o2
WHERE o1.numero_vers = o2.numero_vers - 1
      AND o2.numero_vers % 2 = 0
      AND o1.fin_syl = 12
      AND o2.fin_syl = 12
      AND o1.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V'
          )
      AND o2.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V'
          )  ;
```

On peut modifier cette requête pour concentrer son attention sur le fragment du vers contenant la rime (tableau 75). Le fragment de la clause **SELECT** :

```
v1.vers AS 'vers 1'
```

devient :

TABLEAU 75 – PHÈDRE : paires de vers rimant par auto-jointure

n°	*n°*	*vers 1*	*vers 2*
1	2	... Théramène,	... le Trézène.
3	4	... suis agité,	... n oisiveté.
5	6	... e mon père,	... e si chère ;
7	8	... ent cacher.	... c chercher ?

```
CONCAT("... ", SUBSTRING(v1.vers, LENGTH(v1.vers) - 10))
    AS 'vers 1',
```

Sachant la longueur du vers via [**LENGTH**(v1.vers)], on peut garder la sous-chaîne du vers à partir du 10^{e} caractère avant la fin : la fonction **SUBSTRING** prend comme premier argument la chaîne dont il faut retourner un fragment et comme deuxième le point de départ dans la chaîne. Le recours à **CONCAT** permet d'ajouter des points de suspension avant la fin du vers. On opère de la même manière pour le 2^{e} vers en relation de rime.

Access

R^{1}_{66} *t. 1* p. 131

```
SELECT Occurrences.numero_vers AS n°1,
    Occurrences_1.numero_vers AS n°2,
    Occurrences.occ_car AS mot1,
    Occurrences_1.occ_car AS mot2
FROM Occurrences,
    Occurrences AS Occurrences_1
WHERE (((Occurrences.numero_vers)=[Occurrences_1.numero_vers]-1)
    AND ((Occurrences.cat) NOT LIKE "Sep*"
        AND (Occurrences.cat) <> "Espace")
    AND ((Occurrences_1.cat) NOT LIKE "Sep*"
        AND (Occurrences_1.cat) <> "Espace")
    AND  ((Occurrences.fin_syl)=12)
    AND ((Occurrences_1.fin_syl)=12)
    AND (([Occurrences_1.numero_vers] MOD 2)=0));
```

Cette requête SQL Server se différencie de son homologue MySQL *supra* par la manière de conserver uniquement des « mots » effectifs, par opposition aux espaces et séparateurs de tous ordres. Le motif **NOT LIKE** "Sep*" élimine les occurrences dont la catégorie commence par Sep.

Un autre motif, positif, part du constat que les initiales des catégories à conserver et de celles à éliminer ne sont pas en intersection. Il sélectionne les catégories dont l'initiale convient, comme on le constate en comparant avec la requête MySQL :

```
... AND ((Occurrences.cat)  LIKE "[DCPRANV]*"
```

Un dernier motif table inversement sur le fait que les occurrences à éliminer ont un nom de catégorie qui commence soit par S (c'est l'ensemble des séparateurs) soit par E (Espace) et qu'aucune occurrence à conserver ne présente une catégorie ayant une telle initiale :

```
... AND ((Occurrences.cat)  NOT Like "[ES]*")...
```

$\simeq$ mots plus de 2 fois en relation de rime

R^{1}_{67} *t. 1* p. 131

```
PROJECTION AVEC DOUBLONS(
o1.occ_car TITRE 'mot1'
o2.occ_car TITRE 'mot2')
[<résultat2 R1_66 t. 1 p. 131>]
↪
REGROUPER SUR('mot1', 'mot2')[<résultat1>]
↪
PAR GROUPE(
'mot1',
'mot2',
NOMBRE DE LIGNES() TITRE 'o.'
)
[<résultat2>]
AVEC (NOMBRE DE LIGNES() > 2)
```

MySQL

R^{1}_{67} *t. 1* p. 131

```
SELECT o1.occ_car AS 'mot1',
     o2.occ_car AS 'mot2',
     COUNT(*) AS 'o.'
FROM occurrences AS o1,
     occurrences AS o2
WHERE o1.numero_vers = o2.numero_vers - 1
     AND o2.numero_vers % 2 = 0
     AND o1.fin_syl = 12
     AND o2.fin_syl = 12
     AND o1.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V'
         )
     AND o2.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V'
         )
GROUP BY 'mot1', 'mot2'
     HAVING COUNT(*) > 1 ;
```

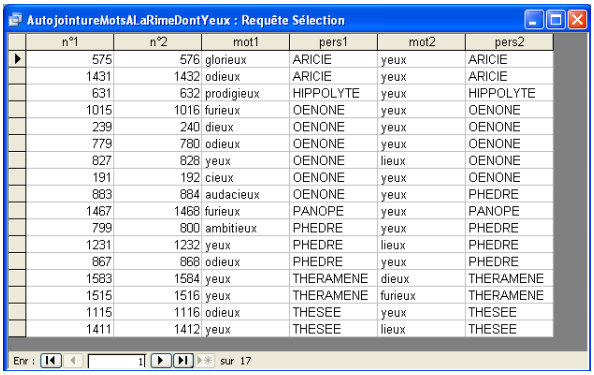
On constate que la condition de rapprochement constitutive de la jointure apparaît au milieu des autres conditions de restriction de la clause **WHERE**. △

Access

R^{1}_{67} *t. 1* p. 131

```
SELECT Occurrences.occ_car AS mot1,
     Occurrences_1.occ_car AS mot2,
     COUNT(*) AS o
FROM Occurrences,
     Occurrences AS Occurrences_1
WHERE (((Occurrences.numero_vers)=[Occurrences_1.numero_vers]-1)
     AND ((Occurrences.cat) Not Like "[ES]*")
     AND ((Occurrences_1.cat) NOT LIKE "[ES]*")
     AND ((Occurrences.fin_syl)=12)
     AND ((Occurrences_1.fin_syl)=12)
     AND (([Occurrences_1.numero_vers] MOD 2)=0))
GROUP BY Occurrences.occ_car, Occurrences_1.occ_car
HAVING COUNT(*) > 1;
```

Le résultat de cette requête figure en 73.



AutojointureMotsALaRimeDontYeux : Requête Sélection

n°1	n°2	mot1	pers1	mot2	pers2
575	576	glorieux	ARICIE	yeux	ARICIE
1431	1432	odieux	ARICIE	yeux	ARICIE
631	632	prodigieux	HIPPOLYTE	yeux	HIPPOLYTE
1015	1016	furieux	OENONE	yeux	OENONE
239	240	dieux	OENONE	yeux	OENONE
779	780	odieux	OENONE	yeux	OENONE
827	828	yeux	OENONE	lieux	OENONE
191	192	cieux	OENONE	yeux	OENONE
883	884	audacieux	OENONE	yeux	PHEDRE
1467	1468	furieux	PANOPE	yeux	PANOPE
799	800	ambitieux	PHEDRE	yeux	PHEDRE
1231	1232	yeux	PHEDRE	lieux	PHEDRE
867	868	odieux	PHEDRE	yeux	PHEDRE
1583	1584	yeux	THERAMENE	dieux	THERAMENE
1515	1516	yeux	THERAMENE	furieux	THERAMENE
1115	1116	odieux	THESEE	yeux	THESEE
1411	1412	yeux	THESEE	lieux	THESEE

Enr : 1 sur 17

4.2. ⊕ Semi-jointure

« Écho » syllabe à l'hémistiche – syllabe à la rime, rime interne

R^1_{68} *t. 1* p. 133

```
JOINTURE(
p1.numero_vers = p2.numero_vers
ET p1.numero_vers = v.numero_vers
)
[positions ALIAS p1, positions ALIAS p2, vers ALIAS v]
↪
RESTRICTION(
p1.position = 6
ET p2.position = 12
ET p1.voy = p2.voy
)
[<résultat1>]
↪
PROJECTION(
v.numero_vers TITRE 'n°',
vers
)
[<résultat2>]
```

MySQL

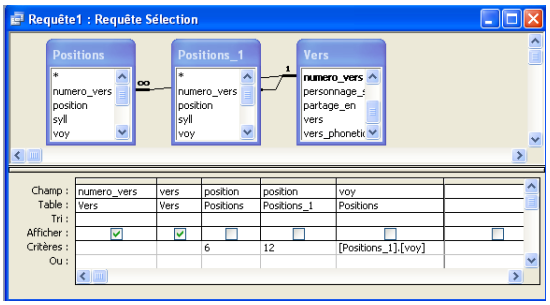
R^1_{68} *t. 1* p. 133

```
SELECT vers.numero_vers AS 'N°',
       vers
FROM vers,
       positions AS p1,
       positions AS p2
WHERE vers.numero_vers = p1.numero_vers
      AND p1.numero_vers = p2.numero_vers
      AND p1.position = 6
      AND p2.position = 12
      AND p1.voy = p2.voy ;
```

Access

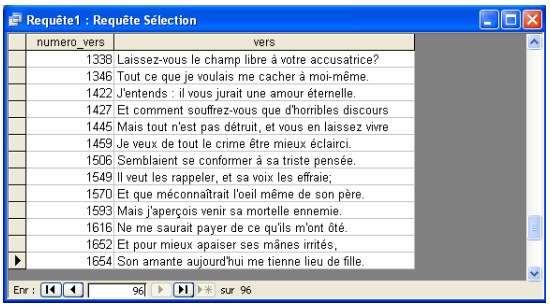
En 46, instanciation de la R^1_{68} *t. 1* p. 133, on constate l'existence d'une jointure « stable » entre chacune des 2 tables Positions et Positions_1 d'une part et la table Vers de l'autre, manifestée à chaque fois par l'arête qui les réunit. La ligne [Critères] spécifie que les deux lignes de la table Positions mises en relation et qui concernent le même vers doivent respectivement porter sur la 6ème syllabe et sur la 12ème syllabe. Elle indique également que la voyelle en 6ème syllabe doit être identique à la voyelle en 12ème position, via le recours au nom qualifié [Positions_1].[voy].

46



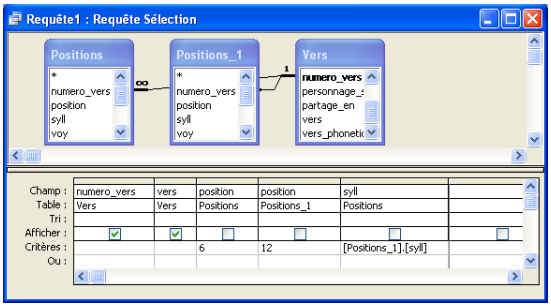
On trouve le résultat en 47.

47



La recherche des vers à rime interne n'implique qu'une modification minime de la requête précédente, comme le montre 48. On vérifie simplement que la syllabe, et non plus la voyelle, en 6ème position est identique à son homologue en 12ème position.

48



PRÉMA

Une jointure peut mettre en relation plus d'une table, comme dans la requête :

TABLEAU 76 – PRÉMA : jointure de 4 tables et résultat

BB	*sexe*	*poids*	...	*inf.*	*âge*	*service*	...	*fiche*	*texte*
38	Garçon	810	...	61	26	Jour	...	361	bb calme

R^2_{95}

JOINTURE(
signaletique_fiches.id = fiches_depart.id
Et bebes.id = id_bebe
Et infirmieres.id = id_infirmiere
)[fiches_depart,
signaletique_fiches,
bebes,
infirmieres]

↪ PROJECTION(
bebes.id TITRE 'BB',
sexe,
poids,
…
infirmieres.id TITRE 'inf.',
age Titre 'âge',
service,
…
signaletiques_fiches.id TITRE 'f.',
…
fiches_depart.texte
)[<résultat$_1$>]

La figure 5, p. 258 montre comment les conditions de rapprochement entre tables créent une table donc chaque ligne combine les informations des 4 tables de départ. Une partie d'une telle ligne est fournie dans le tableau 76, p. 257. Les 4 tables combinées ici supposent 3 conditions de rapprochement. L'attribut id constitue la clé primaire de chacune des tables. Il constitue aussi une clé étrangère pour la table signaletique_fiches par rapport à la table fiches_depart. Les attributs id_bebe et id_infirmiere servent de clés étrangères dans la table signaletique_fiches respectivement par rapport aux tables bebes et infirmieres. △

La jointure peut constituer une opération coûteuse, puisqu'elle revient conceptuellement à l'élagage d'un produit relationnel potentiellement volumineux. Dans le cas présent, la jointure aboutit à retenir 1 017 lignes combinant pour la signalétique d'une fiche donnée, le texte correspondant de fiches_depart ainsi que les métadonnées disponibles sur le bébé en cause et sur l'infirmière qui a rédigé la fiche. Le produit relationnel des 4 tables comprend. . . 5 256 256 698 lignes (1 017 × 1 017 × 121 × 42).

On n'utilise pas toujours l'ensemble des attributs résultants de la jointure. La **semi-jointure** représente le cas particulier où l'on conserve uniquement les attributs d'une des deux relations d'entrée. On peut par contre utiliser pour des restrictions les attributs de la relation non conservés dans le résultat. C'est ainsi qu'on peut conserver seulement l'identifiant et le texte des fiches concernant un bébé déterminé (tableau 77, p. 259) :

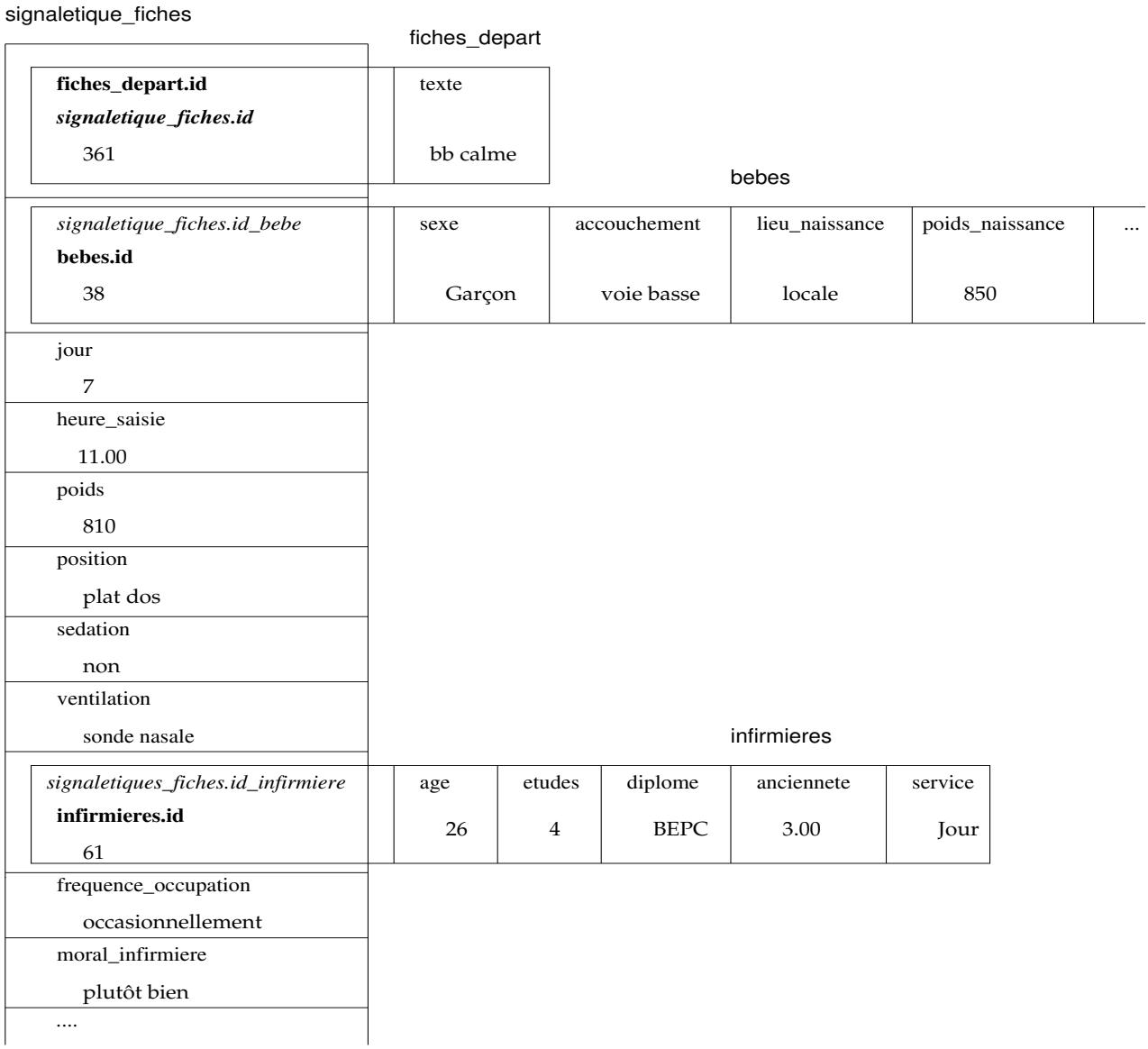


FIGURE 5 – PRÉMA : notion de jointure

TABLEAU 77 – PRÉMA : texte de départ pour le bébé 38

id	*texte*
360	petit bébé bien tonique Reactions adaptées. Pleure pd les soins, agripe ce qui passe à sa porté de main. Bébé calme qui dort entre les soins. Pas agité outre mesure.
361	bb calme
362	BB plus que Tonique lors des Soins. Gigote ds tous les Sens. BB fragile- fatigué. Très Fétif. Contact moyen avec regard qui plafonne parfois.
363	BB très calme et détendu, installé sur le ventre se tortille les fesses. Très peu éveillé, ouvre peu les yeux - Peu participatif aux soins mais reactif aux soins douloureux
358	Bebé tres energique pt les soins. gigote bras et jambes sait se detendre en dehors des periodes de soin. Ouvre les yeux.
359	- c'est un BéBé agréable, tonique, très mignon est très calme. ouvre ses yeux quand on lui parle - c'est très agréable de s'occuper de lui, l'expression de son visage est rigolotte. Il va très Bien et respire seule.

R^2_{96}

JOINTURE NATURELLE[
signaletique_fiches,
fiches_depart]
↪ RESTRICTION(id_bebe = 38)[<résultat$_1$>]
↪
PROJECTION(id, texte)[<résultat$_2$>]

MySQL

R^2_{95} p. 257

```
SELECT bebes.id AS BB, sexe, poids, ...,
       infirmieres.id AS 'inf.', age as 'âge', service, ...,
       fiches_depart.id AS fiche, texte
FROM fiches_depart,
     signaletique_fiches,
     bebes,
     infirmieres
WHERE signaletique_fiches.id = fiches_depart.id
    AND bebes.id = id_bebe
    AND infirmieres.id = id_infirmiere ;
```

R^2_{96} p. 259

```
SELECT f.id,
    f.texte
FROM signaletique_fiches AS s,
    fiches_depart AS f
WHERE s.id = f.id
     AND id_bebe = 38 ;
```

Cette requête pourrait s'exprimer de manière plus concise via une jointure naturelle :

```
SELECT f.id,
       texte
FROM signaletique_fiches
NATURAL JOIN fiches_depart AS f
WHERE id_bebe = 38 ;
```

Exercice n°3 Pour chaque infirmière, fournir son identifiant, le nombre de fiches qu'elle a remplies, le nombre total d'occurrences de mots pleins et le nombre moyen d'occurrences de mots pleins, ainsi que le nombre total de lemmes de mots pleins et le nombre moyen de réalisations par lemme.

PHÈDRE

Mots rimant ensemble

Exercice n°4 Fournir la table des mots qui riment ensemble dans *Phèdre*, avec pour chaque couple, le numéro du premier vers, le numéro du second, les réalisations graphiques des deux mots, ainsi que leurs catégories.

Exercice n°5 Modifier la requête de l'exercice précédent pour obtenir l'ensemble des mots qui riment avec *yeux* et le nombre d'occurrences de chaque couple.

Exercice n°6 À partir de la requête de l'exercice précédent, indiquez le nombre de fois où *yeux* occupe la première position dans le couple de vers rimant, le nombre de fois où il occupe la deuxième position, le nombre de fois où un autre mot que *yeux* est en première position dans un couple de vers rimant et comprenant *yeux* et le nombre de fois où dans un tel couple, un autre mot que *yeux* est en deuxième position.

Écho hémistiche-rime

Exercice n°7 À partir de la requête MySQL :

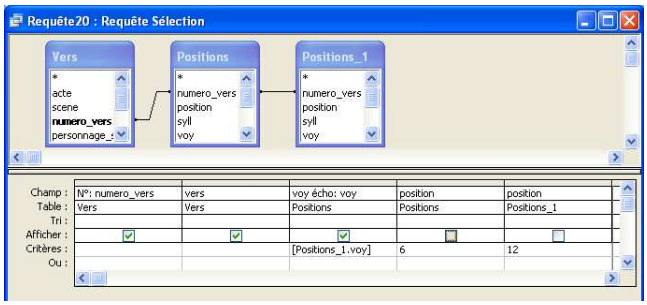
```
SELECT vers.numero_vers AS 'N°',
       vers,
       p1.voy AS 'voy. écho'
FROM vers,
       positions AS p1,
       positions AS p2
WHERE vers.numero_vers = p1.numero_vers
      AND p1.numero_vers = p2.numero_vers
      AND p1.position = 6
      AND p2.position = 12
      AND p1.voy = p2.voy ;
```

ou de son équivalent Access :

TABLEAU 78: PHÈDRE : voyelles en écho hémistiche-rime (extrait)

N°	*vers*	*voy. écho*
342	Le roi n'est plus, Madame ; il faut prendre sa place.	a
873	C'en est fait : on dira que Phèdre, trop coupable,	a
629	Je le vois, je lui parle ; et mon coeur... Je m'égare,	a
1061	Prends garde que jamais l'astre qui nous éclaire	ai
285	Quand ma bouche implorait le nom de la déesse,	ai
46	Lasse enfin d'elle-même et du jour qui l'éclaire,	ai
905	C'est un trésor trop cher pour oser le commettre.	ai
681	Ces dieux qui se sont fait une gloire cruelle	ai
1033	C'est trop laisser la reine à sa douleur mortelle;	ai
229	Quoiqu'il vous reste à peine une faible lumière,	ai

49



qui fournissent l'une et l'autre les voyelles en écho entre hémistiche et rime (tableau 78 p. 261), déterminer la fréquence des voyelles donnant lieu à un tel écho et classer les voyelles et les fréquences correspondantes par fréquence décroissante.

NB : dans la requête Access, pour assurer l'auto-jointure entre les deux instances de la table Positions sur l'attribut numero_vers, on fait glisser le nom de l'attribut numero_vers dans la table Positions_1 sur le nom numero_vers de la table Positions (ou dans l'autre sens). On voit alors en 49 une flèche apparaître entre les 2 tables reliant les deux attributs.

Vers à rime imparfaite

Exercice n°8 Chercher les mots de *Phèdre* qui ne riment pas pour l'oreille, c'est-à-dire où la fin phonétique des deux mots diffère.

ESQUE

5. Repérer les décalages

PRÉMA

PHÈDRE

Les requêtes pour créer la table actes_personnages_possibles, la peupler et agir de même pour la table actes_personnages_effectifs sont les suivantes.

MySQL

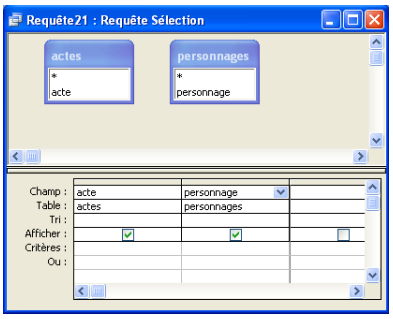
```
CREATE TABLE actes_personnages_possibles (
      acte VARCHAR(5) BINARY NOT NULL DEFAULT "",
      personnage VARCHAR(100) BINARY NOT NULL DEFAULT "",
      PRIMARY KEY (acte, personnage)) ;

INSERT INTO actes_scenes_possibles
      SELECT *
      FROM actes, personnages ;

INSERT INTO actes_personnages_effectifs
      SELECT DISTINCT acte, personnage_s
      FROM vers
      WHERE partage_en = 1 ;
```

La seconde requête utilise les deux tables engendrées au § 1.

Access

Le produit relationnel des deux tables engendrées au § 1 s'obtient par :

50

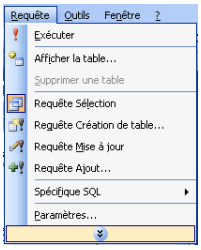


L'équivalent SQL Server est :

```
SELECT actes.acte,
      personnages.personnage
FROM actes,
      personnages;
```

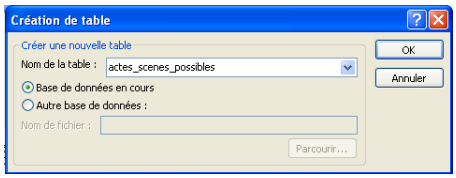
On choisit alors [Requête | Requête création de table] 51.

51



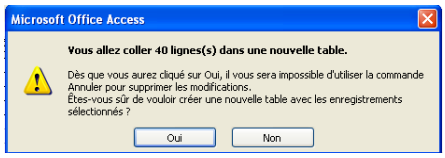
On fournit un nom de table 52.

52



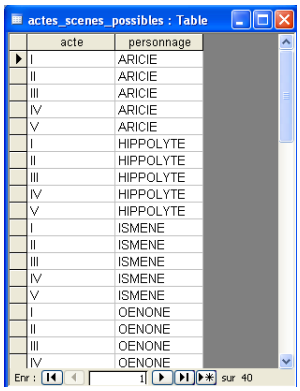
Un avertissement est fourni : 53.

53



En 54, un extrait de la table résultat.

54



L'engendrement de la table actes_personnages_effectifs obéit à la même démarche :

55

La comparaison des deux tables est alors possible.

R$^{1}_{69}$ *t. 1* p. 134

JOINTURE NATURELLE[
actes_personnages_possibles,
actes_personnages_effectifs]

R^1_{70} *t. 1* p. 135

JOINTURE EXTERNE GAUCHE(
gauche.acte = droite.acte
ET gauche.personnage = droite.personnage
)
[actes_personnages_possibles ALIAS gauche,
actes_personnages_effectifs ALIAS droite]

R^1_{71} *t. 1* p. 136

RESTRICTION(e.acte EST NULL)
[<résultat de la R^1_{70} *t. 1* p. 135>]

↪ TRI SUR(gauche.personnage, gauche.acte)[<résultat$_1$>]

La première requête ne retient que les lignes identiques des deux tables. La deuxième requête, qui fait appel à une jointure externe, met en évidence les lignes dans lesquelles une combinaison possible acte–personnage n'a pas de correspondant parmi les combinaisons effectives. La troisième requête raffine la précédente. Elle retient uniquement les lignes de discordance, que manifeste la marque **NULL** pour la colonne personnage de la table actes_personnages_effectifs.

MySQL

R^1_{69} *t. 1* p. 134

```
SELECT *
FROM actes_personnages_possibles
NATURAL JOIN actes_personnages_effectifs ;
```

R^1_{70} *t. 1* p. 135

```
SELECT *
FROM actes_personnages_possibles AS p
LEFT OUTER JOIN actes_personnages_effectifs AS e
ON p.acte = e.acte
    AND p.personnage = e.personnage ;
```

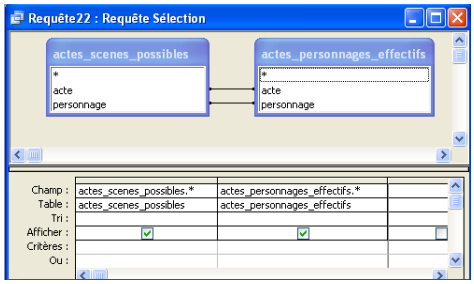
R^1_{71} *t. 1* p. 136

```
SELECT *
FROM actes_personnages_possibles AS p
LEFT OUTER JOIN actes_personnages_effectifs AS e
ON p.acte = e.acte AND p.personnage = e.personnage
WHERE e.personnage IS NULL
ORDER BY p.personnage, p.acte ;
```

Access

La jointure entre les tables actes_personnages_possibles et actes_personnages_effectifs s'opère en faisant glisser les noms des attributs acte et personnage d'une table sur l'autre. La relation (au sens d'Access) « temporaire » résultante est figurée par les deux lignes qui relient les paires d'attributs.

56

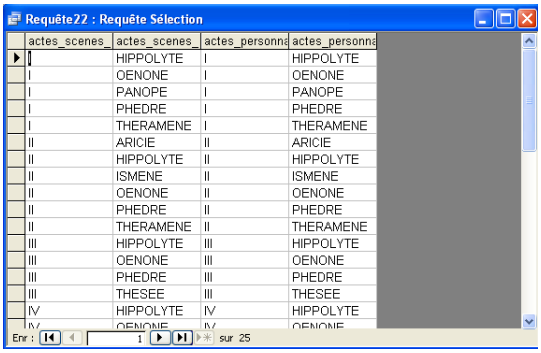


Cette requête Access, proche de la R^1_{69} *t. 1* p. 134, a pour équivalent SQL Server :

```
SELECT actes_scenes_possibles.*,
       actes_personnages_effectifs.*
FROM actes_scenes_possibles
INNER JOIN actes_personnages_effectifs
ON (actes_scenes_possibles.personnage = actes_personnages_effectifs.personnage)
    AND (actes_scenes_possibles.acte = actes_personnages_effectifs.acte);
```

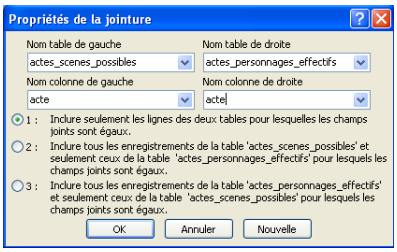
Son résultat comprend 25 lignes, soit l'intersection entre les deux tables.

57



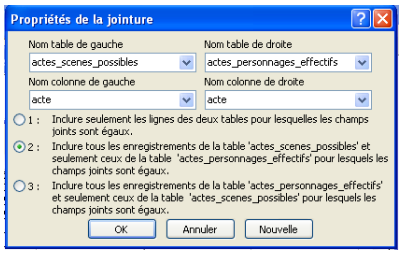
On constate d'ailleurs, lorsqu'on clique avec le bouton droit sur une des deux lignes symbolisant la jointure entre les deux tables que c'est effectivement le comportement attendu de cette jointure :

58

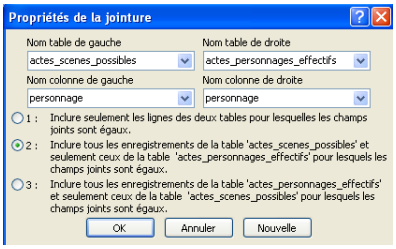


Pour obtenir une jointure externe, on modifie comme suit la jointure pour chacun des attributs mis en relation :

59

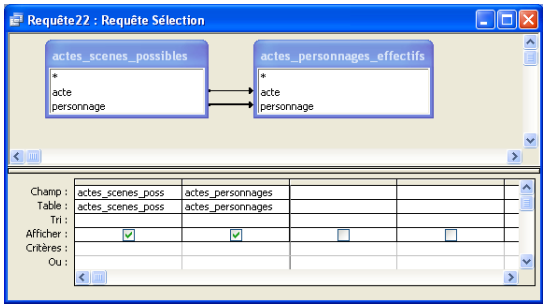


60



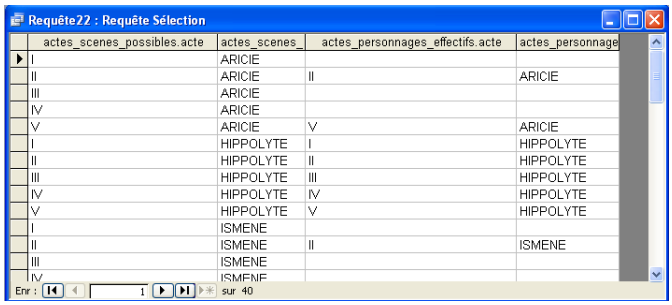
Les lignes reliant les deux tables se transforment en flèches :

61



Le résultat est désormais de 40 lignes. Il contient des lignes mettant en évidence le décalage entre les combinaisons possibles et les combinaisons effectives, **NULL** y étant figuré par le vide :

62



L'équivalent SQL Server montre bien qu'on a formulé, via l'interface, une jointure externe proche de la R^1_{70} *t. 1* p. 135 :

```
SELECT actes_scenes_possibles.*,
       actes_personnages_effectifs.*
FROM actes_scenes_possibles
LEFT JOIN actes_personnages_effectifs
ON (actes_scenes_possibles.personnage = actes_personnages_effectifs.personnage)
       AND (actes_scenes_possibles.acte = actes_personnages_effectifs.acte);
```

TABLEAU 79 – PRÉMA : lemmes par jour : indications globales

Jour	*fiches*	*tot. types*	*min. types*	*max. types*	*moy. types*	*tot. o.*	*Min. o.*	*max. o.*	*moy. o.*	*tok/typ*
1	327	7518	3	100	22.99	9947	3	177	30.42	1.32
3	286	6964	6	66	24.35	9322	6	114	32.59	1.34
7	225	5363	3	65	23.84	7171	3	109	31.87	1.34
15	178	4219	4	58	23.70	5603	4	93	31.48	1.33

ESQUE

6. Au delà du modèle relationnel

PRÉMA

PHÈDRE

ESQUE

7. Solutions

Solution de l'exercice n°1 p. 245 Par type, on entend ici un lemme donné, par occurrences, le nombre de fois où il se rencontre au sein d'une fiche ou d'un ensemble de fiches.

Le tableau 79 p. 267 fournit les résultats. Une jointure est nécessaire avec la table signaletique_fiches pour connaître le jour de rédaction de la fiche :

R_{97}^{2}

```
JOINTURE(
id_fiche = id
)[fiches_types_occurrences_lemmes,
signaletique_fiches]
↪ Regrouper sur(jour)[<résultat1>]
↪
PAR GROUPE(
jour TITRE 'Jour',
NOMBRE DE LIGNES() TITRE 'fiches',
SOMME(types) TITRE 'tot. types',
MINIMUM(types) TITRE 'min. types',
MAXIMUM(types) TITRE 'max. types',
MOYENNE(types) TITRE 'moy. types',
SOMME(occurrences) TITRE 'tot. o.',
MINIMUM(occurrences) TITRE 'min. o.',
MAXIMUM(occurrences) TITRE 'max. o.',
MOYENNE(occurrences) TITRE 'moy. o.',
SOMME(occurrences)/SOMME(types) TITRE 'tok/typ'
)[<résultat2>]
```

TABLEAU 80 – PRÉMA : fiches et sexe des prématurés par jour

Jour	*sexe*	*o.*
1	Fille	151
1	Garçon	176
3	Fille	141
3	Garçon	145
7	Fille	104
7	Garçon	121
15	Fille	83
15	Garçon	95

Jour	*filles*	*%*	*garçons*
1	151	46.18	176
3	141	49.30	145
7	104	46.22	121
15	83	46.63	95

MySQL

```
SELECT jour AS 'Jour',
     COUNT(*) AS 'fiches',
     SUM(types) AS 'tot. types',
     MIN(types) AS 'min. types',
     MAX(types) AS 'max. types',
     FORMAT(AVG(types), 2) AS 'moy. types',
     SUM(occurrences) AS 'tot. o.',
     MIN(occurrences) AS 'Min. o.',
     MAX(occurrences) AS 'max. o.',
     FORMAT(AVG(occurrences), 2) AS 'moy. o.',
     SUM(occurrences) / SUM(types) AS 'tok/typ'
FROM fiche_types_occurrences_lemmes,
     signaletique_fiches
WHERE id_fiche = id
GROUP BY jour ;
```

Solution de l'exercice n°2 p. 245 Le tableau 80 p. 268 donne les résultats des requêtes.

R^2_{98}

```
JOINTURE(
id_bebe = bebes.id
)[signaletique_fiches, bebes]
↪ REGROUPER SUR(jour, sexe)[<résultat1>]
↪
PAR GROUPE(
jour Titre 'Jour',
sexe,
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat2>]
```

R^2_{99}

```
JOINTURE(
id_bebe = bebes.id
)[signaletique_fiches, bebes]
↪ REGROUPER SUR(jour)[<résultat1>]
↪
PAR GROUPE(
jour TITRE 'Jour',
SOMME(SI(sexe = 'Fille', 1, 0)) TITRE 'Filles',
SOMME(SI(sexe = 'Fille', 1, 0)) / NOMBRE DE LIGNES() * 100
TITRE '%',
SOMME(SI(sexe = 'Fille', 0, 1)) TITRE 'garçons'
)[<résultat2>]
```


MySQL

R^2_{98} p. 268

```
SELECT signaletique_fiches.jour AS 'Jour',
    sexe,
    COUNT(*) AS 'o.'
FROM signaletique_fiches,
    bebes
WHERE id_bebe = bebes.id
GROUP BY signaletique_fiches.jour, sexe ;
```

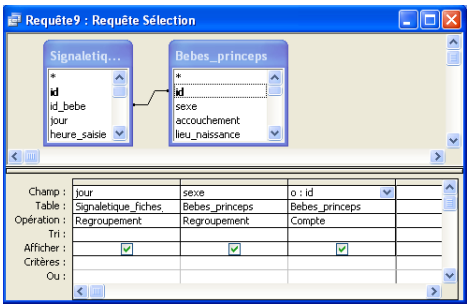
R^2_{99} p. 269

```
SELECT signaletique_fiches.jour AS 'Jour',
    SUM(IF(sexe = 'Fille', 1, 0)) AS 'filles',
    SUM(IF(sexe = 'Fille', 1, 0)) / COUNT(*) * 100 as '%',
    SUM(IF(sexe = 'Fille', 0, 1)) AS 'garçons'
FROM signaletique_fiches,
    bebes
WHERE id_bebe = bebes.id
GROUP BY signaletique_fiches.jour ;
```

Access

En 63, l'équivalent de la R^2_{98} p. 268.

63



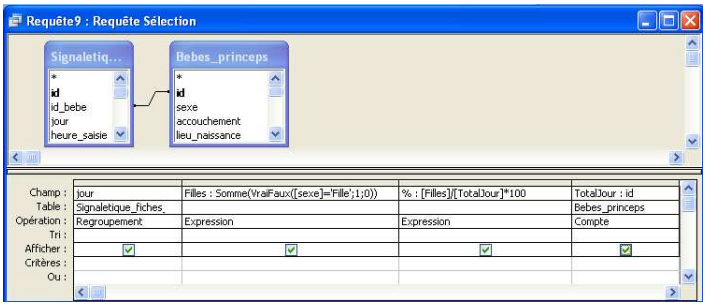
△

En 64, proche de R^2_{99} p. 269, on notera qu'il faut cocher la colonne TotalJour pour qu'elle figure dans le résultat. Si ce n'est pas le cas, la colonne est considérée comme le paramètre

d'une requête paramétrée : l'utilisateur se voit demander la valeur à attribuer au paramètre TotalJour.

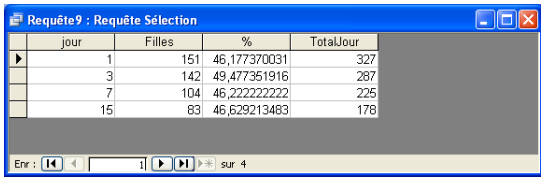
*

64



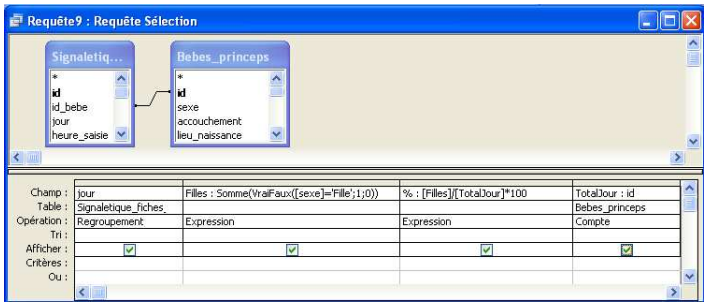
Le résultat figure en 65.

65



Une autre manière d'utiliser le total de bébés par jour sans pour autant le faire figurer dans le résultat consiste à le calculer directement dans la colonne %, comme en 66.

66



Solution de l'exercice n°3 p. 260 Le résultat du tableau 81 p. 272 combine les informations provenant de la table signaletique_fiches (les attributs id_infirmiere et id de la fiche) et celles issues de la table occ_prema : les lemmes et leurs catégories. La requête s'énonce ainsi :

```
R²₁₀₀
        JOINTURE(
        id = fiche
        )[signaletique_fiches,
        occ_prema]
    ↪   REGROUPER SUR(id_infirmiere)[<résultat₁>]
    ↪
        PAR GROUPE(
        id_infirmiere TITRE 'Inf.',
        NOMBRE DE VALEURS DISTINCTES(id) TITRE 'nbre fiches',
        COMPTE(SI(categorie RESSEMBLANT À '[ANRSDVCP]', 1, 0))
        TITRE 'o.',
        COMPTE(SI(categorie RESSEMBLANT À '[ANRSDVCP]', 1, 0))
        / NOMBRE DE VALEURS DISTINCTES(id) TITRE 'o. moy.',
        NOMBRE DE VALEURS DISTINCTES(SI(categorie RESSEM-
        BLANT À '[ANRSDVCP]', lemme, NULL)) TITRE 'lemmes',
        COMPTE(Si(categorie RESSEMBLANT À '[ANRSDVCP]', 1, 0))
        / NOMBRE DE VALEURS DISTINCTES(SI(categorie RESSEM-
        BLANT À '[ANRSDVCP]', lemme, NULL)) TITRE 'o./lemmes'
        )[<résultat₂>]
    ↪
        TRI SUR('o. moy.')[<résultat₃>]
```

MySQL

```
SELECT id_infirmiere AS 'Inf',
     COUNT(DISTINCT id) AS 'nbre fiches',
     COUNT(IF(categorie REGEXP '^[ANRSDVCP]', 1, 0)) AS 'o.',
     COUNT(IF(categorie REGEXP '^[ANRSDVCP]', 1, 0))
          / COUNT(DISTINCT id) AS 'o. moy.',
     COUNT(DISTINCT IF(categorie REGEXP '^[ANRSDVCP]', lemme, NULL))
          AS 'lemmes',
     COUNT(IF(categorie REGEXP '^[ANRSDVCP]', 1, 0))
          / COUNT(DISTINCT IF(categorie REGEXP '^[ANRSDVCP]', lemme, NULL))
          AS 'o./lemmes'
FROM signaletique_fiches,
     occ_prema
WHERE id = fiche
GROUP BY id_infirmiere
ORDER BY 'o. moy.' ;
```

Solution de l'exercice n°4 p. 260 L'auto-jointure d'une table avec elle-même a permis de rapprocher les vers qui riment ensemble dans *Phèdre* (tableau 75, p. 253). On peut opérer de la même manière avec la table occurrences et obtenir les paires de mots en relation de rime (tableau 82, p. 273). Il faut d'abord conserver seulement les « vrais » mots au sein des occurrences, c'est-à-dire les occurrences dont les catégories conviennent. En second lieu, ne sont concernés que les mots en dernière position, qui s'achèvent donc en syllabe 12. La troisième condition rappelle celle de l'auto-jointure sur la table vers : le numéro du deuxième vers doit être pair et correspondre à une unité de plus que celui du premier vers.

TABLEAU 81 – PRÉMA : infirmières et « richesse lexicale »

Inf	*nbre fiches*	*o.*	*o. moy.*	*lemmes*	*o./lemmes*
61	22	520	23.64	60	8.67
70	16	433	27.06	75	5.77
73	10	275	27.50	40	6.88
28	16	474	29.62	82	5.78
46	19	743	39.11	105	7.08
68	27	1071	39.67	100	10.71
18	62	2682	43.26	273	9.82
7	14	612	43.71	86	7.12
81	38	1700	44.74	161	10.56
34	22	1000	45.45	140	7.14
12	41	1895	46.22	177	10.71
14	44	2037	46.30	159	12.81
44	3	143	47.67	38	3.76
8	18	904	50.22	121	7.47
3	29	1491	51.41	216	6.90
16	39	2113	54.18	200	10.56
32	28	1579	56.39	128	12.34
2	7	401	57.29	63	6.37
20	14	804	57.43	127	6.33
9	19	1094	57.58	144	7.60
21	2	119	59.50	32	3.72
65	35	2093	59.80	260	8.05
67	16	976	61.00	163	5.99
97	25	1653	66.12	210	7.87
1	18	1192	66.22	177	6.73
10	14	956	68.29	169	5.66
22	46	3161	68.72	262	12.06
99	21	1463	69.67	172	8.51
39	27	2068	76.59	238	8.69
13	27	2104	77.93	198	10.63
58	29	2303	79.41	229	10.06
62	9	753	83.67	121	6.22
19	42	3564	84.86	280	12.73
4	28	2551	91.11	261	9.77
17	52	4828	92.85	430	11.23
33	19	1876	98.74	237	7.92
47	25	2475	99.00	281	8.81
6	24	2476	103.17	282	8.78
36	23	2538	110.35	340	7.46
24	21	2333	111.10	243	9.60
41	14	1590	113.57	222	7.16
43	12	1533	127.75	232	6.61

TABLEAU 82 – PHÈDRE : mots en relation de rime

v1	*mot1*	*cat1*	*v2*	*mot2*	*cat2*
1	Théramène	Np	2	Trézène	Np
3	agité	Adj	4	oisiveté	Nc
5	père	Nc	6	chère	Adj
7	cacher	V	8	chercher	V
9	crainte	Nc	10	Corinthe	Np
11	bords	Nc	12	morts	Nc
13	Ténare	Np	14	Icare	Np
15	climats	Nc	16	pas	Nc
17	père	Nc	18	mystère	Nc
19	jours	Nc	20	amours	Nc

R^2_{101}

JOINTURE(
o2.numero_vers = o1.numero_vers + 1
)[occurrences ALIAS o1,
occurrences ALIAS o2]
↪
RESTRICTION(
o2.numero_vers % 2 = 0
ET o1.fin_syl = 12
ET o1.cat PARMI ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V')
ET o2.fin_syl = 12
ET o2.cat PARMI ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V')
)[<résultat$_1$>]
↪
PROJECTION(
o1.numero_vers TITRE 'v1',
o1.occ_car TITRE 'mot1'
o1.cat TITRE 'cat1'
o2.numero_vers TITRE 'v2',
o2.occ_car TITRE 'mot2',
o2.cat TITRE 'cat2'
) [<résultat$_1$>]

MySQL

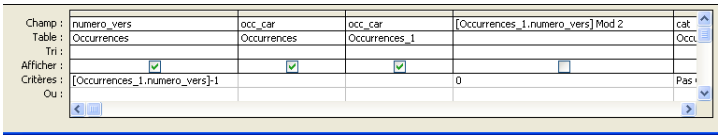
```
SELECT o1.numero_vers AS 'v1',
   o1.occ_car AS 'mot1',
   o1.cat AS 'cat1',
   o2.numero_vers AS 'v2',
   o2.occ_car AS 'mot2',
   o2.cat AS 'cat2'
FROM occurrences AS o1,
     occurrences AS o2
WHERE o1.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv',
          'Adj', 'Nc', 'Np', 'V')
   AND o2.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv',
    'Adj', 'Nc', 'Np', 'V')
```

```
AND o1.fin_syl = 12
AND o2.fin_syl = 12
AND o2.numero_vers % 2 = 0
AND o2.numero_vers = o1.numero_vers + 1 ;
```

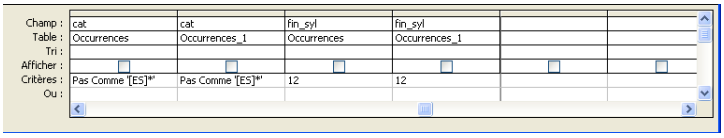
Access

Le nombre des critères oblige à fournir deux copies d'écran :

67



68



Solution de l'exercice n°5 p. 260 Un ajout minime à la requête précédente :

```
... ET  (o1.occ_car = 'yeux' OU  o2.occ_car = 'yeux') ;
```

fournit les couples de mots-rimes dans lesquels *yeux* est impliqué en première ou en deuxième position. Il suffit en effet de rajouter une disjonction indiquant qu'on veut soit que le premier mot soit *yeux* soit que le deuxième mot soit *yeux*. C'est la réalisation de la R_{72}^{1} *t. 1* p. 137.

MySQL

Par rapport à la requête abstraite, la réalisation en MySQL ajoute d'autres colonnes dans le résultat (tableau 83 p. 275). En particulier, elle ajoute le décompte des rimes observées, via un regroupement préalable :

```
SELECT o1.occ_car,
       o1.occ_phon,
       o1.cat,
       o2.occ_car,
       o2.occ_phon,
       o2.cat,
       COUNT(*) AS 'o.'
FROM occurrences AS o1,
       occurrences AS o2
WHERE o1.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj','Nc', 'Np', 'V')
      AND o2.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj','Nc', 'Np', 'V')
      AND o1.fin_syl = 12
      AND o2.fin_syl = 12
      AND o2.numero_vers % 2 = 0
      AND o2.numero_vers = o1.numero_vers + 1
      AND (o1.occ_car = 'yeux' OR o2.occ_car = 'yeux')
GROUP BY o1.occ_car, o2.occ_car ;
```

TABLEAU 83 – PHÈDRE : couples de rime avec *yeux*

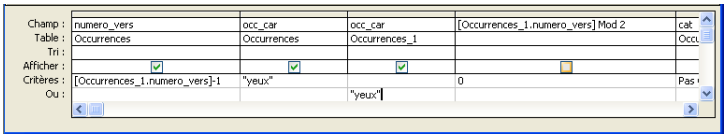
occ_car	*occ_phon*	*cat*	*occ_car*	*occ_phon*	*cat*	*o.*
ambitieux	an |b i |s i |eu	Nc	yeux	y eu	Nc	1
audacieux	oo |d a |s i |eu	Adj	yeux	y eu	Nc	1
cieux	s y eu	Nc	yeux	y eu	Nc	1
dieux	d y eu	Nc	yeux	y eu	Nc	1
furieux	f u |r i |eu	Adj	yeux	y eu	Nc	2
glorieux	g l o |r i |eu	Adj	yeux	y eu	Nc	1
odieux	o |d i |eu	Adj	yeux	y eu	Nc	4
prodigieux	p r o |d i |j i |eu	Adj	yeux	y eu	Nc	1
yeux	y eu	Nc	dieux	d y eu	Nc	1
yeux	y eu	Nc	furieux	f u |r i |eu	Adj	1
yeux	y eu	Nc	lieux	l y eu	Nc	3

L'absence de parenthèses autour de la disjonction finale conduirait le SGBD à chercher *yeux* comme deuxième mot mis en relation avec tous les autres vers de la table, soit . . . un résultat de 1 497 149 lignes (la requête correcte en produit 17).

Access

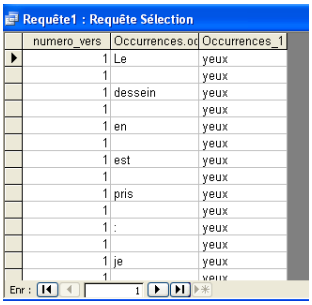
On ajoute une condition sur la colonne correspondant à l'attribut occ_car de la première table. La deuxième condition ne peut pas être mis sur la même ligne, mais cette fois-ci pour l'attribut occ_car de la deuxième table : ce serait chercher les paires de vers où *yeux* rime avec lui-même, ce qui est exclu en versification classique. On peut être tenté de rajouter la condition sur l'attribut occ_car de la deuxième table mais sur la ligne correspondant à un OU logique :

69



Le résultat n'est absolument pas celui qui est souhaité :

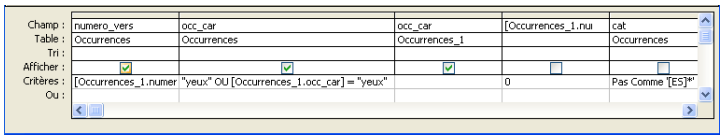
70



Cela revient en effet à chercher soit les mots en situation de rimes tels que le premier soit *yeux* soit à constituer les couples de *yeux* avec toutes les occurrences de la pièce.

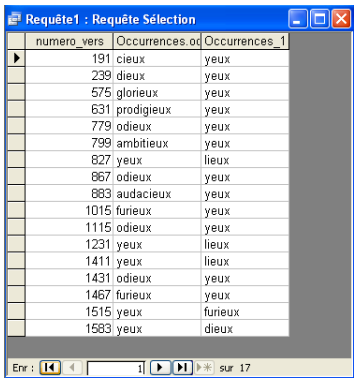
La requête corrigée :

71



donne le résultat souhaité :

72



L'équivalent SQL Server est :

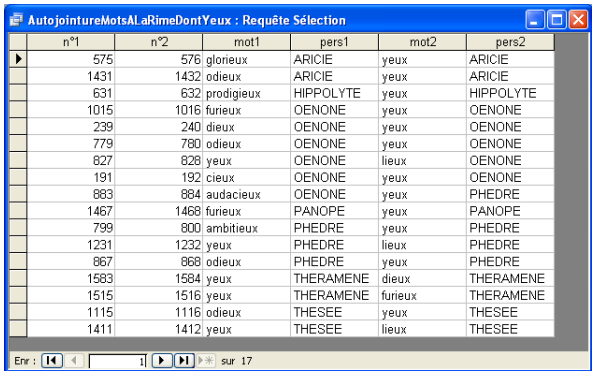
```
SELECT Occurrences.numero_vers,
       Occurrences.occ_car,
       Occurrences_1.occ_car
FROM Occurrences, Occurrences AS Occurrences_1
WHERE (((Occurrences.numero_vers)=[Occurrences_1.numero_vers]-1)
      AND ((Occurrences.cat) Not Like "[ES]*")
      AND ((Occurrences_1.cat) Not Like "[ES]*")
      AND ((Occurrences.fin_syl)=12)
      AND ((Occurrences_1.fin_syl)=12)
      And (([Occurrences_1.numero_vers] MOD 2)=0))
      AND (Occurrences.occ_car = 'yeux'
           OR Occurrences_1.occ_car = 'yeux');
```

Une variante de la requête SQL Server, dont le résultat figure en 73, ajoute le personnage responsable de chacun des mots en relation de rime et organise les résultats en triant par personnage :

```
SELECT Occurrences.numero_vers AS n°1,
       Occurrences_1.numero_vers AS n°2,
       Occurrences.occ_car AS mot1,
       Occurrences.personnage AS pers1,
       Occurrences_1.occ_car AS mot2,
       Occurrences_1.personnage AS pers2
...
ORDER BY Occurrences.personnage, Occurrences_1.personnage;
```

TABLEAU 84 – PHÈDRE : décompte des couples de rime avec *yeux*

v1 = yeux	*v1 ≠ yeux*	*v2 = yeux*	*v2 ≠ yeux*
5	12	12	5



AutojointureMotsALaRimeDontYeux : Requête Sélection

n°1	n°2	mot1	pers1	mot2	pers2
575	576	glorieux	ARICIE	yeux	ARICIE
1431	1432	odieux	ARICIE	yeux	ARICIE
631	632	prodigieux	HIPPOLYTE	yeux	HIPPOLYTE
1015	1016	furieux	OENONE	yeux	OENONE
239	240	dieux	OENONE	yeux	OENONE
779	780	odieux	OENONE	yeux	OENONE
827	828	yeux	OENONE	lieux	OENONE
191	192	cieux	OENONE	yeux	OENONE
883	884	audacieux	OENONE	yeux	PHEDRE
1467	1468	furieux	PANOPE	yeux	PANOPE
799	800	ambitieux	PHEDRE	yeux	PHEDRE
1231	1232	yeux	PHEDRE	lieux	PHEDRE
867	868	odieux	PHEDRE	yeux	PHEDRE
1583	1584	yeux	THERAMENE	dieux	THERAMENE
1515	1516	yeux	THERAMENE	furieux	THERAMENE
1115	1116	odieux	THESEE	yeux	THESEE
1411	1412	yeux	THESEE	lieux	THESEE

Enr : 1 sur 17

Solution de l'exercice n°6 p. 260 Le résultat constitue le tableau 84 p. 277. La requête se formule comme suit :

R^2_{102}

JOINTURE(
o2.numero_vers = o1.numero_vers + 1
)[occurrences ALIAS o1,
occurrences ALIAS o2]
↪ RESTRICTION(
o2.numero_vers % 2 = 0
ET o1.fin_syl = 12
ET o1.cat PARMI ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V')
ET o2.fin_syl = 12
ET o2.cat PARMI ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V')
ET (o1.occ_car = 'yeux' OU o2.occ_car = 'yeux')
)[<résultat$_1$>]
↪ PROJECTION(
SOMME(SI(o1.occ_car = 'yeux', 1, 0)) TITRE 'v1 = yeux',
SOMME(SI(o1.occ_car = 'yeux', 0, 1)) TITRE 'v1 <> yeux',
SOMME(SI(o2.occ_car = 'yeux', 1, 0)) TITRE 'v2 = yeux',
SOMME(SI(o2.occ_car = 'yeux', 0, 1)) TITRE 'v2 <> yeux'
) [<résultat$_1$>]

MySQL

```
SELECT SUM(IF(o1.occ_car = 'yeux', 1, 0)) AS 'v1 = yeux',
       SUM(IF(o1.occ_car = 'yeux', 0, 1)) AS 'v1 <> yeux',
       SUM(IF(o2.occ_car = 'yeux', 1, 0)) AS 'v2 = yeux',
       SUM(IF(o2.occ_car = 'yeux', 0, 1)) AS 'v2 <> yeux'
```

TABLEAU 85 – PHÈDRE : fréquence des voyelles en écho hémistiche-rime

Voy. écho	*o.*
ai	31
i	19
e	15
an	5
u	5
a	3
ou	3
o	3
w a	2
in	1
oe	1


```
FROM occurrences AS o1,
     occurrences AS o2
WHERE o1.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj','Nc', 'Np', 'V')
     AND o2.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj','Nc', 'Np', 'V')
     AND o1.fin_syl = 12
     AND o2.fin_syl = 12
     AND o2.numero_vers % 2 = 0
     AND o2.numero_vers = o1.numero_vers + 1
     AND (o1.occ_car = 'yeux'
          OR o2.occ_car = 'yeux') ;
```

Solution de l'exercice n°7 p. 260 Le résultat figure au tableau 85 p. 278. Il découle de la requête :

R^2_{103}

```
        JOINTURE(
        v.numero_vers = p1.numero_vers
        ET p1.numero_vers = p2.numero_vers
        )[vers ALIAS v,
        positions ALIAS p1,
        positions ALIAS p2]
↪       RESTRICTION(
        p1.position = 6
        ET p2.position = 12
        ET p1.voy = p2.voy
        )[<résultat1>]
↪
        REGROUPER SUR(p1.voy)[<résultat2>]
↪
        PAR GROUPE(
        p1.voy TITRE 'Voy. écho',
        NOMBRE DE LIGNES() TITRE 'o.'
        )[<résultat3>]
↪
        TRI SUR('o.' DÉCROISSANT)[<résultat4>]
```

MySQL

```
SELECT p1.voy AS 'Voy. écho',
     COUNT(*) AS 'o.'
FROM vers,
     positions AS p1,
     positions AS p2
WHERE vers.numero_vers = p1.numero_vers
     AND p1.numero_vers = p2.numero_vers
     AND p1.position = 6
     AND p2.position = 12
     AND p1.voy = p2.voy
GROUP BY p1.voy
ORDER BY 'o.' DESC ;
```

Access

Solution de l'exercice n°8 p. 261 Le principe de rapprochement des mots en relation de rime est le même que dans les requêtes précédentes. S'ajoute une condition de restriction : on retient les mots rimant dont le dernier phonème de la transcription phonétique diffère. On « renverse » pour cela la chaîne de caractères constituant la transcription, c'est-à-dire la valeur de l'attribut `occ_phon` et on garde le premier caractère que l'on peut alors comparer avec son équivalent pour l'autre mot. Le résultat (tableau 86 p. 280) montre d'un côté des « licences » répertoriées ou des sons proches, de l'autre des erreurs de transcription (pour *remords*, *longtemps*, *nom*).

R^2_{104}

JOINTURE(
o2.numero_vers = o1.numero_vers + 1
)[occurrences ALIAS o1, occurrences ALIAS o2]
↪ RESTRICTION(
o1.cat PARMI ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V')
Et o2.cat PARMI ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V') AND o1.fin_syl = 12
ET o2.fin_syl= 12
ET o2.numero_vers % 2 = 0
ET SOUSCHAINE(INVERSER(o1.occ_phon), 1, 2) <> SOUSCHAINE(INVERSER(o2.occ_phon), 1, 2)
)[<résultat$_1$>]
↪
PROJECTION(
o1.numero_vers TITRE 'v1',
o1.occ_car TITRE 'mot1',
o1.occ_phon TITRE 'phon1',
o2.occ_car TITRE 'mot2',
o2.occ_phon TITRE 'phon2',
SOUSCHAINE(INVERSER(o1.occ_phon), 1, 2) TITRE 'fin phon1',
SOUSCHAINE(INVERSER(o2.occ_phon), 1, 2) TITRE 'fin phon2'
)[<résultat$_2$>]
↪
Tri sur('fin phon1', 'fin phon2')[<résultat$_3$>]

TABLEAU 86 – PHÈDRE : rimes pas pour l'oreille

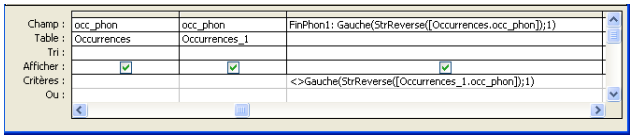
v1	*mot1*	*phon1*	*mot2*	*phon2*	*fin phon1*	*finphon2*
1635	remords	r @ | m o r d	morts	m o r	d	r
895	remords	r @ | m o r d	morts	m o r	d	r
1375	marcher	m a r | sh e	cher	sh ai r	e	r
79	punis	p u | n i	Sinnis	s i | n i s	i	s
1195	avis	a | v i	fils	f i s	i	s
951	finis	f i | n i	fils	f i s	i	s
1647	éclaircis	e | k l air | s i	fils	f i s	i	s
1487	suivis	s yw i | v i	fils	f i s	i	s
435	mépris	m e | p r i	fils	f i s	i	s
1123	asservis	a | s air | v i	fils	f i s	i	s
899	avis	a | v i	fils	f i s	i	s
1403	nom	n o m	Junon	j u | n on	m	on
1303	longtemps	l on | t an p	habitants	a | b i | t an	p	an
1227	transports	t r an s | p o r	remords	r @ | m o r d	r	d
395	fils	f i s	avis	a | v i	s	i
983	fils	f i s	ennemis	ai | n @@ | m i	s	i
1547	fils	f i s	nourris	n ou | r i	s	i
359	fils	f i s	bâtis	b aa | t i	s	i
755	Minos	m i | n oo s	repos	r @ | p oo	s	oo
467	éclatants	e | k l a | t an	longtemps	l on | t an p	an	p
159	noeuds	n oe	cheveux	sh @ | v eu	oe	eu
351	noeuds	n oe	feux	f eu	oe	eu
207	raison	r ai | z on	nom	n o m	on	m
107	rejeton	r @ | j @@ | t on	nom	n o m	on	m
643	flots	f l oo	Minos	m i | n oo s	oo	s

MySQL

```
SELECT o1.numero_vers AS 'v1',
      o1.occ_car AS 'mot1',
      o1.occ_phon AS 'phon1',
      o2.occ_car AS 'mot2',
      o2.occ_phon AS 'phon2',
      REVERSE(SUBSTRING(REVERSE(o1.occ_phon), 1, 2)) AS 'fin␣phon1',
      REVERSE(SUBSTRING(REVERSE(o2.occ_phon), 1, 2)) AS 'finphon2'
FROM occurrences AS o1,
      occurrences AS o2
WHERE o1.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V')
    AND o2.cat IN ('Dét/Pron', 'Conj', 'Prép', 'Rel', 'Adv', 'Adj', 'Nc', 'Np', 'V')
     AND o1.fin_syl = 12
     AND o2.fin_syl= 12
     AND o2.numero_vers % 2 = 0
     AND o2.numero_vers = o1.numero_vers + 1
     AND SUBSTRING(REVERSE(o1.occ_phon), 1, 1) != SUBSTRING(REVERSE(o2.occ_phon), 1,
          1)
ORDER BY 'fin␣phon1', 'fin␣phon2' ;
```

Access

En 74, la partie pertinente de la requête :



Elle a pour équivalent SQL Server :

```
SELECT Occurrences.numero_vers,
       Occurrences.occ_car,
       Occurrences_1.occ_car,
       Occurrences.occ_phon,
       Occurrences_1.occ_phon,
       Left(StrReverse([Occurrences.occ_phon]),1) AS FinPhon1,
       Left(StrReverse([Occurrences_1.occ_phon]),1) AS FinPhon2
FROM Occurrences,
       Occurrences AS Occurrences_1
WHERE (((Occurrences.numero_vers)=[Occurrences_1.numero_vers]-1)
       AND ((Left(StrReverse([Occurrences.occ_phon]),1))
            <>Left(StrReverse([Occurrences_1.occ_phon]),1))
       AND ((Occurrences.cat) Not Like '[ES]*')
       AND ((Occurrences_1.cat) Not Like '[ES]*')
       AND ((Occurrences.fin_syl)=12)
       AND ((Occurrences_1.fin_syl)=12)
       AND (([Occurrences_1.numero_vers] Mod 2)=0));
```

CHAPITRE VIII

ÉTUDE DE CAS 3 : LE SUFFIXE *-ESQUE*

1. Induire la « grammaire » du suffixe *-esque*

2. Formulaire initial

La structure en MySQL de la table esque figure au tableau 87 p. 283. On se reportera au ch. XV § 3 pour le détail des types de données offerts par MySQL et Access pour les attributs des tables.

3. Redondances et incohérences

≃ nombre d'attestations, de combinaisons distinctes de bases et de dérivés avec leurs catégories

R^{1}_{73} *t. 1* p. 147

PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'Attestations',
NOMBRE DE VALEURS DISTINCTES(derive, cat_derive) TITRE 'dérivés',
NOMBRE DE VALEURS DISTINCTES(base, cat_base) TITRE 'bases'
)
[esque]

≃ sens, commentaires des couples base-catégorie ayant plus d'une attestation

TABLEAU 87 – ESQUE : structure de la table esque (MySQL)

Field	*Type*	*Null*	*Key*	*Default*	*Extra*
numero_ligne	int(11)		PRI		auto_increment
derive	varchar(100) binary				
cat_derive	varchar(10) binary	YES			
sens_derive	varchar(100) binary	YES			
variantes_derive	varchar(255) binary	YES			
base	varchar(100) binary	YES			
cat_base	varchar(10) binary	YES			
sens_base	varchar(100) binary	YES			
commentaire_base	blob	YES			
mode_formation	varchar(100) binary	YES			
reference_dans_dictionnaire	varchar(50) binary	YES			
reference	varchar(255) binary	YES			
contexte	blob	YES			
auteur	varchar(100) binary	YES			
reference_complement	varchar(255) binary	YES			
jour	tinyint(4)	YES		0	
mois	tinyint(4)	YES		0	
annee	smallint(6)	YES		0	
annee_complement	varchar(100) binary	YES			
origine	varchar(100) binary	YES			

R^1_{74} *t. 1* p. 148

```
REGROUPER SUR(base, cat_base)[esque]
↪
PAR GROUPE(
base,
cat-base,
sens_base,
commentaire_base
NOMBRE DE LIGNES() TITRE 'o.',
)
AVEC (NOMBRE DE LIGNES() > 1)
[<résultat1>]
```

≃ couples dérivé-catégorie ayant plus d'une base

R^1_{75} *t. 1* p. 150

```
REGROUPER SUR(derive, cat_derive)[esque]
↪
PAR GROUPE(
derive,
NOMBRE DE LIGNES() TITRE 'o.',
NOMBRE DE VALEURS DISTINCTES(base) TITRE 'nbre bases'
)
AVEC (NOMBRE DE VALEURS DISTINCTES(base) > 1)
[<résultat1>]
```

La démarche d'élimination des redondances passe alors par plusieurs étapes. La première est celle du repérage des incohérences dans la table de départ (chapitre XI, § 2). Pour ne pas risquer de corrompre les informations initiales, on crée, dans une deuxième étape, un « miroir » de la table. C'est la table esque_tmp créée et modifiée au ch. XII. On modifie progressivement ce miroir (ajout et suppression de colonnes, altération de lignes). Mais on peut revenir à la table de départ en cas de problème.

△

Le modèle relationnel pousse à éliminer dans la plupart des cas les redondances des tables manipulées. Dans la pratique cependant, on peut volontairement constituer des tables conservant des redondances. C'est le cas par exemple quand on veut faire fréquemment appel à des informations dont la production via une jointure risque de ralentir la consultation.

MySQL

R^1_{73} *t. 1* p. 147

```
SELECT COUNT(*) AS 'Attestations',
       COUNT(DISTINCT derive, cat_derive) AS 'dérivés',
       COUNT(DISTINCT base, cat_base) AS 'bases'
FROM esque ;
```

R^1_{74} *t. 1* p. 148

```
SELECT base,
       cat_base,
       sens_base,
       commentaire_base,
       COUNT(base) AS 'o.'
FROM esque
GROUP BY base, cat_base
HAVING COUNT(base) > 1
ORDER BY 'o.' ;
```

R^1_{75} *t. 1* p. 150

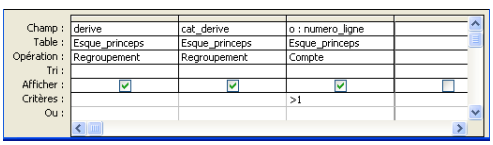
```
SELECT derive,
       COUNT(*) AS 'o.',
       COUNT(DISTINCT base) AS 'nbre_bases'
FROM esque
GROUP BY derive, cat_derive
HAVING COUNT(DISTINCT base) > 1;
```

Access

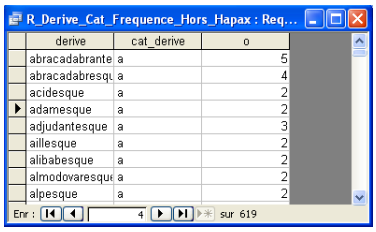
Acces ne dispose pas de l'équivalent de NOMBRE DE VALEURS DISTINCTES(<attribut>). Il faut donc passer par des « détours » pour arriver au résultat souhaité. Nous en donnons un exemple avec la réalisation de la R^1_{75} *t. 1* p. 150.

Une première étape consiste à associer au couple (<dérivé>, <catégorie>) sa fréquence, via un regroupement sur les deux attributs correspondants : 1. On peut d'emblée, en outre, se limiter aux couples qui ne sont pas des hapax. Seul un couple intervenant au moins 2 fois peut avoir des bases distinctes... On sauvegarde la requête correspondante dont le résultat figure en 2.

1

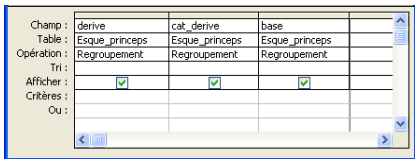


2

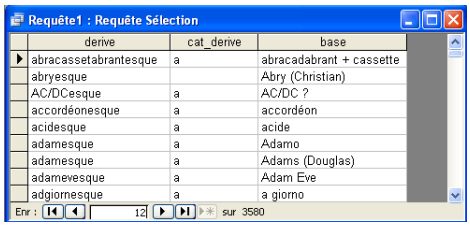


La deuxième étape consiste à constituer l'ensemble des triplets distincts (<dérivé>, <catégorie>, <base>), ce qui passe par un regroupement sur ces trois attributs l'un après l'autre en 3.

3



4

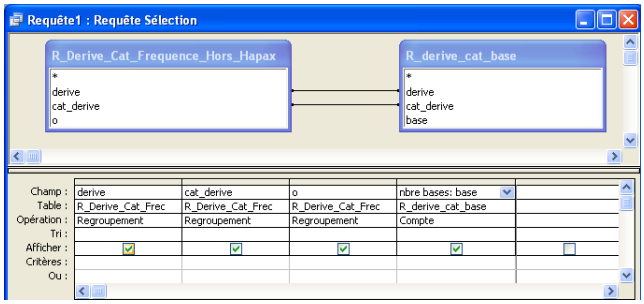


La troisième étape crée une requête à partir des deux requêtes précédentes : 5.

5

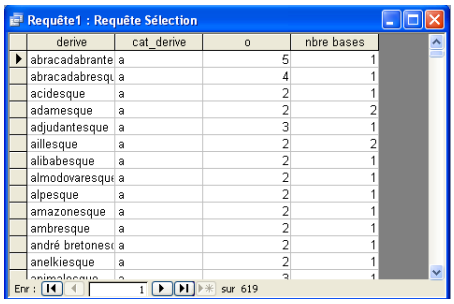
On établit en 6 une jointure entre les deux requêtes sur les attributs derive et cat_derive en faisant glisser avec le clic gauche le nom d'un attribut d'une requête sur son correspondant de l'autre. La nouvelle requête opère un regroupement sur derive, cat_derive et o (le nombre d'occurrences du couple (<dérivé>, <catégorie>). Ce regroupement permet de calculer le nombre de lignes de chaque groupe (COMPTE), c'est-à-dire en définitive le nombre de bases correspondant au couple.

6



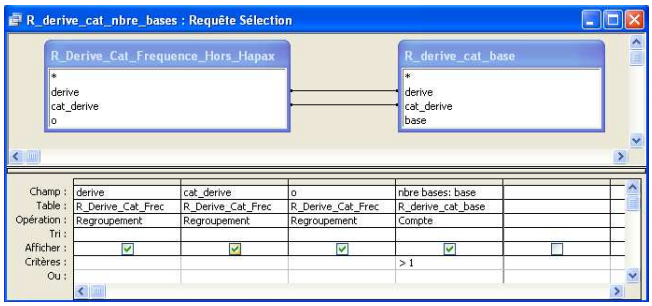
Dans le résultat 7, on voit ainsi qu'*adamesque* a 2 comme attribut nbre bases, ce qui renvoie aux deux bases présentes dans le résultat 3 : *Adamo* et *Adams (Douglas)*.

7

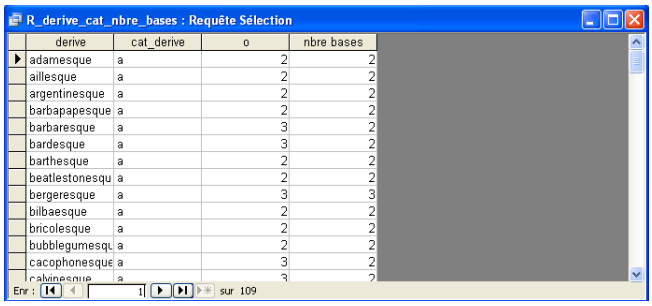


La requête 8 est une simple modification de la requête 6. On ajoute une contrainte de deuxième niveau. On ne garde que les regroupements pour lequel l'attribut nbre bases soit supérieur à 1, ce qui ne garde effectivement que les dérivés ayant des bases multiples, comme on le constate en 9.

8



9



La requête SQL Server sous-jacente à la requête Access 8 est la suivante :

```
SELECT R_Derive_Cat_Frequence_Hors_Hapax.derive,
       R_Derive_Cat_Frequence_Hors_Hapax.cat_derive,
       R_Derive_Cat_Frequence_Hors_Hapax.o,
       Count(R_derive_cat_base.base) AS [nbre bases]
FROM R_Derive_Cat_Frequence_Hors_Hapax
INNER JOIN R_derive_cat_base
      ON (R_Derive_Cat_Frequence_Hors_Hapax.cat_derive = R_derive_cat_base.cat_derive
          )
           AND (R_Derive_Cat_Frequence_Hors_Hapax.derive = R_derive_cat_base.derive)
GROUP BY R_Derive_Cat_Frequence_Hors_Hapax.derive,
       R_Derive_Cat_Frequence_Hors_Hapax.cat_derive,
       R_Derive_Cat_Frequence_Hors_Hapax.o
   HAVING (((Count(R_derive_cat_base.base))>1));
```

Exercice n°1 À partir de la R^1_{75} *t. 1* p. 150 et de la table esque_tmp dont la création et l'évolution sont détaillées au ch. XII, utilisez une requête enchâssée pour obtenir, pour les bases qui ont plusieurs dérivés distincts, les dérivés correspondants et leurs catégories.

Exercice n°2 En recourant à une requête enchâssée et à la table esque_tmp dont la création et l'évolution sont détaillées au ch. XII, fournir les dérivés, leurs catégories et leurs bases pour les dérivés qui ont plusieurs bases distinctes.

4. Solutions

Solution de l'exercice n°1 p. 287 324 bases sont assorties de dérivés multiples. Un extrait des résultats figure au tableau 88, p. 288.

Est utilisée la table esque_tmp dont la création et l'évolution sont détaillées au ch. XII. Des attributs supplémentaires engrangent des informations « nettoyées » par rapport à la table esque. Dans l'attribut base_normalisee, une marque **NULL** pour l'attribut base est remplacée par le dérivé précédé de la chaîne '> '. Cela veut dire que la base, inconnue, a servi à produire le dérivé en question. De manière similaire, la catégorie de la base est donnée parfois comme douteuse, ou bien se trouve assortie de traits morphologiques avec des différences de traitement selon les dérivés. Un attribut base_POS fournit donc la seule partie du discours (*Part of Speech*) du mot base.

Par commodité pédagogique, la requête enchâssée est exprimée à part et la requête enchâssante y réfère. Mais on pourrait remplacer tout aussi bien cette référence par la requête enchâssée elle-même, comme on le constate dans la version MySQL.

R^2_{105}

```
        REGROUPER SUR(
        base_normalisee,
        base_POS
        )[esque_tmp]
        AVEC (NOMBRE DE VALEURS DISTINCTES(derive_cat_derive)
        > 1)
↪       PROJECTION(base_normalisee)[<résultat1>]
```

TABLEAU 88 – ESQUE : bases à dérivés multiples (extraits)

base_normalisee	*derive*	*derive_POS*
> humoresque	humoresque	a
> humoresque	humoresque	n
Ali Baba	ali-babaesque	a
Ali Baba	Alibabesque	a
Ali Baba	alibababesque	a
Ali Baba	alibabesque	a
Ali Baba	ali-babesque	a
Almodovar	almodovardesque	a
Almodovar	almodovaresque	a
Amiga	amiganesque	a
Amiga	amigaesque	a
Amin Dada (Idi)	idi-amin-dadaesque	a
Amin Dada (Idi)	amindadesque	a
Anelka (Nicolas)	anelkiesque	a
Anelka (Nicolas)	Anelkesque	a
Arcimboldo (Guiseppe)	arcimboldesque	a
Arcimboldo (Guiseppe)	archimboldesque	a
Arlequin	arlequinesque	absente
Arlequin	arlequinesque	a
Arrabal (Fernando)	arrabalesque	a
Arrabal (Fernando)	Arrabalesque	a
Arrabal (Fernando)	arrabalesqua	absente
Arrivé (Michel)	arrivéesque	a
Arrivé (Michel)	arrivéesque	absente
Arétin (Pietro Aretino dit l')	aretinesque	a
Arétin (Pietro Aretino dit l')	arétinesque	a
Avignon	avignonesque	a
Avignon	avignonnesque	a
BD	BDesque	absente
BD	bédéesque	a
BD	BDesque	a
Bada	Badadesques	absente
Bada	Badadesque	n
Bada	Badatesques	absente
Barbara	barbaresque	a
Barbara	barbaresque	absente
Bardamu	bardamuesque	a
Bardamu	bardamesque	a
Bardot (Brigitte)	bardesque	a
Bardot (Brigitte)	bardotesque	a
Bardot (Brigitte)	brigidesque	absente
Bassano (Iacopo)	bassanesque	a
Bassano (Iacopo)	bassanesque	n
Ben Laden (Ousmane)	benladenesque	a
Ben Laden (Ousmane)	ben ladesque	a
Bergame	Bergamesque	n
Bergame	bergamesque	a
Bernhardt (Sarah)	sarahbernhardtesque	a
Bernhardt (Sarah)	sarahbernardtesque	a
Bernhardt (Sarah)	bernhardtesque	a
Berni (Francesco)	bernesque	absente
Berni (Francesco)	bernesque	a
Berni (Francesco)	berniesque	a

R^2_{106}
RESTRICTION(
base_normalisee PARMI (<résultat R^2_{105} p. 287)
)[esque_tmp]
↪ PROJECTION(
base_normalisee,
derive,
derive_POS
)[<résultat$_1$>]
↪
TRI SUR(base_normalisee)[<résultat$_2$>]

MySQL

```
SELECT DISTINCT
      base_normalisee ,
      derive ,
      derive_POS
FROM esque_tmp
WHERE base_normalisee  IN
      (SELECT base_normalisee
       FROM esque_tmp
       GROUP BY base_normalisee ,
            base_POS
             HAVING COUNT(DISTINCT derive_cat_derive) > 1)
ORDER BY base_normalisee  ;
```

Le mot-clé **IN** dans la clause **WHERE** attend une liste de valeurs. C'est pourquoi la requête enchâssée doit correspondre à une table ayant une seule colonne, qui constitue elle aussi une liste de valeurs. C'est pour cette raison qu'a été créé dans la table esque_tmp un attribut derive_cat_derive qui est la concaténation des attributs derive et cat_derive.

Solution de l'exercice n°2 p. 287 102 dérivés ont des bases multiples. Un extrait des résultats figure au tableau 89, p. 290. Pour les mêmes raisons qu'à l'exercice précédent, est utilisée la table esque_tmp et la requête enchâssée est énoncée séparément de la requête enchâssante dans laquelle elle figure.

R^2_{107}
REGROUPER SUR(
derive_cat_derive
)[esque_tmp]
AVEC (NOMBRE DE VALEURS DISTINCTES(base_normalisee)
> 1)
↪ PROJECTION(derive_cat_derive)[<résultat$_1$>]

R^2_{108}
RESTRICTION(
derive_cat_derive PARMI (<résultat R^2_{107} p. 289)
)[esque_tmp]
↪ PROJECTION(
derive,
derive_POS,
base_normalisee
)[<résultat$_1$>]
↪
TRI SUR(derive, derive_POS)[<résultat$_2$>]

TABLEAU 89 – ESQUE : dérivés à bases multiples (extraits)

derive	*derive_POS*	*base_normalisee*
adamesque	a	Adamo
adamesque	a	Adams (Douglas)
aillesque	a	aile
aillesque	a	ail
ambresque	a	ambre
ambresque	a	Ambre
argentinesque	a	Argentine
argentinesque	a	Argentin
barbapapesque	a	barbe à papa
barbapapesque	a	barba à papa
barbaresque	a	Barbara
barbaresque	a	Barbarie
bardesque	a	Bardot (Brigitte)
bardesque	a	barde
beatlestonesque	a	Beatles (les)
beatlestonesque	a	Beatles (les), Rolling Stones
bergeresque	a	Berger (Michel)
bergeresque	a	Bergier (Jacques)
bergeresque	a	Bergerac (Cyrano de)
bibiesque	a	Bibi
bibiesque	a	BiBi
bilbaesque	a	Bibao
bilbaesque	a	Bilba (Jim)
bricolesque	a	bricol(é)e ()
bricolesque	a	bricole
bubblegumesque	a	bubblegum
bubblegumesque	a	Bubblegum (James)
cacophonesque	a	cacophie
cacophonesque	a	cacophonie
calvinesque	a	Calvin (Jean Cauvin, dit)
calvinesque	a	Calvin
cambronnesque	a	Cambronne (Pierre)
cambronnesque	a	Cambronne (Pierre Jacques Etienne)
canardesque	a	canard
canardesque	a	Canard enchaîné (Le)
canardesque	a	Canardo
capitalesque	a	capital
capitalesque	a	capitale
carrachesque	a	Carrache
carrachesque	a	Carrache (les)
charentonnesque	a	Charenton 1
charentonnesque	a	Charenton 2
charivaresque	a	charivari
charivaresque	a	Charivari (le)
chattesque	a	chatte
chattesque	a	chat
chaudronnesque	a	chaudron
chaudronnesque	a	Chaudron
chopinesque	a	Chopin (Frédéric)
chopinesque	a	Chopinot (Régine)
clintonesque	a	Clinton ()
clintonesque	a	Clinton (Bill)
clochardesque	a	clachard
clochardesque	a	clochard
conesque	a	con
conesque	a	cone
cuivresque	a	cuivre
cuivresque	a	cuivresque

MySQL

```
SELECT DISTINCT derive,
       derive_POS,
       base_normalisee
FROM esque_tmp
WHERE derive_cat_derive IN
       (SELECT derive_cat_derive
        FROM esque_tmp
        GROUP BY derive_cat_derive
              HAVING COUNT(DISTINCT base_normalisee) > 1)
ORDER BY derive, derive_POS ;
```

CHAPITRE IX

MODÉLISATION

1. Le modèle Entité/Association (E/A)

PRÉMA

En décomposant le domaine sous-jacent à PRÉMA en énoncés élémentaires, on obtiendrait par exemple :

1. Une infirmière rédige une fiche
2. Une fiche concerne un bébé
3. Une fiche comporte des occurrences
4. ...

Ces énoncés mettent en évidence l'existence d'au moins trois types d'entités : les infirmières, les fiches, les bébés. Chaque instance d'une entité est distincte des autres du même type. Chaque type d'entité doit donc être doté d'un attribut permettant de l'identifier. Pour les infirmières et les bébés, ce pourrait être le numéro de sécurité sociale, mais aussi l'association du nom, du prénom et de la date de naissance, ou encore, comme dans la base de données PRÉMA, un identifiant unique arbitraire.

En ce qui concerne le nombre d'entités mises en jeu par chaque association ou cardinalité, une infirmière donnée rédige au minimum 1 fiche et en rédige n au maximum, tandis qu'une fiche est rédigé par une infirmière donnée et une seule. On parle d'association *un-à-plusieurs* ou *un-à-n* (à une infirmière, peuvent correspondre n fiches). De la même manière, une fiche porte sur un bébé déterminé et un seul, mais par contre à un bébé correspond un minimum d'une fiche et un maximum de n fiches (théoriquement $n \leq 12$ – 4 jours $\times$ 3 fiches). C'est encore une association *un-à-plusieurs*. Il en va de même pour une fiche et les occurrences qui la composent. Par contre, l'association entre la signalétique d'une fiche et un état donné de

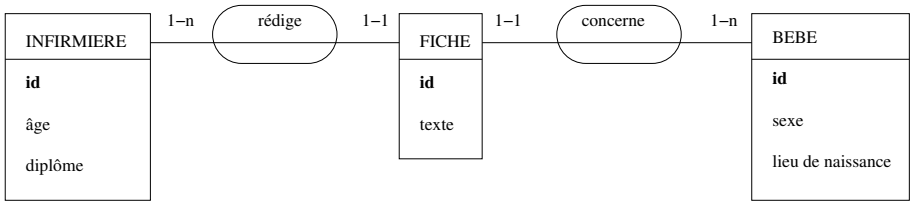


FIGURE 6 – PRÉMA : modèle Entité/Association

cette fiche (état de départ, état normalisé, état lemmatisé, etc.) est une association *un-à-un* : à la signalétique d'une fiche correspond une fiche et une seule.

La figure 6 p. 293 fournit une telle représentation pour un sous-ensemble du domaine de PRÉMA. Les cardinalités sont portées sur les traits reliant les entités : on voit que, par l'association concerne, une instance de BEBE est associée à 1 ou n instances (mention 1-n) de FICHE tandis qu'une instance de FICHE l'est avec 1 et une seule de BEBE (mention 1-1). Un cas de figure n'est pas représenté dans cet exemple : n-n (ou association *plusieurs-à-plusieurs*).

PHÈDRE

Sont toutes des associations *un-à-n* les associations entre un vers et les positions métriques qui le constituent, entre un vers et les occurrences qui le forment, entre une occurrenc et les positions en lesquelles elle se décomposent. Par contre, ressortit aux associations *un-à-un* l'association entre un vers graphique et sa transcription phonétique.

△

Une association peut être établie entre un type d'entité et lui-même.

La relation rime est un exemple d'association entre un type d'entité et lui-même : elle associe soit deux instances de vers soit deux instances d'occurrences.

Le vers 463 de *Phèdre* est « partagé » entre la scène I et la scène II de l'acte II. La représentation choisie ne permet pas de rendre compte de cette situation, puisque le rattachement à un acte et à une scène est opéré globalement, pour le vers entier, dans la table vers. Les attributs acte et scene sont dans l'immédiat des attributs de l'entité vers. Il serait plus judicieux de les rattacher à l'entité occurrences. Une occurrence se rattache de manière univoque à une scène d'un acte, un vers non. Dans *Phèdre*, il n'y a qu'une seule exception au rattachement d'un vers à une scène et une seule. Des comédies en vers comme *Le Tartuffe* obligeraient à une autre modélisation. Le partage d'un vers entre deux scènes voire entre deux actes y est commun.

ESQUE

2. Un modèle Entité/Association pour les mots en *-esque*

3. Relativité des schémas Entité/Association (E/A)

PRÉMA

Certaines entités ne présentent pas assez d'existence autonome ou sont trop marginales pour être conservées en tant que telles : il vaut mieux les représenter sous forme d'attributs.

C'est le cas de l'état global d'un bébé un jour donné. Il associe le score médical (SM) et la classe correspondant à ce score ainsi que le lieu où est le bébé : dans le service ; sorti de l'hôpital ou dans un autre service ; décédé. À une instance de BEBE pourrai(en)t être associée(s) n ($n = 4$) instances de ETAT_BEBE. Ces entités auraient les attributs jour, SM, Classe_SM et lieu. Il a toutefois paru plus naturel d'attacher ces attributs aux entités de type BEBE, en distinguant les jours en cause.

À l'inverse, certaines entités peuvent absorber, via des attributs, des entités qui pourraient exister en tant que telles. On a ainsi pu, comme dans la figure 6 p. 293 représenter le texte d'une fiche comme l'un des attributs de l'entité FICHE. Un tel choix est cohérent avec une analyse donnant plus de place aux données factuelles et n'utilisant le texte qu'à des fins illustratives. La place donné aux impressions des infirmières, la multiplicité des représentations fournies pour chacune d'entre elles a poussé à faire également de chaque état du texte une entité à part entière (tableau 5, p. 30).

Exercice n°1 Fournir le nombre minimum et le nombre maximum d'entités effectifs du côté n de l'association rédige du modèle E/A telle qu'elle est réalisée dans PRÉMA.

Exercice n°2 Donner le nombre minimum et le nombre maximum de fiches qui ont été consacrées à un bébé, ainsi que les bébés concernés. Par jour, donner le maximum de fiches consacrées à un bébé et le ou les bébé(s) correspondant(s).

PHÈDRE

ESQUE

Le repérage des références dans des dictionnaires à des dérivés s'effectue via :

R^2_{109}

```
    RESTRICTION(
    reference_dans_dictionnaire EST PAS NULL
    [esque]
↪   REGROUPER SUR(
    reference_dans_dictionnaire
    [<résultat1>]
↪   PAR GROUPE(
    reference_dans_dictionnaire,
    NOMBRE DE LIGNES() TITRE 'o.'
    )
    [résultat2>]
↪   TRI SUR('o')[<résultat3>]
```

Il met en évidence le faible nombre de dérivés pourvus d'une telle information : 74 :

reference_dans_dictionnaire	*o.*
DSA (sv), 1993	67
Björkman (sv)	3
DSA (sv caramboleur),1993	1
DSA (sv) 1993	1
DSA (sv région), 1993	1

DSA renvoie à l'ouvrage : Le Doran, Serge, Frédéric Pellou & Philippe Rosé, *Dictionnaire San-Antonio*, Paris : Fleuve Noir, 1993. On note d'ailleurs qu'un des renvois à cet ouvrage se trouve séparé des 67 autres, pour cause de virgule manquante. . . L'abréviation *sv* signifie *sub verbo*, c'est-à-dire sous l'entrée considérée. Le dérivé *chatounesque* se trouve ainsi sous l'entrée *région* du *DSA*. Björkman réfère à une thèse suédoise sur les dérivés en -esque : Björkman, Sven, *'L'incroyable, romanesque, picaresque épisode barbaresque'. Etude sur le suffixe français -esque et sur ses équivalents en espagnol, italien et roumain, Stockholm* : Almqvist & Wiksell, 1984.

L'attribut reference_dans_dictionnaire ne s'impose pas (cf. tome 1). Sans doute faudrait-il distinguer les références dans des dictionnaires de langue, du type *Le Petit Robert*, des mentions dans des livres consacrés à un idiolecte, comme le *Dictionnaire San-Antonio*, ou à des études linguistiques, comme celui de Björkman. △

MySQL

La requête précédente se réalise en MySQL par :

```
SELECT reference_dans_dictionnaire ,
     COUNT(*) AS 'o.'
FROM esque
WHERE reference_dans_dictionnaire IS NOT NULL
GROUP BY reference_dans_dictionnaire
ORDER BY 'o.' DESC ;
```

4. D'un schéma E/A à celui d'une base de données

PRÉMA

PHÈDRE

ESQUE

5. Normalisations

6. ⊕ Modélisation et points de vue

On se souvient du livre de Jules Verne, *Vingt mille lieux sous les mers*. Une expédition est organisée pour examiner et chasser un « monstre marin » qui s'en prend aux navires. Il s'agit en fait du sous-marin *Le Nautilus*, du capitaine Nemo. Le double objectif, scientifique et de lutte, de l'expédition explique que figurent parmi ses membres d'une part M. Aronnax, Professeur au Museum d'Histoire Naturelle de Paris, chargé du premier volet de l'opération et d'autre part, Ned Land, harponneur de baleines de son état, et responsable du deuxième volet. Jules Verne met en scène un dialogue entre Conseil, serviteur de M. Aronnax et « classificateur enragé » d'animaux, comme lui, d'une part, et Ned Land d'autre part. Ce dialogue oppose précisément deux points de vue sur la classification des animaux, qui relèvent de deux modélisations distinctes.

- Ami Ned, vous êtes un tueur de poissons, un très habile pêcheur. Vous avez pris un grand nombre de ces intéressants animaux. Mais je gagerais que vous ne savez pas comment on les classe.

- Si, répondit sérieusement le harponneur. *On les classe en poissons qui se mangent et poissons qui ne se mangent pas.*

- Voilà une distinction de gourmand, répondit Conseil. *Mais dites-moi si vous connaissez la différence qui existe entre les poissons osseux et les poissons cartilagineux ?*

- Peut-être bien, Conseil.

- Et la subdivision de ces deux grandes classes ?

- Je ne m'en doute pas, répondit le Canadien.

- Eh bien ! ami Ned, écoutez et retenez ! Les poissons osseux se subdivisent en six ordres : primo, les acanthoptérygiens, dont la mâchoire supérieure est complète, mobile et dont les branchies affectent la forme d'un peigne. Cet ordre comprend quinze familles, c'est-à-dire les trois quarts des poissons connus. Type : la perche commune.

- Assez bonne à manger, répondit Ned Land.

6.1. ⊕ Grain d'analyse variable des données textuelles : du « mot » à l'« énoncé »

Pour PHÈDRE comme pour PRÉMA, le choix a été fait de faire coexister des « grains » d'analyse des données textuelles de taille différentes, le vers et l'occurrence d'une part, le texte d'une fiche et les occurrences qui le constituent d'autre part. La même réalité textuelle est donc conceptualisée à chaque fois de deux manières différentes, ce qui donne lieu à deux types d'entités dans le modèle Entité/Association correspondant.

PHÈDRE

L'analyse a pu laisser dans l'ombre des entités ou des attributs pourtant importants. Pour PHÈDRE, une entité PERSONNAGE permettrait d'ajouter à chaque personnage des propriétés comme le sexe, le statut (personnage principal *vs.* secondaire ou roi/reine/prince(esse) *vs.* confident). Les regroupements issus des propriétés d'une telle entité apporteraient éventuellement un autre regard sur les répartitions des catégories, des longueurs de phrase, etc.

7. Solutions

Solution de l'exercice n°1 p. 294 Cela revient à déterminer dans le premier cas les nombres minimum et maximum de fiches rédigées par une infirmière et dans le second les nombres minimum et maximum de fiches concernant un bébé. Le minimum de fiches rédigées par une infirmière est de 2 et le maximum de 62. Le minimum de fiches pour un bébé est 1 et le maximum observé est 13, donc supérieur au maximum théorique de 12 (3 fiches $\times$ 4 jours).

Le regroupement des fiches par identifiant d'infirmière permet de calculer le nombre de fiches de chaque agrégat, c'est-à-dire de chaque infirmière. Lorsqu'on trie ces nombres par ordre croissant et que l'on garde la première ligne, on obtient le nombre minimal.

R^2_{110}

REGROUPER SUR(id_infirmieres)[fiches_originelles]
↪ PAR GROUPE(
NOMBRE DE LIGNES() TITRE 'f.'
)[<résultat$_1$>]
↪
TRI SUR('f.')[<résultat$_2$>]
↪
LIMITÉ À(1)[<résultat$_3$>]

Si l'on trie par ordre décroissant, on obtient le nombre maximal.

MySQL

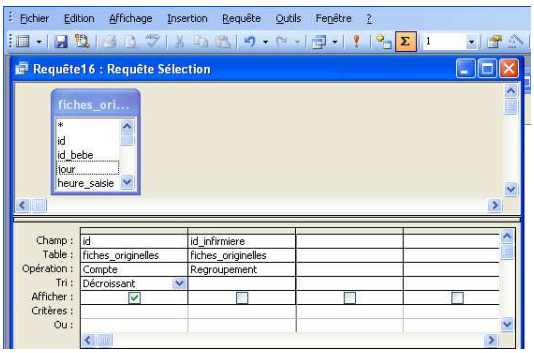
R^2_{110} p. 297

```
SELECT COUNT(*) AS 'f.'
FROM fiches_originelles
GROUP BY id_infirmiere
ORDER BY 'f.'
LIMIT 1 ;

SELECT COUNT(*) AS 'f.'
FROM fiches_originelles
GROUP BY id_infirmiere
ORDER BY 'f.' DESC
LIMIT 1 ;
```

Access

En 1, la requête Access pour obtenir le nombre maximum de fiches par infirmière. On voit que dans la barre de menu du haut, on a indiqué 1 comme le nombre de lignes qu'on voulait voir figurer dans la feuille de données résultante, ce que manifeste 2. On aurait pu également formuler cette contrainte en utilisant le menu [Paramètres] utilisable via un clic droit sur la partie haute de la formulation de la requête.

1



Solution de l'exercice n°2 p. 294 La démarche est la même que dans la R^2_{110} p. 297 pour les fiches concernant un bébé. On change simplement le regroupement initial, qui s'opère sur l'attribut id_bebe et non sur l'attribut id_infirmiere.

MySQL

En classant les bébés par nombre croissant de fiches associées, on constate que les bébés 66 et 103 n'ont qu'une seule fiche.

```
SELECT id_bebe,
     COUNT(id) AS 'o.'
FROM signaletique_fiches
GROUP BY id_bebe
ORDER BY 'o.'
LIMIT 10 ;
```

En opérant de la même manière, mais par ordre décroissant de fiches associées, on isole le bébé 120 avec 13 fiches.

```
SELECT id_bebe,
     COUNT(id) AS 'o.'
FROM signaletique_fiches
GROUP BY id_bebe
ORDER BY 'o.' DESC
LIMIT 10 ;
```

Rajouter un regroupement par jour permet de repérer le bébé 105, auquel sont consacrées 6 fiches le jour 3.

```
SELECT id_bebe,
     COUNT(id) AS 'o.'
FROM signaletique_fiches
GROUP BY id_bebe, jour
ORDER BY 'o.' DESC
LIMIT 10 ;
```

CHAPITRE X

CRÉER ET PEUPLER UNE BASE DE DONNÉES

On appelle **schéma d'une base de données** la définition de sa structure en termes de tables, d'attributs de ces tables (types de valeurs, caractère facultatif ou obligatoire, valeur par défaut), d'identifiant primaire, etc. Cette « ossature » de la base de données est peu volumineuse, par opposition au contenu. Le schéma de départ de ESQUE comprend simplement une seule table, dont la spécification en SQL occupe une vingtaine de lignes. Par contre, cette table contient plus de 4 000 attestations avec les informations associées.

Le schéma évolue *a priori* peu, puisqu'il doit être modifié uniquement lorsqu'on entend altérer la représentation du monde visée, tandis que souvent les tables gagnent, perdent en volume et voient leur contenu amendé. La situation diffère selon la base de données envisagée. Les tables de ESQUE, par définition, sont « ouvertes » : on souhaite ajouter les nouvelles attestations rencontrées, les bases correspondantes si elles n'y figurent pas, etc. Les tables de PHÈDRE et de PRÉMA peuvent nécessiter des corrections (chapitre XI, p. 2), mais elles n'attendent pas de nouvelles entités (il y a peu de chances que la tragédie *Phèdre* s'enrichisse de nouveaux vers. . .). Par contre, ces deux bases, tout comme ESQUE, peuvent gagner à l'ajout de nouvelles tables pour modéliser d'autres volets du domaine d'application.

1. Créer/modifier/supprimer une base de données

Une base de données doit avoir été préalablement créée comme un réceptacle vide pour qu'on puisse y ajouter des tables. Modifier une base de données revient à ajouter, modifier et supprimer des tables, ainsi qu'à ajouter, modifier ou supprimer des entités dans les tables.

MySQL

La création d'une base, sous MySQL, relève d'une commande du type :

CREATE DATABASE <nom> ;

Sous MySQL, la suppression d'une base, via la commande :

DROP DATABASE <nom> ;

entraîne la disparition de toutes les tables qu'elle peut contenir. C'est donc une opération à utiliser avec mesure.

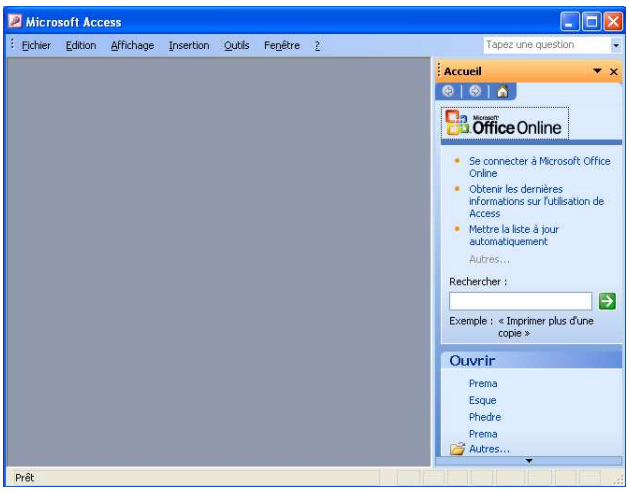
Access

Il faut tout d'abord lancer l'application Access, dont le nom est associée à l'icône d'une clé :

1

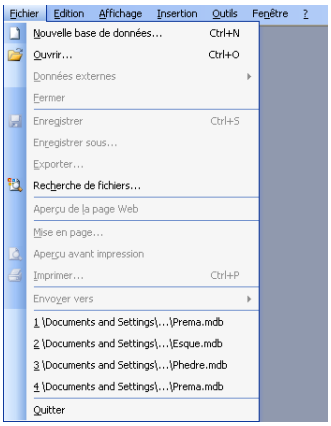
On obtient alors un écran d'accueil :

2



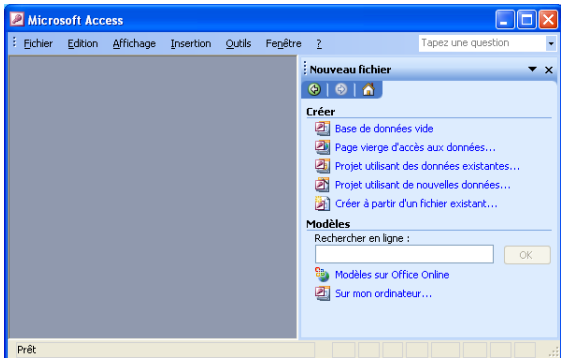
Le choix [Fichier | Nouvelle base de données] permet de créer effectivement une nouvelle base de données :

3



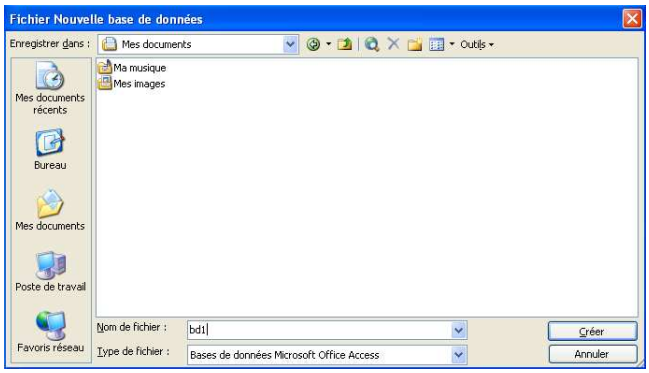
Il faut alors choisir le type de création souhaitée, ici une base de données vide :

4



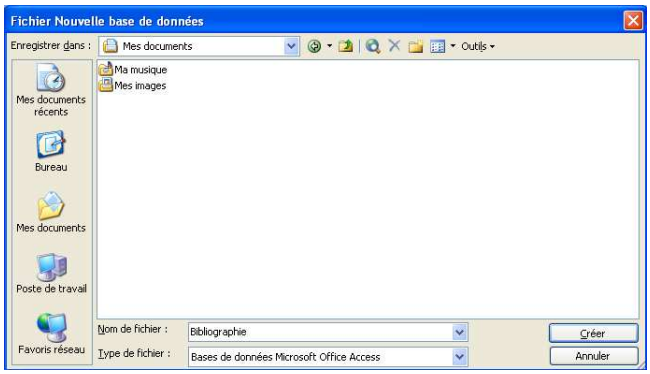
On doit alors sauvegarder le fichier (d'extension .mdb pour Microsoft Data Base, extension ajoutée automatiquement) dans le dossier souhaité. Un nom prédéfini, ici bd1, est proposé :

5



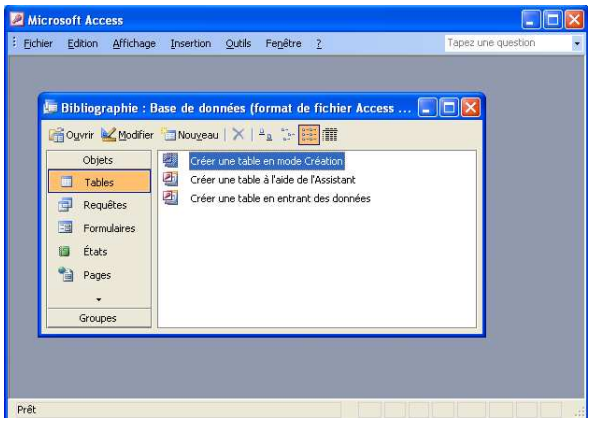
On choisit le nom qui paraît rendre compte clairement des objectifs de la base en cours de création :

6



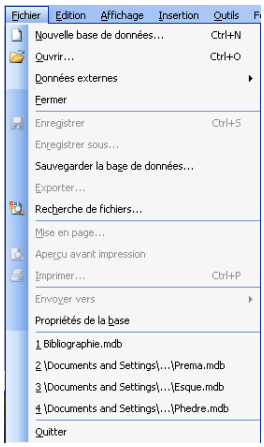
Le fichier (ici Bibliographie.mdb) sauvegardé, la base correspondante est ouverte, vide :

7



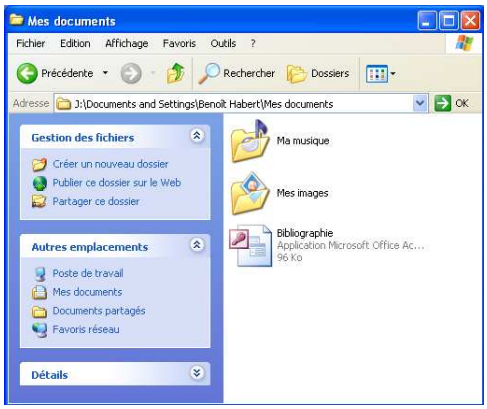
Le menu ⌈Fichier⌋ donne accès aux opérations fondamentales : fermer la base courante, en ouvrir une autre, ouvrir une des dernières bases ouvertes et dont les noms sont proposés, quitter Access (et fermer par là-même la base courante), etc. :

8



La suppression d'une base de données s'effectue une fois Access quitté via ⌈Fichier | Quitter⌋. On ouvre le dossier dans lequel a été sauvegardée la base, reconnaissable à l'icône d'Access (la clé) :

9

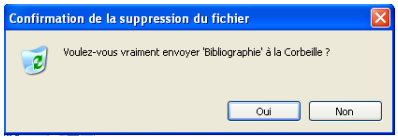


Un clic droit sur cette icône ouvre un menu contextuel où choisir ⌈Supprimer⌋ :

10

Il faut confirmer ce choix :

11

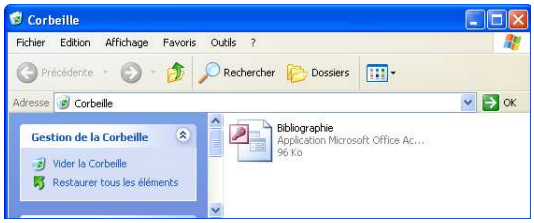


Le fichier constituant la base de données est déplacé dans la corbeille, dont l'icône manifeste qu'elle n'est pas vide :

12

Ouvrir la corbeille montre le fichier en instance de suppression définitive. Choisir ⌈Fichier | Vider la corbeille⌋ fait disparaître le fichier.

13



PRÉMA

PHÈDRE

ESQUE

2. Créer/modifier/supprimer une structure de table

Les opérations qui concernent le contenu d'une table font l'objet du § 3, p. 312.

2.1. ⊕ Créer une table

Le nom d'une table, comme celui de ses attributs, résulte d'un compromis entre un renvoi le plus explicite possible à l'entité/l'attribut correspondant(e) et la commodité d'utilisation.

On retient, pour chaque attribut, dans la palette des types proposés par le SGBD, celui qui semble le plus pertinent pour bien rendre compte de la propriété visée. Les tailles des types doivent être suffisantes pour que les valeurs attendues « tiennent » (une taille trop courte entraîne la troncation de ce qui « dépasse »), sans pour autant tomber dans le risque inverse : « gâcher » de la place disque et ralentir le traitement des données. Ainsi, le contexte le plus long de la table esque initiale fait 1 087 caractères, soit plus d'une centaine de mots.

MySQL

La création d'une table s'effectue par la requête :

```
CREATE TABLE <nom> (
  <définition de colonne>+
  <définition de clé primaire>?) ;
```

CREATE TABLE est suivi du nom de la table à créer, puis entre parenthèses, des descriptifs des attributs, séparés par des virgules, et d'indications valant pour la table entière (comme la mention **PRIMARY KEY**. . .).

Pour chaque attribut, on trouve son nom (par exemple derive), son type (**VARCHAR**, **BLOB**, etc.) et des paramètres réglant le comportement de cet attribut.

```
<définition de colonne> →
  <nom de la colonne>
  <type et longueur éventuelle>
  <caractère obligatoire>?
  <valeur par défaut>?

où
<caractère obligatoire> → NOT NULL

et

<valeur par défaut> → DEFAULT <valeur par défaut>
```

Certains attributs doivent obligatoirement être renseignés, ce que manifeste la mention **NOT NULL**. C'est obligatoirement le cas de l'attribut ou de la combinaison d'attributs qui constitue l'identifiant unique ou clé primaire de la table. La mention
PRIMARY KEY(numero_ligne)
précise cette fonction.

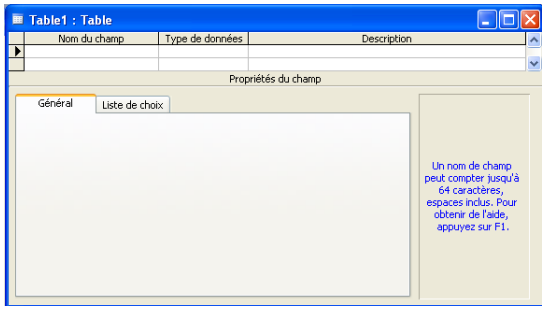
Access

Dans la fenêtre de navigation dans la base de données courante, l'onglet ⌈Tables⌉ donne accès au choix ⌈Créer une table en mode création⌉ :

14

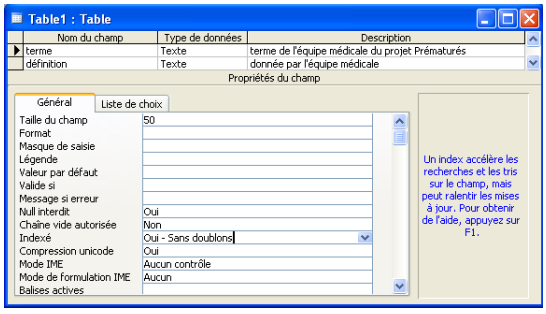
On obtient alors l'écran de création interactive de la structure d'une table :

15



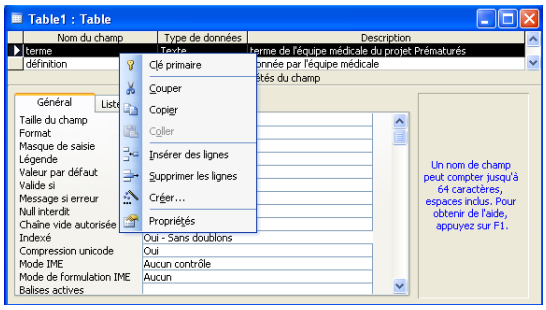
L'exemple choisi est celui d'une table destinée à accueillir le glossaire constitué par l'équipe médicale (ch. I § 11). Le haut de l'écran est destiné à fournir le nom, le type (via un menu déroulant), et une description (des précisions) sur chaque attribut jugé nécessaire, qui occupe une ligne dans cette partie de l'écran. Le bas de l'écran permet de préciser, pour l'attribut courant, ses caractéristiques et son fonctionnement. Dans le cas présent, on a choisi deux attributs, terme et définition. Le bas de l'écran indique les choix qui ont été faits pour l'attribut courant, terme, marqué par le triangle sur la gauche : interdiction de **NULL** (autorisé par défaut), interdiction de la chaîne vide (il ne peut pas y avoir d'entrée de la table sans que l'attribut terme soit renseigné par une chaîne non vide, alors qu'un attribut de type texte accepte par défaut la chaîne vide comme valeur), indexation sans doublons (par défaut pas d'indexation) :

16



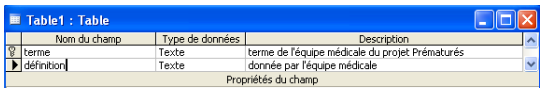
Sélectionner un attribut par un clic gauche donne accès à un menu contextuel par clic droit, qui permet en particulier de faire de l'attribut choisi la clé primaire :

17



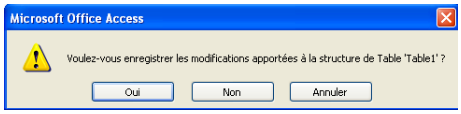
La marque de ce choix est la petite clé qui s'ajoute à gauche du nom de l'attribut :

18



Lorsqu'on ferme l'écran de définition d'une table en cliquant sur la case rouge en haut à droite, on se voit demander si l'on souhaite enregistrer la table qui vient d'être définie :

19

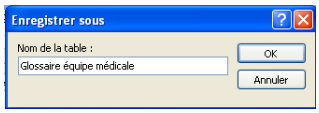


Une réponse positive amène sur un écran permettant de choisir le dossier de destination et proposant un nom par défaut :

20

On choisit alors un nom plus approprié :

21



La table effectivement sauvegardée (réduite à sa structure et donc vide) est visible dans l'onglet ⌈Tables⌉ de la feuille de navigation dans la base de données courante.

22

Exercice n°1 Donner la longueur maximale des attributs `occ_car` et `occ_phon` de la table `occurrences`. Comparer avec la taille allouée à ces attributs dans la définition de la table.

2.2. ⊕ Modifier une table

Si la colonne est facultative, l'ajout d'une colonne s'effectue sans contrainte. Si elle est obligatoire, soit la table doit être vide soit la colonne doit être accompagnée d'une valeur par défaut.

On ne peut supprimer directement une colonne qui intervient dans la clé primaire : on risquerait de modifier profondément la nature de la relation (ce qui distingue les éléments qui y figurent). La suppression d'une colonne participant à une clé étrangère est elle aussi soumise à conditions.

La suppression d'une colonne fait disparaître toutes les informations qui y figuraient.

MySQL

La requête :

```
ALTER TABLE esque
    ADD COLUMN datation DATE AFTER annee ;
```

change la table esque (**ALTER**). Elle lui ajoute une nouvelle colonne de type **DATE**, après la colonne annee (**AFTER**). L'indication sur l'endroit où insérer une nouvelle colonne est d'ailleurs optionnelle. Si elle est abstente, la nouvelle colonne se place en dernier. Le type **DATE** accepte des dates au format américain (AAAA-MM-JJ) entre le 1er janvier 1000 et le 31 décembre 9999 (dans le calendrier grégorien). Par contre, comme on l'a indiqué au chapitre VI, § 3, p. 220, le filtrage de dates impossibles est fruste : 2006-00-00 est une date tout à fait acceptable. En l'occurrence, ce laxisme se révèle bénéfique : pour une attestation issue d'un livre, l'année de publication suffit, et donner comme jour et mois 0 n'affaiblit pas la référence mémorisée.

Une colonne est obligatoire si sa définition comporte la mention **NOT NULL**. La définition d'une valeur par défaut passe par la mention **DEFAULT** <valeur par défaut> suivie de la valeur par défaut souhaitée.

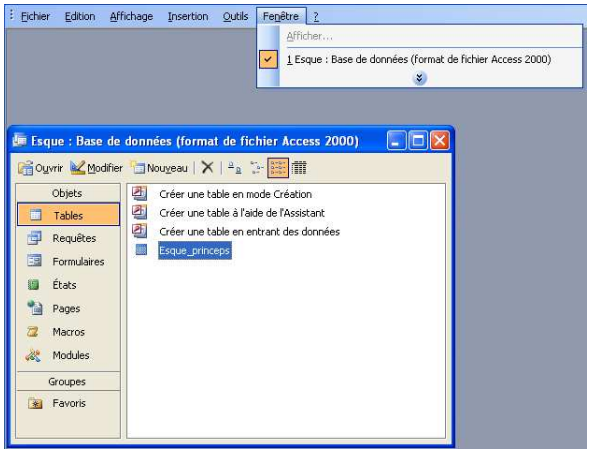
On peut de la même manière supprimer une colonne :

```
ALTER TABLE <nom>
    DROP COLUMN <colonne> ;
```

Access

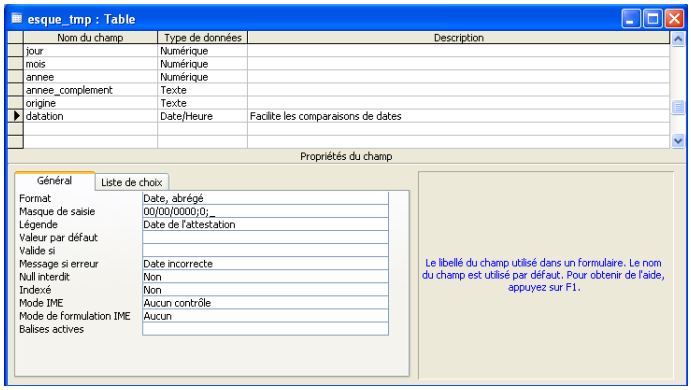
En utilisant le volet [Tables] de la fenêtre de navigation dans la base, on peut modifier la structure d'une table.

23



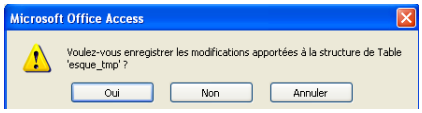
On peut alors ajouter une ligne 24, qui sera une nouvelle colonne de la table, modifier le type ou les caractéristiques d'une colonne.

24



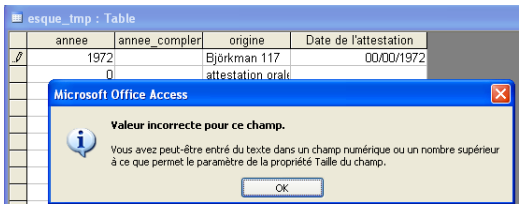
Dans tous les cas, les choix faits doivent être confirmés avant d'être mis en œuvre 25.

25



On note que le type choisi (Date/Heure) ne permet pas d'utiliser 0 comme pour 'jour non connu ou non pertinent' ou pour 'mois non connu ou non pertinent'.

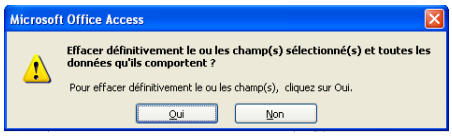
26



Lorsqu'on supprime une colonne de la table, après l'avoir sélectionnée et indiqué qu'on veut la supprimer 27, une confirmation est également demandée 28.

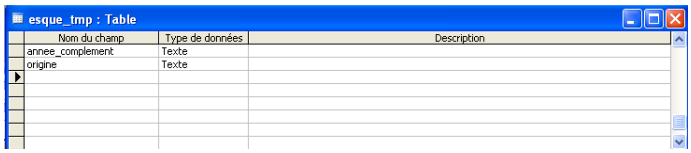
27

28



En 29, la structure de la table après suppression.

29



PRÉMA

Exercice n°2 Ajouter à la table signaletique_fiches un attribut classe_poids destiné à mémoriser la classe de poids du bébé en fonction du tableau fourni :

Classes de poids
poids non fourni
poids < 750 g.
750 g. <= poids <= 1 000 g.
poids > 1 000 g.

Exercice n°3 Ajouter à la table signaletique_fiches un attribut classe_position destiné à mémoriser la classe de position du bébé en fonction du tableau fourni :

position	*classe_position*
Non fournie	position non fournie
plat côté	côté
plat dos	dos
plat ventre	ventre
proclive côté	côté
proclive dorsale	dos
proclive ventrale	ventre

Exercice n°4 Créer une table fiche_occurrences_lemmes pour associer à chaque fiche le nombre de types et d'occurrences de lemmes. Peupler cette table avec les résultats d'une requête sur la table occ_prema (veiller à ne garder que les lemmes de vrais « mots »).

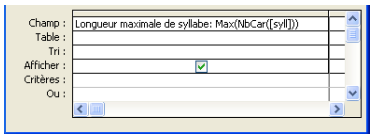
PHÈDRE

MySQL

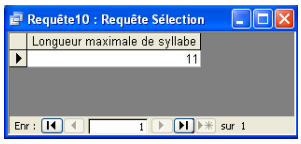
Access

L'objectif est de diminuer la place occupée par une colonne, syll, dévolue à la représentation phonétique d'une syllabe. Dans la version princeps de la table positions, cette colonne est de type texte, de 255 caractères. On imagine aisément que la représentation d'une syllabe n'occupe pas tant de place. On peut calculer la taille maximale de la colonne 30, ce qui donne 11 caractères : 31.

30

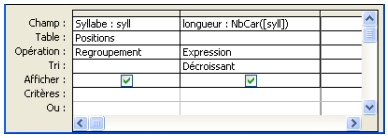


31

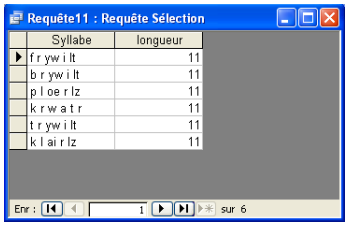


On peut même chercher à fournir les syllabes et leur taille, triées par taille décroissante 32 33.

32

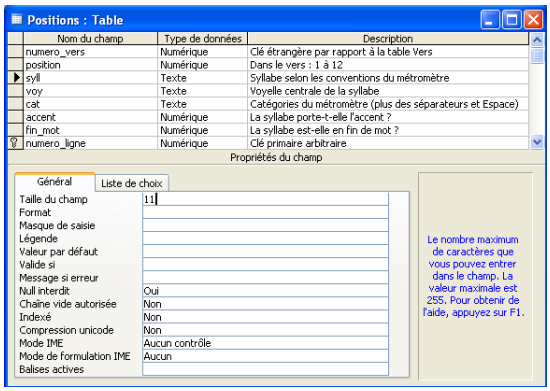


33



Puisqu'on sait désormais qu'une transcription phonétique de syllabe ne dépasse jamais 11 caractères, il est possible de modifier la taille du « champ » syll 34, selon la terminologie d'Access.

34

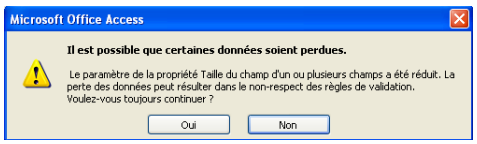


Confirmation est demandée 35.

35

Un message d'avertissement 36 indique la troncation de la colonne, qui peut effectivement faire perdre des informations.

36



2.3. ⊕ Supprimer une table

Lors de la suppression d'une table, à la fois son contenu (les lignes qu'elle contient) et sa structure disparaissent sans retour en arrière possible

MySQL

La requête ⌈**DROP TABLE** <nom> ;⌉ supprime la table dont le nom est indiqué. C'est, tout comme **DROP DATABASE**, une instruction à utiliser avec discernement. . .

Access

On supprime une table en sélectionnant son icône dans le volet [Tables] de la fenêtre de navigation dans la base, puis en appuyant sur la touche Del/Suppr ou bien en passant par [Edition | Supprimer] ou bien encore via le clic droit de la souris et le choix de [Supprimer]. Une confirmation est demandée.

ESQUE

3. Ajouter/modifier/supprimer des entités

3.1. ⊕ Ajouter une entité

Le hasard des lectures a permis de glaner l'attestation suivante (Jacques Jouet, *Cantates de proximité – Scènes et portraits de groupe*, P.O.L., 2005, Paris, p. 44) :

*Descendant des cheveux blonds tintin*esque*s*
le visage est long de sa croissance inachevée qui fatigue
et marqué du bleu éclairant des yeux.

On souhaite ajouter ce contexte à la base ESQUE qui comprend déjà d'ailleurs 3 occurrences du dérivé *tintinesque* (« . . . Le "Tintinéraire" est bouclé : près de 180 000 kilomètres à travers tous les pays des péripéties "tintinesques". . . », « . . . mus par une foi très tintinesque à soulever des montagnes. . . », « . . . houppe tintinesque au vent. . . »).

MySQL

L'insertion s'effectue par :

```
INSERT INTO <table>
(<liste de noms d'attributs>)
VALUES
(<liste␣de␣valeurs␣correspondantes>)␣;
```

comme dans :

```
  INSERT INTO attestations
  (
derive,
cat_derive,
base,
cat_base,
reference,
contexte,
auteur,
datation)
  VALUES
  (
'tintinesque',
'a',
'Tintin',
'npr',
```

```
5 'Cantates␣de␣proximité␣–␣Scènes␣et␣portraits␣de␣groupe,␣P.O.L.,␣2005,␣Paris,␣p.␣44'
    ,
6 'Descendant␣des␣cheveux␣blonds␣<tintinesques>#le␣visage␣est␣long␣de␣sa␣croissance␣
    inachevée␣qui␣fatigue#et␣marqué␣du␣bleu␣éclairant␣des␣yeux.',
7 'Jacques␣Jouet',
8 '2005–00–00')  ;
```

Sont mentionnés dans une première liste les attributs pour lesquels une valeur va être fournie. Les attributs non mentionnés prendront la valeur par défaut qui est la leur dans la définition de la table ou seront positionnés à **NULL** si cette possibilité est ouverte. Les attributs mentionnés sont remplis en fonction des valeurs indiquées dans la liste suivant **VALUES**, la valeur de rang *k* étant celle de l'attribut de rang *k* dans la liste des noms d'attributs. Des indices ont été ajoutés manuellement dans l'exemple aux éléments des deux listes pour faciliter la compréhension des correspondances. Ainsi 'Tintin', de rang 3 est la valeur de l'attribut de rang 3 dans la liste des noms d'attributs : base. △

Quand on utilise un attribut auto-incrémenté dans une table, chaque ajout d'une ligne affecte à cet attribut dans la ligne la valeur suivante de la dernière valeur qui a été affecté à cet attribut. Par contre, si on supprime des lignes, les valeurs correspondantes de l'attribut ne sont pas réutilisées ultérieurement. Il y a des « trous » dans les valeurs successives de l'attribut.

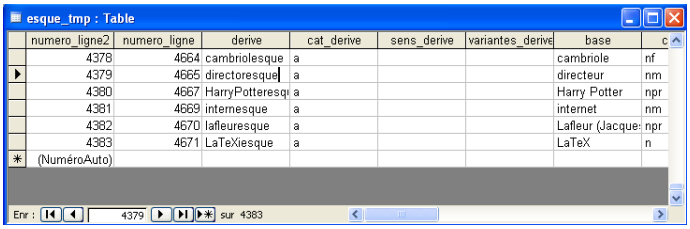
Access

Lorsqu'on ouvre une table en mode feuille de données, la ligne du bas de la feuille de données permet de naviguer dans la table. Le triangle vers la gauche ou vers la droite permet de se déplacer d'une ligne vers le haut ou vers le bas. Accompagné d'un trait vertical, il positionne en tout début ou en toute fin de table. Enfin, le triangle assorti d'une étoile positionne en fin de table, sur une nouvelle ligne vierge attendant de nouvelles informations. C'est donc le bouton qu'il faut cliquer pour insérer une nouvelle ligne. On peut par ailleurs taper directement le numéro de la ligne que l'on souhaite examiner dans la fenêtre entre les deux triangles où apparaît le numéro de ligne courant.

37

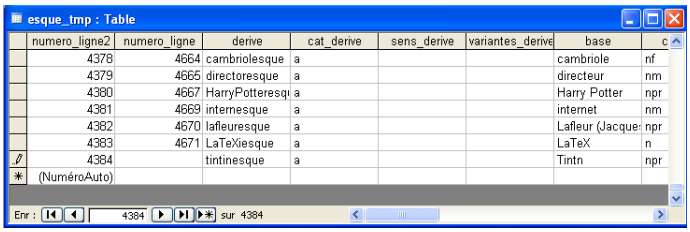
Une fois positionné sur une nouvelle ligne vierge 38, on est en mesure de la modifier pour ajouter l'attestation fournie comme exemple. La mention NuméroAuto entre parenthèses, dans la colonne d'extrême gauche, indique que la nouvelle ligne recevra automatiquement un nouveau numéro d'ordre, sans que l'on ait à le fournir.

38



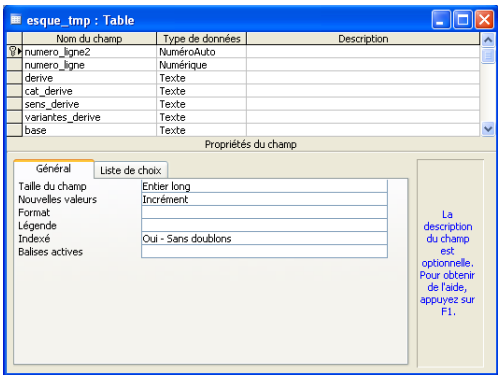
En 39, le résultat de l'insertion, avec la nouvelle valeur du numéro d'ordre.

39



On trouve en 40 un extrait de la structure d'une copie de la table esque. On voit que l'attribut numero_ligne2 est défini comme NuméroAuto et que les nouvelles valeurs qu'il prendra sont de type Autoincrement. Il s'agit d'un attribut dont les lignes successives correspondent à la valeur d'un compteur incrémenté à chaque ajout. On constate d'ailleurs le décalage entre ce numéro d'ordre, qui va de 1 à 4384, et qui correspond effectivement au nombre de lignes de la table, et la dernière valeur de la colonne numero_ligne, 4671. Cette dernière colonne correspond à l'import sous Access d'une table MySQL et n'est pas de type NuméroAuto, mais de type Entier. Elle correspond aux numéros d'ordre de la table sous MySQL et aux « trous » mentionnés supra.

40



3.2. ⊕ Modifier une entité

Lorsqu'on examine les catégories de départ des dérivés dans ESQUE (tableau 90, p. 315, en haut), on constate une incohérence. Certains dérivés, l'immense majorité, ont pour catégorie a(djectif) tandis qu'un très petit nombre est étiqueté A. L'examen des dérivés en question confirme qu'il s'agit bien d'une erreur : *néo-mondesque*, *julesque*, *quatorze juillesque*, *Freudesque*, *Vicoesque*.

MySQL

On modifie l'étiquette de ces dérivés par la requête :

```
UPDATE esque
SET cat_derive = 'a'
WHERE cat_derive = 'A' ;
```

La valeur attribuée à un attribut pour un ensemble d'entités donné, ensemble déterminé par la clause **WHERE**, peut résulter d'un calcul sur la valeur précédente de cet attribut ou sur celle d'autres attributs.

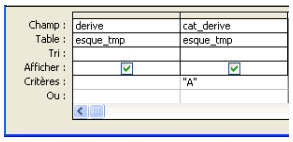
TABLEAU 90 – ESQUE : catégories de départ des dérivés

cat_derive	*COUNT(cat_derive)*
NULL	134
?	3
A	5
a	4137
adv	13
n	20
n?	2
nf	27
nfpl	1
nm	38
npl	1
v	1
vintr	1

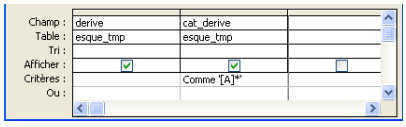
Access

Le repérage des attestations où la catégorie du dérivé est A et non a est difficile sous Access. △
En effet, les deux requêtes 41 et 42 retournent l'ensemble des dérivés dont la catégorie est soit A soit a. Access ne semble pas prendre en compte la casse dans ces requêtes.

41

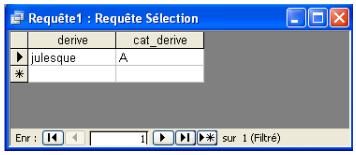


42



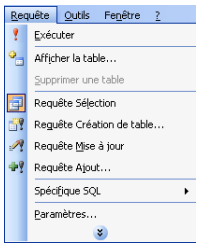
Pourtant, la table examinée présente bien des dérivés étiquetés A :

43



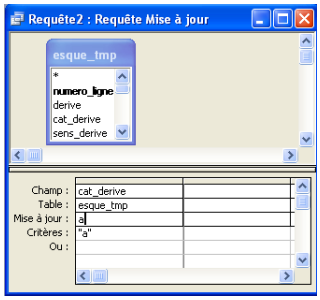
Pour effectuer la modification souhaitée, il faut modifier la requête pour qu'elle devienne une requête Mise à jour 44.

44



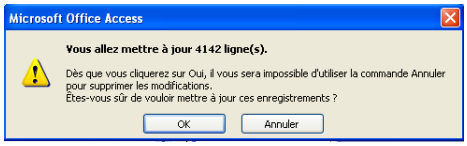
La sous-fenêtre de formulation d'une requête dispose alors d'une ligne ⌈Mise à jour⌉ destinée à donner la nouvelle valeur souhaitée 45.

45



Le déclenchement de la requête entraîne une mise en garde 46.

46

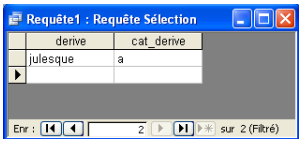


L'équivalent SQL Server est :

```
UPDATE esque_tmp
SET esque_tmp.cat_derive = "a"
WHERE (((esque_tmp.cat_derive)="a"))
```

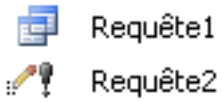
La clause **WHERE** semble ne chercher que les a minuscules. Néanmoins, les 4 142 lignes concernées par la requête correspondent aux 4 137 avec a minuscule et aux 5 avec A majuscules. On constate d'ailleurs en 47 le changement effectif de la catégorie pour l'exemple donné supra en 43.

47



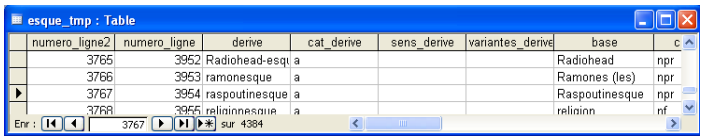
L'icône d'une requête Mise à jour est précédée d'un petit crayon (Requête 2), contrairement à une requête ordinaire (Requête 1) 48.

48

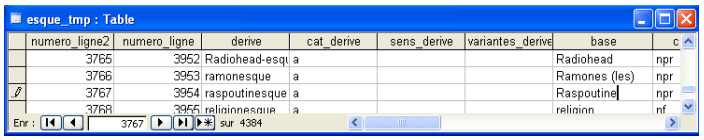


Il est également possible de modifier directement une table ouverte en mode Feuille de données. Il suffit alors de changer les cellules souhaitées. C'est le cas en 49 où la base erronée *Raspoutinesque* est corrigée en *Raspoutine* 50.

49

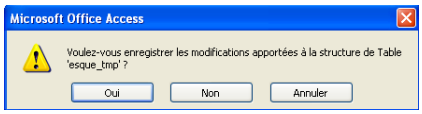


50



Les modifications ne deviennent toutefois effective qu'après fermeture de la feuille de données et confirmation des changements apportés 51.

51



Préma

Exercice n°5 Une fois l'attribut classe_poids ajouté à la table signaletique_fiches, mettez à jour les valeurs de cet attribut en fonction de la valeur de la colonne poids.

Exercice n°6 Une fois l'attribut classe_position ajouté à la table signaletique_fiches, mettez à jour les valeurs de cet attribut en fonction de la valeur de la colonne position.

Phèdre

Esque

Exercice n°7 On appelle *dérivation impropre* l'opération morphologique qui crée un mot en le changeant de catégorie sans changer sa forme (c'est le cas de *doux* dans *le doux du ventre*), par opposition à une dérivation, dans laquelle intervient un affixe, comme *-esque*.

Repérer les mots en relation de dérivation impropre dans la table esque et indiquer les cas où il s'agit probablement d'erreurs à corriger.

Exercice n°8 Ajouter les attestations suivantes :

1. Jacques Roubaud, *La Bibliothèque de Warburg*, Seuil, 2002, Fiction & Cie, Paris, Version mixte, p. 277 [...] le bord du bout de la langue fjordesque du Loch Dunvegan[...]
2. Laura Lippman, *La colline des bouchers*, J'ai Lu, 1999, Policier, Paris, Butchers hill Traduction de Thierry Arson, p. 47-48 [Kitty est la tante, haute en couleur, séductrice, de l'héroïne] En une seconde Will se perdit dans l'univers kittyesque, ce petit royaume de douceurs féminines dont le drapeau avait la couleur de boucles blond-roux, dont le parfum officiel était le freesia du Jardin Botanique et où le seul son autorisé était un murmure de contralto.
3. Michel Rocard, *Entretien avec Judith Waintraub*, Flammarion, 2001, Histoire, Paris,
   (a) p. 87 Je m'en ouvre aussitôt à Depreux, patron du PSU, pagailleur, foutoiresque, mais très généreux.
   (b) p. 215 J'ai été victime d'une histoire abracadabrantesque.
4. Jean-Pierre Maurel, *Malaver à l'hôtel*, Viviane Hamy, 1996, Chemins nocturnes, Paris,
   (a) p. 70 Les murs dont la peinture saumon a été essuyée au gant servent de cimaises à quelques grands pastels reposants et à un tableau qui l'est moins, dans lequel trois figures diabolico-carnavalesques peintes à grands traits d'un pinceau coloré, épais et circulaire, se tiennent les coudes pour me signifier tout le grotesque de notre condition.
   (b) p. 176 Il pige au quart de tour, ne se perd pas en vaines questions et son oeil renardesque pétille d'ironie complice.
5. Gabriela Avigur-Rotem, *Canicule et oiseaux fous*, Actes Sud, 2006, Arles, Traduit de l'hébreu par Arlette Pierrot et Ziva Avran, p. 472 Encore un instant, encore un autre - l'avion tremble de colère, concentre ses forces pour décoller, commence sa course sur la piste, lent, lourd, canardesque, mais son décollage grogne déjà dans ses entrailles ; je le sens dans la plante de mes pieds qui se contractent pour le capter, pour le propulser vers l'azur - vole, métal, vole, je reprends mon incantation contre la gravitation, vole, métal, vole, et voilà - j'ai encore raté ce moment merveilleux de la séparation [...]

Exercice n°9 Constituer le tableau des catégories des bases de ESQUE avec leur nombre d'occurrences. Vous constaterez que figure, une fois seulement, la catégorie `nf>`. Le tableau de la base et du dérivé pour cette catégorie vous permettra de constater que c'est bien une faute de frappe pour `nf`. Vous modifierez alors la table `esque` en conséquence.

3.3. ⊕ Supprimer une entité

MySQL

Supprimer une ou plusieurs entités s'opère par la requête :

```
DELETE FROM <table>
[WHERE <condition>] ;
```

où la clause **WHERE**, optionnelle, opère la sélection des entités à détruire. Par conséquent, la requête sans clause **WHERE** supprime toutes les entrées de la table, mais conserve sa structure, à la différence de :

DROP <table> ;

qui efface table et structure. Comme **DROP**, **DELETE** est à utiliser avec parcimonie et précautions : la destruction est sans retour.

Par exemple, la requête :

```
DELETE FROM esque
WHERE cat_derive = 'adv' ;
```

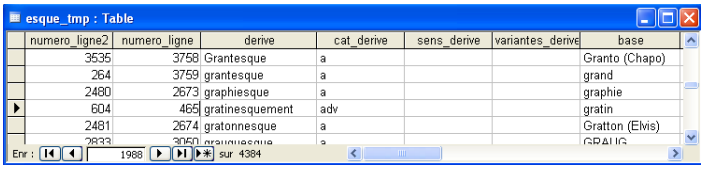
aboutirait à effacer les 13 entrées de `esque` qui sont des adverbes (*gratinesquement*, *outragesquement*, *gidesquement*, etc.) et dont on peut estimer qu'il s'agit de dérivés de dérivés en *-esque*, de « deuxième niveau » en somme.

Access

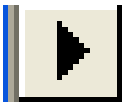
Deux approches sont possibles pour supprimer des entités, par interaction directe sur la feuille de données ou par requête.

La première consiste, en mode Feuille de données, à se positionner sur la ligne concernée [52], à la sélectionner en cliquant sur le triangle dans la marge de gauche [53].

[52]

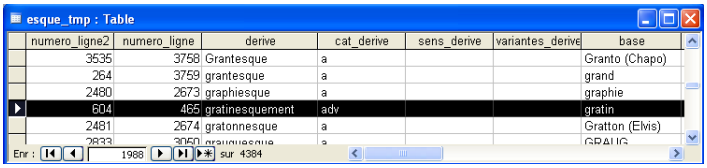


[53]



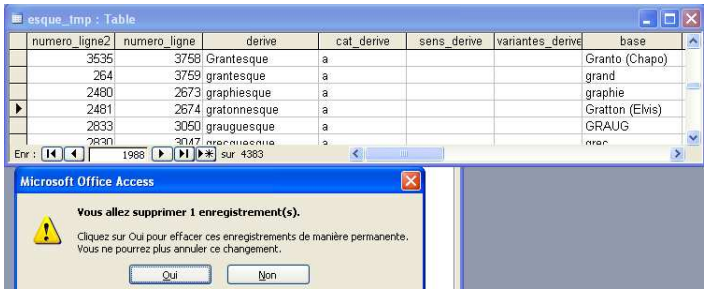
La ligne passe alors en inverse vidéo [54].

[54]



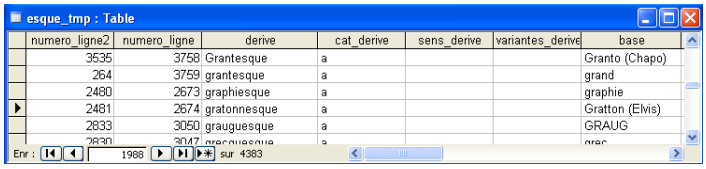
Appuyer sur la touche Del/Suppr ou passer par ⌈Edition | Supprimer⌉ conduit à une demande de confirmation [55].

55



Une fois la confirmation faite, la ligne a définitivement disparu de la table 56.

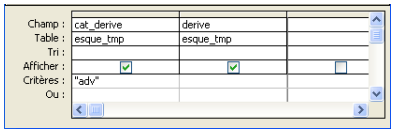
56



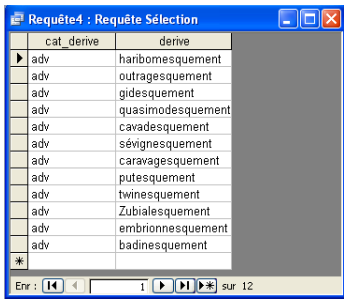
La seconde manière de supprimer des entrées opère par requête.

Si l'on commence par chercher les dérivés étiquetés comme `Adv` 57, on obtient comme résultat 58

57

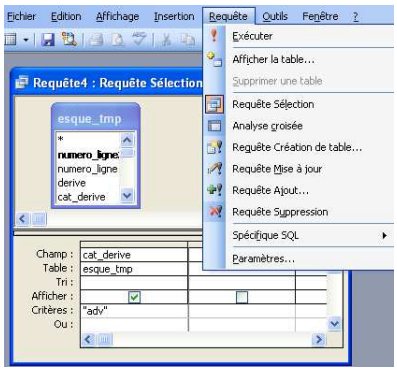


58



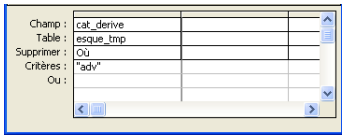
On crée donc une nouvelle requête, cherchant les dérivés en `Adv` et on sélectionne ⌈Requête | Requête Suppression⌉ 59.

59



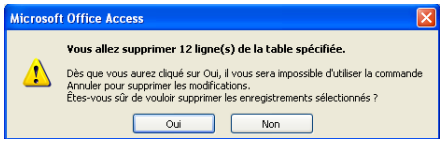
La sous-fenêtre de formulation de requête change dans la barre de menu du haut et par l'ajout d'une ligne [Supprimer] 60.

60



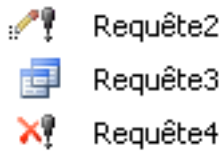
Le déclenchement de la requête débute par une demande de confirmation 61.

61



L'icône d'une requête Suppression est une croix en rouge suivie d'un point d'exclamation 62.

62



4. ⊕ Sauvegarder une base ou une table

MySQL

Une des manières de sauvegarder consiste à engendrer un fichier comprenant toutes les instructions MySQL permettant de recréer la table ou la base et d'y insérer les lignes qui y figuraient.

Pour PRÉMA, voici un extrait d'un tel fichier :

```
DROP TABLE IF EXISTS `infirmieres_princeps`;

CREATE TABLE `infirmieres_princeps` (
...
) TYPE=MyISAM;
```

```
INSERT INTO `infirmieres_princeps` VALUES (41,26,3,'BEPC','0.00',NULL);
INSERT INTO `infirmieres_princeps` VALUES (2,29,3,'BEPC','2.50','Jour');

...
INSERT INTO `infirmieres_princeps` VALUES (22,31,3,'BAC','9.00','Jour');
INSERT INTO `infirmieres_princeps` VALUES (73,29,3,'BAC','1.50','Jour');
```

On se reportera au chapitre I § 5 p. 24 pour le détail de la création de la structure de la table infirmieres_princeps.

On note que la table est supprimée avant d'être recréée : la présence de cette suppression est optionnelle, elle doit être choisie au moment de la sauvegarde (option *--add-drop-table* ci-dessous). Suivent les instructions permettant de peupler la table en ajoutant autant de lignes qu'il y en avait dans la table sauvegardée.

Sous Linux/Unix, l'instruction de sauvegarde est la suivante :

```
mysqldump --add-drop-table -u <utilisateur> -p <base>  > <fichier de sauvegarde>
```

comme dans :

```
mysqldump --add-drop-table -u habert -p prema  > prema.sql
```

L'option -u introduit l'utilisateur qui veut utiliser la base de données prema, -p indique qu'un mot de passe sera fourni avant de pouvoir accéder à la base. Le chevron fermant précède le nom du fichier dans lequel sont écrites les instructions de reconstruction de la base. C'est une manière, sous Unix/Linux, d'écrire dans un fichier.

Sous Linux/Unix, on peut également sauvegarder une ou plusieur table(s) sans sauvegarder la base tout entière :

```
mysqldump --add-drop-table -u <utilisateur> -p <base> <$table$>+ > <fichier de
    sauvegarde>
```

comme dans :

```
mysqldump --add-drop-table -u habert -p prema  bebes > bebes.sql
```

Access

La sauvegarde consiste à faire une copie du fichier d'extension .mdb (se terminant par .mdb). Ce fichier peut être volumineux. Prema.mdb, Phedre.mdb et Esque.mdb « pèsent » respectivement 8,4, 12 et 7.5Mo (Mo = Méga-octets = million de caractères).

L'extension des fichiers (ici en l'occurrence .mdb) n'est pas forcément visible sous Windows et dépend du paramétrage de l'affichage. △

5. ⊕ Rester efficace : les index

Les lignes d'une table ne sont pas « rangées » (chapitre VI, § 2). Elles ne suivent pas un ordre particulier. Y chercher une information peut supposer de parcourir toute la table, opération longue si la table est volumineuse. Les **index** permettent d'accélérer les recherches. Un index est une structure auxiliaire qui établit la correspondance entre les valeurs éventuellement

ordonnées d'une ou de plusieurs colonnes et les numéros de lignes correspondantes dans la table indexée (par exemple, *tintinesque* occupe les lignes 963, 1295 et 1934 de la table esque initiale). Un index des dérivés évite d'avoir à parcourir les 4 384 lignes de la table esque. Le tri préalable de l'index et les structures sous-jacentes donnent en effet accès à des méthodes d'accès plus rapides que la lecture ligne à ligne. Si le dérivé figure dans l'index, les numéros associés sont utilisés pour accéder aux informations des lignes correspondantes. Dans le cas contraire, la recherche a été plus rapide que celle dans la table complète, par définition non triée. Il en résulte néanmoins un léger surcoût lors de l'ajout, de la modification ou de la suppression de lignes dans la table, puisqu'il faut modifier en même temps les index associés. △

On associe à une table autant d'index que nécessaires à sa manipulation rapide. Malgré le léger surcoût à la création/destruction de lignes, il est souvent fructueux d'indexer chaque clé primaire et les clés étrangères, puisque ce sont des ingrédients centraux des opérations de jointure, les plus coûteuses.

MySQL

On peut déclarer, lors de la création d'une table, que certains attributs ou combinaisons d'attributs doivent être indexé(e)s en ajoutant la mention
INDEX(<colonne ou combinaison de colonnes>)
dans les options de la table. On peut également ajouter *a posteriori* un index à une table, en particulier lorsqu'on pense que cette table ne va plus guère évoluer mais qu'il faut optimiser les requêtes qui y font accès :

```
ALTER TABLE infirmieres_princeps
    ADD INDEX (id) ;

ALTER TABLE infirmieres_princeps
    ADD INDEX infirmieres_princeps_ndx (id) ;
```

Dans le second cas, l'index créé reçoit en outre un nom.

La création séparée d'un index s'opère par :

```
CREATE [UNIQUE] INDEX <nom de l'index>
ON␣<table>␣(<colonne␣ou␣combinaison␣de␣colonnes>)␣;
```

Le mot-clé **UNIQUE** contraint la colonne (ou la combinaison de colonnes) utilisées pour l'indexation à avoir des valeurs toutes différentes. Ce serait le cas par exemple pour l'attribut vers de la table du même nom :

```
CREATE UNIQUE INDEX vers_ndx
ON vers (vers) ;
```

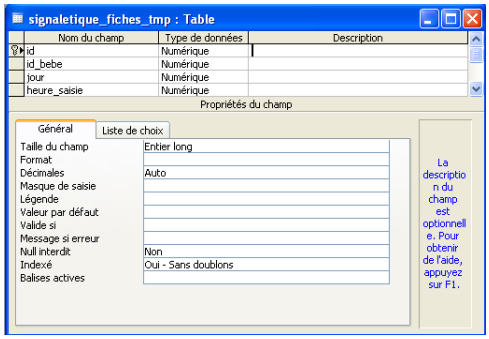
La destruction d'un index s'effectue via :

```
DROP INDEX <nom de l'index>␣;
```

Access

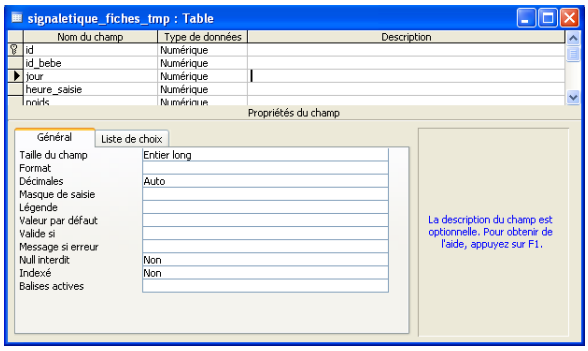
Quand on examine la structure de la table signaletique_fiches et l'attribut id, qui sert de clé primaire, on constate que cet attribut est indexé sans doublons et que **NULL** est interdit 63.

63



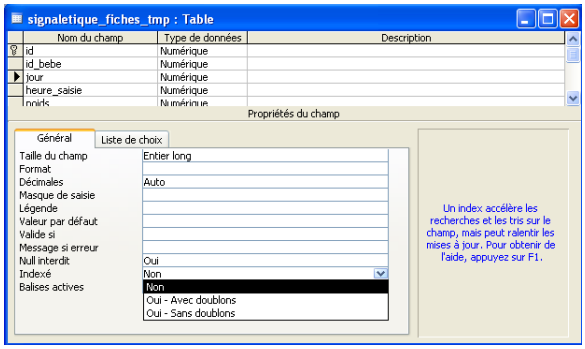
En 64, on voit l'état de départ de l'attribut jour, non indexé et où **NULL** est permis.

64



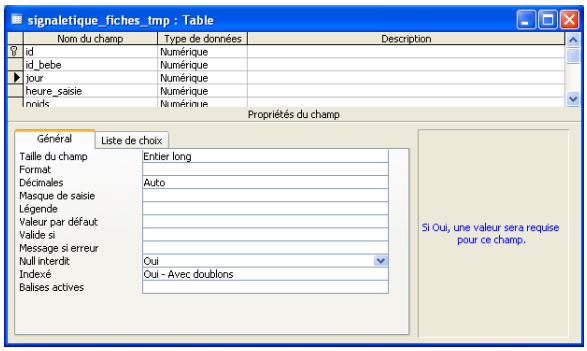
Pour l'indexation, il faut choisir entre pas d'indexation, l'indexation avec doublons et l'indexation sans doublons 65.

65



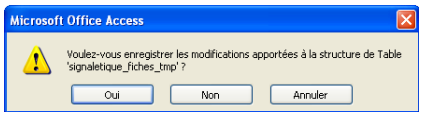
C'est l'indexation avec doublons qui convient dans le cas présent (de nombreuses fiches ont la même valeur pour l'attribut jour) et l'interdiction de la marque **NULL** 66.

66



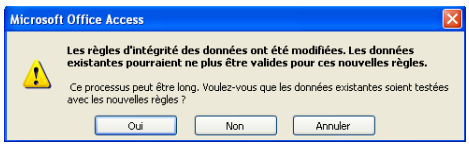
La fermeture de la fenêtre de modification de la structure de la table entraîne une demande de confirmation 67.

67



En outre, une deuxième fenêtre indique que l'ajout de l'indexation modifie les relations entre tables 68.

68



△

Dans certains cas, l'indexation peut être refusée par Access dans la mesure où elle contredit des relations préexistantes.

PRÉMA

PHÈDRE

ESQUE

6. Solutions

Solution de l'exercice n°1 p. 306 Le résultat est le suivant :

mot graph. max.	*mot phon. max.*
14	26

Les valeurs de l'attribut `occ_car` et de l'attribut `occ_phon` occupent au maximum 14 et 26 caractères respectivement, alors que leur sont alloués 100 caractères au maximum. La requête correspondante est :

R^2_{111}

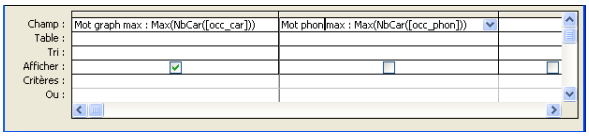
PAR GROUPE(
MAXIMUM(LONGUEUR(occ_car)) TITRE 'mot graph. max.',
MAXIMUM(LONGUEUR(occ_phon)) TITRE 'mot phon. max.'
)[occurrences]

MySQL

```
SELECT MAX(LENGTH(occ_car)) AS 'mot graph. max.',
       MAX(LENGTH(occ_phon)) AS 'mot phon. max.'
FROM occurrences ;
```

Access

La requête :



69

a pour équivalent SQL Server :

```
SELECT Max(Len([occ_car])) AS [Mot graph max],
       Max(Len([occ_phon])) AS [Mot phon max]
FROM Occurrences;
```

Solution de l'exercice n°2 p. 309

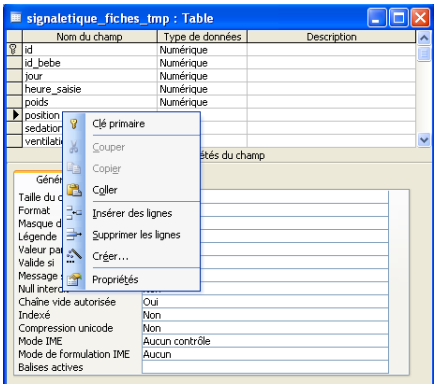
MySQL

```
ALTER TABLE signaletique_fiches
      ADD COLUMN classe_poids VARCHAR(50) AFTER poids ;
```

Access

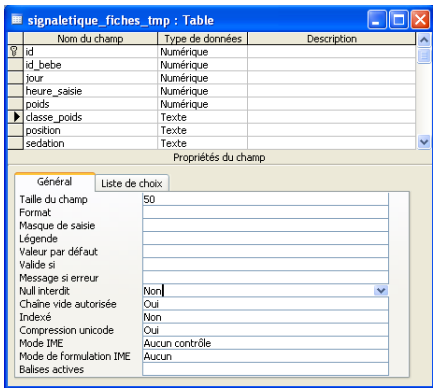
Pour ajouter un attribut en mode Modification de la structure d'une table, on se positionne avant l'attribut devant lequel on veut ajouter le nouvel attribut et on utilise le clic droit pour avoir accès au menu permettant d'insérer une ligne 70.

70



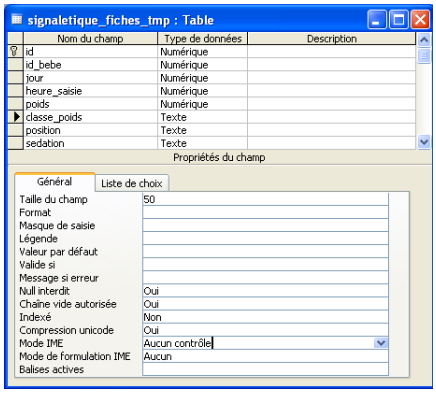
Ce choix fait, on indique le type de l'attribut, ici Texte, et sa longueur, ici 50 caractères. Par défaut, **NULL** n'est pas interdit et la chaîne vide est autorisée 71.

71



On modifie ces deux choix 72.

72



Solution de l'exercice n°3 p. 309

MySQL

```
ALTER TABLE signaletique_fiches
      ADD COLUMN classe_position VARCHAR(50) AFTER position ;
```

Access

La solution est similaire à celle de l'exercice précédent.

Solution de l'exercice n°4 p. 309

MySQL

```
CREATE TABLE fiche_types_occurrences_lemmes (
       id_fiche INT NOT NULL,
       types INT NOT NULL,
       occurrences INT NOT NULL,
      PRIMARY KEY (id_fiche)) ;

INSERT INTO fiche_types_occurrences_lemmes
      SELECT fiche ,
```

TABLEAU 91 – ESQUE : dérivations impropres

base	*cat_base*	*derive*	*cat_derive*
arabesque	a	arabesque	nf
burlesque	a	burlesque	nm
funambulesque	a	funambulesque	nm
gigantesque	a	gigantesque	nm
grotesque	a	grotesque	nm
mauresque	a	mauresque	nf
picaresque	a	picaresque	nm
pittoresque	a	pittoresque	nm
putanesque	a	putanesque	n
romanesque	a	romanesque	nf
romanesque	a	romanesque	nm
soldatesque	a	soldatesque	nf
arabesque	a	arabesque	nf
picaresque	a	picaresque	nm
soldatesque	**NULL**	soldatesque	nf
cuivresque	nm	cuivresque	a
cartoon-esque	nm	cartoon-esque	a


```
        COUNT(DISTINCT lemme),
        COUNT(*)
    FROM occ_prema
    WHERE categorie REGEXP '[ANRSDVCP]'
    GROUP BY fiche ;
```

Access

Solution de l'exercice n°7 p. 317 Le résultat figure au tableau 91 p. 328. On constate que les adjectifs en *-esque* donnent naissance à des noms : *arabesque* donne *une arabesque*. On peut se demander si les deux dernières lignes ne correspondent pas à des erreurs, à l'interversion des catégories entre la base et le dérivé et, pour *soldatesque*, à l'omission de la catégorie de la base.

MySQL

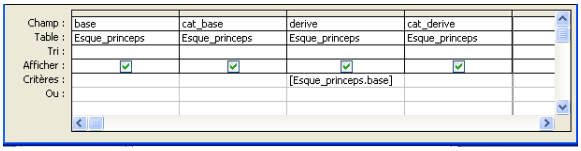
```
SELECT base,
       cat_base,
       derive,
       cat_derive
FROM esque
WHERE derive = base ;
```

Access

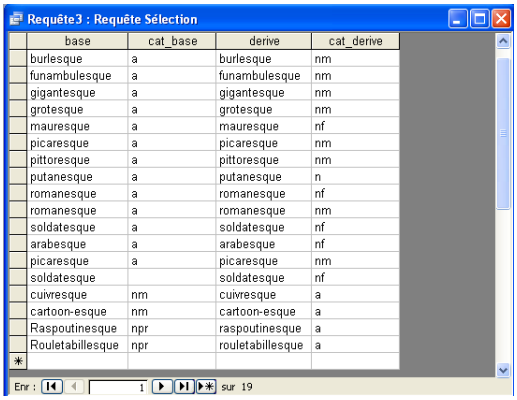
La requête 73 débouche sur un résultat légèrement différent de celui de la requête MySQL 74. On constate là encore qu'Access semble insensible à la casse, puisque *Raspoutinesque* et *raspoutinesque* sont considérés comme identiques. Il semble d'ailleurs qu'il s'agisse pour *raspoutinesque* et *rouletabillesque* d'une erreur en ce qui concerne la base, erreur qui serait également à corriger. L'obtention du même résultat avec MySQL suppose de modifier la clause **WHERE** en :

```
WHERE derive = LOWER(base)
```

73



74



Solution de l'exercice n°5 p. 317

MySQL

On passe de 164 poids différents dans la table signaletique_fiches à 4 classes.

```
UPDATE signaletique_fiches
      SET classe_poids = 'poids non fourni'
      WHERE poids = 0 ;

UPDATE signaletique_fiches
      SET classe_poids = 'poids < 750 g.'
      WHERE poids < 750 AND poids > 0 ;

UPDATE signaletique_fiches
      SET classe_poids = '750 g. <= poids <= 1 000 g.'
      WHERE poids >= 750 AND poids <= 1000 ;

UPDATE signaletique_fiches
      SET classe_poids = 'poids > 1 000 g.'
      WHERE poids > 1000 ;
```

On remarque que la deuxième modification suppose de vérifier que le poids est non seulement inférieur à 750 grammes mais qu'il est aussi supérieur à 0, sans quoi les fiches qui ont été étiquetés 'poids non fourni' par la première modification se verraient attribuer la catégorie 'poids < 750 g.' par la seconde.

Une modification « en un coup » s'effectue par l'appel à l'aiguillage **CASE** :

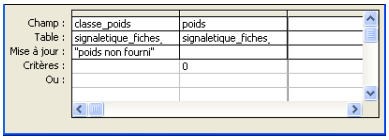
```
UPDATE signaletique_fiches
SET classe_poids =
      CASE
```

```
        WHEN poids = 0 THEN 'poids non fourni'
        WHEN poids > 0 AND poids < 750 THEN 'poids < 750 g.'
        WHEN poids > 1000 THEN 'poids > 1000 g.'
        ELSE '750 g. <= poids <= 1000 g.'
    END ;
```

Access

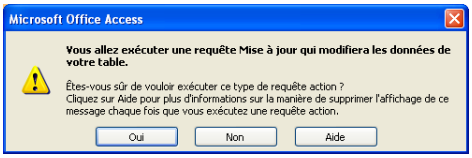
La solution passe par une requête Mise à jour du type de :

75

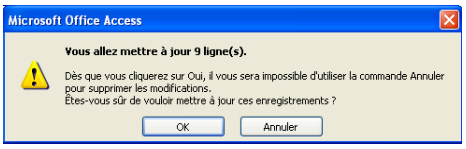


Le déclenchement de cette requête s'accompagne de deux demandes de confirmation :

76



77



Il faut autant de requêtes que de modifications à effectuer.

Solution de l'exercice n°6 p. 317

MySQL

On passe de 7 positions différentes dans la table signaletique_fiches à 4 classes.

```
UPDATE signaletique_fiches
    SET classe_position = 'Non fournie'
    WHERE position = 'position non fournie' ;

UPDATE signaletique_fiches
    SET classe_position = 'côté'
    WHERE position = 'plat côté' ;

UPDATE signaletique_fiches
    SET classe_position = 'dos'
    WHERE position = 'plat dos' ;

UPDATE signaletique_fiches
    SET classe_position = 'ventre'
    WHERE position = 'plat ventre' ;

UPDATE signaletique_fiches
```

```
    SET classe_position = 'côté'
    WHERE position = 'proclive côté' ;

UPDATE signaletique_fiches
    SET classe_position = 'dos'
    WHERE position = 'proclive dorsale' ;

UPDATE signaletique_fiches
    SET classe_position = 'ventre'
    WHERE position = 'proclive ventrale' ;
```

On pourrait là aussi faire appel à **CASE**.

Access

La solution est similaire à celle de l'exercice précédent.

Solution de l'exercice n°8 p. 317 Cet exercice n'appelle pas de solution, puisqu'il s'agit simplement de reprendre les manœuvres expliquées.

Solution de l'exercice n°9 p. 318 Les tableaux donnant les catégories de départ des bases avec leur nombre d'occurrences et fournissant la base de catégorie 'nf>' figurent dans le tableau 92 p. 332.

MySQL

L'enchaînement des trois instructions suivantes permet d'obtenir le résultat souhaité :

```
SELECT cat_base,
    COUNT(cat_base)
FROM esque
GROUP BY cat_base ;

SELECT derive,
    base
FROM esque
WHERE cat_base = 'nf>' ;

UPDATE esque
    SET cat_base = 'nf'
WHERE cat_base = 'nf>' ;
```

Access

La transformation de la requête Sélection en requête Mise à jour s'obtient via le menu [Requête] de la barre de menu du haut, mais aussi en utilisant le clic droit sur la sous-fenêtre du haut de formulation des requêtes :

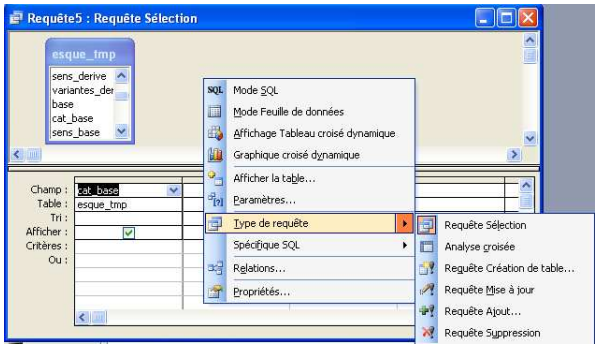
TABLEAU 92 – ESQUE : catégories de départ des bases

cat_base	*COUNT(cat_base)*
NULL	339
?	7
a	130
a ?	10
a/nm	4
acron	1
adv	4
adv ?	1
interj	8
ladj	1
ladv	3
loc	2
n	58
n ?	2
nf	517
nf?	1
nf/nm	1
nf>	1
nfpl	5
nm	1306
nm ?	11
nm/nf	1
nmpl	9
nnpr	1
npl	2
npr	1879
npr ?	22
npr ?	1
onomat	3
sigle	49
sigle ?	2
vtr	2

dérivé catégorisé 'nf>'

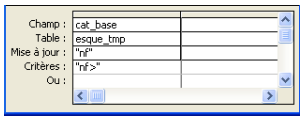
derive	*base*
barbesque	barbe

78



La requête Mise à jour est alors :

79



CHAPITRE XI

PRATIQUES ET BONNES PRATIQUES

1. Prévenir les incohérences

2. Vérifier la justesse des données

PRÉMA

La requête suivante :

R^2_{112} PROJECTION(poids)[fiches_originelles]
$\hookrightarrow$ TRI SUR(poids)[<résultat$_1$>]

permet si les valeurs pour le poids dans la table fiches_originelles sont « raisonnables ». Cette vérification souligne le problème de la représentation des non-réponses par 0 pour le poids. On a vu au chapitre IV § 4 qu'une représentation par **NULL** est probablement plus appropriée.

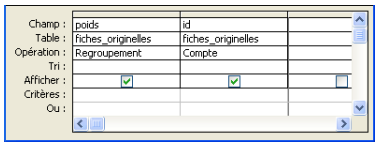
MySQL

```
SELECT DISTINCT poids
FROM fiches_originelles
ORDER BY poids ;
```

Access

La requête Access :

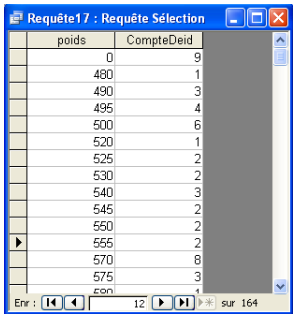
1



repose sur une démarche différente. À la place d'une projection, c'est un regroupement qui permet de n'obtenir qu'un exemplaire de chaque poids. On profite de ce regroupement pour calculer le nombre d'occurrences de chaque valeur.

Son résultat est :

2



Il y a 164 poids distincts, dont 9 occurrences de 0.

Cette requête a pour équivalent MySQL :

```
SELECT DISTINCT fiches_originelles.poids,
    Count(fiches_originelles.id) AS CompteDeid
FROM fiches_originelles
GROUP BY fiches_originelles.poids;
```

Une autre vérification à effectuer est celle de la justesse du regroupement en classes des valeurs d'une variable, continue ou non. Les requêtes suivantes :

R^2_{113} PROJECTION(poids, classe_poids)[signaletique_fiches]
↪ TRI SUR(classe_poids, poids)[<résultat$_1$>]

R^2_{114} PROJECTION(position, classe_position)[signaletique_fiches]
↪ TRI SUR(classe_position, position)[<résultat$_1$>]

permettent une telle vérification pour le lien poids–classe_poids et position–classe_position dans la table signaletique_fiches. Cf. les solutions des exercices n°5 et 6, p. 317.

MySQL

```
SELECT DISTINCT poids, classe_poids
FROM signaletique_fiches
ORDER BY classe_poids, poids ;

SELECT DISTINCT position, classe_position
FROM signaletique_fiches
ORDER BY classe_position, position ;
```

Exercice n°1 Vérifier s'il n'existe pas, pour un bébé donné, des journées où il est bien présent dans le service de réanimation mais où aucune fiche n'a été rédigée sur lui.

Exercice n°2 Les lemmes des formes verbales sont des infinitifs. Il peut s'agir de mots simples (*être*, *agripper*) comme de « mots en plusieurs mots » (*aller bien*, *donner envie*, *ficher la paix*). On veut vérifier si les lemmes de toutes les formes verbales de la table occ_prema sont bien des infinitifs. Pour se centrer sur les erreurs possibles de lemmatisation et ne pas avoir à vérifier toutes les formes verbales, on veut éliminer les formes verbales dont les lemmes qui peuvent constituer des infinitifs, c'est-à-dire qui se terminent par une terminaison d'infinitif (*er*, *ir*, *re*) ou qui comprennent un infinitif fréquent dans les verbes « en plusieurs mots » (*aller*, *être*, *avoir*, *faire*, *prendre*, *mettre*). On veut donc éliminer les formes verbales qui semblent disposer d'un lemme idoine.

Exercice n°3 Repérer s'il existe des formes normalisées de la table occ_prema qui sont associées à plusieurs lemmes distincts.

Si c'est le cas, fournir chacun de ces lemmes, avec les formes normalisées associées, leur catégorie et leur fréquence.

PHÈDRE

La recherche des fins de phrase qui ne sont pas suivies d'une majuscule s'effectue par :

R^1_{76} *t. 1* p. 177

```
RESTRICTION(
vers RESSEMBLANT À '[.] *[^A-Z]'
)
[vers]
```

↪ PROJECTION(numero_vers, vers)[<résultat$_1$>]

MySQL

```
SELECT numero_vers, vers
FROM vers
WHERE vers REGEXP '[.] *[a–zéèêëàâîïôùüûç]' ;
```

La clause **WHERE** pourrait également également s'énoncer ainsi :

```
...
WHERE vers REGEXP '\\. *[a–zéèêëàâîïôùüûç]' ;
```

Le point, caractère spécial des expressions régulières, y est échappé par une double contre-oblique et non par les crochets.

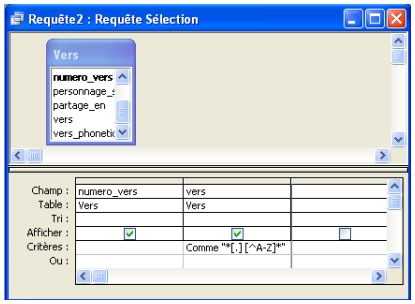
Une simple contre-oblique ne dé-spécialise pas le point et aboutit à lister les 1 654 vers de la pièce, puisque le motif filtre les vers comprenant un caractère quelconque (point) suivi optionnellement d'espaces et suivi d'un caractère minuscules. Tous les vers répondent à cette contrainte.

Access

La requête 3 procède à l'inverse de la requête MySQL ci-dessus. Elle retient les vers qui comprennent 0 ou n caractères, suivis d'un point (entre crochets pour le traiter comme un caractère ordinaire et non comme un caractère spécial ou méta-caractère), d'une espace et d'un

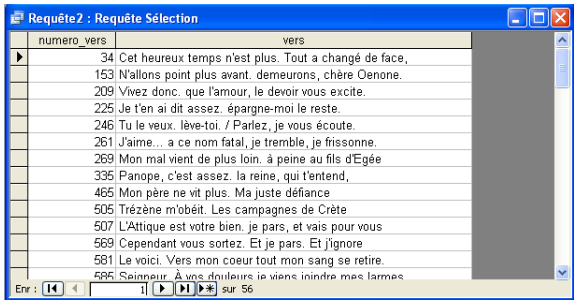
caractère qui n'est pas une majuscule. Cette requête est plus proche de la R_{76}^{1} *t. 1* p. 177. Cependant, il n'est pas possible sous Access d'indiquer qu'un caractère donné peut être présent 0 ou n fois (rôle de l'étoile post-posée dans la notation abstraite et en MySQL). Par ailleurs la définition d'un ensemble de caractères par la négative, comme souvent, engendre du bruit. En effet, les obliques qui séparent les fragments dans un vers partagé ne correspondent pas à des majuscules. Les vers partagés sont donc rapatriés par la requête Access, même lorsque les obliques y sont suivis par des majuscules (début d'une nouvelle réplique). La requête Access ramène d'ailleurs 56 vers quand la requête MySQL supra en restitue 20.

3



En 4 un extrait des résultats.

4



Exercice n°4 Fournir le numéro et le vers pour les vers ne commençant pas par une majuscule.

Exercice n°5 Le tableau 61 p. 210 présente un « intrus » au sein des noms propres : *il*. On peut donc chercher les mots qui sont étiquetés comme des noms propres alors qu'ils débutent en minuscules. Pour rendre les résultats plus parlants, on fournira à chaque fois, le mot, le numéro de vers concerné et le vers en question.

Exercice n°6 Le vers 1252 *Ils ne se verront plus. / Ils s'aimeront toujours.*, où interviennent successivement Oenone et Phèdre, voit sa transcription phonétique effectuée par le métromètre marquée par une liaison impossible :

```
i l n @ s @ v ai r on p l u lz i l s ai m @@ r on t ou j ou r
```

Cette liaison intervient en effet entre deux tirades de deux personnages différents. Utiliser la table occurrences pour détecter de tels cas de figure. À titre d'aide, indiquons que la solution fait appel à une autojointure. . .

ESQUE

On s'en souvient, dans la table esque, 0 est le moyen de noter que l'année de l'attestation n'est pas connue. On peut donc chercher si les autres valeurs fournies sont « raisonnables » par la requête :

R^1_{77} *t. 1* p. 177

```
RESTRICTION(
annee > 0
ET ANNEE PAS ENTRE 1000 ET 2006
)
[esque]
```

Cette requête met en évidence des « années » erronées (186, 18, 16, 2201).

MySQL

```
SELECT annee
FROM attestations
WHERE annee > 0
    AND annee NOT BETWEEN 1000 AND 2006 ;
```

Dans le même esprit, la requête :

R^2_{115}

```
REGROUPER SUR(derive, cat_derive)[esque]
AVEC(Nombre de valeurs distinctes(base) > 1)
↪ PAR GROUPE(
derive,
Nombre de lignes() TITRE 'o.',
Nombre de valeurs distinctes(base) TITRE 'nbre bases')
)[<résultat1>]
```

permet de trouver les dérivés pour lesquels il existe plus d'une base.

MySQL

```
SELECT derive,
    COUNT(*) AS 'o.',
    COUNT(DISTINCT base) AS 'nbre bases'
FROM esque
GROUP BY derive, cat_derive
  HAVING COUNT(DISTINCT base) > 1;
```

3. Maintenir la cohérence

4. La base de données : un maillon dans une chaîne

Donnons-en deux exemples. Si l'on s'intéresse à la régularité des patrons rythmiques des vers de *Phèdre*, on peut chercher le nombre d'accents en positions 3 et 9, en se limitant aux vers prononcés par un seul personnage pour ne pas introduire les probables irrégularités dues au partage de la parole. On compare alors ce résultat avec la répartition par personnage

TABLEAU 93 – PHÈDRE : accents et personnages

Vers	*3*	*%*	*9*	*%*
1612	679	42.12	860	53.35

Personnage	*vers*	*3*	*%*	*9*	*%*
ARICIE	133	55	41.35	70	52.63
HIPPOLYTE	348	138	39.66	189	54.31
ISMENE	31	10	32.26	17	54.84
OENONE	202	71	35.15	113	55.94
PANOPE	34	17	50.00	22	64.71
PHEDRE	464	201	43.32	238	51.29
THERAMENE	176	88	50.00	90	51.14
THESEE	224	99	44.20	121	54.02

TABLEAU 94 – PHÈDRE : proportions des catégories chez les personnages

Personnage	*Mots*	*Adj*	*%*	*Nc*	*%*	*Np*	*%*	*V*	*%*
ARICIE	1174	107	9.11	217	18.48	20	1.70	200	17.04
HIPPOLYTE	3006	283	9.41	534	17.76	77	2.56	509	16.93
ISMENE	281	30	10.68	47	16.73	14	4.98	44	15.66
OENONE	1770	160	9.04	332	18.76	23	1.30	323	18.25
PANOPE	317	29	9.15	63	19.87	8	2.52	52	16.40
PHEDRE	4177	421	10.08	732	17.52	65	1.56	753	18.03
THERAMENE	1517	157	10.35	302	19.91	42	2.77	247	16.28
THESEE	1972	216	10.95	355	18.00	30	1.52	336	17.04

(tableau 93 p. 339, respectivement haut et bas). Il n'est pas possible simplement en examinant ces deux tableaux de savoir si certaines différences de proportions d'un personnage à l'autre sont effectivement significatives. Il en va de même lorsqu'on examine la répartition des catégories morpho-syntaxiques principales par personnage (tableau 94 p. 339).

PRÉMA

PHÈDRE

ESQUE

5. Solutions

Solution de l'exercice n°4 p. 337 Le résultat de la requête :

R^2_{116}
RESTRICTION(
vers RESSEMBLANT À '^[^A-Z]]'
)[vers]
↪ PROJECTION(
numero_vers TITRE 'N°',
vers
)[<résultat$_1$>]

TABLEAU 95 – PHÈDRE : vers ne commençant pas par une majuscule

N°	*vers*
195	À quel affreux dessein vous laissez-vous tenter ?
203	À ce fier ennemi de vous, de votre sang,
240	À l'horreur de vous voir expirer à mes yeux ?
249	Ô haine de Vénus ! O fatale colère !
266	Ô désespoir ! Ô crime ! Ô déplorable race !
417	Ô toi qui me connais, te semblait-il croyable
599	À votre inimitié j'ai pris soin de m'offrir.
813	Ô toi, qui vois la honte où je suis descendue,
877	À votre accusateur que pourrai-je répondre ?
956	Ô ciel, de ma prison pourquoi m'as-tu tiré ?
970	À ses monstres lui-même a servi de pâture ;
982	A-t-elle au criminel accordé quelque asile ?
1005	O tendresse ! O bonté trop mal récompensée !
1226	À quel nouveau tourment je me suis réservée !
1367	À nos amis communs portons nos justes cris ;
1480	O ciel ! Oenone est morte, et Phèdre veut mourir ?
1498	A peine nous sortions des portes de Trézène,
1541	À travers des rochers la peur les précipite ;
1561	"Le ciel, dit-il, m'arrache une innocente vie.
1571	O mon fils ! cher espoir que je me suis ravi !
1573	À quels mortels regrets ma vie est réservée !
1576	À la face des dieux l'accepter pour époux.

figure au tableau 95 p. 340. On constate qu'il ne s'agit pas vraiment de caractères commençant par une minuscule, mais de vers débutant soit par un guillemet (v. 1561) soit par une majuscule accentuée. On isole cependant un vers erroné, mais pour une tout autre raison : un accent superflu sur le A majuscule de *A-t-elle* au v. 982. Aurait-on intégré les majuscules accentuées qu'on n'aurait pas repéré cette scorie. À quelque chose bruit est bon, parfois...

MySQL

```
SELECT numero_vers AS 'N°',
       vers
FROM vers
WHERE vers REGEXP '^[^A–Z]' ;
```

La clause **WHERE** manifeste les deux emplois de l'accent circonflexe comme caractère spécial au sein d'une expression régulière, en premier lieu pour l'ancrer en indiquant le début de vers, ensuite, en première position au sein de crochets pour indiquer le complémentaire d'un ensemble de caractères.

Une autre manière est de recourir dans la clause **WHERE** à une classe de caractères (*character-class*) pré-définie, lower, qui englobe l'ensemble des caractères minuscules, accentués ou non :

```
WHERE vers REGEXP '^[:lower:]'
```

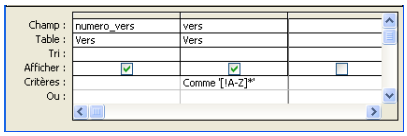
Avec cette restriction, plus forte que la précédente, le résultat est la table vide. Il n'y a pas de vers qui commencent par une minuscule.

Access

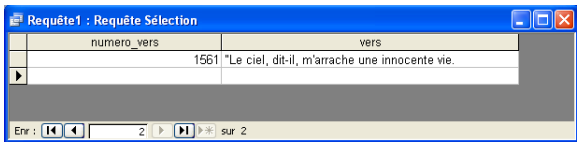
La requête 5 semble le strict équivalent de la première requête MySQL ci-dessus, à part

le passage de l'accent circonflexe au point d'exclamation en première position entre crochets pour indiquer le complémentaire d'un ensemble de caractères. Le résultat 6 ne comprend cependant qu'un vers, celui qui commence par un guillemet. Pour Access, les majuscules accentuées sont comprises dans l'ensemble des majuscules, ce n'est pas le cas pour MySQL.

5



6



Solution de l'exercice n°1 p. 336 Il s'agit de repérer des décalages. La solution aura donc recours à une jointure externe. Il faut au préalable constituer les tables à mettre en relation.

La première table, bebe_jour_nombre_fiches, associe à un bébé et un jour le nombre de fiches rédigé ce jour pour ce bébé. Une deuxième table, bebe_jour_present, associe à un bébé et à un jour un indicateur logique selon que le bébé est présent ou non dans le service de réanimation le jour en question.

La table bebe_jour_nombre_fiches comprend 378 lignes tandis que la table bebe_jour_present en comprend 484 (121 bébés × 4 jours), dont 384 correspondant à une présence. Il y a donc au moins 6 journées sans fiches.

MySQL

On peut créer d'abord la structure et ensuite la peupler. Il est également possible de la créer à la volée, ce qui équivaut aux requêtes Création de table d'Access :

```
CREATE TABLE bebe_jour_nombre_fiches
    (SELECT id_bebe,
            jour,
            COUNT(*)  AS 'nombre_fiches'
     FROM signaletique_fiches
     GROUP BY id_bebe, jour ) ;
```

On n'omettra pas de donner un alias aux colonnes calculées. C'est cet alias qui devient le nom de la colonne dans la table engendrée. Sans un tel alias, la troisième colonne s'appellerait... COUNT(*).

La structure de la table engendrée est la suivante :

Field	*Type*	*Null*	*Key*	*Default*	*Extra*
id_bebe	int(11)			0	
jour	int(11)			0	
nombre_fiches	bigint(21)			0	

La deuxième table est également engendrée à la volée pour les informations concernant le premier jour :

```
CREATE TABLE bebe_jour_present
      (SELECT id,
            1 AS 'jour',
            IF(etat_J1 = 'réa', 1, 0)  AS 'present'
       FROM bebes_princeps) ;
```

Elle est ensuite complétée pour les jours suivants :

```
INSERT INTO bebe_jour_present
      (SELECT id,
            3,
            IF(etat_J3 = 'réa', 1, 0)
       FROM bebes_princeps) ;
```

en changeant à chaque fois la valeur littérale de la deuxième colonne et le test de la troisième.

Une jointure externe permet d'isoler les 6 journées postulées :

```
SELECT *
FROM bebe_jour_present AS p
LEFT OUTER JOIN bebe_jour_nombre_fiches AS f
ON p.id = f.id_bebe
     AND p.jour = f.jour
WHERE p.present = 1
     AND f.id_bebe IS NULL ;
```

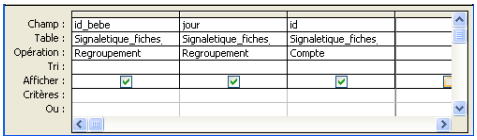
Le résultat est le suivant :

id	*jour*	*present*	*id_bebe*	*jour*	*nombre_fiches*
40	1	1	**NULL**	**NULL**	**NULL**
55	3	1	**NULL**	**NULL**	**NULL**
64	7	1	**NULL**	**NULL**	**NULL**
118	7	1	**NULL**	**NULL**	**NULL**
59	15	1	**NULL**	**NULL**	**NULL**
122	15	1	**NULL**	**NULL**	**NULL**

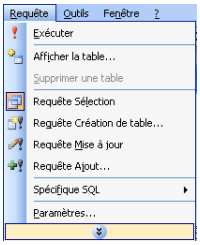
Access

La création de la table bebe_jour_nombre_fiches, qui repose sur les critères fournis en 7 demande que la requête soit transformée en ⌈Requête Création de table⌋ par l'appel au menu 8.

7

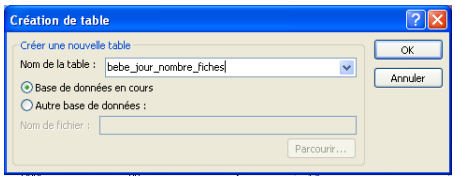


8



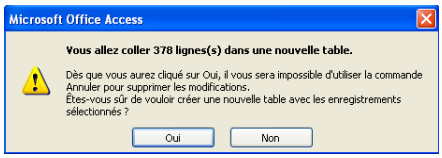
On fournit alors le nom de la table à créer : 9.

9



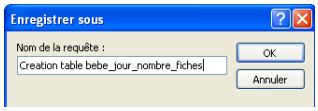
On est averti de l'ajout de lignes dans une nouvelle table : 10.

10

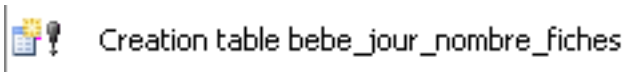


Lorsqu'on enregistre la requête 11, l'icône résultante pour une telle requête porte une table, un éclat de lumière et un point d'exclamation : 12.

11

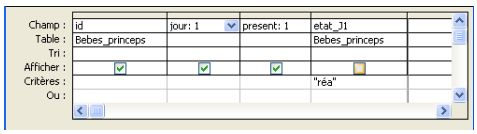


12



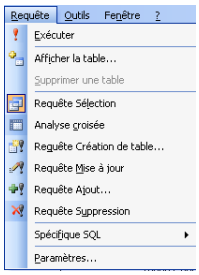
La création de la table bebe_jour_present s'effectue par une première requête Création de table : 13, pour le jour 1. Pour chaque identifiant de bébé tel que etat_J1 a pour valeur 'réa', on crée une ligne avec l'identifiant, la valeur 1 pour l'attribut jour et la valeur 1 pour l'attribut present.

13



Pour le jour 3, on crée ensuite une deuxième requête, cette fois-ci une requête Ajout : 14. Il s'agit en effet de compléter la table bebe_jour_present.

14

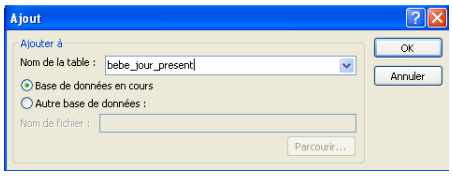


Cette requête 15 a une barre du haut différente et une ligne [Ajouter] en plus. Il faut indiquer les attributs dans lesquels se font les ajouts.

15

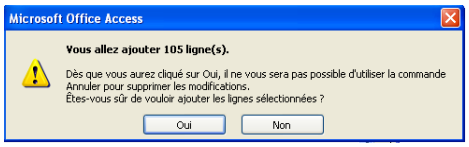
On doit ensuite choisir la table qui va être complétée : 16.

16



Au moment du déclenchement de la requête, une demande de confirmation est faite : 17.

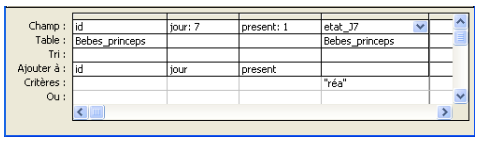
17



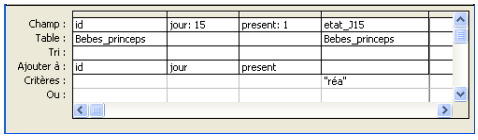
Les requêtes 18 et 19 sont une simple modification de la requête Ajout : on change à chaque fois la valeur littérale qui est ajoutée pour l'attribut `jour` (respectivement 3, 7 et 15) et l'attribut qui sert de critère de restriction (respectivement `etat_J3`, `etat_J3` et `etat_J15`). Sont ajoutées, pour le jour, 3 105 lignes, pour le jour 7, 85 et pour le jour 15, 73.

On note que la démarche diffère, à dessein, du traitement en MySQL. On insère dans la table `bebe_jour_present` uniquement des lignes pour les présences. L'attribut `present` a toujours pour valeur 1. La table `bebe_jour_present` comporte au total 384 lignes.

18

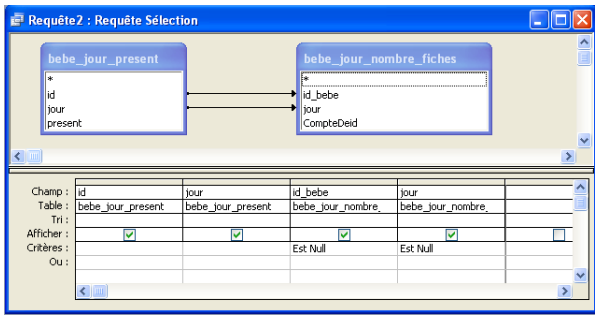


19



La comparaison entre les deux tables fait appel à une jointure externe, comme le montrent les flèches à la place de lignes dans la sous-fenêtre du haut de 20. La jointure « locale » a été obtenue en faisant glisser le nom d'un attribut d'une table sur son correspondant de l'autre table.

20



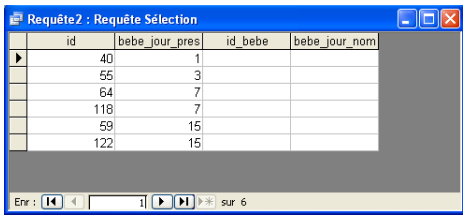
En outre, via le clic droit sur chaque ligne unissant deux noms, on a pu en 21 et en 22 choisir le type de jointure souhaité.

21

22

Le résultat figure en 23.

23



Solution de l'exercice n°2 p. 336 On conserve les lemmes dont la catégorie commence par V, c'est-à-dire les seuls verbes. On élimine les lemmes qui comprennent les verbes *aller*, *être*, etc. Enfin et surtout, on élimine les lemmes qui se terminent par *ir*, *re* ou *er*. Pour cela, on vérifie que les deux derniers caractères du lemme ne figurent pas dans cette liste. Si l'on examine au ralenti la manière d'obtenir ce résultat, on commence par inverser le lemme, on garde les deux premières lettres du lemme inversé (donc les deux dernières du lemme non inversé) et on réinverse ces lettres pour pouvoir les comparer aisément avec les terminaisons visées.

TABLEAU 96 – PRÉMA : lemmes de verbes à vérifier

lemme	*forme_normalisee*	*o.*
attaché	attaché	1
c'est	c'est	179
changer d'idée	changer d'idée	1
dialogue	dialogue	1
donne l'impression	donne l'impression	1
donner envie	donne envie	3
donner l'impression	donne l'impression	2
ennuie	ennuie	1
entrer en contact	entrer en contact	2
ficher la paix	fiche la paix	1
il s'agit de	il s'agit d'	2
il y a	il y a	5
perdre de le poids	a perdu de le poids	1
se passer mal	se passe mal	1
se porter bien	se porte bien	1
se sentir en sécurité	se sent en sécurité	1
tirer la langue	tirer la langue	1
était	était	4

R^{2}_{117}

RESTRICTION(
categorie RESSEMBLANT À '^V'
ET INVERSER(SOUSCHAÎNE(INVERSER(lemme), 1, 2)) PAS PARMI ('ir', 're', 'er')
ET lemme NE RESSEMBLANT PAS À 'aller | être | avoir | faire | prendre | mettre'
)[occ_prema]
↪ REGROUPER SUR(lemme)[<résultat$_1$>]
↪
PAR GROUPE(
lemme,
forme_normalisee,
NOMBRE DE LIGNES() TITRE 'o.'
)[<résultat$_2$>]

Le résultat, au tableau 96 p. 346, fournit pour le lemme, la forme normalisée, et le nombre d'occurrences de cette forme avec ce lemme. On trouve quelques verbes en plusieurs mots qui n'ont pas été éliminés (*changer d'idée*, *entrer en contact*, etc.). Les lignes qui ne sont pas des hapax correspondent généralement à des choix de lemmatisation intentionnels : le lemme de *c'est* est... *c'est*, avec 179 o. Il en va de même pour *il s'agit de* ou *il y a*. Mais on isole par ailleurs de vraies erreurs : *attaché*, *dialogue*, *ennuie*, *donne l'impression* et surtout *était* avec 4 o.

MySQL

```
SELECT lemme,
       forme_normalisee,
       COUNT(*) AS 'o.'
FROM occ_prema
WHERE categorie REGEXP '^V.*'
      AND REVERSE(SUBSTRING(REVERSE(lemme), 1, 2))
```

```
        NOT IN ('ir', 're', 'er')
    AND lemme
        NOT REGEXP 'aller|être|avoir|faire|prendre|mettre'
GROUP BY lemme ;
```

Une autre manière d'extraire la terminaison du lemme pour vérifier sa non présence dans les terminaisons d'infinitifs est la suivante :

```
... SUBSTRING(lemme, LENGTH(lemme) -1, 2)
        NOT IN ('ir', 're', 'er')
```

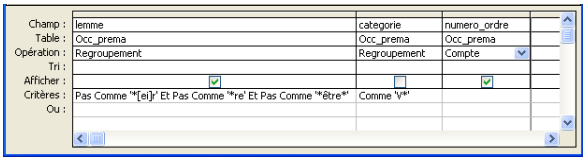
Access

La requête 24 constitue à dessein une variante par rapport à la version MySQL supra. On a recours à un motif, on cherche l'absence de 'ir', 'er' ou 're' en fin de lemme, plutôt qu'à des inversions et troncations de chaîne. On constate par ailleurs que les motifs d'Accesss et de MySQL Server sont plus frustes que ceux de MySQL et qu'il faudrait rajouter :

Et Pas Comme '*<verbe>*'

pour chacun des verbes courants pouvant figurer dans des « mots en plusieurs mots » et dont les lignes correspondantes sont donc à éliminer.

24



Solution de l'exercice n°3 p. 336 On trouve au tableau 97 p. 348 les formes normalisées ayant des lemmes distincts. Le tableau 98 p. 349 donne les formes normalisées, leur partie du discours, le lemme correspondant et la fréquence de l'association. La présence de la partie du discours (ici le premier caractère de la catégorie) met en évidence l'ambiguïté catégorielle de certains mots : *fait*, nom et verbe ; *calme*, nom et adjectif ; *continue*, adjectif et verbe. Le tableau met cependant en évidence des incohérences : + comme adverbe (R) ayant pour lemme tantôt ++ tantôt *plus* ; *donne l'impression* renvoyé à *donner l'impression* mais aussi à *donne l'impression* ; *est arrivé* dont le lemme est tantôt l'infinitif actif tantôt l'infinitif passif. Il en va de même encore pour *était* et *éveillé*.

MySQL

La délimitation des formes normalisées ayant plusieurs lemmes distincts découle de :

```
SELECT forme_normalisee
FROM occ_prema
GROUP BY forme_normalisee
    HAVING COUNT(DISTINCT lemme) > 1 ;
```

Pour obtenir les lemmes correspondants, deux démarches sont possibles.

La première passe par la création d'une table temporaire, « à la volée », pour mémoriser les formes normalisées en question :

```
CREATE TABLE formes_normalisees_a_plusieurs_lemmes_tmp
      SELECT forme_normalisee
      FROM occ_prema
      GROUP BY forme_normalisee
          HAVING COUNT(DISTINCT lemme) > 1 ;
```

TABLEAU 97 – PRÉMA : formes normalisées ayant des lemmes distincts

forme_normalisee
+
-
/
approche
calme
caresse
ce
change
continue
couche
de
donne l'impression
droite
est arrivé
fait
grimace
pose
présente
remuant
reste
sécurisé
une
écoute
était
éveillé

On peut alors, par jointure, obtenir les lemmes correspondants et les fréquences des formes normalisées associées :

```
SELECT o.forme_normalisee,
       lemme,
       SUBSTRING(categorie, 1, 1) AS POS,
       COUNT(*) AS 'o.'
FROM occ_prema AS o,
       formes_normalisees_a_plusieurs_lemmes_tmp AS t
WHERE o.forme_normalisee = t.forme_normalisee
GROUP BY o.forme_normalisee, lemme ;
```

Rien n'empêche enfin, c'est la deuxième démarche, de faire l'économie d'une table temporaire et de recourir à une requête enchâssée :

```
SELECT forme_normalisee,
       lemme,
       SUBSTRING(categorie, 1, 1) AS POS,
       COUNT(*) AS 'o.'
FROM occ_prema
WHERE forme_normalisee IN
       (SELECT forme_normalisee
       FROM occ_prema
       GROUP BY forme_normalisee
           HAVING COUNT(DISTINCT lemme) > 1)
GROUP BY forme_normalisee, lemme ;
```

On notera que les requêtes enchâssées, même relativement simples comme celle-ci, sont longues à exécuter. À titre de comparaison, avec la version 14.7 de MySQL, sous Linux (De- △

TABLEAU 98 – PRÉMA : lemmes distincts pour une même forme

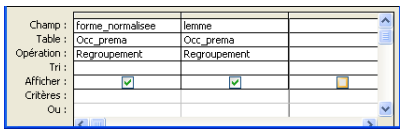
forme_normalisee	*lemme*	*POS*	*o.*
+	++	R	2
+	et	C	4
+	plus	R	3
-	-	Y	901
-	moins	R	1
/	,	Y	2
/	par rapport à	S	1
approche	approche	N	1
approche	approcher	V	1
calme	calme	A	427
calme	calmer	V	1
caresse	caresse	N	3
caresse	caresser	V	8
ce	ce	D	42
ce	cela	P	4
change	change	N	4
change	changer	V	4
continue	continu	A	3
continue	continuer	V	1
couche	couche	N	1
couche	coucher	V	1
de	de	S	430
de	un	D	4
donne l'impression	donne l'impression	V	1
donne l'impression	donner l'impression	V	2
droite	droit	A	1
droite	droite	N	1
est arrivé	arriver	V	1
est arrivé	être arrivé	V	1
fait	faire	V	51
fait	fait	N	1
grimace	grimace	N	1
grimace	grimacer	V	20
pose	pose	N	4
pose	poser	V	7
présente	présent	A	15
présente	présenter	V	3
remuant	remuant	A	1
remuant	remuer	V	1
reste	reste	N	1
reste	rester	V	36
sécurisé	sécuriser	V	1
sécurisé	sécurisé	A	1
une	K	D	1
une	un	D	128
écoute	écoute	N	1
écoute	écouter	V	15
était	était	V	4
était	être	V	3
éveillé	éveiller	A	1
éveillé	éveillé	A	47

bian), la requête précédente, qui passe par une table temporaire s'est exécutée en 1.47 seconde, la création de la table temporaire ayant demandé 1.93 seconde, tandis que la requête enchâssée en a demandé littéralement des heures.

Access

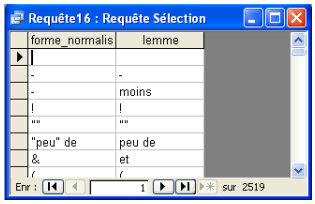
Access ne permettant pas de calculer directement le nombre de valeurs distinctes d'une colonne donnée, l'approche est un peu plus contournée. Une première requête 25 opère un regroupement par forme normalisée et par lemme.

25



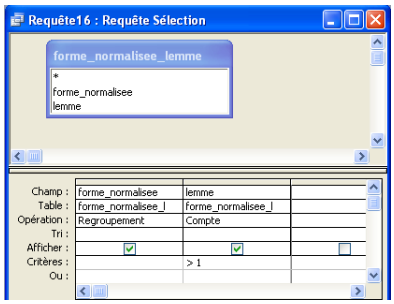
On observe dans le début des résultats 26 une forme normalisée ayant des lemmes distincts : le tiret.

26

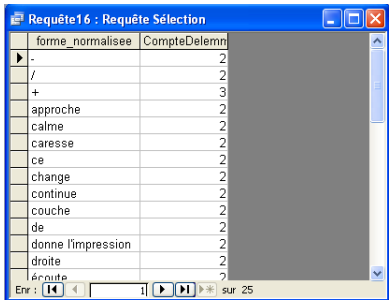


La deuxième requête 27 prend en entrée la requête 25 : elle opère cette fois un regroupement par forme normalisée et un décompte du nombre de lemmes associés : 28.

27



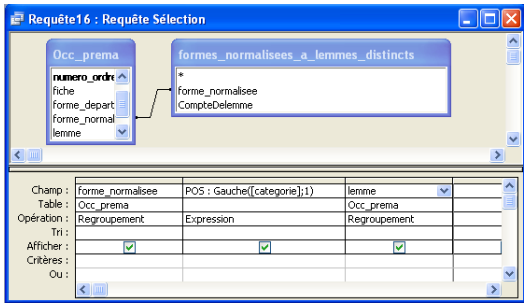
28



La requête suivante 29 opère une jointure « locale » entre la table occ_prema et la requête précédente, sur la valeur de la colonne forme_normalisee (en faisant glisser le nom de l'attribut

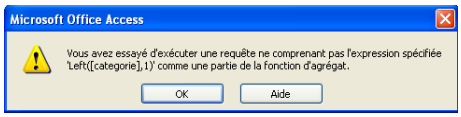
d'une table sur le nom identique de l'autre table). C'est le moyen de restreindre l'examen de la table occ_prema aux seules formes normalisées qui ont plus d'un lemme.

29



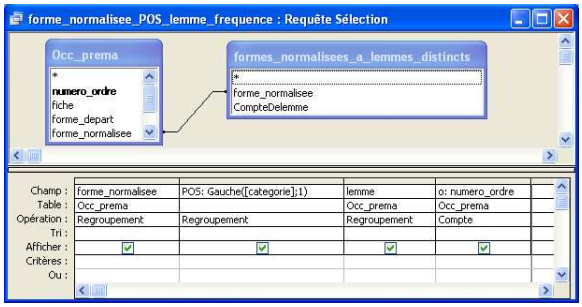
Cette troisième requête, pourtant proche de son équivalent MySQL, n'est cependant pas exécutable par Access : 30, qui ne semble pas accepter une colonne calculée entre deux colonnes utilisées pour des regroupements.

30

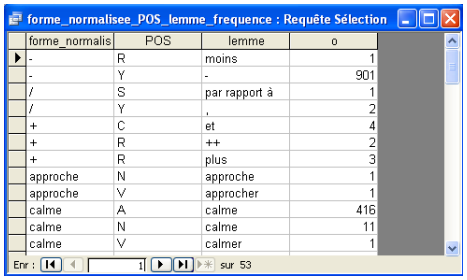


La requête corrigée 31 qui déplace le calcul de la partie du discours après les regroupements, aboutit par contre au résultat souhaité : 32.

31

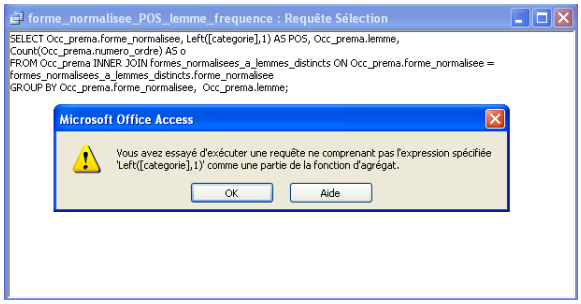


32



On notera que cette différence de comportement entre MySQL et Access n'est pas propre à l'interface que constitue Access. La requête SQL Server qui interpose un attribut calculé entre deux attributs servant par ailleurs aux regroupements déclenche aussi une erreur : 33.

33



Solution de l'exercice n°5 p. 337 Le résultat est fourni au tableau 99 p. 353. Les informations concernant 23 vers sont à corriger.

Il s'agit d'isoler les occurrences de catégorie Np (pour nom commun) et dont la représentation graphique ne commence pas par une majuscule. La jointure avec la table vers permet de resituer le mot mal étiqueté dans son contexte.

R^2_{118}
JOINTURE(
v.numero_vers = o.numero_vers
)[vers ALIAS v, occurrences ALIAS o]
↪ RESTRICTION(
cat = 'Np'
ET occ_car NE RESSEMBLANT PAS À '^[A-Z]'
)[<résultat$_1$>]
↪
PROJECTION(
o.numero_vers,
occ_car,
vers
)[<résultat$_2$>]

MySQL

```
SELECT o.numero_vers,
       occ_car,
       vers
FROM occurrences AS o,
       vers AS v
WHERE v.numero_vers = o.numero_vers
      AND cat = 'Np'
      AND occ_car NOT REGEXP '^[A-Z]' ;
```

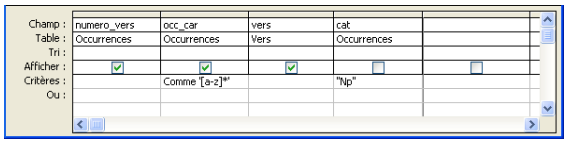
Access

La requête qui vient immédiatement à l'esprit 34, qui teste la présence de minuscules en début de l'attribut occ_car, ne fonctionne pas : 35. Access ne distingue en effet pas les majuscules des minuscules dans les motifs.

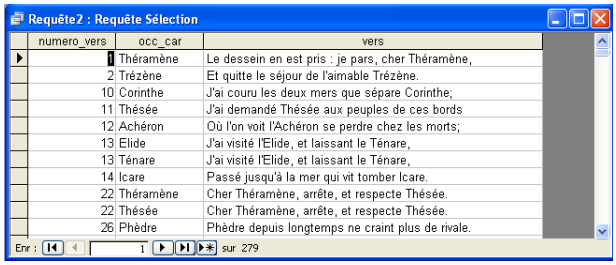
TABLEAU 99 – PHÈDRE : noms propres sans majuscules

numero_vers	*occ_car*	*vers*
140	est	Seigneur? / C'est mon dessein : tu peux l'en avertir.
334	il	Qu'il n'entraîne après lui tout un peuple volage.
384	avec	Qu'avec Pirithoüs aux enfers descendu,
399	il	Qu'il plaindra mes malheurs? / Madame, je le croi.
440	il	Qu'il méprise lui-même, et qu'il semble ignorer.
488	une	Qu'une superbe loi semble me rejeter.
546	envi	Tout vous livre à l'envi le rebelle Hippolyte.
562	ignore	Elle vous cherche. / Moi? / J'ignore sa pensée.
617	un	Qu'un soin bien différent me trouble et me dévore!
801	il	Qu'il mette sur son front le sacré diadème;
823	il	Qu'il aime... mais déjà tu reviens sur tes pas,
1080	ils	Qu'ils m'ôtent la parole et m'étouffent la voix.
1221	offensait	Qu'offensait le respect, qu'importunait la plainte,
1261	il	Qu'il ne se borne pas à des peines légères :
1282	il	Lorsqu'il verra sa fille à ses yeux présentée,
1289	dieu	Pardonne. Un dieu cruel a perdu ta famille;
1307	entends	Qu'entends-je? Quels conseils ose-t-on me donner?
1469	entends	Qu'entends-je? / Son trépas n'a point calmé la reine :
1481	on	Qu'on rappelle mon fils, qu'il vienne se défendre!
1482	il	Qu'il vienne me parler, je suis prêt de l'entendre.
1493	ai	Dieux! / J'ai vu des mortels périr le plus aimable,
1540	dieu	Un dieu qui d'aiguillons pressait leur flanc poudreux.
1645	une	Elle expire, Seigneur! / D'une action si noire

34



35



Solution de l'exercice n°6 p. 337 L'examen attentif des lignes de la table occurrences correspondant au vers 1252 (tableau 100 p. 354) permet d'abstraire un « motif » de liaison « mal't à propos » : une liaison est indiquée pour une occurrence suivie presque immédiatement après d'une ligne dont l'attribut personnage a pour valeur ENTRE_PERSONNAGES, c'est-à-dire qu'on change dans les faits de locuteur. Le mot à liaison malencontreuse est séparé de la ligne portant ENTRE_PERSONNAGES par une ou plusieurs lignes qui correspond(ent) à une ponctuation ou à des espaces. Dans le cas du vers 1252, le mot *plus* prononcé par Œnone et censé lié au mot suivant qui est prononcé en fait par Phèdre (*Ils*), est séparé de la ligne correspondante à ENTRE_PERSONNAGES par la ligne du point final de la tirade d'Œnone.

Une auto-jointure de la table occurrences avec elle-même permet de chercher les mots à liaison tels qu'il y ait à droite un changement de personnage. Il faut ensuite affiner les contraintes

TABLEAU 100 – PHÈDRE : liaison en fin de tirade

numero_vers	*personnage*	*occ_car*	*occ_phon*	*liaison*
1252	OENONE	Ils	i l	-
1252	OENONE		-	-
1252	OENONE	ne	n @	-
1252	OENONE		-	-
1252	OENONE	se	s @	-
1252	OENONE		-	-
1252	OENONE	verront	v ai |r on	-
1252	OENONE		-	-
1252	OENONE	plus	p l u	lz
1252	OENONE	.	-	-
1252	ENTRE_PERSONNAGES		-	-
1252	ENTRE_PERSONNAGES	/	-	-
1252	ENTRE_PERSONNAGES		-	-
1252	PHEDRE	Ils	i l	-
1252	PHEDRE		-	-
1252	PHEDRE	s'	s	-
1252	PHEDRE	aimeront	ai |m @@ |r on	-
1252	PHEDRE		-	-
1252	PHEDRE	toujours	t ou |j ou r	-
1252	PHEDRE	.	-	-

pour éviter le bruit.

MySQL

Une première requête s'impose :

```
SELECT o1.numero_vers,
     o1.occ_car, vers,
     vers_phonetique
FROM occurrences AS o1, occurrences AS o2, vers AS v
WHERE o1.numero_vers = o2.numero_vers
     AND o1.liaison REGEXP '^l.*'
     AND o2.personnage = 'ENTRE_PERSONNAGES'
     AND o1.numero_ligne < o2.numero_ligne
     AND v.numero_vers = o1.numero_vers ;
```

On cherche toutes les combinaisons d'occurrences (auto-jointure) au sein d'un même vers telles que :

- la première occurrence comporte une liaison (le l est la marque conventionnelle d'une liaison, le phonème venant après indiquant la nature de la liaison – cf. tableau 56 p. 205) ;
- la deuxième occurrence est une marque de transition entre personnages (ENTRE_PERSONNAGES) ;
- la première occurrence intervient « avant » la seconde (son numéro de ligne est inférieur à celui de la seconde).

Pour que le résultat soit utilisable (tableau 101 p. 355), on a ajouté une jointure avec la table vers pour pouvoir afficher le vers lui-même et sa transcription phonétique. Nous ne gardons que 12 lignes sur les 48 du résultat effectif. Deux constats : cette requête isole effectivement des liaisons malencontreuses (v. 151 et v. 1493) mais elle « bégaie ». Un même vers est présenté plusieurs fois. On comprend qu'il y a autant de « combinaisons gagnantes » que de lignes «

TABLEAU 101 – PHÈDRE : repérage liaisons malencontreuses (version 1)

numero_vers	*occ_car*	*vers*	*vers_phonetique*
151	vient	Elle vient. / Il suffit : je la laisse en ces lieux,	ai l @@ v y in lt i l s u f i j @ l a l ai s an s e l y eu
151	vient	Elle vient. / Il suffit : je la laisse en ces lieux,	ai l @@ v y in lt i l s u f i j @ l a l ai s an s e l y eu
151	vient	Elle vient. / Il suffit : je la laisse en ces lieux,	ai l @@ v y in lt i l s u f i j @ l a l ai s an s e l y eu
⋮	⋮	⋮	⋮
1252	plus	Ils ne se verront plus. / Ils s'aimeront toujours.	i l n @ s @ v ai r on p l u lz i l s ai m @@ r on t ou j ou r
1252	plus	Ils ne se verront plus. / Ils s'aimeront toujours.	i l n @ s @ v ai r on p l u lz i l s ai m @@ r on t ou j ou r
1252	plus	Ils ne se verront plus. / Ils s'aimeront toujours.	i l n @ s @ v ai r on p l u lz i l s ai m @@ r on t ou j ou r
1493	Dieux	Dieux! / J'ai vu des mortels périr le plus aimable,	d y eu lz j ai v u d e m o r t ai l p e r i l @ p l u lz ai m a b l
1493	Dieux	Dieux! / J'ai vu des mortels périr le plus aimable,	d y eu lz j ai v u d e m o r t ai l p e r i l @ p l u lz ai m a b l
1493	Dieux	Dieux! / J'ai vu des mortels périr le plus aimable,	d y eu lz j ai v u d e m o r t ai l p e r i l @ p l u lz ai m a b l

suivant » le mot avec liaison au sein du même vers et marquées ENTRE_PERSONNAGES. Les 3 lignes de cette sorte du v. 1252 donnent effectivement naissance à 3 lignes dans le tableau 101 p. 355. La solution est simple : il suffit de garder un seul exemplaire de chaque combinaison :

```
SELECT DISTINCT o1.numero_vers, o1.occ_car, vers, vers_phonetique
...
```

Le résultat (tableau 102 p. 356) ne comprend plus que 15 lignes. La répétition du v. 463 n'est pas une erreur dans l'élimination des doublons : il y a 2 mots avec liaisons (*vous* et *vais*) qui sont successivement mis en correspondance avec la ligne portant ENTRE_PERSONNAGES qui « suit ». Au v. 763, il y a deux liaisons erronées liées à un vers partagé en trois et d'autant plus erronées que les phonèmes suivant ces liaisons sont dans les deux cas des consonnes. Le v. 927 surprend : pas de liaison malencontreuse dans la transcription phonétique. En fait, c'est une incohérence entre la transcription du vers qui a probablement été corrigée quant à cette liaison inopportune, tandis que la table occurrences n'a pas été amendée parallèlement, comme permet de le vérifier la requête (tableau 103 p. 357) :

```
SELECT numero_vers, personnage, occ_car, liaison
FROM occurrences
WHERE numero_vers = 927 ;
```

On peut contraindre enfin la ligne portant ENTRE_PERSONNAGES à n'être pas trop éloignée de la ligne avec liaison. On ajoute alors :

```
... AND o2.numero_ligne < o1.numero_ligne + 3...
```

dans les contraintes de la requête de repérage. Le doublon pour le v. 463 disparaît. Il en reste 2 pour le v. 763, pour des raisons proches du v. 927 : une des liaisons inélégantes a été enlevée de la ligne idoine de la table vers, il en reste une, par contre, la table occurrences n'a pas été corrigée en cohérence et les deux liaisons fâcheuses y figurent.

L'examen de la situation « prototypique » qu'est le vers 1252 amène d'ailleurs à se demander s'il ne serait pas nécessaire de vérifier s'il n'y a pas des liaisons après certaines fins de phrase.

TABLEAU 102 – PHÈDRE : repérage liaisons malencontreuses (version 2)

numero_vers	*occ_car*	*vers*	*vers_phonetique*
151	vient	Elle vient. / Il suffit : je la laisse en ces lieux,	ai l @@ v y in lt i l s u f i j @ l a l ai s an s e l y eu
205	dieux	Cet Hippolyte... / Ah, dieux! / Ce reproche vous touche.	s ai i p o l i t a d y eu lz s @ r @ p r o sh @@ v ou t ou sh
259	-vous	Aimez-vous? / De l'amour j'ai toutes les fureurs.	e m e v ou lz d @ l a m ou r j ai t ou t @@ l e f u r oe r
264	dieux	Hippolyte? grands dieux! / C'est toi qui l'as nommé.	i p o l i t @@ g r an d y eu lz s ai t w a k i l a n o m e
399	malheurs	Qu'il plaindra mes malheurs? / Madame, je le croi.	k i l p l in d r a m e m a l oe r lz m a d a m @@ j @ l @ k r w a
463	vient	Il vient à vous. / Madame, avant que de partir,	i l v y in lt a v ou lz m a d a m a v an k @ d @ p a r t i r
463	vous	Il vient à vous. / Madame, avant que de partir,	i l v y in lt a v ou lz m a d a m a v an k @ d @ p a r t i r
670	vais	Et je vais... / Ah! cruel, tu m'as trop entendue.	e j @ v ai lz a k r u ai l t u m a t r oo lp an t an d u
763	meurs	Quand je me meurs! / Fuyez. / Je ne le puis quitter.	k an j @ m @ m oe r lz f yw i y ai lz j @ n @ l @ p yw i k i t e
763	Fuyez	Quand je me meurs! / Fuyez. / Je ne le puis quitter.	k an j @ m @ m oe r lz f yw i y ai lz j @ n @ l @ p yw i k i t e
839	mourez	Vous mourez? / Juste ciel! qu'ai-je fait aujourd'hui?	v ou m ou r ai lz j u s t @ s y ai l k ai j @ f ai lt oo j ou r lt yw i
914	met	Madame; et dans vos bras met... / Arrêtez, Thésée,	m a d a m e d an v oo b r aa m ai lt a r e t e t e z e
927	quitter	Vous, mon fils, me quitter? / Je ne la cherchais pas :	v ou m on f i s m @ k i t e j @ n @ l a sh ai r sh ai p aa
1252	plus	Ils ne se verront plus. / Ils s'aimeront toujours.	i l n @ s @ v ai r on p l u lz i l s ai m @@ r on t ou j ou r
1493	Dieux	Dieux! / J'ai vu des mortels périr le plus aimable,	d y eu lz j ai v u d e m o r t ai l p e r i l @ p l u lz ai m a b l

TABLEAU 103 – PHÈDRE : repérage liaisons malencontreuses (version 3)

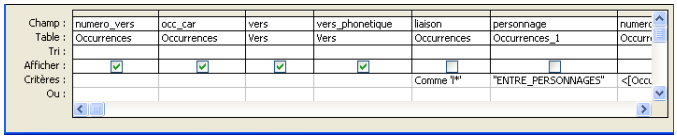
numero_vers	*personnage*	*occ_car*	*liaison*
927	THESEE	Vous	-
927	THESEE	,	-
927	THESEE		-
927	THESEE	mon	-
927	THESEE		-
927	THESEE	fils	-
927	THESEE	,	-
927	THESEE		-
927	THESEE	me	-
927	THESEE		-
927	THESEE	quitter	lr
927	THESEE	?	-
927	ENTRE_PERSONNAGES		-
927	ENTRE_PERSONNAGES	/	-
927	ENTRE_PERSONNAGES		-
927	HIPPOLYTE	Je	-
927	HIPPOLYTE		-
927	HIPPOLYTE	ne	-
927	HIPPOLYTE		-
927	HIPPOLYTE	la	-
927	HIPPOLYTE		-
927	HIPPOLYTE	cherchais	-
927	HIPPOLYTE		-
927	HIPPOLYTE	pas	-
927	HIPPOLYTE	:	-

Normalement, une phrase constitue un groupe rythmique autonome, qui ne se lie pas avec la suivante.

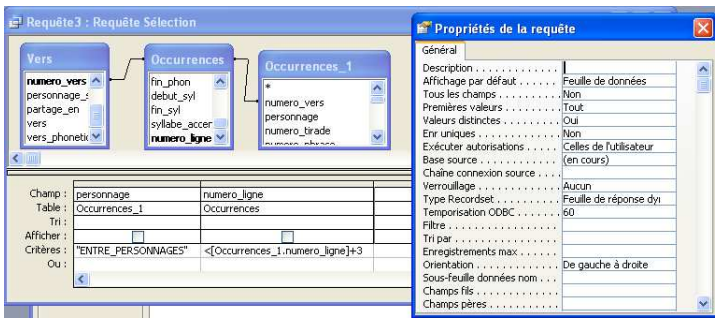
Access

Les écrans 36 et 37 décomposent la requête sous Access.

36



37



En tout état de cause, cet exercice fait toucher du doigt la difficulté des bases de données

relationnelles à prendre en compte les phénomènes de séquentialité, puisqu'il s'agit de repérer dans les lignes de la table occurrences correspondant à un même vers celles qui comprennent par un mot à liaison et qui sont suivies par une marque de changement de personnage, malgré l'interposition d'espaces ou de ponctuation. L'auto-jointure est un palliatif dont on a vu les limites.

Cet exercice montre aussi que la traque aux erreurs est sans fin. La requête a d'ailleurs mis au jour des incohérences insoupçonnées entre les tables vers et occurrences. Elle témoigne aussi du louvoiement nécessaire entre silence et bruit. Elle montre enfin que les SGBD sont les premiers moyens de repérer les incohérences des bases qu'ils servent à utiliser.

CHAPITRE XII

⊕ RÉORGANISER UNE BASE DE DONNÉES

L'objectif est d'abord de détecter automatiquement le maximum d'incohérences et de redondances dans la table esque, d'y remédier dans la mesure du possible et de marquer les problèmes rencontrés et les amendements apportés (§ 1). Une deuxième étape, après transferts de données d'un environnement à un autre (§ 2), est de vérifier les résultats des premières modifications automatiques et d'effecter des corrections manuelles (§ 3). Il s'agit ensuite, toujours guidé par le schéma Entités/Associations dégagé au ch. IX, de créer une base de données articulant plusieurs tables à partir des informations « consolidées » de la table esque (§ 4). On peut enfin tirer les leçons de cette tentative de remaniement des informations issues de la table esque (§ 5).

La réalisation de ce chapitre associe MySQL et Access d'une manière différente des autres chapitres de ce deuxième tome, non plus en parallèle, mais de manière alternée. MySQL a été privilégié pour l'analyse, le signalement et les corrections automatiques des redondances et incohérences. Le mode ligne de commande et les requêtes SQL sont plus rapides que le passage par de multiples menus, choix, etc. Les atouts d'Access ont par contre été pleinement exploités pour les corrections manuelles fines, où l'interface graphique est particulièrement cruciale. Il faut l'avouer tout net : l'interface offerte par Phpmyadmin est relativement sommaire et insuffisante pour des ajustements minutieux et nombreux.

1. Traquer et signaler les incohérences

Il s'agit d'une démarche de longue haleine, non exempte de tâtonnements et de décisions erronées. On s'appuie au maximum sur les opérations des SGBD pour cette « chasse ».

L'objectif général est de préparer la transformation de la table unique de départ esque en une véritable base de données relationnelle, en fonction de la modélisation Entité/Association du ch. IX. Pour ce faire, il faut :

1. « atomiser » les valeurs. Il est préférable que la valeur d'un attribut soit atomique, c'est-à-dire ne rassemble pas plusieurs informations hétérogènes. On l'a vu, dans la table esque de départ, le mot-base rassemble plusieurs facettes distinctes : le mot lui-même, mais éventuellement l'origine étrangère, et, parfois, le degré de confiance avec lequel cette base a été attribuée au dérivé, etc. Il en va de même de la catégorie des bases et des dérivés qui conjoint partie du discours et traits morphologiques. On n'a dès lors pas accès à chacune des facettes indépendamment. Il convient alors de séparer les informations, pour aboutir à plusieurs attributs contenant chacun une information « minimale », atomique.
2. éliminer les redondances, surtout quand elles débouchent sur des incohérences. Dans la table esque de départ, les informations sur une base ou sur un dérivé sont dupliquées entre les lignes qui les comportent, et ce, avec des décalages dans les informations fournies. On s'attache alors à « consolider » les bases et les dérivés, en fusionnant les données les concerant.

On crée une nouvelle table esque_tmp qui, au départ, est une simple copie de la table esque. L'objectif est d'abord de ne pas risquer de détériorer les données de départ : il est toujours possible d'y revenir en reprenant celles de la table esque. En second lieu, la structure de cette table est enrichie au fur et à mesure d'attributs tmémorisant des versions consolidées automatiquement des attributs de départ (par exemple un jeu de catégories nettoyé pour les attributs cat_base et cat_derive). Enfin, on souhaite pouvoir vérifier manuellement les corrections faites automatiquement et corriger manuellement les lignes problématiques. On ajoute alors à la table esque_tmp des attributs permettant de signaler les lignes corrigées automatiquement ou considérées comme problématiques, le premier pour ce qui concerne la base, le second pour le dérivé, le troisième pour le contexte :

```
ALTER TABLE esque_tmp
     ADD COLUMN verif_base VARCHAR(255)  DEFAULT "";

ALTER TABLE esque_tmp
     ADD COLUMN verif_derive VARCHAR(255)  DEFAULT "";

ALTER TABLE esque_tmp
     ADD COLUMN verif_contexte VARCHAR(255)  DEFAULT "";
```

Ces attributs reçoivent pour valeur par défaut la chaîne vide. Au fur et à mesure des constats d'incohérence ou des amendements automatiques, des requêtes vont ajouter à chaque fois une chaîne indiquant une correction ou un problème. Les mises à jour sont de la forme :

```
UPDATE esque_tmp SET verif_... = CONCAT(verif_..., " <chaîne ajoutée>")
```

Pour chacun des aspects examinés dans cette section, la démarche sera la même :

1. constat grâce aux opérations des SGBD sur les incohérences, manques, etc. ;
2. éventuellement, début de correction automatique ;
3. marquage des lignes qui ont été repérées et/ou amendées.

1.1. Jeux de catégories pour les mots bases et les dérivés

1.1.1. Jeux de départ : problèmes

Les requêtes :

```
SELECT cat_base AS 'Bases',
     COUNT(*) AS 'o'
FROM esque
GROUP by cat_base ;

SELECT cat_derive AS 'Dérivés',
     COUNT(*) AS 'o'
FROM esque
GROUP by cat_derive ;
```

isolent les catégories employées pour les bases et pour les dérivés ainsi que leurs occurrences. L'examen du tableau 104 p. 362 met en évidence les problèmes suivants :

1. dans un nombre de cas non négligeable, la catégorie n'est pas renseignée. La présence de la marque **NULL** ne permet pas de savoir s'il s'agit d'un oubli ou d'une incertitude sur la catégorie à employer. On rencontre par contre le point d'interrogation comme marque d'incertitude assumée.
2. Une catégorie peut être attribuée, mais avec un certain degré d'incertitude, ce que marque un point d'interrogation post-posé : npr?, etc.
3. Il y a hésitation entre deux catégories, séparées alors par une barre oblique : a/nm.
4. Des variantes graphiques ou de notation renvoient à une même catégorie : nnpr et npr pour nom propre, acron et sigle également.
5. Certaines précisions, de genre ou nombre pour les noms (nms, nfpl, etc.), ou sur la nature locutionnelle (« mots en plusieurs mots ») de certains adjectifs ou adverbes notés ladj et ladv pour locution adjective et locution adverbiale, émiettent certaines catégories et compliquent les rapprochements : compter les noms parmi les dérivés, c'est compter les lignes dont la catégorie figure dans l'ensemble : {'n', 'nf', 'nfpl', 'nm', 'npl', 'n?'}. On trouve d'ailleurs loc pour locution comme seule catégorie, sans qu'on connaisse la partie du discours dont relèvent les locutions en question.

1.1.2. Réorganisation des catégories

Dérivés Pour les catégories des dérivés, on propose de distinguer les trois nouveaux attributs suivants :

1. derive_POS : la partie du discours, abrégée en POS, qui prend ses valeurs dans l'ensemble : {absente, ?, a, adv, n, v}. La valeur absente correspond à la marque **NULL** dans l'attribut cat_derive, et le point d'interrogation, décalque du point d'interrogation de l'attribut cat_derive, signifie que la catégorie du dérivé est douteuse.
2. derive_POS_douteuse, qui aura pour valeur 1 quand la partie du discours attribuée reste douteuse (c'est le cas pour deux noms) et 0 sinon.
3. derive_POS_traits : les traits spécifiant une catégorie (par exemple le genre et le nombre pour un nom, la construction pour un verbe).

On ajoute donc les colonnes idoines à la table esque_tmp via les requêtes suivantes :

```
ALTER TABLE esque_tmp
     ADD COLUMN derive_POS varchar(10) NOT NULL DEFAULT "absente" ;

ALTER TABLE esque_tmp
     ADD COLUMN derive_POS_douteuse INT NOT NULL DEFAULT 0 ;

ALTER TABLE esque_tmp
     ADD COLUMN derive_POS_traits varchar(10) ;
```

TABLEAU 104 – ESQUE : catégories initiales de la table esque

Bases	*o*
NULL	339
?	7
a	130
a ?	10
a/nm	4
acron	1
adv	4
adv ?	1
interj	8
ladj	1
ladv	3
loc	2
n	58
n ?	2
nf	518
nf ?	1
nf/nm	1
nfpl	5
nm	1306
nm ?	11
nm/nf	1
nmpl	9
nnpr	1
npl	2
npr	1880
npr ?	22
npr?	1
onomat	3
sigle	49
sigle?	2
vtr	2

Dérivés	*o*
NULL	134
?	3
a	4143
adv	13
n	20
n ?	2
nf	27
nfpl	1
nm	38
npl	1
v	1
vintr	1

Les mises à jour des attributs rajoutés en fonction des informations déjà présentes s'effectue par les requêtes :

```
UPDATE esque_tmp
     SET derive_POS = 'n',
          derive_POS_douteuse = 1
     WHERE cat_derive = 'n␣?' ;

UPDATE esque_tmp
     SET derive_POS = cat_derive
     WHERE cat_derive IN ('?', 'a', 'adv', 'n', 'v') ;

UPDATE esque_tmp
     SET derive_POS = SUBSTRING(cat_derive, 1, 1),
          derive_POS_traits = SUBSTRING(cat_derive, 2, 10)
WHERE cat_derive REGEXP '^[nv][a-z]+' ;
```

La dernière requête revient à prendre la première lettre de l'attribut cat_derive via :

```
SUBSTRING(cat_derive, 1, 1)
```

et à en faire la valeur de l'attribut derive_POS. On garde le reste de la chaîne cat_derive, à partir du 2ème caractère jusqu'au 10ème, pour en faire la valeur de l'attribut derive_POS_traits. △

De telles créations par requêtes de nouvelles valeurs pour de nouveaux attributs peuvent aisément engendrer des erreurs, soit qu'un motif employé dans une condition de restriction se revèle trop restrictif ou au contraire trop laxiste, soit que la ou les modification(s) spécifiée(s) s'avère(nt) erronée(s). Il est donc nécessaire de vérifier la correspondance entre les catégories de départ et les catégories réorganisées. C'est l'objectif de la requête suivante :

```
SELECT cat_derive,
     derive_POS,
     derive_POS_douteuse,
     derive_POS_traits,
     COUNT(*) AS 'o.'
FROM esque_tmp
GROUP BY cat_derive,
     derive_POS,
     derive_POS_douteuse,
     derive_POS_traits ;
```

Le tableau 105 p. 364 montre qu'en l'occurrence les transformations effectuées sont conformes à ce qui était souhaité.

Mots bases On opère de la même manière pour la catégorie de la base, à ceci près qu'on ajoute un attribut base_locution, valant 1 si la base est une locution (un « mot en plusieurs mots ») et 0 sinon. Il est d'ailleurs possible, voire probable, que la présence de locutions dans les bases soit sous-estimée.

Cela qui conduit à 4 requêtes de modification de la structure de la table esque_tmp :

```
ALTER TABLE esque_tmp
     ADD COLUMN base_POS varchar(10) NOT NULL DEFAULT "absente" ;

ALTER TABLE esque_tmp
     ADD COLUMN base_POS_douteuse INT NOT NULL DEFAULT 0 ;

ALTER TABLE esque_tmp
```

TABLEAU 105 – ESQUE : transformation de l'attribut cat_derive

cat_derive	*derive_POS*	*derive_POS_douteuse*	*derive_POS_traits*	*o.*
NULL	absente	0		134
?	?	0		3
a	a	0		4143
adv	adv	0		13
n	n	0		20
n?	n	1		2
nf	n	0	f	27
nfpl	n	0	fpl	1
nm	n	0	m	38
npl	n	0	pl	1
v	v	0		1
vintr	v	0	intr	1

```
    ADD COLUMN base_POS_traits varchar(10) ;

ALTER TABLE esque_tmp
    ADD COLUMN base_locution INT NOT NULL DEFAULT 0 ;
```

On peut alors mettre à jour les attributs ajoutés en fonction des informations présentes dans l'attribut cat_base :

```
UPDATE esque_tmp
    SET base_POS_douteuse = 1
WHERE cat_base REGEXP '^.+[?]' ;

UPDATE esque_tmp
    SET base_POS = cat_base
WHERE cat_base IN ("?", "a", "adv", "interj", "n", "npr", "onomat", "sigle") ;

UPDATE esque_tmp
    SET base_POS = SUBSTRING(cat_base, 2, 10),
        base_locution = 1
WHERE cat_base IN ("ladv", "ladj") ;
```

Cette dernière requête revient à enlever le l initial de ladv et ladj, puisqu'on recopie la valeur de l'attribut cat_base à partir de son deuxième caractère, après le l donc. Elle indique également que la base est une locution.

L'examen des contextes d'emploi de certains cas problématiques :

base	*cat_base*	*contexte réduit*
Gnole	nnpr	du language [sic] <gnolesque>).
SOCRATE	acron	is cela est un bug <socratesque> et une réalité tarifaire st
bouf(f)re	nf/nm	Allons, tais-toi, <bouffresque>. Nous allons maintenant, me
maboul	a/nm	oupant les tirades <maboulesques> de Viviani [...]
maboul	a/nm	et de paix dans le <maboulesque> dessein de venir racheter u
sauvage	a/nm	La mort du prince impérial, qui m'a frappé comme une image d
simien	a/nm	pilants où la gent <simiesque> démontre sa faculté à utilise
sit(-)com	nm/nf	ans les niaiseries <sitcomesques> du type "Hélène et les gar
tout-autour-du-monde	loc	sio ; les aventures <tout-autour-du-mondesques> d'un irascibl
à la fin	loc	loser-vos-oreilles-<à-la-fin-esque> "le dernier pas" font de

permet d'autres mises à jour :

```
UPDATE esque_tmp
     SET base_POS = "npr"
WHERE cat_base = "nnpr" ;

UPDATE esque_tmp
     SET base_POS = "sigle"
WHERE cat_base = "acron" ;

UPDATE esque_tmp
     SET base_POS = "n",
          base_POS_traits = 'f/m'
WHERE cat_base IN ("nf/nm", "nm/nf") ;

UPDATE esque_tmp
     SET base_POS = "a"
WHERE cat_base = "a/nm" ;

UPDATE esque_tmp
     SET base_POS = "adv",
          base_locution = 1
WHERE cat_base = "loc" ;
```

On peut alors effectuer les dernières mises à jour :

```
UPDATE esque_tmp
     SET base_POS = "a"
WHERE cat_base = "a␣?" ;

UPDATE esque_tmp
     SET base_POS = "a"
WHERE base_POS = "adj" ;

UPDATE esque_tmp
     SET base_POS = "adv"
WHERE cat_base = "adv␣?" ;

UPDATE esque_tmp
     SET base_POS = "n"
WHERE cat_base = "n␣?" ;

UPDATE esque_tmp
     SET base_POS = "npr"
WHERE cat_base = "npr␣?" ;

UPDATE esque_tmp
     SET base_POS = "npr"
WHERE cat_base = "npr?" ;

UPDATE esque_tmp
     SET base_POS = "sigle"
WHERE cat_base = "sigle?" ;
```

On décompose dans les requêtes qui suivent la valeur de `cat_base` entre les traits et la partie du discours que l'on place chacun dans l'attribut idoine.

```
UPDATE esque_tmp
     SET base_POS = "n",
          base_POS_traits = 'f'
```

```
WHERE cat_base = "nf" ;

UPDATE esque_tmp
     SET base_POS = "n",
          base_POS_traits = 'f'
WHERE cat_base = "nf␣?" ;

UPDATE esque_tmp
     SET base_POS = "n",
          base_POS_traits = 'm'
WHERE cat_base = "nm␣?" ;

UPDATE esque_tmp
     SET base_POS = "n",
          base_POS_traits = 'm'
WHERE cat_base = "nm" ;

UPDATE esque_tmp
     SET base_POS = "n",
          base_POS_traits = 'mpl'
WHERE cat_base = "nmpl" ;

UPDATE esque_tmp
     SET base_POS = "n",
          base_POS_traits = 'fpl'
WHERE cat_base = "nfpl" ;

UPDATE esque_tmp
     SET base_POS = "n",
          base_POS_traits = 'pl'
WHERE cat_base = "npl" ;

UPDATE esque_tmp
     SET base_POS = "v",
          base_POS_traits = 'tr'
WHERE cat_base = "vtr" ;
```

Comme pour les dérivés, on effectue une vérification de la transformation des catégories des mots bases (tableau 106 p. 367).

Sachant, grâce à la requête :

```
SELECT COUNT(DISTINCT base, base_POS)
FROM esque_tmp ;
```

le nombre total de combinaisons base/POS : 3 169, on peut obtenir la répartition de ces combinaisons selon les parties du discours (tableau 107 p. 367), via la requête :

```
SELECT base_POS AS 'Bases',
     COUNT(DISTINCT base) AS 'o.',
     COUNT(DISTINCT base) / 3169 * 100 AS '%'
FROM esque_tmp GROUP BY base_POS ;
```

On peut obtenir de la même manière :

```
SELECT derive_POS AS 'Dérivés',
     COUNT(DISTINCT derive) AS 'o.',
     COUNT(DISTINCT derive) / 3479 * 100 AS '%'
FROM esque_tmp
```

TABLEAU 106 – ESQUE : transformation de l'attribut cat_base

cat_base	*base_POS*	*base_POS_traits*	*base_POS_douteuse*	*base_locution*	*o.*
NULL	absente		0	0	339
?	?		0	0	7
a	a		0	0	130
a?	a		1	0	10
a/nm	a		0	0	4
acron	sigle		0	0	1
adv	adv		0	0	4
adv?	adv		1	0	1
interj	interj		0	0	8
ladj	adj		0	1	1
ladv	adv		0	1	3
loc	adv		0	1	2
n	n		0	0	58
n?	n		1	0	2
nf	n	f	0	0	518
nf?	n	f	1	0	1
nf/nm	n	f/m	0	0	1
nfpl	n	fpl	0	0	5
nm	n	m	0	0	1306
nm?	n	m	1	0	11
nm/nf	n	f/m	0	0	1
nmpl	n	mpl	0	0	9
nnpr	npr		0	0	1
npl	n	pl	0	0	2
npr	npr		0	0	1880
npr?	npr		1	0	22
npr?	npr		1	0	1
onomat	onomat		0	0	3
sigle	sigle		0	0	49
sigle?	sigle		1	0	2
vtr	v	tr	0	0	2

TABLEAU 107 – ESQUE : répartition des bases par partie du discours

Bases	*o.*	*%*
?	7	0.22
a	120	3.79
absente	228	7.19
adv	8	0.25
interj	7	0.22
n	1384	43.67
npr	1370	43.23
onomat	3	0.09
sigle	40	1.26
v	2	0.06

TABLEAU 108 – ESQUE : répartition des dérivés par partie du discours

Dérivés	*o.*	*%*
?	3	0.09
a	3249	93.39
absente	134	3.85
adv	13	0.37
n	78	2.24
v	2	0.06

```
GROUP BY derive_POS ;
```

la répartition des 3 479 combinaisons distinctes de dérivé/POS (tableau 108 p. 368).

On remarque que les décomptes ne sont pas les mêmes que ceux du tableau 104 p. 362. On ne s'intéresse ici qu'aux types de bases ou de dérivés (d'où l'appel à **DISTINCT**) et non à leurs occurrences. On constate la part écrasante du nom dans les mots bases (à égalité entre noms propres et noms communs) et d'autre part, à côté de la domination sans partage de l'adjectif au sein des dérivés, une petite proportion (2,24%) de dérivés nominaux (du type *une arabesque*, *la soldatesque*).

1.1.3. Signalement des changements de catégorie

On note dans l'attribut verif_derive lorsque la catégorie du dérivé a été effectivement changée, par la requête :

```
UPDATE esque_tmp
    SET verif_derive = CONCAT(verif_derive, "␣changement␣catégorie")
    WHERE cat_derive IS NULL
        OR cat_derive <> derive_POS ;
```

On peut ensuite examiner en tant que telles toutes les entrées dont les catégories de dérivées ont été effectivement changées, par une requête du type :

```
SELECT derive,
    cat_derive,
    derive_POS
FROM esque_tmp
WHERE verif_derive REGEXP 'changement␣catégorie' ;
```

qui produit un résultat de la forme :

derive	*cat_derive*	*derive_POS*
grotesque	nf	n
kaplanesque	**NULL**	absente
ducciesque	**NULL**	absente
fanfaronnesque	n?	n
moresque	nf	n
papillonnesque	**NULL**	absente
computeresque	**NULL**	absente
galarettesque	**NULL**	absente
Carmividesque	**NULL**	absente
Dr Quinnesque	**NULL**	absente

La démarche est similaire pour la catégorie des mots bases :

```
UPDATE esque_tmp
    SET verif_base = CONCAT(verif_base, ' changement catégorie' )
    WHERE cat_base IS NULL
        OR cat_base <> base_pos ;
```

1.2. Discordances de catégorisation pour une base ou un dérivé spécifique

1.2.1. Constats

En dehors de l'ensemble d'étiquettes, de catégories employées, il peut se faire que des incohérences se soient introduites dans l'attribution d'une catégorie à une base donnée ou à un dérivé donné. Là encore, on souhaite utiliser les requêtes du SGBD pour déceler de telles incohérences.

Pour ce qui concerne les catégories des mots bases, une première requête, basée sur une autojointure, isole les bases identiques pour lesquelles deux catégories distinctes existent et décompte les occurrences de chaque cas :

```
SELECT e1.base AS 'Base',
    e1.cat_base AS 'cat1',
    e2.cat_base AS 'cat2',
    COUNT(*) AS 'o.'
FROM esque_tmp AS e1, esque_tmp AS e2
WHERE e1.base = e2.base
    AND e1.cat_base <> e2.cat_base
GROUP BY e1.base, e1.cat_base, e2.cat_base ;
```

Le résultat, au tableau 109 p. 371, est rendu peu lisible du fait qu'une même paire de catégories distinctes donne naissance à deux couples. Ainsi, pour la discordance nf–sigle de *BD*, on trouve le couple nf sigle et le couple sigle nf. La solution est d'ordonner lexicographiquement les deux catégories (en cherchant, grâce à < lequel est le premier dans l'ordre lexicographique) et de souder, de concaténer (**CONCAT**) le couple résultant :

```
SELECT e1.base AS 'Base',
    IF(IFNULL(e1.cat_base, "--") < IFNULL(e2.cat_base, "--"), CONCAT(IFNULL(e1.
        cat_base, "--"), "-", IFNULL(e2.cat_base, "--")), CONCAT(IFNULL(e2.cat_base
        , "--"), "-", IFNULL(e1.cat_base, "--"))) AS 'catégories',
    COUNT(*) AS 'o.'
FROM esque_tmp AS e1, esque_tmp AS e2
WHERE e1.base = e2.base
    AND IFNULL(e1.cat_base, "--") <> IFNULL(e2.cat_base, "--")
GROUP BY e1.base,
    IF(IFNULL(e1.cat_base, "--") < IFNULL(e2.cat_base, "--"), CONCAT(IFNULL(e1.
        cat_base, "--"), "-", IFNULL(e2.cat_base, "--")), CONCAT(IFNULL(e2.cat_base
        , "--"), "", IFNULL(e1.cat_base, "--"))) ;
```

Par ailleurs, comme la marque **NULL** n'est égale à aucune valeur et n'est donc pas comparable telle quelle, on transforme cette marque par l'appel à la fonction **IFNULL**. Si l'attribut cat_base contient la marque **NULL**, celle-ci est remplacée par la chaîne "–". Dans le cas contraire, la chaîne de caractères qui symbolise la catégorie de la base est retournée.

Le tableau 110 p. 372 est effectivement plus compact et plus parlant. On distingue plusieurs cas de figure :

1. deux catégories en fait concordantes, l'une spécifiant l'autre. C'est le cas de *vaudou* avec nm et n. L'alignement sur l'une des deux catégories paraît légitime. On peut ramener éventuellement à ce cas la présence ou non d'une hésitation, comme pour *Amiga* ou *Ra Dada*, npr ou npr ? Rentre peut-être aussi dans cet ensemble le cas où s'oppose l'absence de catégorie, symbolisée par la marque **NULL**, et une catégorie. Reste cependant à choisir dans ces différents cas de figure la catégorie la plus générale ou la plus spécifique.
2. deux spécifications distinctes d'une même catégorie, correspondant probablement à deux mots distincts. C'est le cas d'*un faune*, nm, qui s'oppose effectivement à *la faune*, nf. Les deux mots doivent demeurer tels quels, et, partant, les deux catégories.
3. deux catégories distinctes potentiellement légitimes, comme *bidon* a dans *une excuse bidon* et nm dans *un bidon d'essence*, ou dont on peut se demander s'il ne s'agit pas d'une simple incohérence au moment de l'entrée des informations, comme pour *BD*, nf et sigle. Dans le premier cas, on peut recourir à une catégorie composite, qui indique l'ambivalence, par exemple a/nm pour *bidon*.

Une requête équivalente (où base est remplacé par derive et cat_base par cat_derive) isole les discordances catégorielles pour les dérivés (tableau 111, p. 373).

1.2.2. Signalement

Pour la discordance de catégorisation des bases, une première requête, reposant sur un enchâssement :

```
UPDATE esque_tmp
    SET verif_base =
          CONCAT(verif_base, '␣discordance␣catégorie')
    WHERE base IN
         (SELECT e1.base
          FROM esque_tmp AS e1,
                  esque_tmp AS e2
          WHERE e1.base = e2.base
                  AND IFNULL(e1.cat_base, "--")
                  <> IFNULL(e2.cat_base, "--")) ;
```

débouche sur une erreur :

```
ERROR 1093 (HY000) : You can't specify target table 'esque_tmp' for update
in FROM clause
```

On procède alors en deux temps : 1) création à la volée d'une table engrangeant les bases à catégories multiples ; 2) utilisation de cette table pour mettre à jour en conséquence la table esque_tmp :

```
CREATE TABLE base_a_categories_discordantes
     (SELECT e1.base
      FROM esque_tmp AS e1,
           esque_tmp AS e2
      WHERE e1.base = e2.base
           AND IFNULL(e1.cat_base, "--")
                <> IFNULL(e2.cat_base, "--")) ;

UPDATE esque_tmp
    SET verif_base =
          CONCAT(verif_base, '␣discordance␣catégorie')
    WHERE base IN
         (SELECT *
          FROM base_a_categories_discordantes) ;
```

TABLEAU 109 – ESQUE : bases ayant plusieurs catégories (version 1)

Base	*cat1*	*cat2*	*o.*
Amiga	npr	npr ?	1
Amiga	npr ?	npr	1
BD	nf	sigle	2
BD	sigle	nf	2
E.T.	npr ?	sigle	1
E.T.	sigle	npr ?	1
Gnome	nm	npr	1
Gnome	npr	nm	1
Molière	nm	npr	5
Molière	npr	nm	5
Ra Dada	npr	npr ?	1
Ra Dada	npr ?	npr	1
bidon	a	nm	2
bidon	nm	a	2
canon	a	nm	1
canon	nm	a	1
chocolat	nm	npr	1
chocolat	npr	nm	1
espagnol	a	nm ?	1
espagnol	nm ?	a	1
faune	nf	nm	4
faune	nm	nf	4
gaucho	nf	nm	1
gaucho	nm	nf	1
hurluberlu	a	nm	1
hurluberlu	nm	a	1
louffe	n	nm	1
louffe	nm	n	1
ovale	a	nm	1
ovale	nm	a	1
philosophe	n	nm	1
philosophe	nm	n	1
roller	n	nm	1
roller	nm	n	1
somnambule	n	nm	4
somnambule	nm	n	4
vampire	n	nm	2
vampire	nm	n	2
vaudou	n	nm	2
vaudou	nm	n	2

TABLEAU 110 – ESQUE : bases ayant plusieurs catégories (version 2)

Base	*catégories*	*o.*
?	– npr	212
Amiga	npr npr ?	2
BD	nf sigle	4
Bibi	– npr	2
Bouygues (Francis)	– npr	2
E.T.	npr ? sigle	2
Gnome	nm npr	2
Mercredi	– nm	2
Molière	nm npr	10
Ra Dada	npr npr ?	2
baba cool	– nm	2
bidon	a nm	4
calembour + bourde	– nm	4
canon	a nm	2
chocolat	nm npr	2
con	– nm	12
espagnol	a nm ?	2
faramineux ?	– a	2
farfelu	– a	2
faune	nf nm	8
foutre	– vtr	2
fratesco [it.]	– a	4
furbesco [it.]	– a	6
gaucho	nf nm	2
glamour	– n ?	2
gourou	– nm	2
grotesque	– a	16
grottesco [it.]	– a	2
hurluberlu	a nm	2
latin	– nm	2
louffe	n nm	2
morisco [esp.]	– a	20
noble	– a	2
ovale	a nm	2
philosophe	n nm	2
pittoresco [it.]	– a	4
plateresco [esp.]	– a	4
premier avril ?	– nm	2
prout	– a	2
quincailler ?	– nm	2
roller	n nm	2
sauvage	– a/nm	2
simien	– a/nm	8
soldatesque	– a	2
somnambule	n nm	8
todesco, tedesco [it.]	– nm	4
turc	– nm	12
vampire	n nm	4
vaudou	n nm	4

TABLEAU 111 – ESQUE : dérivés ayant plusieurs catégories

Dérivé	*catégories*	*o.*
BDesque	– a	2
Fela-esque	– a	2
abracadabrantesque	a nm	10
arabesque	a nf	8
arlequinesque	– a	4
arrivéesque	– a	2
asinesque	– a	2
barbaresque	– a	6
baronesque	– a	2
barthesque	– a	4
bassanesque	a n	2
bernesque	– a	2
boudinesque	a nf	4
bramantesque	– a	2
burlesque	a nm	2
calamitesque	a nm	4
caravagesque	a n	6
carnavalesque	a nm	8
chevaleresque	a nm	12
churrigueresque	a nm	4
cocoriquesque	a nm	6
confituresque	– a	2
corsesque	a nf	4
crotesque	a n	6
cyclonesque	– a	4
danaesque	– a	2
delaruesque	– a	2
ellroyesque	– a	2
faunesque	a nm	8
flandresque	a nf	2
foresque	a nf	2
foresque	a nm	2
foresque	nf nm	2
fourbesque	a nm	6
funambulesque	a nm	8
gallianesque	– a	2
gallimardesque	a n	4
gigantesque	a nm	6
giorgionesque	a n	2
giottesque	a n	4
giottesque	a nm	4
giottesque	n nm	2
grenouillesque	– a	4

Dérivé	*catégories*	*o.*
grotesque	a nm	4
grotesque	nf nm	2
grottesque	a nf	2
halloweenesque	– a	2
humoresque	a n	2
humoresque	a nf	2
humoresque	n nf	2
joliesque	a nm	2
langueste	– a	2
machinesque	a nm	2
mauresque	a nf	8
miraillesque	– a	2
moresque	a n	2
moresque	a nf	2
moresque	n nf	2
noblesque	a n ?	2
pagnolesque	– a	6
papillonnesque	– a	2
papinesque	– a	4
perlimpimpesque	– a	2
piratesque	– a	2
pittoresque	a nm	6
plantardesque	– nm	2
porphyresque	a nm	2
proutesque	– a	2
putanesque	a n	4
rembranesque	a n	4
romanesque	a nf	12
romanesque	a nm	12
romanesque	nf nm	2
serpentesque	– a	2
sitcomesque	– a	2
soldatesque	a nf	12
spawnesque	– a	2
titanesque	a nm	10
titianesque	a n	2
tombesque	– a	2
truandesque	a nm	6
zappaesque	– a	2
zidanesque	– a	2
écranesque	a nm	4

On procède de la même manière pour les discordances de catégorisation portant sur les dérivés :

```
CREATE TABLE derive_a_categories_discordantes
     (SELECT DISTINCT e1.derive
      FROM esque_tmp AS e1,
           esque_tmp AS e2
      WHERE e1.derive = e2.derive
           AND IFNULL(e1.cat_derive, "--")
                <> IFNULL(e2.cat_derive, "--")) ;

UPDATE esque_tmp
     SET verif_derive =
          CONCAT(verif_derive, ' discordance catégorie')
     WHERE derive IN
          (SELECT * FROM derive_a_categories_discordantes) ;
```

1.3. Bases absentes ou douteuses

1.3.1. Constat

Dans 74 cas (tableau 112 p. 375), la base du dérivé et sa catégorie sont manquantes, ce qui est signalé par la marque **NULL** ou bien par la seule mention du point d'interrogation. Dans d'autres cas, au nombre de 190 (tableau 113 p. 376 pour un extrait), la base proposée est assortie d'un doute, symbolisé par un point d'interrogation.

On peut chercher s'il n'existe pas, pour un dérivé dont la base est manquante, une autre occurrence de ce dérivé avec une base non manquante. C'est la requête :

```
SELECT base,
     cat_base,
     derive
FROM esque
WHERE base IS NOT NULL
     AND base NOT REGEXP '^ *\\? *$'
     AND derive IN
          (SELECT derive
           FROM esque
           WHERE base IS NULL
                OR base REGEXP '^ *\\? *$') ;
```

Elle repose sur une requête enchâssée dans les conditions de restriction. Ces dernières réclament une base qui ne contient ni la marque **NULL** ni uniquement un point d'interrogation précédé et suivi optionnellement d'espaces et pour laquelle le dérivé figure dans l'ensemble des dérivés pour lesquels la base contient soit la marque **NULL** soit uniquement un point d'interrogation précédé et suivi optionnellement d'espaces, c'est la requête enchâssée. L'examen du résultat (tableau 114 p. 376) est décevant : on ne trouve en effet que 9 occurrences de dérivés répondant à ces conditions, qui couvrent 8 dérivés distincts (*simonesque* est fourni avec deux bases possibles : *Simone (Marco)* et *Simon (Claude)*). On ne peut donc pas se servir de ce résultat pour suppléer aux manques constatés.

La notation actuelle des bases est problématique :

– une base qui manque est toujours représentée par un couple de marques **NULL** ou par un point d'interrogation et une marque **NULL** pour les attributs `base` et `cat_base`, si bien que les multiples dérivés de ces bases manquantes semblent relever de seulement deux « bases ».

TABLEAU 112 – ESQUE : bases manquantes (extrait)

base	*cat_base*	*derive*
NULL	**NULL**	humoresque
?	**NULL**	CANesque
?	**NULL**	chamallowesque
NULL	**NULL**	tégévéesque
?	**NULL**	auvesque
?	**NULL**	bavesque
?	**NULL**	boulardesque
?	**NULL**	bourrelesque
?	**NULL**	charbonnièresque
?	**NULL**	ellesque
?	**NULL**	jamesque
?	**NULL**	joséphiéresque
?	**NULL**	rubesque
?	**NULL**	simonesque
?	**NULL**	tunequesque
NULL	**NULL**	bonhamesque
NULL	**NULL**	conandoylesque
NULL	**NULL**	cocodinguesque
NULL	**NULL**	bottonesque
?	**NULL**	monteputinesque
?	**NULL**	Carmividesque
?	**NULL**	brétesque
?	**NULL**	robesque
NULL	**NULL**	tocambolesque
NULL	**NULL**	latunesque
?	**NULL**	humiesque
NULL	**NULL**	avionnesque
?	**NULL**	baribaudesque
?	npr	Benny-Hillesque
?	**NULL**	bonesque
?	**NULL**	caldoresque
NULL	**NULL**	colombanesque
?	**NULL**	colombesque
?	**NULL**	cornucopiesque
?	**NULL**	delmièsque
?	**NULL**	Dr Quinnesque
NULL	**NULL**	escargotesque
?	**NULL**	explosiantesque
?	**NULL**	gothico-seborgiesque
?	**NULL**	mentonnesque
?	**NULL**	moriartyesque
NULL	**NULL**	papagalliesque
?	**NULL**	pistolesque
?	**NULL**	plonesque
NULL	**NULL**	post-lichesque
NULL	**NULL**	spawnesque
?	**NULL**	stryeresque
?	**NULL**	varitétochiesque (ou variétochiesque)
NULL	**NULL**	wolfesque
?	**NULL**	morandinesque
?	**NULL**	champignolesque
NULL	**NULL**	ardissonnesque
?	**NULL**	bonaldiesque
?	**NULL**	catenacciesque
?	npr	flo-joesque

TABLEAU 113 – ESQUE : bases douteuses (extrait)

base	*cat_base*	*derive*
Bourget (Paul) ?	npr	bourgetesque
bricol(é)e (?)	**NULL**	bricolesque
Charpini (Jean) ?	npr	Charpinesque
Charpini (Jean) ?	npr	charpiniesque
compile ?	nf	compilesque
coutumier ?	a	coutumiesque
croquignole(t) ?	**NULL**	croquignolesque
Del Duca (Gino) (?)	npr	delducatesque
déprédateur ?	**NULL**	déprédatesque
faramineux ?	a	faraminesque
flicard (+ picaresque ?)	**NULL**	flicaresque
°grandiloqueur ?	**NULL**	grandiloqueste
libidineux ?	a	libidinesque
loukoum ou loukoumien ?	**NULL**	loukoumiesque
obu (+ ubuesque ?)	nm	obuesque
olympien ?	a	olympiesque
pataugeur ?	a	pataugesque
peinturlure ?	nf	peinturluresque
°polardin (ou polar ?)	**NULL**	polardinesque
quincailler ?	**NULL**	quincaillesque
Scapula (François) (+ scapulaire ?)	npr	scapularesque
sénile ?	**NULL**	sénilesque
tintinnabulant (?)	**NULL**	tintinnabulesque
Valentin (Saint-) (?)	npr	valentinesque
vidéo (?)	**NULL**	vidéotesque
Albanie ?	npr	dalbanesque
spongieux ?	a	spongesquer
zapper ?	**NULL**	zappesque
mercantesco [it.] ?	**NULL**	mercantesque
ventre ?	**NULL**	ventresque
enchanteur ?	a ?	enchanteresque
show bizness ?	nm	showbizenesque
avril ?	nm	aprilesque
Corleon ?	npr	corléonesque
Clinton (?)	npr	clintonesque
faramineux ?	**NULL**	faraminesque
quincailler ?	nm	quincaillesque
bouquin ?	nm	boucquinesque
Ferry (Jules) ?	npr	ferrychonnesque
Péladan +asinus ?	**NULL**	péladasinesque

TABLEAU 114 – ESQUE : dérivés dont la base n'est pas toujours manquante

base	*cat_base*	*derive*
Humoreske [all.]	npr	humoresque
Morandini (Jean-Marc)	npr	morandinesque
Manson (Charles) ?	npr	mansonnesque
Simone (Marco)	npr	simonesque
Spawn	npr	spawnesque
Simon (Claude)	npr	simonesque
Sex Pistols ?	**NULL**	pistolesque
Fela ?	**NULL**	Fela-esque
Jam	sigle	jamesque

– l'attribut base, pour les bases douteuses, n'est pas « atomique ». Il conjoint plusieurs informations distinctes : la base proposée, l'indication du doute (un ou plusieurs points d'interrogation), et éventuellement, préposé, le dito (°), la marque utilisée en morphologie pour indiquer une base reconstruite et non attestée en tant que telle (c'est le cas pour °*grandiloqueur ?* reconstruit, et donné comme douteux, comme base de *grandiloqueste*).

Un autre problème, mineur, mais réel, est l'indication de deux bases possibles dans l'attribut base, ce qui est marqué par la barre oblique ou par le mot *ou*. La requête :

```
SELECT DISTINCT base
FROM esque_tmp
WHERE base REGEXP '/|␣[(]?ou␣' ;
```

met en évidence le petit nombre des bases concernées.

base
loukoum ou loukoumien ?
°polardin (ou polar ?)
Babar/Barre (Raymond)
ange/angélique
temple (ou templier ?)
Champolion ou Champollion ?
Sion (pierre de) ou Prieuré de Sion (?)
Dive ou diva ?
face (ou farce)
AC/DC ?
bouche ou boucher ?
Titanic/titan
cucul/culte ?
moto / moteur ?

On notera que la barre oblique a visiblement plusieurs emplois : elle peut faire partie de la base (*AC/DC*), ajouter une précision sémantique en permettant de distinguer des homonymes (*Babar/Barre (Raymond)* à ne pas confondre avec le personnage pour enfants) ou effectivement indiquer une ambivalence quant à la base effective.

1.3.2. Réorganisation

Pour aboutir à une notation plus claire, on se propose d'abord de rajouter un attribut statut_base, qui aura pour valeur par défaut sûre, et pour autres valeurs possibles douteuse et absente. Pour pouvoir contrôler les transformations de l'attribut base, on introduit un attribut base_normalisee destinée à mémoriser la version normalisée de la base. La normalisation porte sur deux points :

1. une base manquante, quelle que soit sa représentation de départ, est symbolisée par le dérivé précédé du chevron fermant, utilisé souvent en morphologie pour indiquer une relation de dérivation. Ainsi ?, base manquante de *mobydiesque*, est remplacé par '> *modybiesque*' (on peut gloser cette notation de la manière suivante : 'la base de *mobydiesque* est la forme, non connue ou non notée, qui a donné naissance à *mobydiesque*') et l'attribut statut_base prend pour valeur absente.
2. le ou les point(s) d'interrogation manifestant le doute dans l'attribut base est/sont effacé(s) de cet attribut, par contre, l'attribut statut_base prend pour valeur douteuse.

Voici les requêtes nécessaires pour transformer la structure de la table esque_tmp et pour mettre à jour les nouveaux attributs :

```
ALTER TABLE esque_tmp
     ADD COLUMN base_normalisee VARCHAR(100) NOT NULL DEFAULT "base␣manquante" ;

ALTER TABLE esque_tmp
     ADD COLUMN statut_base VARCHAR(15) NOT NULL DEFAULT "sûre" ;

UPDATE esque_tmp
     SET base_normalisee = base ;

UPDATE esque_tmp
     SET statut_base = 'absente',
          base_normalisee = CONCAT("><␣", derive)
WHERE base IS NULL
     OR base REGEXP '^␣*\\?␣*$' ;

UPDATE esque_tmp
     SET base_normalisee = RTRIM(REPLACE(base, "?", '')),
          statut_base = 'douteuse'
WHERE base REGEXP '\\?'
     AND base NOT REGEXP '^␣*\\?␣*$' ;
```

Au total, 202 lignes de la table esque_tmp ont une valeur pour l'attribut base_normalisee qui diffère de celle de l'attribut base. On peut vérifier que la normalisation s'est effectué correctement en examinant précisément les lignes correspondantes (tableau 115 p. 379 pour un extrait), via la requête :

```
SELECT base,
      base_normalisee,
      statut_base,
      derive
FROM esque_tmp
WHERE base <> base_normalisee
     OR base IS NULL ;
```

On notera que la condition de restriction :
base **IS NULL**
est strictement nécessaire. En effet la marque **NULL** n'est pas une valeur et n'est donc pas comparable à une valeur effective. Sa présence doit être testée en tant que telle, grâce aux opérateurs appropriés.

La modification qui vient d'être faite aboutit à donner à l'attribut statut_base la valeur douteuse quand le point d'interrogation figure dans la base. Mais ce point d'interrogation peut figurer tout seul après une base, comme dans *cocotte ?* ou bien entre parenthèses après la base, comme dans *Twain (Shania) (?)*. Dans ce dernier cas, la base normalisée comprend des parenthèses vides, à effacer : *Twain (Shania) ()*, comme le montre le tableau 115 p. 379. C'est l'objet de la requête :

```
UPDATE esque_tmp
     SET base_normalisee =
          REPLACE(base_normalisee, "()", "")
WHERE base_normalisee LIKE '%()%' ;
```

1.3.3. Signalement

TABLEAU 115 – ESQUE : bases normalisées (extrait)

base	*base_normalisee*	*statut_base*	*derive*
mou ?	mou	douteuse	mollesque
Péladan +asinus ?	Péladan +asinus	douteuse	péladasinesque
?	> auvesque	absente	auvesque
zapper ?	zapper	douteuse	zappesque
?	> brétesque	absente	brétesque
NULL	> tégévéesque	absente	tégévéesque
?	> stryeresque	absente	stryeresque
?	> bavesque	absente	bavesque
mon-voisin-totor ?	mon-voisin-totor	douteuse	mon-voisin-totoresque
plagiat ?	plagiat	douteuse	plagiesque
loukoum ou loukoumien ?	loukoum ou loukoumien	douteuse	loukoumiesque
?	> colombesque	absente	colombesque
léonin ?	léonin	douteuse	leonesque
philia ? [grec]	philia [grec]	douteuse	philianesque
Charpini (Jean) ?	Charpini (Jean)	douteuse	charpiniesque
estudiantin ?	estudiantin	douteuse	estudiantesque
fliper ?	fliper	douteuse	flipesque
?	> boulardesque	absente	boulardesque
temple (ou templier ?)	temple (ou templier)	douteuse	templiesque
Bloody Valentino ?	Bloody Valentino	douteuse	Bloody Valentinesque
?	> akira-esque	absente	akira-esque
croquignole(t) ?	croquignole(t)	douteuse	croquignolesque
enchanteur ?	enchanteur	douteuse	enchanteresque
Electronica ?	Electronica	douteuse	Electronica-esque
marine ?	marine	douteuse	marinesque
?	> humiesque	absente	humiesque
Gladiator ?	Gladiator	douteuse	gladiatoresque
show bizness ?	show bizness	douteuse	showbizenesque
scimmiesco [it. ?]	scimmiesco [it.]	douteuse	simiesque
quellegrosseconne ?	quellegrosseconne	douteuse	quellegrosseconnesque
tintinnabulant (?)	tintinnabulant ()	douteuse	tintinnabulesque
Morgan ?	Morgan	douteuse	morganesque
(ra)caille ?	(ra)caille	douteuse	caillesque
?	> lichesque	absente	lichesque
ventre ?	ventre	douteuse	ventresque
?	> murgesque	absente	murgesque
Manson (Charles) ?	Manson (Charles)	douteuse	mansonnesque
Crimson ?	Crimson	douteuse	Crimson-esque
Barthez (Fabien) ?	Barthez (Fabien)	douteuse	barthesque
grimpe ?	grimpe	douteuse	grimpesque

```
UPDATE esque_tmp
     SET verif_base =
          CONCAT(verif_base, "␣normalisation␣base")
     WHERE base <> base_normalisee
          OR base IS NULL ;
```

On marque spécifiquement les bases « doubles », pour lesquelles la personne qui a rédigé la ligne hésite entre deux bases possibles. On indique par ailleurs qu'il s'agit de bases douteuses.

```
UPDATE esque_tmp
     SET verif_base =
          CONCAT(verif_base, "␣2␣bases")
WHERE base  REGEXP '/|␣ou␣' ;
```

1.4. Plusieurs mots comme base

1.4.1. Constat

Il ne s'agit pas ici des cas où la base postulée est un « mot en plusieurs mots », ce que permet d'isoler l'attribut base_locution de la table esque_tmp :

```
SELECT DISTINCT base,
     derive
FROM esque_tmp
WHERE base_locution = 1 ;
```

Cette requête met en évidence un petit nombre de bases et de dérivés :

base	*derive*
a giorno	adgiornesque
à la con	àlaconesque
catimini (en-)	catiminesque
a giorno	à giornesque
à la fin	à-la-fin-esque
tout-autour-du-monde	tout-autour-du-mondesque

La situation plus fréquente est celle où la base conjoint éventuellement deux mots, où elle constitue peut-être un mot-valise, ce qui est signalé par un + (et un point d'interrogation en cas de doute) :

```
SELECT DISTINCT base,
     derive
FROM esque_tmp
WHERE base REGEXP '[+]'
```

On obtient (tableau 116 p. 381) 38 couples base-dérivé.

1.4.2. Signalement

On marque les bases qui constituent peut-être des mots-valises :

```
CREATE TABLE base_mot_valise
     (SELECT DISTINCT base
      FROM esque_tmp
      WHERE base REGEXP '[+]') ;

UPDATE esque_tmp
     SET verif_base =
          CONCAT(verif_base, "␣mot–valise")
WHERE base IN (SELECT * FROM base_mot_valise) ;
```

TABLEAU 116 – ESQUE : base = mot-valise

base	*derive*
calembour + bourde	calembourdesque
carambole (+ Rocambole)	carambolesque
clitoris + pittoresque	clitoresque
Clochemerle + merde	clochemerdesque
flicard (+ picaresque ?)	flicaresque
gigantesque + tomate	gigantomatesque
hurluberlu + ubuesque	hurlubuesque
obu (+ ubuesque ?)	obuesque
rhum + romanesque	rhumanesque
Scapula (François) (+ scapulaire ?)	scapularesque
sofa + affalé	sofalesque
sous + table	sous-tablesque
télé + éléphantesque	téléphantesque
abracadabrant + cassette	abracassetabrantesque
chier + picaresque	chicarresque
raffiné + fignolé	rafignolesque
sensualo + lolita	sensualolitesque
Péladan +asinus ?	péladasinesque
château + Chateaubriand (François René de) ?	chateaubriesque
calembour + bourde	calembourdinesque
zoo + man ?	zoomanesque
cucul + Ku Klux Klan	cucul-klanesque
cul + culture	CULturesque
Flammarion + Gallimard + Seuil	flammario-galligrasseuilesque
piste + pittoresque	pistoresque
Ravachol + gloupinesque ?	ravacholinesque
Amiga + Atari	amigataresque
bordel + drome	bordelodromesque
calembour + barnum	calembarnumesque
humano + simiesque	humanosimiesque
farfelu + fada ?	farfadesque
altissime + simiesque	altissimiesque
javanais + java ?	javanesque
cavalier + chevaleresque ?	cavaleresque
Do Quichotte + shoot ?	Dom Qui Shootesque
formule + lune ?	formulunesque
Hue (Robert) + Hubu	Hu(bu)esque ?
abracadabra + dantesque	abracadantesque

1.5. Indication d'une origine autre que le français moderne

1.5.1. Constat

Quand la base n'est pas un mot du français moderne, une mention entre crochets dans la base vient le préciser :

1. [angl.] ou [angl] anglais ;
2. [it.] italien ;
3. [a.f.] ancien français ;
4. [prov.] provençal ;
5. [lat.] latin ;
6. [all.] allemand ;
7. [esp.] espagnol.

Ces mentions touchent 63 bases (tableau 117 p. 117), comme le montre la requête :

```
SELECT DISTINCT base_normalisee
FROM esque_tmp
WHERE base_normalisee REGEXP '\\[' ;
```

Dans un cas, grisé dans le tableau, ce qui est entre crochets ne sert pas à noter une langue, mais une variante.

1.5.2. Réorganisation et signalement

On commence par ajouter à la table esque_tmp un attribut textuel langue_base qui prend comme valeur par défaut fr(ançais) :

```
ALTER TABLE esque_tmp
    ADD COLUMN langue_base
        VARCHAR(10) NOT NULL DEFAULT "fr"
        AFTER base_locution ;
```

On opère une modification complexe, en utilisant l'aiguillage **CASE**. On ajoute à l'attribut verif_base l'indication que la base n'est pas du français moderne. On efface la chaîne représentant la langue de l'attribut base_normalisee. On change en conséquence la valeur de langue_base.

```
UPDATE esque_tmp
    SET verif_base =
        CONCAT(verif_base, "␣base␣non␣français␣moderne"),
            base_normalisee =
              CASE
               WHEN base_normalisee REGEXP '\\[angl\\.\\]' THEN  REPLACE(
                   base_normalisee, "[angl.]", "")
                WHEN base_normalisee REGEXP '\\[it\\.\\]' THEN  REPLACE(
                    base_normalisee, "[it.]", "")
                WHEN base_normalisee REGEXP '\\[a\\.f\\.\\]' THEN  REPLACE(
                    base_normalisee, "[a.f.]", "")
                WHEN base_normalisee REGEXP '\\[prov\\.\\]' THEN  REPLACE(
                    base_normalisee, "[prov.]", "")
                WHEN base_normalisee REGEXP '\\[lat\\.\\]' THEN  REPLACE(
                    base_normalisee, "[lat.]", "")
```

TABLEAU 117 – ESQUE : origine de base autre que le français moderne

base_normalisee
bummamus [lat.]
fetuccine [it.]
lourd [a.f.]
Michelangelo [it.]
muliebris [lat.]
ottocento [it.]
picaro [esp.]
tailor [taylor]
todesco, tedesco [it.]
plateresco [esp.]
pittoresco [it.]
pavone [it.]
morisco [esp.]
lama [esp.]
Humoreske [all.]
grottesco [it.]
gigantesco [it.]
furbesco [it.]
churrigueresco [esp.]
burlesco [it.]
asinesco [it.]
boschereccio [it.]
bertesca [it.]
cagnesca [it.]
cancelleresco [it.]
corsesca [it.]
cortigianesco [it.]
escoubas [prov.]
fantesca [it.]
fiabesco [it.]
forum, foris [lat.]
forfantesco [it.]

fratesco [it.]
graeciscus [lat.]
grottesca [it.]
levre [a.f.]
mercantesco [it.]
pintoresco [esp.]
principesco [it.]
cinquecento [it.]
quattrocento [it.]
trecento [it.]
scimmiesco [it.]
settecento [it.]
Japan [angl.]
philia [grec]
thug [angl.]
villanelle [it.]
musical [angl.]
friend [angl]
lupus [lat.]
screenshot [angl]
two tone [angl]
Amnesiac [angl.]
background [angl.]
balloon [angl.]
banda [esp.]
calor [esp.]
comic [angl.]
dance floor [angl.]
game [angl.]
new order [angl]
commediante [it.]

```
        WHEN base_normalisee REGEXP '\\[all\\.\\]' THEN  REPLACE(
            base_normalisee, "[all.]", "")
        WHEN base_normalisee REGEXP '\\[esp.\\]' THEN  REPLACE(
            base_normalisee, "[esp.]", "")
      END,
    langue_base =
      CASE
        WHEN base REGEXP '\\[angl\\.\\]' THEN  "angl"
        WHEN base  '\\[it\\.\\]' THEN "it"
        WHEN base REGEXP '\\[a\\.f\\.\\]' THEN  "af"
        WHEN base REGEXP '\\[prov\\.\\]' THEN "prov"
        WHEN base REGEXP '\\[lat\\.\\]' THEN "lat"
        WHEN base REGEXP '\\[all\\.\\]' THEN "all"
        WHEN base REGEXP '\\[esp.\\]' THEN "esp"
      END
  WHERE  base_normalisee REGEXP '\\[.+[.]\\]' ;
```

On remarque que dans le deuxième aiguillage **CASE**, on s'appuie sur la valeur de l'attribut base et non sur celle de l'attribut base_normalisee. En effet, au moment où le deuxième aiguillage est examiné, les valeurs de l'attribut base_normalisee ont déjà été changées et l'indication de langue en a disparu. Une autre solution, qui permettrait d'utiliser dans les deux cas la valeur de l'attribut base_normalisee, serait d'inverser les deux aiguillages : le second ne change pas la valeur de l'attribut base_normalisee, mais celle de l'attribut langue_base.

L'examen d'un échantillon :

```
SELECT base,
       base_normalisee,
      langue_base
FROM esque_tmp
WHERE verif_base REGEXP 'moderne'
ORDER BY RAND()
LIMIT 10 ;
```

met en évidence les changements opérés :

base	*base_normalisee*	*langue_base*
corsesca [it.]	corsesca	it
thug [angl.]	thug	angl
levre [a.f.]	levre	af
morisco [esp.]	morisco	esp
pavone [it.]	pavone	it
background [angl.]	background	angl
pittoresco [it.]	pittoresco	it
game [angl.]	game	angl
settecento [it.]	settecento	it
graeciscus [lat.]	graeciscus	lat

On peut alors examiner la place des langues dans les bases :

```
SELECT langue_base,
     COUNT(*) AS 'o.'
FROM esque_tmp
GROUP BY langue_base
ORDER BY 'o.' DESC ;
```

Même s'il sous-estime probablement la part des bases étrangères (la notation de cette facette ne semblant pas avoir été systématique), le résultat met en évidence le rôle de l'italien et de l'espagnol :

TABLEAU 118 – ESQUE : bases avec et sans sens associé

base_normalisee	*base_POS*	*o.*
Boingo (Oingo)	npr	1
ballon	n	2
bombe	n	1
boulevard	n	1
cardinal	n	4
corsesca [it.]	absente	1
mirliton	n	2
morisco [esp.]	a	3
nana	n	2
pavone [it.]	n	1
pion	n	1
todesco, tedesco [it.]	absente	1
turlupin	n	2
zadjal	n	1

langue_base	*o.*
fr	4290
it	49
esp	21
angl	11
lat	8
af	3
all	1
prov	1

1.6. Sens associés aux bases

1.6.1. Constat

On opère pour les bases une vérification similaire à celle pour les catégories, à ceci près qu'on recourt aux attributs base_normalisee et base_POS plutôt qu'aux attributs de départ base et cat_base. Ces deux derniers, on le sait, sont peu fiables. La requête, dont le résultat figure au tableau 118 p. 385 est de la forme :

```
SELECT base_normalisee,
       base_POS,
       COUNT(*) AS 'o.'
FROM esque_tmp
WHERE CONCAT(base_normalisee, base_POS) IN
       (SELECT CONCAT(base_normalisee, base_POS)
        FROM esque_tmp
        WHERE sens_base IS NOT NULL)
      AND sens_base IS NULL
GROUP BY base_normalisee, base_POS ;
```

1.6.2. Signalement

```
CREATE TABLE base_a_sens_ou_non
       (SELECT CONCAT(base_normalisee, base_POS)
        FROM esque_tmp
        WHERE CONCAT(base_normalisee, base_POS) IN
```

TABLEAU 119 – ESQUE : bases à sens associé ou non

base	*base_POS*	*sens_base*	*o.*
Boingo (Oingo)	npr	**NULL**	1
Boingo (Oingo)	npr	chanteur	1
ballon	n	**NULL**	2
ballon	n	aviation	1
bombe	n	**NULL**	1
bombe	n	bombe sexuelle	2
boulevard	n	**NULL**	1
boulevard	n	genre de spectacles légers et populaires	1
cardinal	n	**NULL**	4
cardinal	n	couleur	1
corsesca [it.]	absente	**NULL**	1
corsesca [it.]	absente	arme à lame fourchue, originaire de la Corse	1
mirliton	n	**NULL**	2
mirliton	n	espèce de flûte	1
morisco [esp.]	a	**NULL**	3
morisco [esp.]	a	"arabe, musulman ; hispano-arabe"	2
nana	n	**NULL**	2
nana	n	"fille [publique]"	1
pavone [it.]	n	**NULL**	1
pavone [it.]	n	"paon"	1
pion	n	**NULL**	1
pion	n	enseign.	1
todesco, tedesco [it.]	absente	**NULL**	1
todesco, tedesco [it.]	absente	"allemand"	1
turlupin	n	**NULL**	2
turlupin	n	"faiseur de mauvais jeux de mots"	1
zadjal	n	**NULL**	1
zadjal	n	ou zéjel, "type de poésie arabe"	1

```
            (SELECT CONCAT(base_normalisee, base_POS)
              FROM esque_tmp
              WHERE sens_base IS NOT NULL)
            AND sens_base IS NULL
          GROUP BY base_normalisee, base_POS) ;

UPDATE esque_tmp
      SET verif_base =
            CONCAT(verif_base, "␣sens␣ou␣non")
      WHERE CONCAT(base_normalisee, base_POS) IN
            (SELECT * FROM base_a_sens_ou_non) ;
```

La mise en évidence des décalages possibles découle de la requête :

```
SELECT base,
      base_POS,
      sens_base,
      COUNT(*) AS 'o.'
FROM esque_tmp
WHERE verif_base REGEXP 'sens␣ou␣non'
GROUP BY base, base_POS, sens_base ;
```

qui s'appuie sur la présence, issue de la pénultième requête, de « sens ou non » dans l'attribut verif_base. Le résultat figure au tableau 119 p. 386.

1.7. Commentaires associés aux bases

1.7.1. Constat

Même démarche pour l'association ou non de commentaire à une base donnée :

```
SELECT base_normalisee,
       base_POS,
       COUNT(*) AS 'o.'
FROM esque_tmp
WHERE CONCAT(base_normalisee, base_POS) IN
      (SELECT CONCAT(base_normalisee, base_POS)
       FROM esque_tmp
       WHERE LENGTH(commentaire_base) > 0)
      AND LENGTH(commentaire_base) = 0
GROUP BY base_normalisee, base_POS ;
```

Le tableau 120 p. 388 fournit les bases problématiques sous cet angle.

1.7.2. Signalement

```
CREATE TABLE base_a_commentaire_ou_non
      (SELECT CONCAT(base_normalisee, base_POS)
        FROM esque_tmp
        WHERE CONCAT(base_normalisee, base_POS) IN
              (SELECT CONCAT(base_normalisee, base_POS)
                FROM esque_tmp WHERE LENGTH(commentaire_base) > 0)
             AND LENGTH(commentaire_base) = 0 GROUP BY base_normalisee, base_POS) ;

UPDATE esque_tmp
      SET verif_base  =
           CONCAT(verif_base, "␣commentaire␣ou␣non")
      WHERE CONCAT(base_normalisee, base_POS) IN
            (SELECT *
             FROM base_a_commentaire_ou_non) ;
```

Les bases correspondantes figurent au tableau 121 p. 389.

1.8. Variantes de dérivés

1.8.1. Constat

On prend en compte le fait que l'absence de variantes est représentée tantôt par la marque **NULL** tantôt par la chaîne vide. Le résultat de la requête suivante se trouve au tableau 122 p. 390 :

```
SELECT derive,
       cat_derive,
       base,
       base_POS
       COUNT(*) AS 'o.'
FROM esque_tmp
WHERE CONCAT(derive, cat_derive, base, base_POS) IN
```

TABLEAU 120 – ESQUE : bases avec et sans commentaire (extrait)

base_normalisee	*base_POS*	*o.*
Amiga	npr	1
Aragon (Louis)	npr	1
Aristophane	npr	3
Arétin (Pietro Aretino dit l')	npr	1
Barbarie	npr	1
Bardot (Brigitte)	npr	3
Barthez (Fabien)	npr	1
Bergame	npr	3
Berni (Francesco)	npr	1
Bernini (Gian Lorenzo)	npr	1
Boingo (Oingo)	npr	1
Bramante (Donato di Angelo Lazzari, dit il)	npr	1
Burton (Tim)	npr	1
Calvin (Jean Cauvin, dit)	npr	1
Camoëns (Luis de)	npr	1
Capra (Frank)	npr	1
Caravage (Michelangelo Meresi, dit le)	npr	3
Cassandre	npr	1
Cendrillon	npr	3
Cervantes (Miguel de)	npr	2
Chaban-Delmas (Jacques)	npr	2
Chaplin (Charlie)	npr	6
Charlemagne	npr	1
Chateaubriand (François René de)	npr	3
Chatwin (Bruce)	npr	1
Chopin (Frédéric)	npr	1
Clochemerle	npr	2
Cocteau (Jean)	npr	3
Courteline (Georges)	npr	2
D'Annunzio (Gabriele)	npr	1
Danaé	npr	1
Dante (Alighieri)	npr	6
David	npr	1
Demille (Cecil B.)	npr	1
Di Rupo (Elio)	npr	1
Diafoirus	npr	1
Don Quichotte	npr	6
Ellroy	npr	1
Fellini (Federico)	npr	3
Flash Gordon	npr	1
Florian (Jean-Pierre Claris de)	npr	1
Fragonard (Jean-Honoré)	npr	1
Gargantua	npr	3
Gaulle (Charles de)	npr	1
Giono (Jean)	npr	2
Giorgione	npr	1
Giotto	npr	2
Giuliani (Rudy)	npr	1
Goya (Francisco de)	npr	2
Harry Potter	npr	1
Hoffmann (Ernst)	npr	1

TABLEAU 121 – ESQUE : bases à commentaire ou non (extrait)

derive	*cat_derive*	*commentaire_base*	*base*	*base_POS*	*o.*
Cecil B Demillesque	a		Demille (Cecil B.)	npr	1
Cecil B. de millesque	a	cinéaste américain, 1881-1959	Demille (Cecil B.)	npr	1
Flash-Gordonesque	a	Film fantastique américain, britannique (1980).	Flash Gordon	npr	1
HarryPotteresque	a		Harry Potter	npr	1
Queen-esque	a	Groupe de musique	Queen	npr	1
Tim Burtonesque	a	Scénariste, enfant terrible du cinéma fantastique américain	Burton (Tim)	npr	1
Wooesque	a		Woo (John)	npr	1
acidesque	a		acide	n	1
acidesque	a	Drogue.	acide	n	1
amigaesque	a		Amiga	npr	1
amiganesque	a	Nom du site ?	Amiga	npr	1
annunziesque	a	Ecrivain italien (1863-1938).	D'Annunzio (Gabriele)	npr	1
arabesque	a		arabe	n	1
arabesque	a	Formé sur l'it. Arabesco.	arabe	n	1
aragonesque	a		Aragon (Louis)	npr	1
aragonesque	a	Ecrivain français (1897-1982).	Aragon (Louis)	npr	1
aretinesque	a		Arétin (Pietro Aretino dit l')	npr	1
aristophanesque	a		Aristophane	npr	3
aristophanesque	a	v. 445-v.386 av J.C.	Aristophane	npr	1
arétinesque	a	Ecrivain italien (1492-1556).	Arétin (Pietro Aretino dit l')	npr	1
ayatollesque	a		ayatollah	n	4
ayatollesque	a	Chef religieux chiite.	ayatollah	n	1
bamboulesque	a		bamboula	n	2
bamboulesque	a	Danse africaine, pop. "ripaille".	bamboula	n	1
barbaresque	a		Barbarie	npr	1
barbaresque	a	Ancienne Barbarie, c.a.d. Afrique du Nord.	Barbarie	npr	1
bardesque	a		Bardot (Brigitte)	npr	2
bardotesque	a		Bardot (Brigitte)	npr	1
barnumesque	a		barnum	n	2
barnumesque	a	"Forain, tapage, désordre", de Phineas Taylor Barnum, directeur de cirque américain (1810-1891).	barnum	n	1
barthesque	a	Allusion probable au baiser de Laurent Blanc sur le crâne de Fabien Barthez, goal de l'équipe de France, lors du Mondial 98.	Barthez (Fabien)	npr	1
barthesque	a		Barthez (Fabien) ?	npr	1
bergamesque	a		Bergame	npr	2

TABLEAU 122 – ESQUE : dérivés avec et sans variantes (extrait)

derive	*cat_derive*	*base*	*base_POS*	*o.*
abracadabresque	a	abracadabra	absente	3
backgroundesque	a	background [angl.]	n	1
bardesque	a	Bardot (Brigitte)	npr	1
bondesque	a	James Bond	npr	2
borgiesque	a	Borgia (Cesare)	npr	1
boy-scoutesque	a	boy-scout	n	1
brandoesque	a	Brando (Marlon)	npr	1
brouillonesque	nf	brouillon	n	1
canulardesque	a	canular	n	2
canularesque	a	canular	n	2
cauchemardesque	a	cauchemar	n	5
cauchemaresque	a	cauchemar	n	3
cendrillonesque	a	Cendrillon	npr	2
chateaubrianesque	a	Chateaubriand (François René de)	npr	1
churrigueresque	a	churrigueresco [esp.]	absente	1
clipesque	a	clip	n	2
clochemerlesque	a	Clochemerle	npr	2
cocteauesque	a	Cocteau (Jean)	npr	2
conesque	a	con	n	2
cornichonesque	a	cornichon	n	1

```
      (SELECT CONCAT(derive, cat_derive, base, base_POS)
       FROM esque_tmp
       WHERE variantes_derive IS NOT NULL
           AND variantes_derive <> '')
      AND (variantes_derive IS NULL
           OR variantes_derive = '')
GROUP BY derive, cat_derive, base, base, base_POS ;
```

1.8.2. Signalement

```
CREATE TABLE derive_a_variantes_ou_non
      (SELECT CONCAT(derive, cat_derive, base, base_POS)
        FROM esque_tmp
        WHERE CONCAT(derive, cat_derive, base, base_POS) IN
              (SELECT CONCAT(derive, cat_derive, base, base_POS)
               FROM esque_tmp
               WHERE variantes_derive IS NOT NULL
                    AND variantes_derive <> '')
           AND (variantes_derive IS NULL
                variantes_derive = '')
         GROUP BY derive, cat_derive, base, base_POS) ;

UPDATE esque_tmp
      SET verif_derive =
           CONCAT(verif_derive, "␣variantes␣ou␣non")
       WHERE CONCAT(derive, cat_derive, base, base_POS) IN
             (SELECT *
              FROM derive_a_variantes_ou_non) ;
```

1.9. Sens associés aux dérivés

1.9.1. Constat

Si l'on examine les sens du dérivé *soldatesque*, on obtient les résultats suivants :

base	*cat_base*	*derive*	*cat_derive*	*sens_derive*	*o.*
soldat	nm	soldatesque	a	vx	1
soldat	nm	soldatesque	a	**NULL**	2
soldatesque	a	soldatesque	nf	**NULL**	2

On constate que l'adjectif *soldatesque* est donné comme vx, vieux, dans un cas seulement sur 3. Il s'agit là d'une inégalité de traitement qui aboutit à une incohérence.

Par ailleurs, les inégalités de traitement peuvent ne pas dissimuler des incohérences mais des dérivés distincts (des homonymes). C'est la situation avec *romanesque* :

base	*cat_base*	*derive*	*cat_derive*	*sens_derive*	*o.*
romain	a	romanesque	a	vx	1
roman	nm	romanesque	a	**NULL**	5
romanesque	a	romanesque	nf	vx, "femme romaine"	1
romanesque	a	romanesque	nm	**NULL**	1

L'adjectif *romanesque* est vieux quand il provient de *romain*, pas quand il a pour base *roman*. Vérification faite, l'adjectif *soldatesque* a toujours pour base *soldat*, tandis que le nom *soldatesque* a pour base l'adjectif *soldatesque*.

En tout état de cause, il s'agit de repérer les cas où un couple dérivé/catégorie se rencontre parfois assorti d'un sens parfois non (tableau 123 p. 392 – le nombre d'occurences en colonne de droite indique le nombre de fois où le dérivé n'a pas de sens alors qu'il existe au moins une occurrence avec un sens associé). C'est encore une requête enchâssée qui permet d'isoler ces cas. La requête enchâssée produit l'ensemble des chaînes de caractères concaténant le dérivé et la catégorie quand ce couple a un sens associé. On est obligé d'utiliser la concaténation dans la mesure où l'opérateur **IN** attend un ensemble d'éléments et non pas un ensemble de couples. On recherche les lignes telles qu'une chaîne dérivé/catégorie retournée par la requête enchâssée se rencontre sans sens associé :

```
SELECT derive,
       cat_derive,
       base,
       base_POS,
       COUNT(*) AS 'o.'
FROM esque_tmp
WHERE CONCAT(derive, cat_derive, base, base_POS) IN
      (SELECT CONCAT(derive, cat_derive, base, base_POS)
       FROM esque_tmp
       WHERE sens_derive IS NOT NULL)
      AND sens_derive IS NULL
GROUP BY derive, cat_derive, base ;
```

Au total, c'est une dizaine de couples dérivé/catégorie qu'il faudra regarder de près pour éliminer les incohérences tout en préservant les homonymies.

TABLEAU 123 – ESQUE : dérivés avec et sans sens associé

derive	*cat_derive*	*base*	*base_POS*	*o.*
arabesque	a	arabe	n	1
calvinesque	a	Calvin (Jean Cauvin, dit)	npr	1
charlatanesque	a	charlatan	n	3
fanfaresque	a	fanfare	n	2
matelotesque	a	matelot	n	2
mazarinesque	a	Mazarin (Jules)	npr	1
philosophesque	a	philosophe	n	1
priapesque	a	Priape	npr	2
soldatesque	a	soldat	n	2
turlupinesque	a	turlupin	n	2

1.9.2. Signalement

```
CREATE TABLE derive_a_sens_ou_non
      (SELECT CONCAT(derive, cat_derive, base, base_POS)
       FROM esque_tmp
       WHERE CONCAT(derive, cat_derive, base, base_POS) IN
            (SELECT CONCAT(derive, cat_derive, base, base_POS)
             FROM esque_tmp
             WHERE sens_derive IS NOT NULL)
           AND sens_derive IS NULL
       GROUP BY derive, cat_derive, base  ;

UPDATE esque_tmp
     SET verif_derive =
          CONCAT(verif_derive, "␣sens␣ou␣non")
     WHERE CONCAT(derive, cat_derive, base, base_POS) IN
           (SELECT *
            FROM derive_a_sens_ou_non) ;
```

Les dérivés avec et sans sens associés sont fournis dans le tableau 123 p. 392.

1.10. Mode de formation des dérivés

1.10.1. Constat

Le tableau 125 p. 393 fournit le résultat de la requête :

```
SELECT derive,
      cat_derive,
      base,
      base_POS,
      COUNT(*) AS 'o.'
FROM esque_tmp
WHERE CONCAT(derive, cat_derive, base, base_POS) IN
      (SELECT CONCAT(derive, cat_derive, base, base_POS)
       FROM esque_tmp
       WHERE mode_formation IS NOT NULL)
      AND mode_formation IS NULL
GROUP BY derive, cat_derive, base, base_POS ;
```

TABLEAU 124 – ESQUE : dérivés avec et sans sens associé

derive	*cat_derive*	*sens_derive*	*base*	*base_POS*	*o.*
arabesque	a	vx	arabe	n	1
arabesque	a	**NULL**	arabe	n	1
calvinesque	a	vx	Calvin (Jean Cauvin, dit)	npr	1
calvinesque	a	**NULL**	Calvin (Jean Cauvin, dit)	npr	1
charlatanesque	a	vx	charlatan	n	1
charlatanesque	a	**NULL**	charlatan	n	3
fanfaresque	a	vx	fanfare	n	1
fanfaresque	a	**NULL**	fanfare	n	2
matelotesque	a	vx	matelot	n	1
matelotesque	a	**NULL**	matelot	n	2
mazarinesque	a	vx	Mazarin (Jules)	npr	1
mazarinesque	a	**NULL**	Mazarin (Jules)	npr	1
philosophesque	a	vx	philosophe	n	1
philosophesque	a	**NULL**	philosophe	n	1
priapesque	a	vx	Priape	npr	1
priapesque	a	**NULL**	Priape	npr	2
soldatesque	a	vx	soldat	n	1
soldatesque	a	**NULL**	soldat	n	2
turlupinesque	a	vx	turlupin	n	1
turlupinesque	a	**NULL**	turlupin	n	2

TABLEAU 125 – ESQUE : dérivés avec et sans mode de formation

derive	*cat_derive*	*base*	*base_POS*	*o.*
soldatesque	a	soldat	n	2

TABLEAU 126 – ESQUE : dérivés avec mode de formation ou non

derive	*cat_derive*	*mode_formation*	*base*	*base_POS*	*o.*
soldatesque	a	**NULL**	soldat	n	2
soldatesque	a	formé sur l'it. Soldatesco	soldat	n	1

1.10.2. Signalement

```
CREATE TABLE derive_a_formation_ou_non
       (SELECT CONCAT(derive, cat_derive, base, base_POS)
        FROM esque_tmp
        WHERE CONCAT(derive, cat_derive, base, base_POS) IN
              (SELECT CONCAT(derive, cat_derive, base, base_POS)
               FROM esque_tmp
               WHERE mode_formation IS NOT NULL)
          AND mode_formation IS NULL
      GROUP BY derive, cat_derive, base, base_POS) ;

UPDATE esque_tmp
      SET verif_derive =
            CONCAT(verif_derive, " formation ou non")
      WHERE CONCAT(derive, cat_derive, base, base_POS) IN
            (SELECT *
              FROM derive_a_formation_ou_non) ;
```

Le tableau 125 p. 393 montre qu'un seul dérivé est concerné. Faut-il en conclure qu'il y a deux adjectifs *soldatesque*, l'un ayant pour origine le français *soldat* et l'autre l'italien *soldatesco* ?

1.11. Caractéristiques générales des attestations

La table esque_tmp compte 4 384 lignes.

- 2 292 n'ont pas d'auteur (la marque **NULL** le signale) ;
- 362 n'ont pas de contexte ;
- 120 n'ont pas de référence ;
- 73 ont une valeur pour reference_dans_dictionnaire ;
- 9 ont une valeur pour reference_complement ;
- 3 039 ont une valeur pour origine ;
- 87 ont une valeur pour annee_complement.

1.12. Origine des attestations

La requête suivante permet d'examiner la répartition des 3 039 valeurs différentes de l'attribut origine :

```
SELECT origine, COUNT(*) AS 'o.'
FROM esque_tmp
GROUP BY origine ;
```

Son résultat figure au tableau 127 p. 395.

TABLEAU 127 – ESQUE : valeurs de l'attribut origine

origine	*o.*
NULL	1345
AFC	72
AFC, Björkman 118	2
AFC, MNC 1	1
Björkman (sv), DDL 47	1
Björkman 100	11
Björkman 101	14
Björkman 102	10
Björkman 103	13
Björkman 104	11
Björkman 105	18
Björkman 106	15
Björkman 107	15
Björkman 108	19
Björkman 109	18
Björkman 110	16
Björkman 111	17
Björkman 112	10
Björkman 113	16
Björkman 114	16
Björkman 115	14
Björkman 116	18
Björkman 117	13
Björkman 118	14
Björkman 119	13
Björkman 120	16
Björkman 121	11
Björkman 122	15
Björkman 123	11
Björkman 28	3
Björkman 30	2
Björkman 31	1
Björkman 32	9
Björkman 33	7
Björkman 34	7
Björkman 35	4
Björkman 36	4
Björkman 55	3
Björkman 65	3
Björkman 66	7
Björkman 67	1
Björkman 71	2
Björkman 72	1
Björkman 73	3
Björkman 74	5
Björkman 79	2
Björkman 81	5
Björkman 85	1
Björkman 86	5
Björkman 87	4
Björkman 89	2
Björkman 93	2
Björkman 94	1
Björkman 99	7
Bornéo	247
Bornéo, MNC 2	1
DDL 10	1
DDL 10, Björkman 116	1
DDL 12	3
DDL 13	8
DDL 14	8
DDL 15	38
DDL 15, Björkman 110	1
DDL 15, Björkman 120	1
DDL 17	11
DDL 18	2
DDL 2	1
DDL 20	2
DDL 21	5
DDL 22	2
DDL 24	12
DDL 25	2
DDL 25 ; Björkman 34	1
DDL 26	4
DDL 27	3
DDL 28	3
DDL 30	2
DDL 31	4
DDL 33	3
DDL 34	12
DDL 34, Björkman 112	1
DDL 35	2
DDL 35, Björkman 116	2
DDL 35, Björkman 120	1
DDL 35, Björkman 31	1
DDL 35, Björkman 74	1
DDL 37	4
DDL 39	4
DDL 40	12
DDL 40, Björkman 93	1
DDL 42	7
DDL 43	2
DDL 46	1
DDL 47	2
DDL 5	4
DDL 6, Björkman 123	1
DDL 7	5
DDL 9	1
DDl 14	1
FEW V, 472	1
Frantext	44
Frantext, DDL 35, Björkman 35	1
GlossaNet	3
MNC 1	17
MNC 2	9
WEB	1
attestaion orale	3
attestation orale	92
correspondance privée	12
web	1913
ww	1

TABLEAU 128 – Esque : références aux dérivés dans des recueils

reference_dans_dictionnaire	*o.*
DSA (sv), 1993	67
Björkman (sv)	3
DSA (sv région), 1993	1
DSA (sv caramboleur),1993	1
DSA (sv) 1993	1

1.13. Références aux dérivés dans des recueils

1.13.1. Constat

Dans la table esque, pour chaque ligne, figure la propriété reference_dans_dictionnaire, le plus souvent mise à **NULL**, mais qui a pour fonction, semble-t-il, de noter les mentions de dérivés en *-esque* dans deux ouvrages de référence (si bien que le nom de l'attribut n'est pas adapté), comme le montre la requête :

R^{2}_{119}

RESTRICTION(
reference_dans_dictionnaire EST PAS NULL
[esque]
↪ REGROUPER SUR(
reference_dans_dictionnaire
[<résultat$_1$>]
↪ PAR GROUPE(
reference_dans_dictionnaire,
NOMBRE DE LIGNES() TITRE 'o.'
)
[résultat$_2$>]
↪ TRI SUR('o')[<résultat$_3$>]

dont le tableau 128 p. 396 donne le résultat et dont l'équivalent MySQL est :

```
SELECT reference_dans_dictionnaire,
     COUNT(*) AS 'o.'
FROM esque
WHERE reference_dans_dictionnaire IS NOT NULL
GROUP by reference_dans_dictionnaire
ORDER BY 'o.' DESC ;
```

La requête suivante vise à isoler les dérivés qui tantôt ont la marque **NULL** pour cet attribut, tantôt non.

```
SELECT derive,
      cat_derive,
      base,
      base_POS,
     COUNT(*) AS 'o.'
FROM esque_tmp
WHERE CONCAT(derive, cat_derive, base, base_POS) IN
      (SELECT CONCAT(derive, cat_derive, base, base_POS)
        FROM esque_tmp
```

TABLEAU 129 – ESQUE : dérivés avec et sans références dans des recueils

derive	*cat_derive*	*base*	*base_POS*	*o.*
auberginesque	a	aubergine	n	1
bananiesque	a	Banania	npr	1
berlusconesque	a	Berlusconi (Silvio)	npr	1
bondesque	a	James Bond	npr	2
braquemardesque	a	braquemard	n	1
cacophonesque	a	cacophonie	n	1
caméléonesque	a	caméléon	n	4
casanovesque	a	Casanova (Francesco)	npr	3
chaplinesque	a	Chaplin (Charlie)	npr	5
connesque	a	con	n	2
gaguesque	a	gag	n	2
hiroshimiesque	a	Hiroshima	npr	2
jocondesque	a	Joconde (la)	npr	1
kangouresque	a	kangourou	n	2
mandolinesque	a	mandoline	n	1
muranesque	a	Murano	npr	3
ménageresque	a	ménagerie	n	4
niagaresque	a	Niagara	npr	3
olidesque	a	Olida	npr	3
panthéonesque	a	Panthéon	npr	3
patatesque	a	patate	n	2
plumesque	a	plume	n	1
pététesque	a	PTT	sigle	2
rocambolesque	a	Rocambole	npr	4
récamiesque	a	Récamier (Madame de)	npr	1
tarzanesque	a	Tarzan	npr	1
tobogganesque	a	toboggan	n	1
tétonnesque	a	téton	n	1
vermotesque	a	Vermot (almanach)	npr	1

```
      WHERE reference_dans_dictionnaire IS NOT NULL)
  AND reference_dans_dictionnaire IS NULL
GROUP BY derive, cat_derive, base, base_POS ;
```

Le résultat de la requête qui précède se trouve au tableau 129 p. 397.

1.13.2. Signalement

```
CREATE TABLE derive_dans_dic_ou_non
     (SELECT CONCAT(derive, cat_derive, base, base_POS)
       FROM esque_tmp
       WHERE CONCAT(derive, cat_derive, base, base_POS) IN
            (SELECT CONCAT(derive, cat_derive, base, base_POS)
              FROM esque_tmp
              WHERE reference_dans_dictionnaire IS NOT NULL)
         AND reference_dans_dictionnaire IS NULL
       GROUP BY derive, cat_derive, base, base_POS) ;

UPDATE esque_tmp
      SET verif_derive =
          CONCAT(verif_derive, "␣dictionnaire␣ou␣non")
     WHERE CONCAT(derive, cat_derive, base, base_POS) IN
          (SELECT *
```

```
        FROM derive_dans_dic_ou_non) ;
```

Le tableau 130 p. 399 fournit les dérivés correspondants.

1.14. Datation des contextes

1.14.1. Constat

L'examen des attributs jour, mois et annee met en évidence d'une part des conventions de notation à respecter, d'autre part des incohérences à rectifier. Pour les jours, 0 sert à indiquer que le jour n'est pas connu, mais on trouve, outre les jours de 1 à 31, le jour 127... De la même manière, 0 note la non-connaissance du mois, mais on trouve le mois 15. Enfin, pour les années, la situation est plus confuse, comme le montre le tableau 131 p. 131, qui provient de la requête :

```
SELECT
    CASE
        WHEN annee >= 1300 AND annee <= 1399 THEN '14e␣siècle'
        WHEN annee >= 1400 AND annee <= 1499 THEN '15e␣siècle'
        WHEN annee >= 1500 AND annee <= 1599 THEN '16e␣siècle'
        WHEN annee >= 1600 AND annee <= 1699 THEN '17e␣siècle'
        WHEN annee >= 1700 AND annee <= 1799 THEN '18e␣siècle'
        WHEN annee >= 1800 AND annee <= 1899 THEN '19e␣siècle'
        WHEN annee >= 1900 AND annee <= 1999 THEN '20e␣siècle'
        ELSE CONCAT("'année'␣", annee)
    END AS 'période/année',
    COUNT(*) AS 'o.'
FROM esque_tmp
GROUP BY
    CASE
        WHEN annee >= 1300 AND annee <= 1399 THEN '14e'
        ...
        WHEN annee >= 1900 AND annee <= 1999 THEN '20e'
        ELSE annee
    END
ORDER BY 'période/année' ;
```

On peut imaginer que les « années » 16, 18, 186 et 2201 correspondent à des erreurs et sont donc à rectifier, par exemple en indiquant que l'année n'est pas connue. On note par ailleurs le déséquilibre de « représentation » des différents siècles : la place écrasante des 20^{e} et 21^{e} siècles peut renvoyer à la fois à une plus grande productivité du suffixe *-esque* et à la plus grande disponibilité pour cette période de corpus numériques et d'outils de moissonnage automatique, sur la Toile en particulier.

La requête suivante :

```
SELECT jour,
    mois,
    annee
FROM esque_tmp
WHERE
    (annee <> 0
        AND (annee NOT BETWEEN 1300 AND 2005))
    OR (mois NOT BETWEEN 0 AND 12)
    OR (jour NOT BETWEEN 0 AND 31) ;
```

isole les lignes dont la datation « boite ». Le résultat figure au tableau 132 p. 400.

TABLEAU 130 – ESQUE : dérivés avec et sans référence dictionnairique

derive	*cat_derive*	*reference_dans_dictionnaire*	*base*	*base_POS*	*o.*
auberginesque	a	**NULL**	aubergine	n	1
auberginesque	a	DSA (sv), 1993	aubergine	n	1
bananiesque	a	**NULL**	Banania	npr	1
bananiesque	a	DSA (sv), 1993	Banania	npr	1
berlusconesque	a	**NULL**	Berlusconi (Silvio)	npr	1
berlusconesque	a	DSA (sv), 1993	Berlusconi (Silvio)	npr	1
bondesque	a	**NULL**	James Bond	npr	2
bondesque	a	DSA (sv), 1993	James Bond	npr	1
braquemardesque	a	**NULL**	braquemard	n	1
braquemardesque	a	DSA (sv), 1993	braquemard	n	1
cacophonesque	a	**NULL**	cacophonie	n	1
cacophonesque	a	DSA (sv), 1993	cacophonie	n	1
caméléonesque	a	**NULL**	caméléon	n	4
caméléonesque	a	DSA (sv), 1993	caméléon	n	1
casanovesque	a	**NULL**	Casanova (Francesco)	npr	3
casanovesque	a	DSA (sv), 1993	Casanova (Francesco)	npr	1
chaplinesque	a	**NULL**	Chaplin (Charlie)	npr	5
chaplinesque	a	DSA (sv), 1993	Chaplin (Charlie)	npr	1
connesque	a	**NULL**	con	n	2
connesque	a	DSA (sv), 1993	con	n	1
gaguesque	a	**NULL**	gag	n	2
gaguesque	a	Björkman (sv)	gag	n	1
hiroshimiesque	a	**NULL**	Hiroshima	npr	2
hiroshimiesque	a	DSA (sv), 1993	Hiroshima	npr	1
jocondesque	a	**NULL**	Joconde (la)	npr	1
jocondesque	a	DSA (sv), 1993	Joconde (la)	npr	1
kangouresque	a	**NULL**	kangourou	n	2
kangouresque	a	DSA (sv), 1993	kangourou	n	1
mandolinesque	a	**NULL**	mandoline	n	1
mandolinesque	a	DSA (sv), 1993	mandoline	n	1
muranesque	a	**NULL**	Murano	npr	3
muranesque	a	DSA (sv), 1993	Murano	npr	1
ménageresque	a	**NULL**	ménagerie	n	4
ménageresque	a	DSA (sv), 1993	ménagerie	n	1
niagaresque	a	**NULL**	Niagara	npr	3
niagaresque	a	DSA (sv), 1993	Niagara	npr	1
olidesque	a	**NULL**	Olida	npr	3
olidesque	a	DSA (sv), 1993	Olida	npr	1
panthéonesque	a	**NULL**	Panthéon	npr	3
panthéonesque	a	DSA (sv), 1993	Panthéon	npr	1
patatesque	a	**NULL**	patate	n	2
patatesque	a	DSA (sv), 1993	patate	n	1
plumesque	a	**NULL**	plume	n	1
plumesque	a	DSA (sv), 1993	plume	n	1
pététesque	a	**NULL**	PTT	sigle	2
pététesque	a	DSA (sv), 1993	PTT	sigle	1
rocambolesque	a	**NULL**	Rocambole	npr	4
rocambolesque	a	DSA (sv) 1993	Rocambole	npr	1
récamiesque	a	**NULL**	Récamier (Madame de)	npr	1
récamiesque	a	DSA (sv), 1993	Récamier (Madame de)	npr	1
tarzanesque	a	**NULL**	Tarzan	npr	1
tarzanesque	a	DSA (sv), 1993	Tarzan	npr	1
tobogganesque	a	**NULL**	toboggan	n	1
tobogganesque	a	DSA (sv), 1993	toboggan	n	1
tétonnesque	a	**NULL**	téton	n	1
tétonnesque	a	DSA (sv), 1993	téton	n	1
vermotesque	a	**NULL**	Vermot (almanach)	npr	1
vermotesque	a	DSA (sv), 1993	Vermot (almanach)	npr	1

TABLEAU 131 – Esque : répartition des contextes par « année » ou siècle

période/année	*o.*
'année' 0	97
'année' 16	1
'année' 18	1
'année' 186	1
'année' 2000	207
'année' 2001	645
'année' 2002	873
'année' 2003	36
'année' 2004	6
'année' 2005	8
'année' 2201	1
14e siècle	1
15e siècle	1
16e siècle	63
17e siècle	36
18e siècle	10
19e siècle	235
20e siècle	2162

TABLEAU 132 – ESQUE : contextes à datation « boiteuse »

jour	*mois*	*annee*
9	8	186
0	0	18
0	0	16
127	10	1999
20	3	2201
10	15	1999

1.14.2. Réorganisation

On peut, sur la base de ces constats, mettre en place une représentation parallèle des datations des contextes qui soit plus cohérente. On commence par ajouter un attribut datation de type date qui peut contenir la marque **NULL**. La marque **NULL** signifie en l'occurrence soit que la datation minimale, l'année, est inconnue, ce qui est signifié conventionnellement par 0 dans la table esque_princeps, soit que les informations sur l'année de la table esque_princeps ne sont pas fiables.

```
ALTER TABLE esque_tmp
    ADD COLUMN datation DATE ;
```

Il reste à mettre à jour cette colonne en fonction des informations collectées dans les attributs jour, mois et annee, en tenant compte des incohérences repérées :

```
UPDATE esque_tmp
    SET datation = CONCAT(annee, "-", mois, "-", jour)
WHERE (annee = 0 OR annee BETWEEN 1300 AND 2005)
    AND (mois BETWEEN 0 AND 12)
    AND (jour BETWEEN 0 AND 31) ;
```

Au total, sur 4 384 contextes, seuls 6 sont dépourvus d'une date (les 4 « années » erronées, plus les deux dates dont soit le jour soit le mois sont incohérents).

1.14.3. Signalement

```
CREATE TABLE contexte_a_datation_boiteuse
     (SELECT numero_ligne
      FROM esque_tmp
      WHERE (annee <> 0
                AND (annee NOT BETWEEN 1300 AND 2005))
           OR (mois NOT BETWEEN 0 AND 12)
           OR (jour NOT BETWEEN 0 AND 31)) ;

UPDATE esque_tmp
    SET verif_contexte = CONCAT(verif_contexte, ' datation boiteuse' )
    WHERE numero_ligne IN
          (SELECT *
           FROM contexte_a_datation_boiteuse) ;
```

1.15. Bilan des changements et problèmes

Les trois requêtes suivantes permettent de saisir l'ampleur des problèmes rencontrés :

```
SELECT verif_base,
    COUNT(*) AS 'o.'
FROM esque_tmp
GROUP BY verif_base ;

SELECT verif_derive,
    COUNT(*) AS 'o.'
FROM esque_tmp
GROUP BY verif_derive ;

SELECT verif_contexte,
    COUNT(*) AS 'o.'
```

TABLEAU 133 – ESQUE : changements et problèmes (1/2)

verif_base	*o.*
	1744
2 bases	1
base non français moderne	16
changement catégorie	1806
changement catégorie 2 bases	2
changement catégorie base non français moderne	38
changement catégorie commentaire ou non	50
changement catégorie commentaire ou non base non français moderne	8
changement catégorie discordance catégorie	62
changement catégorie discordance catégorie base non français moderne	12
changement catégorie discordance catégorie commentaire ou non	7
changement catégorie discordance catégorie mot-valise	3
changement catégorie discordance catégorie normalisation base	58
changement catégorie discordance catégorie sens ou non base non français moderne	2
changement catégorie mot-valise	23
changement catégorie normalisation base	124
changement catégorie normalisation base 2 bases	5
changement catégorie normalisation base base non français moderne	3
changement catégorie normalisation base commentaire ou non	1
changement catégorie normalisation base mot-valise	11
changement catégorie sens ou non	21
changement catégorie sens ou non base non français moderne	2
changement catégorie sens ou non commentaire ou non	5
changement catégorie sens ou non commentaire ou non base non français moderne	2
commentaire ou non	266
discordance catégorie	27
discordance catégorie base non français moderne	6
discordance catégorie commentaire ou non	6
discordance catégorie normalisation base	3
discordance catégorie sens ou non base non français moderne	5
mot-valise	4
normalisation base	49
normalisation base 2 bases	3
normalisation base commentaire ou non	5
normalisation base mot-valise	2
sens ou non commentaire ou non	2

```
FROM esque_tmp
GROUP BY verif_contexte ;
```

Leurs résultats figurent dans les tableaux 133 et 134 p. 402 et p. 403.

2. Préparer les corrections manuelles

2.1. Objectifs généraux

L'objectif est de reprendre les données de la table `esque` pour préparer l'éclatement de cette table en plusieurs tables, en fonction de la modélisation Entités/Associations qui a été conduite au ch. IX. Les redondances entre lignes de la table actuelle doivent laisser place à des informations distribuées entre plusieurs tables. Là où les lignes *charentonnesque* et *charentonesque*, dérivés adjectivaux, provenant tous deux du nom propre *Charenton*, dans le sens d'/hôpital psychiatrique d'une ville du sud-est de Paris/' répètent les informations sur

TABLEAU 134 – ESQUE : changements et problèmes (2/2)

verif_base	*o.*
	1765
changement catégorie	1869
changement catégorie commentaire ou non	58
changement catégorie discordance catégorie	77
changement catégorie discordance catégorie commentaire ou non	7
changement catégorie discordance catégorie normalisation base	58
changement catégorie discordance catégorie sens ou non	2
changement catégorie normalisation base	143
changement catégorie normalisation base commentaire ou non	1
changement catégorie sens ou non	23
changement catégorie sens ou non commentaire ou non	7
commentaire ou non	266
discordance catégorie	33
discordance catégorie commentaire ou non	6
discordance catégorie normalisation base	3
discordance catégorie sens ou non	5
normalisation base	54
normalisation base commentaire ou non	5
sens ou non commentaire ou non	2

verif_derive	*o.*
	3697
changement catégorie	128
changement catégorie discordance catégorie	72
changement catégorie variantes ou non	4
dictionnaire ou non	75
dictionnaire ou non variantes ou non	13
discordance catégorie	138
discordance catégorie sens ou non	2
discordance catégorie sens ou non formation ou non	3
discordance catégorie variantes ou non	5
sens ou non	22
variantes ou non	225

verif_contexte	*o.*
	4378
datation boiteuse	6

cette base, il s'agit au contraire de fournir une seule fois les informations sur cette base et simplement y renvoyer dans les lignes de la table correspondant aux dérivés. Le principe est le même pour les attestations d'un même dérivé.

Pour pouvoir distribuer les informations entre plusieurs tables, il faut au préalable faire en sorte que des discordances non voulues ne viennent pas éclater entre 2 bases (ou entre 2 dérivés) qui relèvent en fait d'un même chef. Un premier type de discordance se concrétise par le fait de fournir conceptuellement la même information, mais sous une forme légèrement différente. Dans l'exemple utilisé pour la 2^{e} forme norme normale (ch. IX § 5), les commentaires concernant la base *Tonton*, npr, divergent en fait un tantinet : *Surnom donné à François Mitterrand.* dans un cas (le point fait partie de la valeur de l'attribut), *surnom de François Mitterrand* dans l'autre. Il y a 3 points d'éloignement entre les deux valeurs : majuscule ou non à l'initiale, point final ou non, *donné à* vs. de. Le second type de discordance, le plus fréquent, de loin, se marque par l'alternance, pour un dérivé ou une base donné(e) et pour un attribut particulier, d'une version où l'attribut a une valeur et d'une version où il n'en a pas.

La répartition des informations sur les mots en *-esque* peut supposer à l'inverse de « dégrouper » les bases et les dérivés quand il y a lieu.

On fait le pari que les opérations cruciales pour faciliter l'identification correcte des bases et des dérivés ont été effectuées supra :

- normalisation des mots bases avec enlèvement de l'indication d'hésitation sur la nature de la base ;
- éclatement des catégories en partie du discours et traits ;
- mise à part des informations concernant la langue de départ du mot-base.

On veut vérifier manuellement d'abord les bases, puis les dérivés, enfin les attestations. On va utiliser pour ce faire Access et donc exporter vers Access les tables nécessaires.

La première table, bases_tmp, est une table provisoire (cf. tmp comme abréviation de temporaire) qui va recevoir une ébauche des bases définitives. On la crée via les requêtes suivantes :

```
CREATE TABLE bases_tmp
     (SELECT base_normalisee AS 'base',
          base_POS AS 'POS',
          base_POS_traits AS 'traits',
          base_POS_douteuse AS 'POS_douteuse',
          0 AS 'numero_sens',
          sens_base AS 'sens',
          commentaire_base AS 'commentaire',
          base_locution AS 'locution',
          langue_base AS 'langue',
          COUNT(*) AS 'nbre_attestations',
          base AS 'base_depart',
          cat_base AS 'cat_depart',
          verif_base
     FROM esque_tmp
     WHERE statut_base <> 'absente'
     GROUP BY base_normalisee, base_POS) ;

ALTER TABLE bases_tmp
     ADD COLUMN id_base INT UNSIGNED NOT NULL AUTO_INCREMENT FIRST,
     ADD INDEX(id_base) ;
```

La première requête opère un regroupement de la table esque_tmp sur les attributs base_normalisee et base_POS, c'est-à-dire sur la version « nettoyée » tant de la forme-même de la base que de sa catégorie. Les attributs conservés sont à la fois ceux qui résultent du nettoyage (se

rajoutent aux deux attributs qui viennent d'être cités base_POS_traits, base_POS_douteuse, numero_sens, avec comme valeur par défaut 0, langue_base et base_locution) et d'autres conservés de la table esque de départ : sens_base, commentaire_base, la forme de départ (base_depart), la catégorie de départ (cat_depart). On pourrra ainsi examiner de manière fine si les transformations effectuées l'ont été à juste escient. L'attribut verif_base, quant à lui, indique les réorganisations effectuées et les problèmes décelés. On note qu'on renomme les attributs pour rendre les noms plus « transparents » et plus manipulables.

La deuxième requête ajoute un attribut auto-incrémenté, id_base : chaque base potentielle se trouve désormais dotée d'un identifiant numérique unique, qui sert de clé primaire.

La structure de départ de la table bases_tmp est donc la suivante.

Field	*Type*	*Null*	*Key*	*Default*	*Extra*
id_base	int(10) unsigned		MUL		auto_increment
base	varchar(100)			base manquante	
POS	varchar(10)			absente	
traits	varchar(10)	YES			
POS_douteuse	int(11)			0	
numero_sens	bigint(1)			0	
sens	varchar(100)	YES			
commentaire	blob	YES			
locution	int(11)			0	
langue	varchar(10)			fr	
nbre_attestations	bigint(21)			0	
base_depart	varchar(100)	YES			
cat_depart	varchar(10)	YES			
verif_base	varchar(255)	YES			

On modifie en parallèle la table esque_tmp, en lui ajoutant un attribut id_base. On met à jour la valeur de cet attribut en profitant d'une jointure avec la table bases_tmp : on donne à ce nouvel attribut la valeur qu'il a dans la table bases_tmp quand il y a partage des valeurs des attributs base_normalisee et base d'un côté et base_POS de l'autre. On donne donc à un couple (<base>, <partie du discours>) le même identifiant dans les deux tables, ce qui permet d'établir entre elles une jointure $1-n$: à une entrée de la table bases_tmp correspond 1 ou n entrées de la table esque_tmp. On peut par ailleurs détailler le nombre d'entrées associées aux bases grâce à la requête :

```
SELECT nbre_attestations AS 'Nombre d\'attestations par base',
     COUNT(*) AS 'o.'
FROM bases_tmp
GROUP BY nbre_attestations ;
```

Si l'immense majorité des bases n'a qu'une attestation, 2 bases en ont 10.

Nombre d'attestations par base	*o.*
1	2449
2	435
3	152
4	63
5	28
6	11
7	7
8	1
10	2

2.2. Exporter les données à partir de MySQL en format délimité

Pour chacune des deux tables bases_tmp et esque_tmp, on produit un fichier délimité pour l'import sous Access. Le fichier est délimité d'une part par le caractère tabulation entre les colonnes et, entre les lignes, par la suite de caractères \r\n, qui correspond au changement de lignes sous Windows.

Le ch. XIII § 2 détaille la notion de fichier délimité.

L'export est ici effectué sous Linux et profite des échanges possibles entre commandes. La commande echo suivie d'une chaîne et suivie d'un pipe, la barre verticale, permet d'envoyer à la commande mysql| la requête à effectuer. Le résultat de cette commande est renvoyé par MySQL sous la forme d'une suite de caractères qu'il faut sauver dans un fichier, via la commande de redirection qu'est le chevron fermant. On se voit demander le mot de passe d'accès à la base de données correspondant à l'utilisateur indiqué.

Pour un attribut pouvant contenir la marque **NULL**, on exporte non pas cette marque qui deviendrait la simple chaîne de caractères 'NULL' dans le fichier de sortie, mais la chaîne vide '', qui sera remplacée par la marque **NULL** par Access. C'est le rôle de

IFNULL(<attribut>, <valeur remplaçant la marque **NULL**>)

qui retourne la valeur de l'attribut quand il ne s'agit pas de la marque **NULL** et qui sinon renvoie la valeur donnée comme deuxième argument.

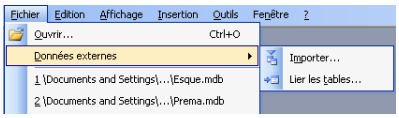
Les commandes :
|sed −e "s/.∗/&~/"|tr '~' '\r'
insèrent par ailleurs un retour-chariot (\r) avant le caractère \n.

```
echo "SELECT␣id_base,␣base,␣POS,␣IFNULL(traits,␣'')␣AS␣'traits',␣POS_douteuse,␣
    numero_sens,␣IFNULL(sens,␣'')␣AS␣'sens',␣commentaire,␣locution,␣langue,␣
    nbre_attestations,␣base_depart,cat_depart,␣verif_base␣FROM␣bases_tmp␣"|mysql −u
    habert −p esque  |sed −e "s/.∗/&~/"  |tr '~' '\r'  > bases_tmp.txt

echo "SELECT␣numero_ligne␣AS␣numero_attestation,␣␣id_base,␣base_normalisee,␣base_POS
    ,␣IFNULL(base_POS_traits,␣'')␣AS␣'base_POS_traits',␣base_POS_douteuse,␣
    base_locution,␣langue_base,␣IFNULL(sens_base,␣'')␣AS␣'sens_base',␣IFNULL(
    commentaire_base,␣'')␣AS␣'commentaire_base',␣IFNULL(base,␣'')␣AS␣'base_depart',␣
    IFNULL(cat_base,␣'')␣AS␣'cat_base_depart',␣verif_base,␣␣statut_base,␣id_derive,␣
    derive,␣derive_POS,␣IFNULL(derive_POS_traits,␣'')␣AS␣'derive_POS_traits',␣
    derive_POS_douteuse,␣IFNULL(sens_derive,␣'')␣AS␣'sens_derive',␣IFNULL(cat_derive
    ,␣'')␣AS␣'cat_derive_depart',␣IFNULL(variantes_derive,␣'')␣AS␣'variantes_derive
    ',␣IFNULL(mode_formation,␣'')␣AS␣'mode_formation',␣verif_derive,␣␣IFNULL(
    contexte,␣'')␣AS␣'contexte',␣IFNULL(reference,␣'')␣AS␣'reference',␣IFNULL(
    reference_dans_dictionnaire,␣'')␣AS␣'reference_dans_dictionnaire',␣IFNULL(auteur
    ,␣'')␣AS␣'auteur',␣IFNULL(reference_complement,␣'')␣AS␣'reference_complement',␣
    IFNULL(jour,␣'')␣AS␣'jour',␣IFNULL(mois,␣'')␣AS␣'mois',␣IFNULL(annee,␣'')␣AS␣'
    annee',␣IFNULL(datation,␣'')␣AS␣'datation',␣IFNULL(annee_complement,␣'')␣AS␣'
    annee_complement',␣IFNULL(origine,␣'')␣AS␣'origine',␣verif_contexte␣FROM␣
    esque_tmp" |mysql −u habert −p esque |sed −e "s/.∗/&~/"  |tr '~' '\r'  >
    esque_tmp.txt
```

2.3. Importation d'un fichier tabulé sous Access

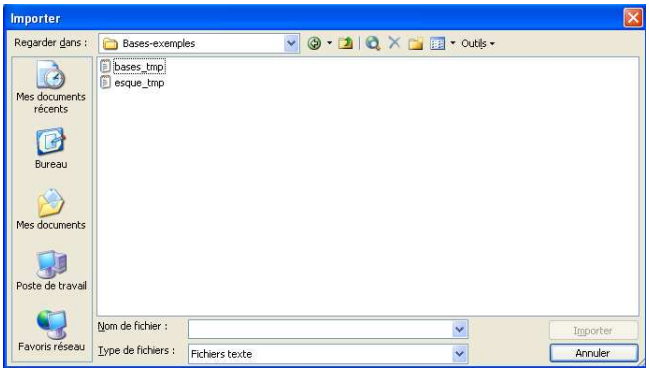
L'importation s'effectue via [Fichier | Données externes | Importer].

1



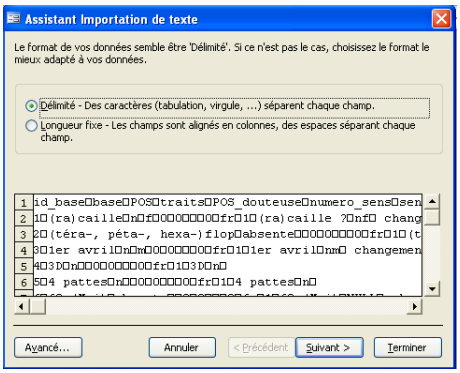
Au sein du dossier où figure le fichier tabulé (auquel est souvent donnée l'extension .txt, pour indiquer que c'est du texte « brut »), on choisit de visualiser les fichiers textes. On double-clique alors sur le nom de fichier pertinent, ici Bases_tmp.txt.

2



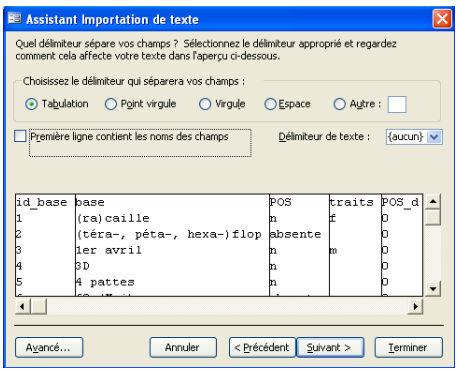
L'import proprement dit commence. Le choix par défaut, correct en l'occurrence, est l'importation d'un fichier délimité par des caractères spéciaux séparant les colonnes et les lignes.

3



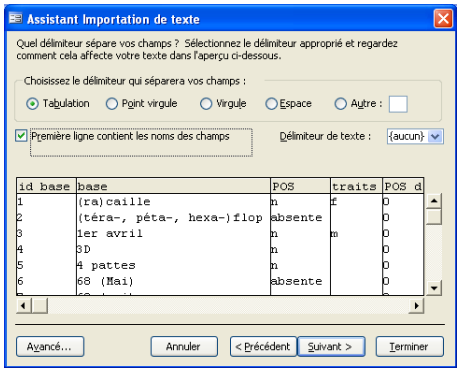
On peut choisir le caractère de délimitation des colonnes, par défaut la tabulation, ce qui convient ici. Une case permet d'indiquer si la première ligne contient les noms des colonnes.

4



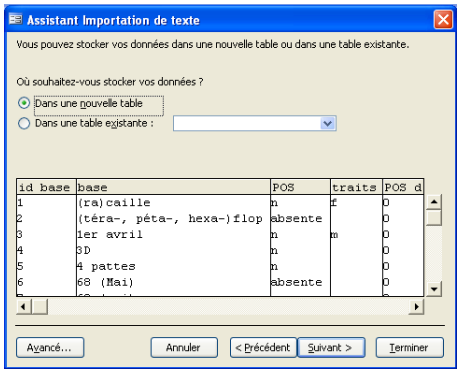
Comme c'est le cas, la case est cochée et Access utilise la première ligne pour nommer chacune des colonnes. Dans le cas contraire, il faut donner un nom à chaque colonne.

5



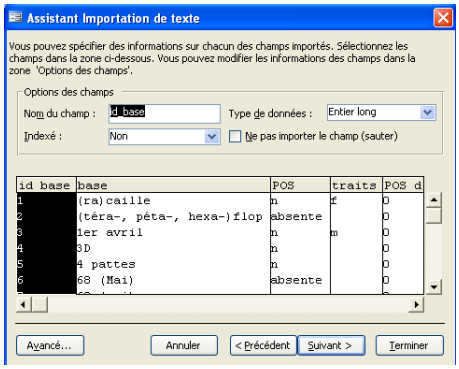
L'import se fait par défaut dans une nouvelle table. Il pourrait également compléter une table existante.

6



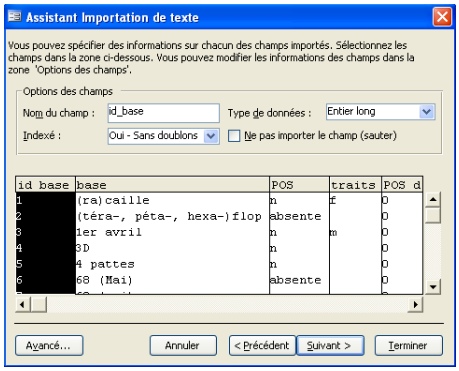
On peut ajuster ensuite les facettes de chaque attribut (les « champs » pour Access).

7



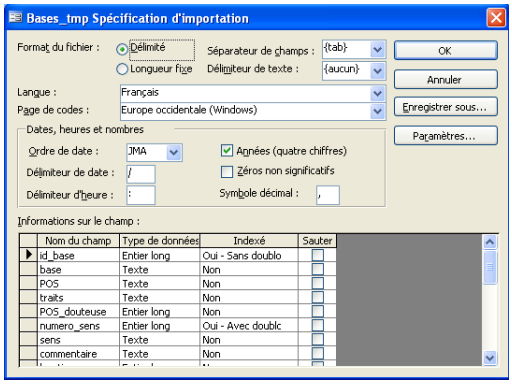
Par rapport à 7, on choisit d'indexer sans doublons l'attribut `id_base`, qui servira d'identifiant, de clé primaire.

8



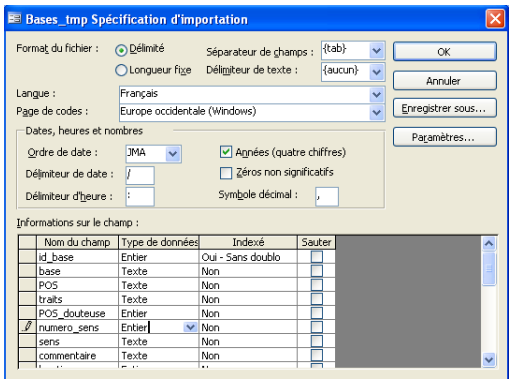
Cliquer sur le bouton [Avancé] permet de voir tous les attributs ensemble, ce qui facilite les réglages. On peut indiquer qu'on ne souhaite pas importer une colonne du fichier de départ. En fonction des valeurs qu'il a rencontrées, Access propose un type pour chaque attribut. On peut le modifier.a

9



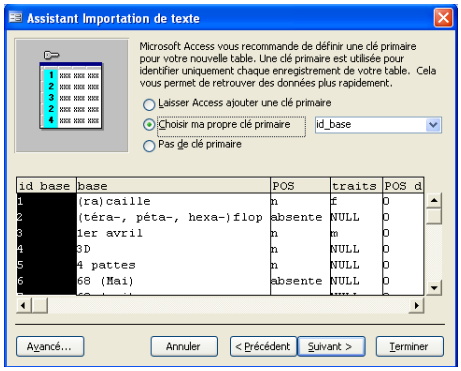
Par rapport à 9, on change les entiers longs en simples entiers dans la mesure où les entiers à mémoriser dans les attributs correspondants ne sont jamais très grands (un peu plus de 3 000 pour `id_base`, par exemple).

10



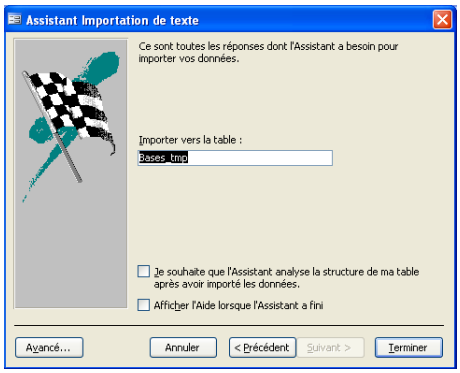
On choisit une clé primaire parmi les attributs existants, ici `id_base`. Ne pas en choisir conduirait à l'apparition d'un attribut auto-incrémenté servant de clé primaire.

11



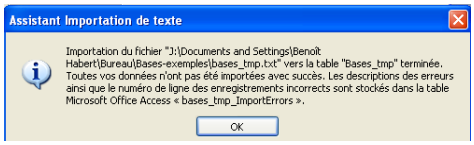
On choisit le nom de la table à créer. Par défaut, c'est celui du fichier utilisé pour l'importation amputé de son extension.

12



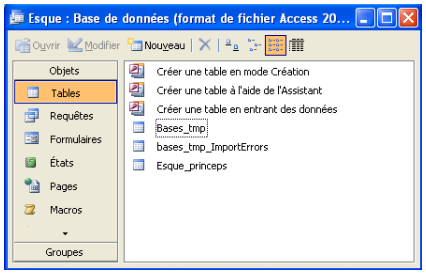
Une fois tous ces réglages faits, Access crée la structure de la table et y transfère les informations du fichier délimité. Access signale les problèmes éventuels. Dans le cas présent, deux des lignes du fichier d'entrée ont une valeur pour l'attribut commentaire qui dépasse les 255 autorisés par le type Texte. La valeur stockée dans la base pour cet attribut est donc tronquée pour ces deux lignes.

13



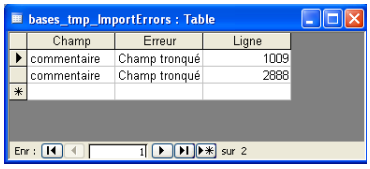
L'onglet [Tables] de la fenêtre de navigation dans la base courante montre les deux nouvelles tables créées : Bases_tmp et Bases_tmp_ImportErrors. Cette dernière table contient des indications sur les erreurs d'importation.

14



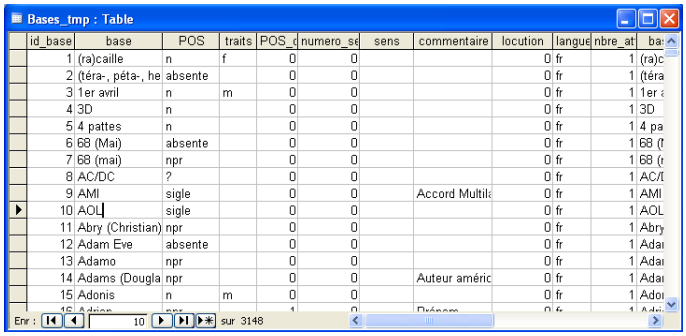
L'affichage de la table Bases_tmp_ImportErrors permet éventuellement d'aller examiner les lignes et les attributs qui ont été « abîmés » et de décider des conclusions à en tirer.

15



La table Bases_tmp donne le point de départ pour établir une liste aussi définitive que possible des mots-bases, assortis des informations nécessaires.

16



L'importation du fichier délimité Esque_tmp.txt s'effectue dans les mêmes conditions. L'attribut datation est omis : son format, américain, n'est pas celui attendu par la version d'Access utilisée. Comme dans le cas précédent, deux valeurs de l'attribut commentaire_base ont été tronquées.

Il est aisé sous MySQL de retrouver les deux commentaires « à l'étroit ». Ce sont forcément les deux plus longs. La requête suivante donne les mots-bases correspondants, la taille de ces deux commentaires et manifeste la partie qui est conservée et celle qui est perdue lors de l'importation :

```
SELECT base,
       LENGTH(commentaire) AS 'car.',
       LEFT(commentaire, 255) AS 'gardé',
       RIGHT(commentaire, LENGTH(commentaire) - 255) AS 'perdu'
FROM bases_tmp
ORDER BY LENGTH(commentaire) DESC
LIMIT 2 ;
```

Son résultat est le suivant :

base	*car.*	*gardé*	*perdu*
shtru	282	Shtru : Courant artistique musical caractérisé par une forte aptitude à chanter vite et haut. Les chanteurs shtrus, dont un grand mystere plane autour de la personnalité, reprennent des célèbres chansons à leur sauce, et dans un langage incompréhensible do	nt eux seuls ont le secret.
NIN	267	Derrière Nine Inch Nails se cache essentiellement une seule personne : Trent Reznor , il cumule les talents au sein de NIN ou il est à la fois compositeur , chanteur, clavier, guitariste, bassiste, producteur (http ://www.modenation.com/html/artiste_du_mom	ent_NIN.htm)

3. Corrections manuelles

3.1. « Consolider » les bases

3.1.1. Identifier les mots-bases, mettre en cohérence leurs facettes

L'objectif premier est l'identification des mots-bases effectifs

Lorsque deux mots-bases A et B sont à fusionner dans la table bases_tmp (il s'agit en fait du même mot-base) :

- les dérivés du mot-base B prennent comme id_base celui du mot A ;
- le mot-base B est enlevé de la table bases_tmp.

Lorsqu'un mot-base A est à séparer en A et B (il y a en fait deux mots-bases différentes) :

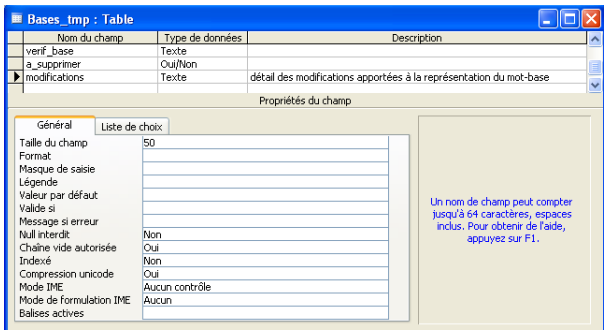
- on crée un mot-base B, qui reçoit automatiquement une valeur pour id_base ;
- on examine tous les dérivés de A et soit on leur laisse l'id_base de A soit on leur donne l'id_base de B.

Il faut, dans un deuxième temps, assurer la mise en cohérence des informations associées à un mot-base

Si ce mot-base correspond à une seule attestation, donc à une seule ligne dans la table esque_tmp, on vérifie les informations correspondant au mot-base dans cette ligne. Dans le cas contraire, on vérifie s'il ne faut pas rassembler des informations présentes dans plusieurs lignes. Ce problème et son traitement sont détaillés pour les dérivés.

3.1.2. Mémoriser les amendements

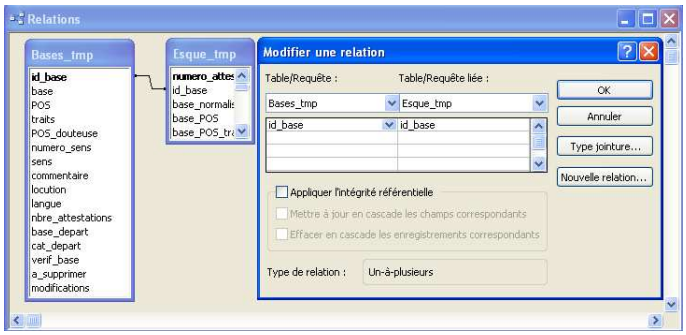
On souhaite par ailleurs garder la trace des amendements réalisés, pour pouvoir les vérifier, avoir une idée de leur nombre, etc. On modifie en conséquence la table Bases_tmp. On lui ajoute deux attributs. Le premier, a_supprimer, booléen, qui par défaut a pour valeur Faux, indique si la ligne est à supprimer (pour cause de fusion de ses informations avec une autre). Le second, modifications, de type Texte, permet de noter le détail des modifications apportées.



3.1.3. Marquage des lignes de esque_tmp dont la base est à changer

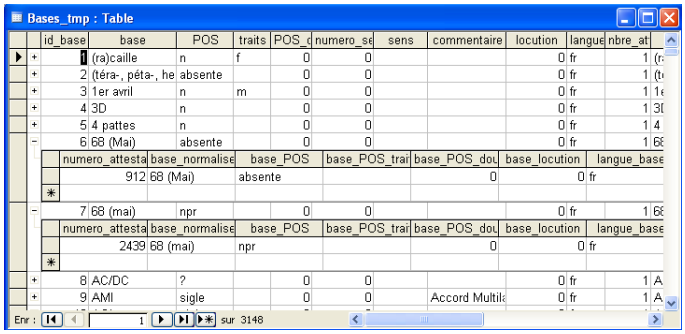
On établit une jointure de type $1-n$ entre les tables Bases_tmp et Esque_tmp, sur l'attribut id_base. À une ligne de la table Bases_tmp, c'est-à-dire à un mot base donné, correspondent une ou plusieurs lignes de la table Esque_tmp, c'est-à-dire une ou plusieurs attestations.

18



On le voit en 19, quand on ouvre en mode feuille de données la table Bases_tmp, une croix sur la gauche de chaque ligne nous indique une jointure avec une autre table. Lorsqu'on clique sur cette croix, on obtient la ou les ligne(s) correspondante(s) de l'autre table.

19

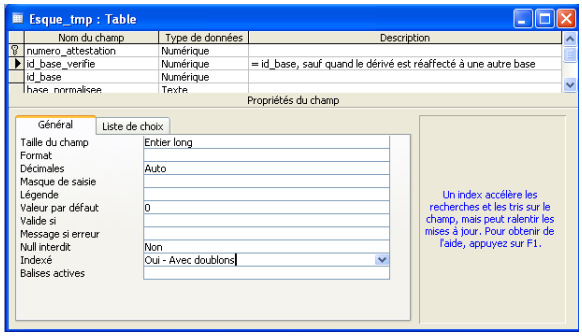


L'exemple de 19 montre bien les problèmes à résoudre. On constate les incohérences entre les lignes d'id_base 6 et 7, concernant tant l'orthographe de *Mai 68* que sa catégorie morpho-syntaxique. L'objectif est donc d'une part de ne garder qu'une base '*68 (Mai)*' et d'autre part de « rapatrier » vers cette base le ou les dérivé(s) qui dépendai(en)t de la base à éliminer.

Supposons qu'on souhaite garder la base d'id_base 6, et ramener à elle le dérivé dépendant de la base d'id_base 7. On ne peut pas modifier dans la ligne de la table Esque_tmp l'attribut id_base puisqu'il est précisément utilisé pour assurer la jointure.

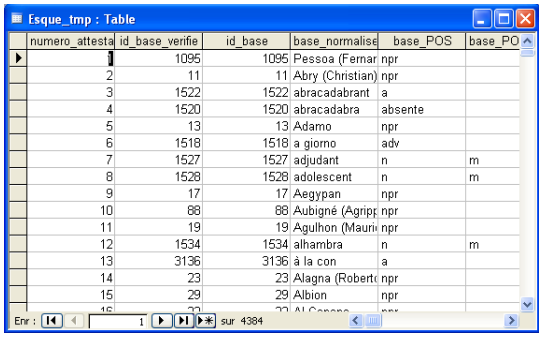
On ajoute un attribut à la table Esque_tmp, id_base_verifie.

20

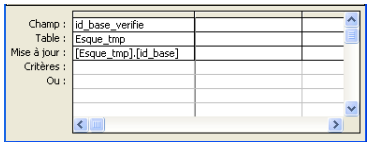


Au départ, il contient la même valeur que l'attribut id_base 21, grâce à une requête Mise à jour 22.

21

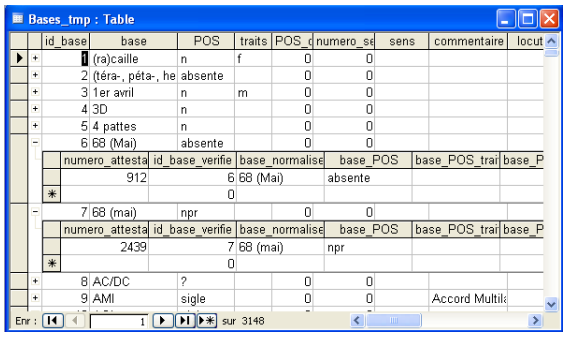


22

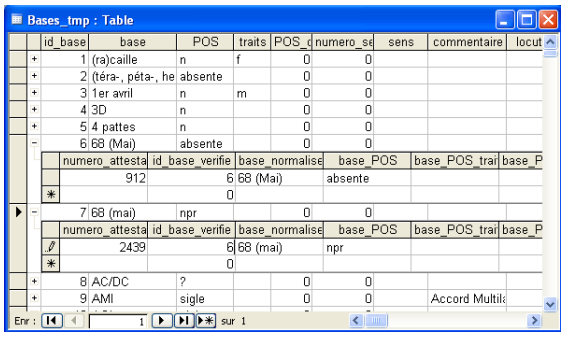


Si l'attestation est réaffectée à une autre base, on lui donne manuellement la valeur de l'identifiant de cette autre base. On observe ainsi, pour l'attestation 2439, la différence entre 23, où id_base_verifie vaut 7 et 24 où l'attribut vaut maintenant 6, ce qui indique que cette attestation est rattachée à la base d'id_base 6.

23



24



3.1.4. Correction et formulaires

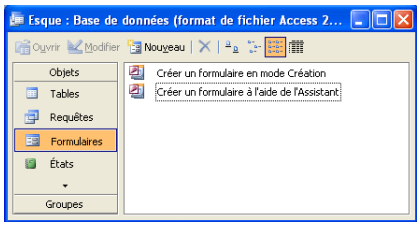
Les formulaires sont sous Access le moyen privilégié d'afficher et de changer les données d'une table, voire de deux tables liées. Un formulaire permet de réunir en un écran des zones de saisie, de les nommer, de les disposer à volonté, de les assortir de contrôles, etc.

La fabrication de formulaires sophistiqués relève des manuels de l'utilisateur d'Access. Elle n'est donc pas détaillée dans ces pages.

Nous allons simplement dans un premier temps montrer rapidement la création assistée d'un formulaire rudimentaire.

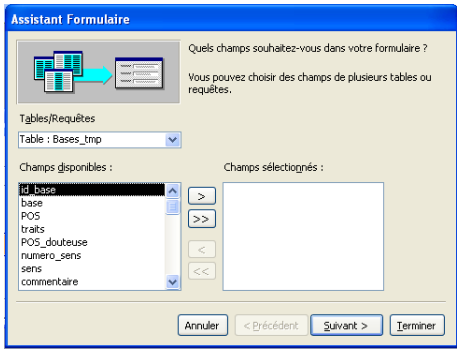
L'onglet [Formulaires] de la fenêtre de navigation dans la base de données est le point d'entrée. On va ici recourir à l'Assistant.

25



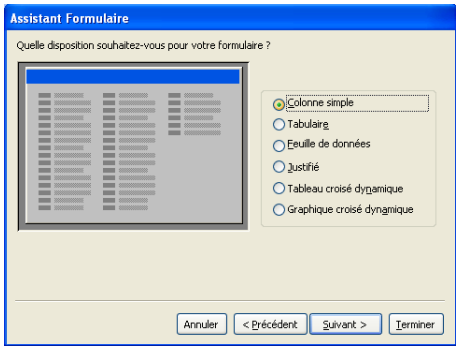
Le premier pas revient à choisir la ou les table(s)/requête(s) qui seront affichée(s) par le formulaire, les attributs qui y figureront et l'ordre dans lequel ils interviendront.

26

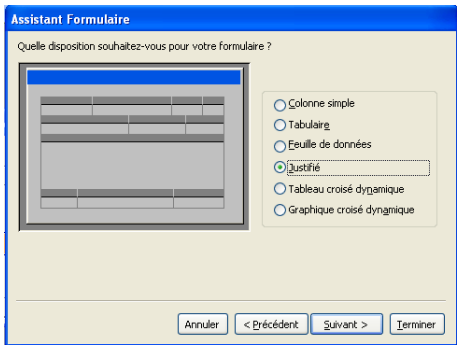


On choisit alors un style de formulaire, en colonne comme en 27, ou justifié comme en 28. À chaque fois la partie gauche de l'écran fournit un aperçu.

27

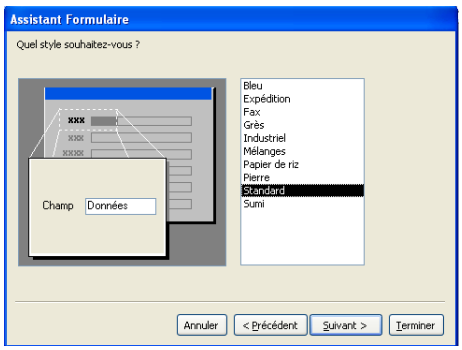


28



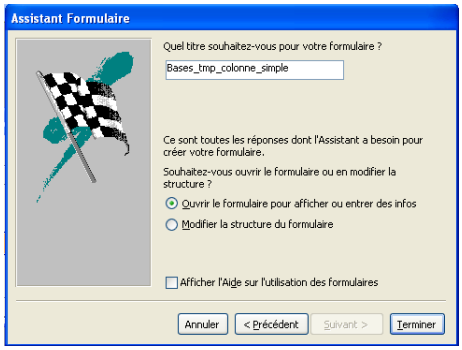
Ce choix fait (en l'occurrence, en colonne), il reste à choisir le type de fond, les caractères des titres, etc.

29



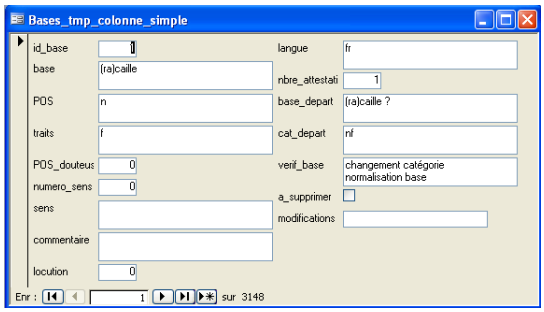
On détermine le nom du formulaire et on décide soit de l'utiliser immédiatement soit de se servir de cette ébauche pour parvenir à une mise en forme plus élaborée.

30



Dans le premier cas, on obtient un formulaire plus agréable que la saisie en mode Feuille de données.

31

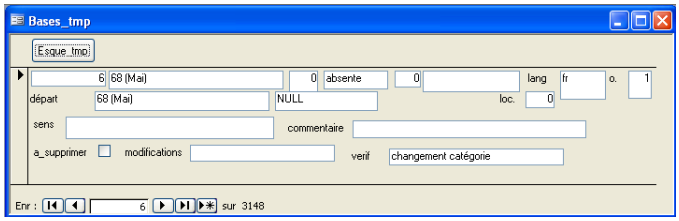


Ce premier résultat montre une idée de l'appui que constituent les formulaires pour séparer l'utilisateur du format concret de la table support. On peut ne pas montrer tous les attributs, changer leur ordre, donner des titres plus clairs que ceux des attributs aux zones de saisie, etc.

3.1.5. Formulaire à sous-formulaire

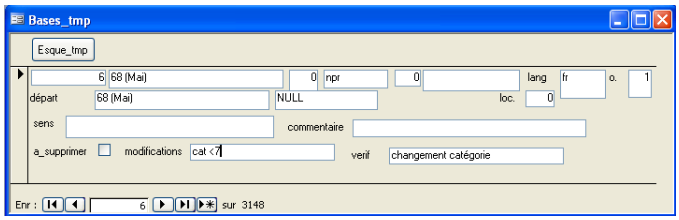
On va dans un deuxième temps faire appel à des formulaires plus complexes qui permettent à la fois de garder le lien entre les deux tables liées et de rendre co-présentes les informations à maîtriser et à modifier. On crée un formulaire à sous-formulaire. Le formulaire principal correspond à la table « maîtresse », du côté 1 de la relation $1-n$, dans le cas présent Bases_tmp. Le sous-formulaire correspond à la table « auxiliaire », du côté n de la relation $1-n$, ici Esque_tmp. L'exemple de *Mai 68* est repris. Le formulaire principal montre la base d'id_base 6 avant toute modification. La première ligne fournit dans l'ordre les attributs suivants : id_base, base, numero_sens, POS, POS_douteuse, traits, locution, nbre_attestations.

32



En 33, on voit plus commodément qu'en mode Feuille de données les modifications effectuées : changement de catégorie, indication que le mot d'id_base 7 a été absorbé (<7).

33



En haut du formulaire principal, on trouve un bouton permettant de « déplier » le formulaire auxiliaire. C'est le cas en 34 qui correspond au dérivé dont le numéro d'id_base a été changé de 7 à 6. L'identifiant vérifié figure après la mention OK. Dans le formulaire principal, on indique que cette base est à supprimer et qu'elle est fusionnée avec celle d'id_base 6 (>6).

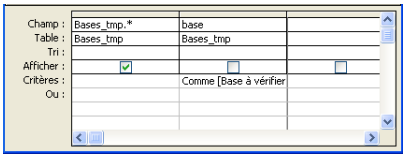
34

3.1.6. Rechercher les doublons séparés

Il est relativement aisé de confronter les informations sur deux mots-bases quand ils se suivent dans l'ordre de la table Bases_tmp, c'est-à-dire par tri sur les attributs base et POS. C'est le cas par exemple avec *sit com* et *sit(-)com*. D'autres doublons sont dispersés : *Le Monde* et *Monde (Le)* ; *SOCRATE* et *Socrate* ; *Led Zeppelin* et *Zeppelin (Led)* ; *legume* et *légume*.

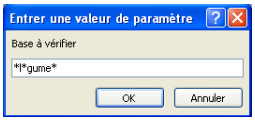
Pour faciliter la recherche de ce cas de figure, on développe une requête paramêtrée. Elle affiche tous les attributs de la table Bases_tmp et attend de l'utilisateur un motif qu'elle confronte avec la valeur de l'attribut base.

35



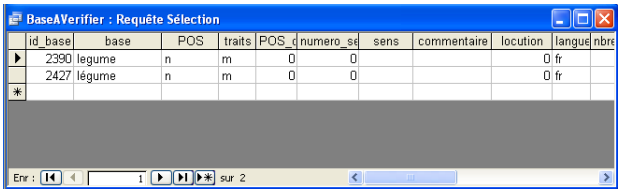
L'interrogation correspond au dernier exemple fourni.

36



La réponse permet effectivement d'isoler un mot-base à faire fusionner avec un autre. Si le mot-base *légume* n'avait pas existé dans la table, on se serait contenté de rectifier la graphie fâcheuse.

37

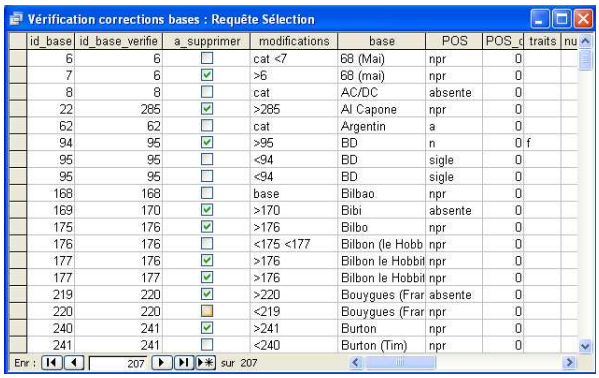


3.1.7. Contrôler les corrections manuelles

La correction manuelle est fastidieuse et sujette aux erreurs. Il faut donc la contrôler. Ainsi, une autre requête conserve uniquement les lignes de la table Bases_tmp qui ont été manuellement changées, c'est-à-dire celles dans lesquelles soit l'attribut a_supprimer a pris la

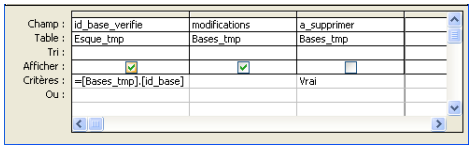
valeur Vrai soit l'attribut modifications ne contient pas la marque **NULL**. Les 207 modifications de 38 correspondent à la fusion de 70 à 80 mots-bases : chaque fusion donne lieu à deux modifications, l'une d'une côté de la ligne supprimée, l'autre du côté de la ligne « augmentée », comme on le voit sur les deux premières lignes en ce qui concerne *Mai 68*. L'examen du résultat met en évidence des incohérences rémanentes. Ainsi l'entrée d'id_base 176 absorbe-t-elle ses voisines 175 et 177. Pour 175, la valeur d'id_base_verifie a bien été mise à jour à 176, ce n'est pas le cas pour 177.

38



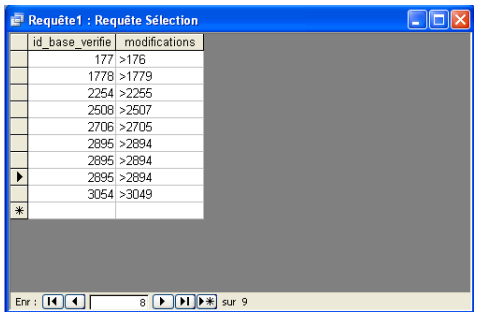
Une autre requête permet d'isoler rapidement les erreurs de ce type.

39



On constate que 9 erreurs de ce genre ont été commises.

40



On rectifie manuellement en ce sens Bases_tmp.

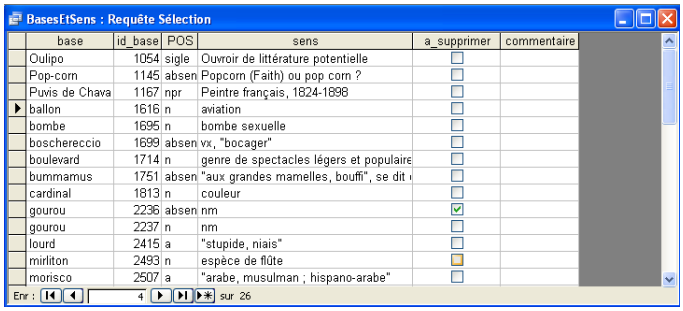
3.1.8. Fusionner les attributs sens et commentaire

Dans la modélisation du ch. IX, on a décidé d'une part de disposer d'un attribut numero_sens pour distinguer les bases homonymes jusqu'à la partie du discours compris et par ailleurs, d'un seul attribut commentaire, destiné à accueillir aussi ce qui figure dans l'attribut sens de la table esque de départ.

Lorsqu'on examine la valeur que prend l'attribut sens dans esque_tmp quand il n'équivaut pas à la marque **NULL**, en 41 et en 41, on constate d'une part que, sauf une exception (*turlupin*), l'attribut commentaire n'a pas de valeur, d'autre part, qu'une des bases est marquée

comme à supprimer et enfin que dans deux cas (*gourou*), le « sens » correspond en fait à des traits morphologiques.

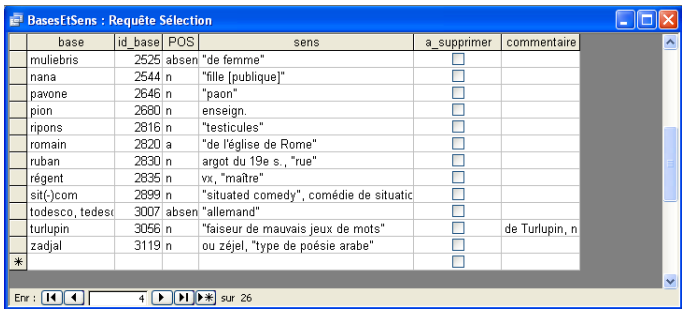
41


BasesEtSens : Requête Sélection

base	id_base	POS	sens	a_supprimer	commentaire
Oulipo	1054	sigle	Ouvroir de littérature potentielle	☐	
Pop-corn	1145	absen	Popcorn (Faith) ou pop corn ?	☐	
Puvis de Chava	1167	npr	Peintre français, 1824-1898	☐	
ballon	1616	n	aviation	☐	
bombe	1695	n	bombe sexuelle	☐	
boschereccio	1699	absen	vx, "bocager"	☐	
boulevard	1714	n	genre de spectacles légers et populaire	☐	
bummamus	1751	absen	"aux grandes mamelles, bouffi", se dit	☐	
cardinal	1813	n	couleur	☐	
gourou	2236	absen	nm	☑	
gourou	2237	n	nm	☐	
lourd	2415	a	"stupide, niais"	☐	
mirliton	2493	n	espèce de flûte	☐	
morisco	2507	a	"arabe, musulman ; hispano-arabe"	☐	

Enr : 4 sur 26

42

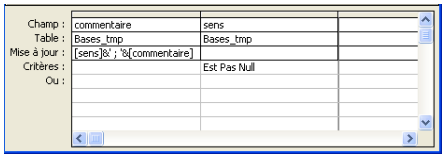

BasesEtSens : Requête Sélection

base	id_base	POS	sens	a_supprimer	commentaire
muliebris	2525	absen	"de femme"	☐	
nana	2544	n	"fille [publique]"	☐	
pavone	2646	n	"paon"	☐	
pion	2680	n	enseign.	☐	
ripons	2816	n	"testicules"	☐	
romain	2820	a	"de l'église de Rome"	☐	
ruban	2830	n	argot du 19e s., "rue"	☐	
régent	2835	n	vx, "maître"	☐	
sit(-)com	2899	n	"situated comedy", comédie de situatic	☐	
todesco, tedesc	3007	absen	"allemand"	☐	
turlupin	3056	n	"faiseur de mauvais jeux de mots"	☐	de Turlupin, n
zadjal	3119	n	ou zéjel, "type de poésie arabe"	☐	

Enr : 4 sur 26

On décide de retirer à la base *gourou* destinée à rester dans les bases consolidées le « sens » nm. Par ailleurs, pour les bases dont le sens n'équivaut pas à la marque **NULL**, on choisit, via une requête Mise à jour, de donner à l'attribut commentaire la valeur de l'attribut sens concaténée à la chaîne ' ; ' et à la valeur précédente de l'attribut commentaire.

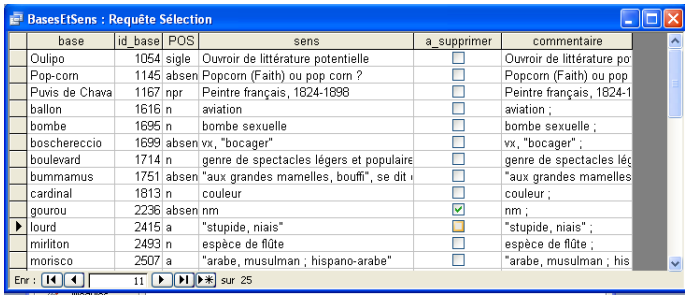
43



Champ :	commentaire	sens	
Table :	Bases_tmp	Bases_tmp	
Mise à jour :	[sens]&' ; '&[commentaire]		
Critères :		Est Pas Null	
Ou :			

Les deux écrans qui suivent montrent la transformation opérée. La concaténation est particulièrement claire pour la ligne *turlupin*.

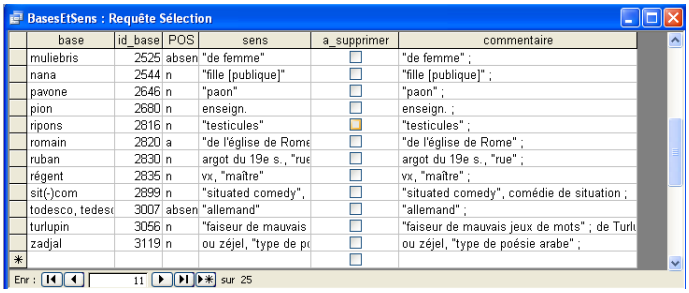
44


BasesEtSens : Requête Sélection

base	id_base	POS	sens	a_supprimer	commentaire
Oulipo	1054	sigle	Ouvroir de littérature potentielle	☐	Ouvroir de littérature po
Pop-corn	1145	absen	Popcorn (Faith) ou pop corn ?	☐	Popcorn (Faith) ou pop
Puvis de Chava	1167	npr	Peintre français, 1824-1898	☐	Peintre français, 1824-1
ballon	1616	n	aviation	☐	aviation ;
bombe	1695	n	bombe sexuelle	☐	bombe sexuelle ;
boschereccio	1699	absen	vx, "bocager"	☐	vx, "bocager" ;
boulevard	1714	n	genre de spectacles légers et populaire	☐	genre de spectacles lé
bummamus	1751	absen	"aux grandes mamelles, bouffi", se dit	☐	"aux grandes mamelles
cardinal	1813	n	couleur	☐	couleur ;
gourou	2236	absen	nm	☑	nm ;
lourd	2415	a	"stupide, niais"	☐	"stupide, niais" ;
mirliton	2493	n	espèce de flûte	☐	espèce de flûte ;
morisco	2507	a	"arabe, musulman ; hispano-arabe"	☐	"arabe, musulman ; his

Enr : 11 sur 25

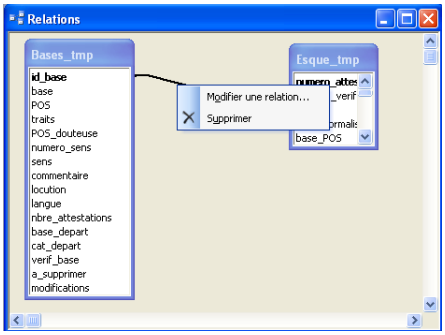
45



3.1.9. Rattacher les attestations aux bases consolidées

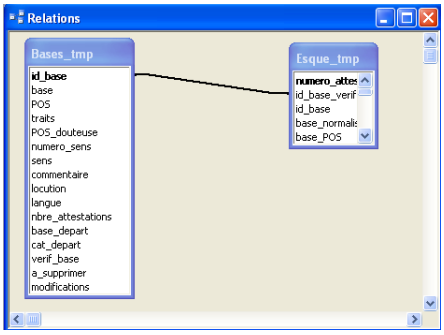
Une fois ces erreurs corrigées ainsi que les autres incohérences dans la correction, on peut « profiter » de cette version « consolidée » des bases. La jointure qui liait les tables Bases_tmp et Esque_tmp, sur l'attribut id_base, n'est plus valide, puisque certaines lignes de Bases_tmp sont destinées à être supprimées (70 sur les 3 148 initiales). Les renvois corrects figurent désormais dans l'attribut id_base_verifie de la table Esque_tmp. Par un clic droit sur l'arête matérialisant la jointure dans la fenêtre des relations (onglet [Outils] de la barre de menu du haut, on peut supprimer la jointure obsolète.

46



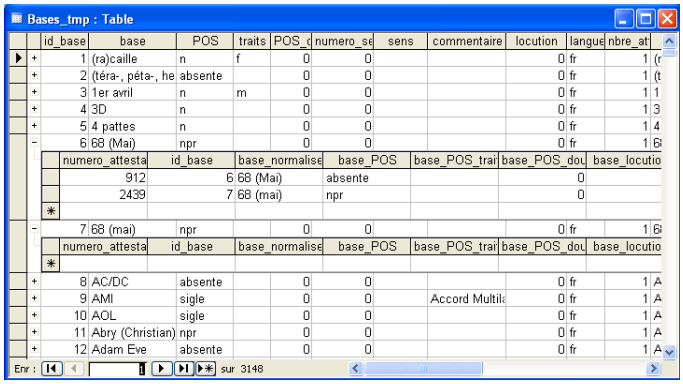
On la remplace par une jointure d'id_base_verifie vers id_base.

47



On constate en examinant la table Bases_Tmp que les dérivés de *Mai 68* relèvent maintenant d'une seule base (l'autre ayant une table vide pour les dérivés associés).

48

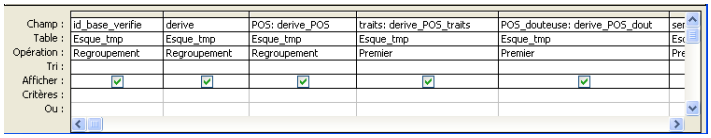


3.2. « Consolider » les dérivés

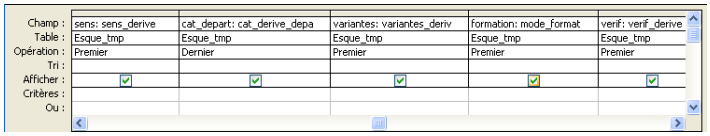
3.2.1. Création d'une table temporaire des dérivés

On peut passer à la « consolidation » des dérivés, sur le modèle de ce qui a été réalisé pour les bases. Il s'agit pour ce faire d'obtenir une table Derives_Tmp. La requête sous-jacente regroupe les lignes de la table Esque_tmp successivement sur id_base, id_derive, derive, et derive_POS. Les autres attributs concernant le dérivé sont rajoutés : derive_POS_traits, derive_POS_douteuse, sens_derive, cat_derive_depart, variantes_derives, mode_formation et verif_derive.

49

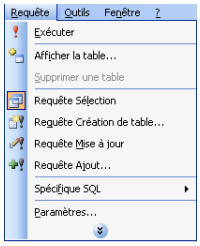


50



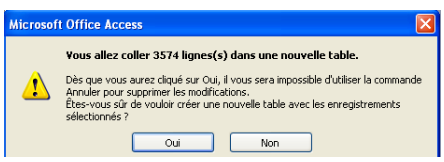
On déclenche une requête Création de table.

51



Après avoir fourni le nom de la table résultante, on reçoit l'avertissement concernant le nombre de lignes qui vont être ajoutées dans la table créée à la volée.

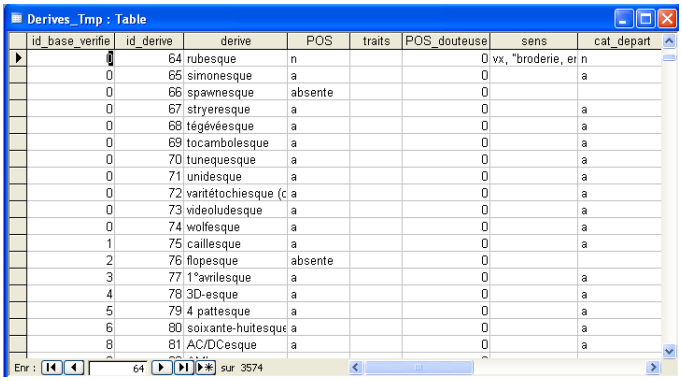
52



On ajoute par ailleurs à cette table d'une part un attribut id_derive, de type NumeroAuto, et on en fait la clé primaire, et d'autre part un attribut commentaire, de type Texte, destiné à recueillir des informations éparses.

L'aperçu de la table résultat permet de se souvenir que lorsqu'un dérivé n'a pas de base, il prend 0 comme valeur d'id_base.

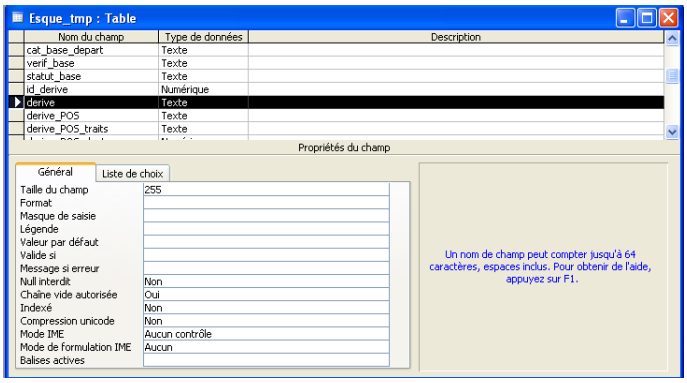
53



3.2.2. Modifications induites pour la table Esque_tmp

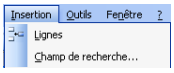
On modifie parallèlement la table Esque_tmp pour ajouter deux attributs id_derive et id_derive_verifie. Via l'onglet [Tables] de la fenêtre de navigation dans la base de données, on choisit [Modifier] et on se positionne à l'endroit où l'on souhaite ajouter une ligne, c'est-à-dire un attribut, dans la structure.

54



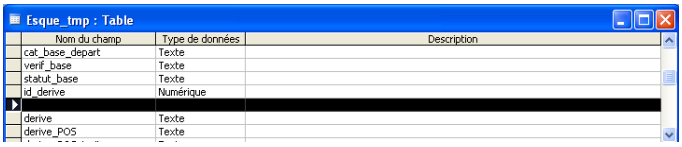
On suit le chemin [Insertion | Lignes] de la barre de menu du haut.

55



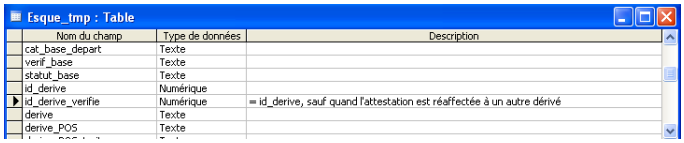
La ligne est insérée.

56



Il reste à nommer cet attribut, le doter d'un type, le documenter. On sauvegarde la structure ainsi modifiée.

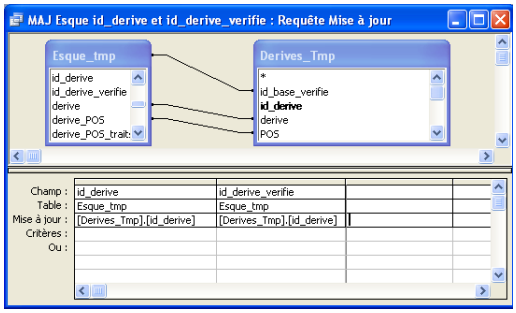
57



Il faut donner donner à ces nouveaux attributs la valeur qu'ils ont dans la table Derives_Tmp.

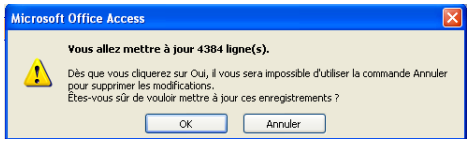
Pour cela, on établit une jointure « temporaire » entre les tables Derives_Tmp et Esque_tmp, sur les attributs id_base_verifie, derive, derive_POS. Cette jointure permet de créer une requête Mise à jour qui donne à id_derive et id_derive_verifie dans la table Esque_tmp la valeur d'id_derive dans la table Derives_tmp.

58



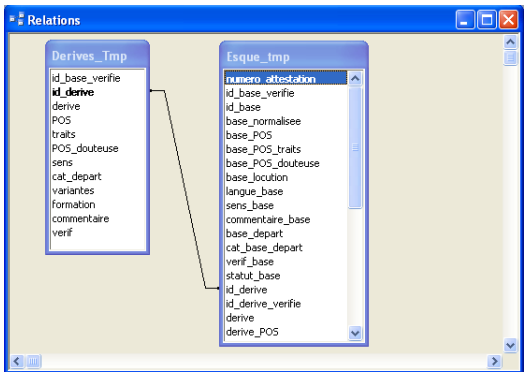
On est averti des modifications qui vont en résulter.

59



Les deux tables partagent maintenant les mêmes identifiants pour les dérivés, l'attribut id_derive étant la clé primaire de la table Derives_Tmp et la clé étrangère de la table Esque_tmp. On établit une jointure de type $1-n$ entre les tables Derives_Tmp et Esque_Tmp sur l'attribut partagé id_derive.

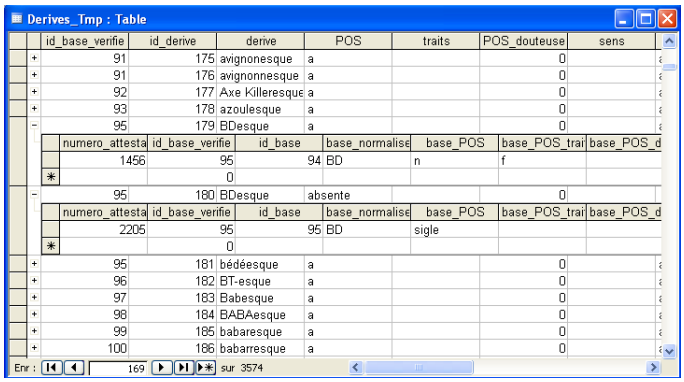
60



3.2.3. Fusion de dérivés et rattachements de lignes de Esque_tmp

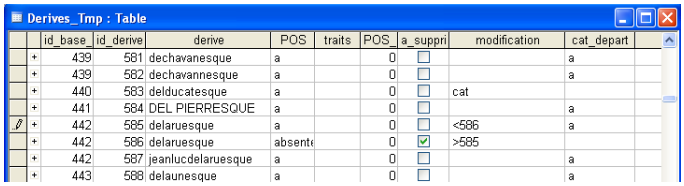
On voit en 61 un exemple de deux dérivés distincts qui sont à fusionner : ils partagent la même base et, sauf erreur, ne diffèrent que par leur catégorie, a(djectif) dans un cas, absente dans l'autre.

61



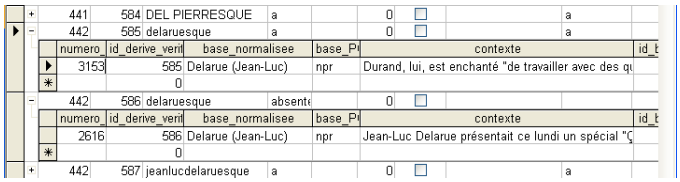
On voit en 62 les deux dérivés *delaruesque*, qui ont même base, mais l'un est étiqueté a(djectif), l'étiquette de l'autre est absente.

62



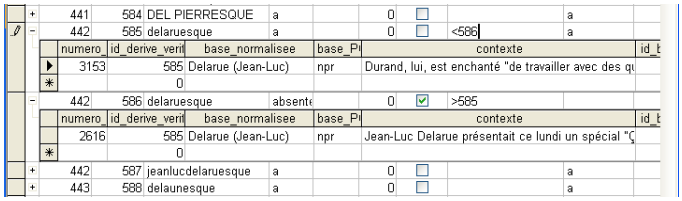
L'ouverture des lignes liées de la table Esque_tmp confirme le constat.

63



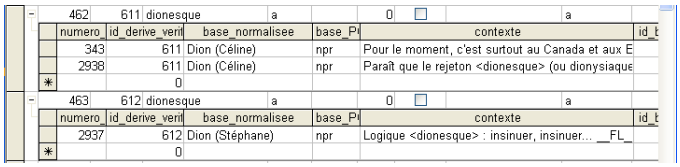
Les corrections sont alors portées. Le deuxième dérivé est marqué comme à supprimer. On indique qu'il fusionne avec le précédent, et qu'inversement ce dernier accueille la ou les attestation(s) qui en dépendai(en)t. L'attestation rattachée au second dérivé est maintenant rattachable au premier, grâce au changement de valeur de l'attribut id_derive_verifie.

64



L'écran 65 correspond par contre à deux bases distinctes.

65



462	611	dionesque	a		0		a

numero_	id_derive_verif	base_normalisee	base_P	contexte	id_t
343	611	Dion (Céline)	npr	Pour le moment, c'est surtout au Canada et aux E	
2938	611	Dion (Céline)	npr	Paraît que le rejeton <dionesque> (ou dionysiaque	
	0				

463	612	dionesque	a		0		a

numero_	id_derive_verif	base_normalisee	base_P	contexte	id_t
2937	612	Dion (Stéphane)	npr	Logique <dionesque> : insinuer, insinuer... __FL_	
	0				

3.2.4. Retours en arrière

L'examen des dérivés met en évidence des incohérences rémanentes au niveau des bases, malgré l'étape de corrections manuelles. Le cas de figure est le suivant : deux dérivés identiques pour le dérivé et sa partie du discours, ainsi que pour le mot-base, diffèrent pour la catégorie du mot-base, parce que, dans un cas, cette dernière est absente.

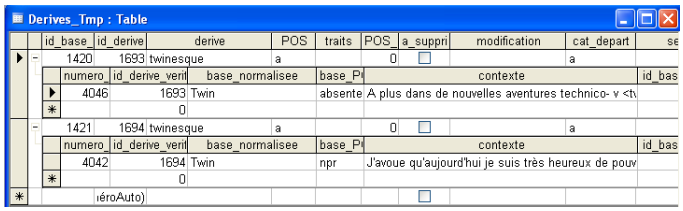
Il faut donc :

- revenir sur la table Bases_tmp pour vérifier si ces mots-bases à partie du discours absente peuvent effectivement être supprimées ;
- pour les attestations des dérivés dont la base peut effectivement être supprimée, les rattacher au dérivé fait sur l'autre base (celle où la partie du discours est présente).

Les écrans suivants illustrent le problème et sa solution.

On constate pour l'adjectif *twinesque* une telle discordance :

66



Derives_Tmp : Table

id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
1420	1693	twinesque	a		0			a	

numero_	id_derive_verif	base_normalisee	base_P	contexte	id_bas
4046	1693	Twin	absente	A plus dans de nouvelles aventures technico- v <t	
	0				

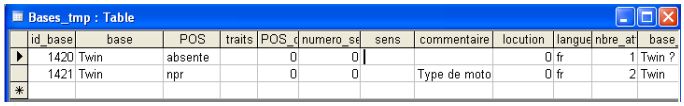
1421	1694	twinesque	a		0		a

numero_	id_derive_verif	base_normalisee	base_P	contexte	id_bas
4042	1694	Twin	npr	J'avoue qu'aujourd'hui je suis très heureux de pouv	
	0				

éroAuto)

L'examen des bases *Twin* manifeste la discordance de catégorie. On marque la base sans catégorie comme à supprimer.

67

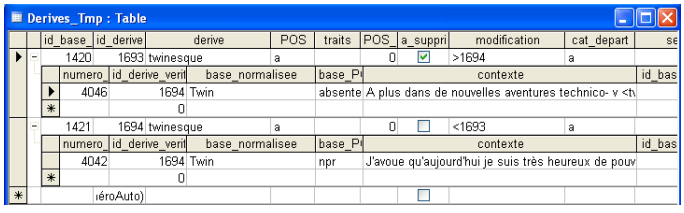


Bases_tmp : Table

id_base	base	POS	traits	POS_d	numero_se	sens	commentaire	locution	langue	nbre_at	base_
1420	Twin	absente		0	0			0	fr	1	Twin ?
1421	Twin	npr		0	0		Type de moto	0	fr	2	Twin

On rattache l'attestation dépendant du dérivé rattaché à cette base à supprimer au dérivé de la base *Twin npr*. On indique que le dérivé rattaché à la base à supprimer est également à supprimer.

68



Derives_Tmp : Table

id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
1420	1693	twinesque	a		0	☑	>1694	a	

numero_	id_derive_verif	base_normalisee	base_P	contexte	id_bas
4046	1694	Twin	absente	A plus dans de nouvelles aventures technico- v <t	
	0				

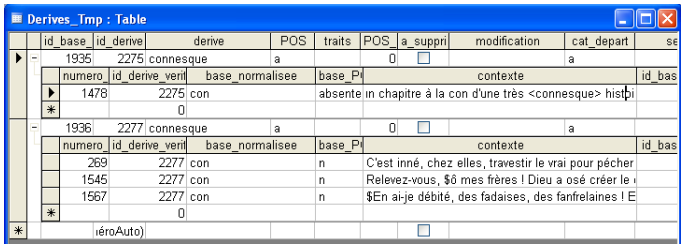
1421	1694	twinesque	a		0		<1693	a	

numero_	id_derive_verif	base_normalisee	base_P	contexte	id_bas
4042	1694	Twin	npr	J'avoue qu'aujourd'hui je suis très heureux de pouv	
	0				

éroAuto)

La table Derives_tmp contient au moins 6 autres exemples de cette situation, qui sont détaillés dans les écrans qui suivent.

69


Derives_Tmp : Table

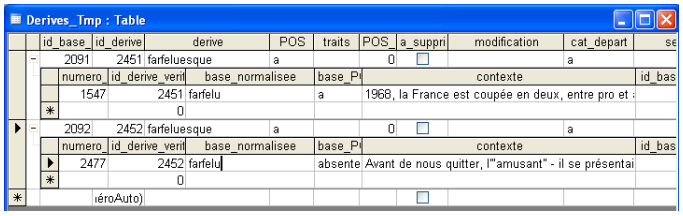
id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
1935	2275	connesque	a		0			a	

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_bas
1478	2275	con	absente	in chapitre à la con d'une très <connesque> histoi	
	0				

id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
1936	2277	connesque	a		0			a	

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_bas
269	2277	con	n	C'est inné, chez elles, travestir le vrai pour pécher	
1545	2277	con	n	Relevez-vous, $ô mes frères ! Dieu a osé créer le	
1567	2277	con	n	$En ai-je débité, des fadaises, des fanfrelaines ! E	
	0				

70


Derives_Tmp : Table

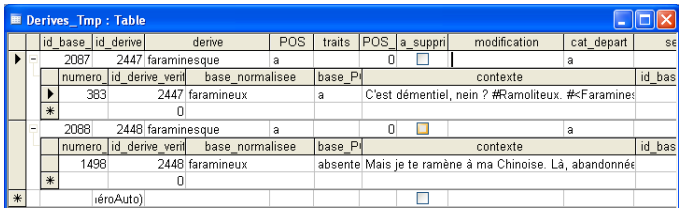
id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
2091	2451	farfeluesque	a		0			a	

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_bas
1547	2451	farfelu	a	1968, la France est coupée en deux, entre pro et	
	0				

id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
2092	2452	farfeluesque	a		0			a	

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_bas
2477	2452	farfelu	absente	Avant de nous quitter, l'"amusant" - il se présentai	
	0				

71


Derives_Tmp : Table

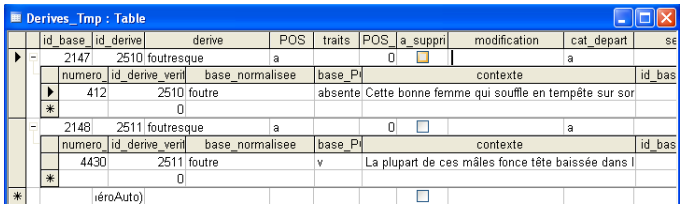
id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
2087	2447	faraminesque	a		0			a	

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_bas
383	2447	faramineux	a	C'est démentiel, nein ? #Ramoliteux. #<Faramines	
	0				

id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
2088	2448	faraminesque	a		0			a	

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_bas
1498	2448	faramineux	absente	Mais je te ramène à ma Chinoise. Là, abandonnée	
	0				

72


Derives_Tmp : Table

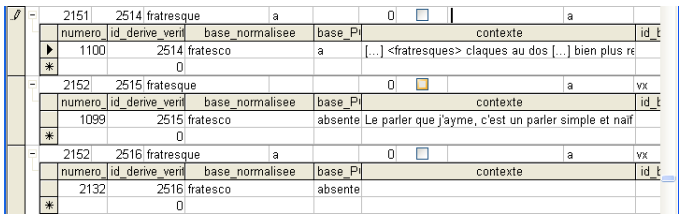
id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
2147	2510	foutresque	a		0			a	

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_bas
412	2510	foutre	absente	Cette bonne femme qui souffle en tempête sur sor	
	0				

id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
2148	2511	foutresque	a		0			a	

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_bas
4430	2511	foutre	v	La plupart de ces mâles fonce tête baissée dans l	
	0				

73



id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
2151	2514	fratresque	a		0			a	

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_b
1100	2514	fratesco	a	[...] <fratresques> claques au dos [...] bien plus re	
	0				

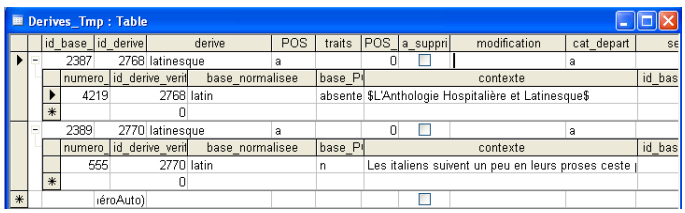
id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
2152	2515	fratesque	a		0			a	vx

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_b
1099	2515	fratesco	absente	Le parler que j'ayme, c'est un parler simple et naif	
	0				

id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
2152	2516	fratresque	a		0			a	vx

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_b
2132	2516	fratesco	absente		
	0				

74


Derives_Tmp : Table

id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
2387	2768	latinesque	a		0			a	

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_bas
4219	2768	latin	absente	$L'Anthologie Hospitalière et Latinesque$	
	0				

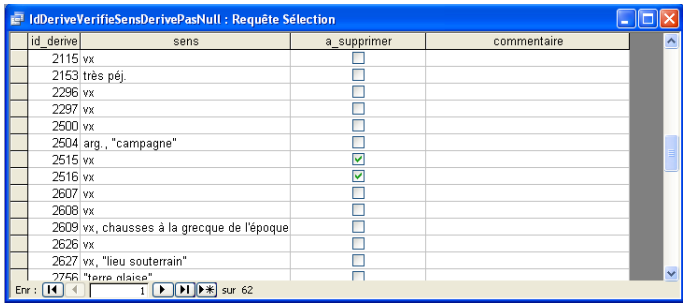
id_base_	id_derive	derive	POS	traits	POS_	a_suppri	modification	cat_depart	se
2389	2770	latinesque	a		0			a	

numero_	id_derive_verit	base_normalisee	base_P	contexte	id_bas
555	2770	latin	n	Les italiens suivent un peu en leurs proses ceste	
	0				

3.2.5. Fusion des attributs sens et commentaire

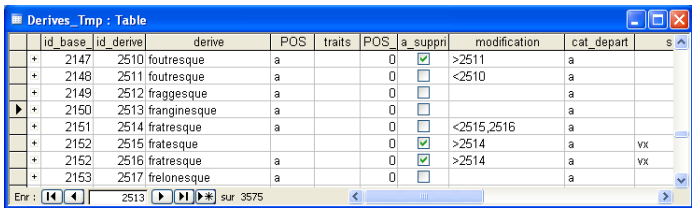
Sur le modèle de ce qui a été fait pour les bases, les informations présentes dans l'attribut sens sont à ajouter à celles présentes dans l'attribut commentaire. On remarque cependant que certains dérivés sont marqués comme à supprimer.

75



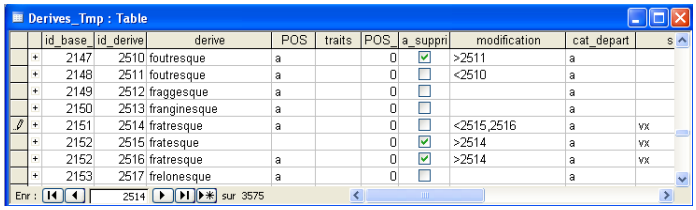
En regardant les lignes correspondantes, on constate que le dérivé qui va être conservé, à la différence de ceux marqués à supprimer dont les attestations lui seront rattachées, ne dispose pas de la marque vx (pour 'vieux').

76



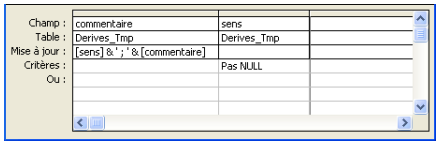
On lui ajoute cette marque.

77



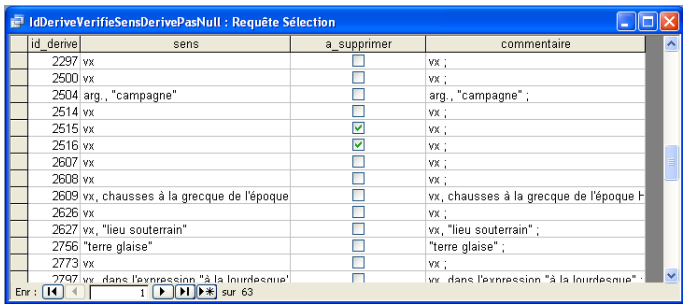
On crée alors une requête Mise à jour qui ajoute avant la valeur de l'attribut commentaire et séparée de lui par la chaîne ' ; ', la valeur de l'attribut sens, quand cet attribut ne porte pas la marque **NULL**.

78



Le résultat est conforme à ce qui était attendu.

79

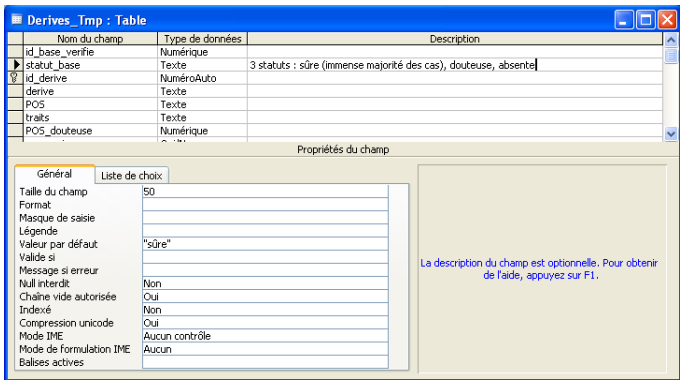


3.2.6. Rajout de l'attribut statut_base à la table Derives_tmp

Dans la table esque_tmp, on dispose de l'attribut statut_base, qui vaut très généralement sûre, mais peut valoir absente ou douteuse, ce qui correspond à un point d'interrogation dans la forme initiale du dérivé dans la table esque.

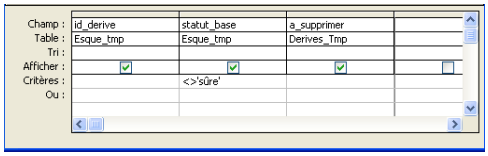
Cet attribut a par mégarde été laissé de côté lors de la création de la table Derives_tmp. Il faut donc l'y rajouter.

80



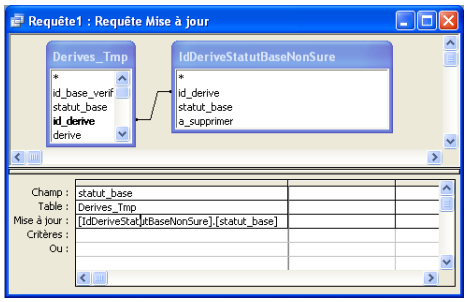
Une requête Création de table permet de faire la correspondance entre id_derive et statut_base quand cet attribut n'a pas pour valeur la chaîne 'sûre', c'est-à-dire quand la valeur était soit 'douteuse' soit 'absente'.

81



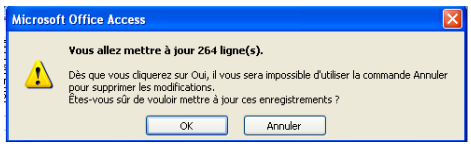
Une requête Mise à jour a pour fonction de donner à l'attribut statut_base de la table Derives_Tmp la valeur de son homologue de la table qui vient d'être créée : IdDeriveStatutBaseNonSure. Elle repose sur la jointure « locale » sur l'attribut partagé id_derive.

82



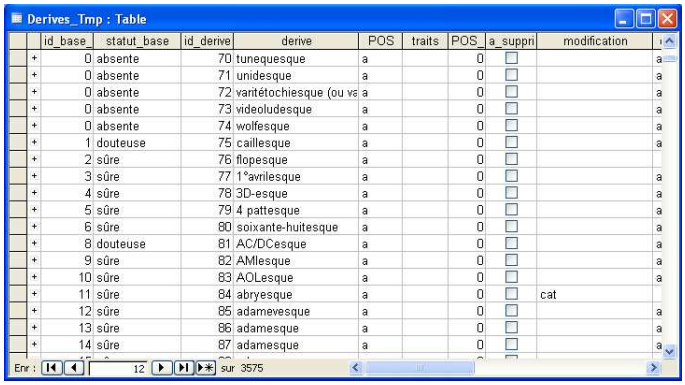
Le déclenchement de la requête entraîne un avertissement.

83



La valeur de l'attribut statut_base est effectivement mis à jour.

84



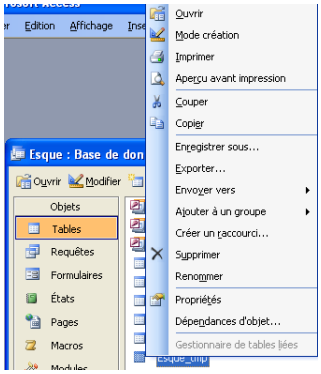
3.3. Examiner les attestations

Il s'agit simplement de donner la mesure des vérifications et transformations à consentir. Nous ne ferons en effet qu'esquisser cette dernière étape, dont la finition demanderait trop de temps.

3.3.1. Création d'une table temporaire des attestations

Sur le modèle suivi pour les bases comme pour les dérivés, on crée une table Attestations_tmp. Une manière de faire est de copier la table Esque_tmp via un clic droit 85 sous un nouveau nom, puis de supprimer les attributs inutiles.

85

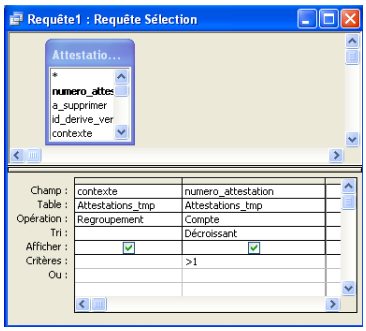


La table résultant comprend alors les attributs suivants hérités de la table Esque_tmp : numero_attestation, id_derive_verifie, contexte, reference, reference_dans_dictionnaire, auteur, reference_complement, jour, mois, annee, annee_complement, origine, verif_contexte. On ajoute deux attributs : modifications, pour noter les modifications apportées à une attestation, a_supprimer, booléen, ayant pour valeur par défaut non.

3.3.2. Attestations doublons

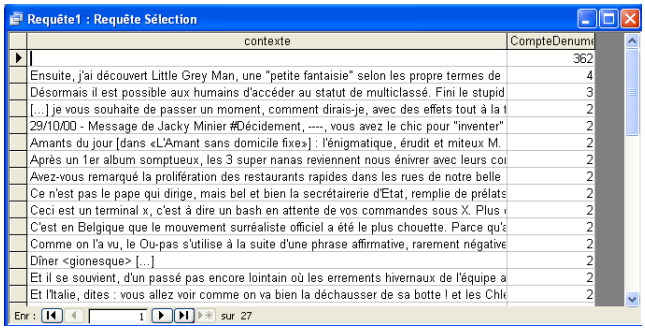
La requête :

86



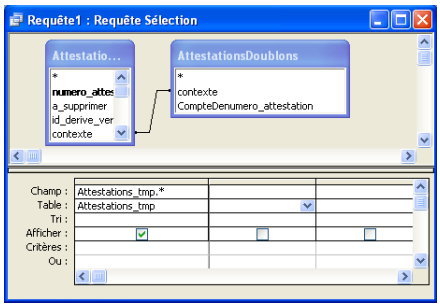
permet de repérer un nombre non négligeable de doublons. En dehors des 362 contextes « vides », on compte 26 contextes présents pour l'essentiel 2 fois. Mais un contexte non vide totalise 4 occurrences...

87



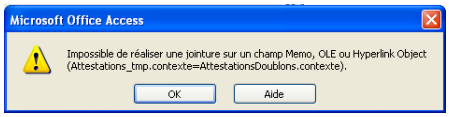
On sauvegarde cette requête pour pouvoir faire une jointure avec la table résultante et examiner en détail les attestations redoublées. La première tentative

88



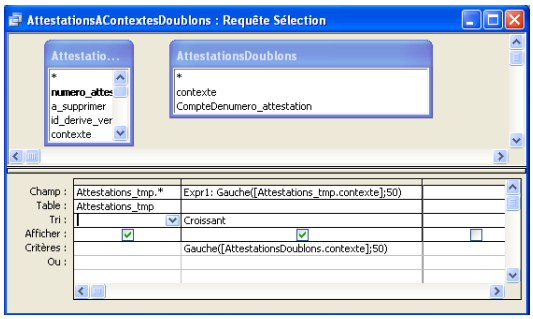
échoue dans la mesure où l'on cherche à effectuer une jointure sur des attributs de type Memo, ce qui n'est pas possible, comme le montre le message d'erreur :

89



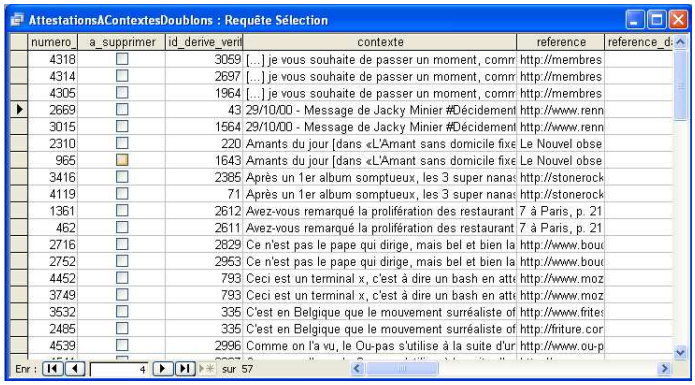
Une approximation est alors de faire une jointure sur l'égalité des 50 premiers caractères de l'attribut `contexte` des deux tables :

90



La feuille de données produite :

91

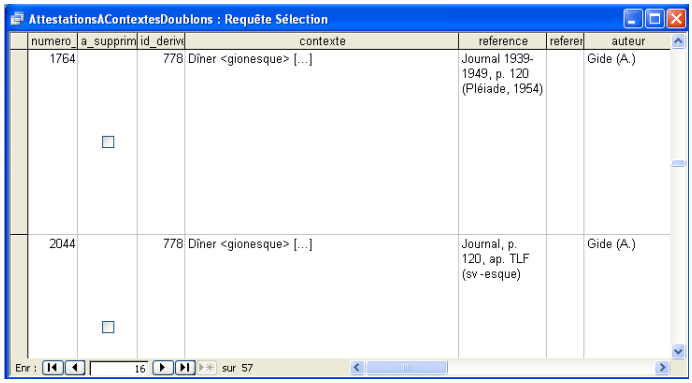


pousse à regarder de plus près les situations. À côté d'attestations partageant un même identifiant de dérivé, on en trouve d'autres dont les dérivés diffèrent.

Lorsque l'identifiant diverge, il s'agit le plus souvent d'une attestation qui conjoint plusieurs dérivés et qui est donc légitimement répertoriée sous chacun d'eux.

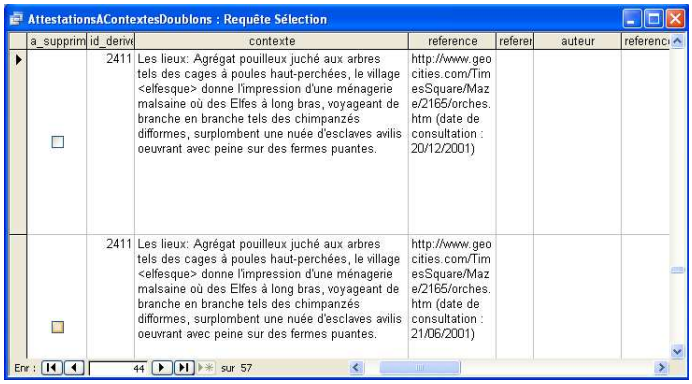
Lorsque deux attestations partagent le même identifiant de dérivé, on se trouve en général face à une répétition indue, comme dans :

92



où il s'agit vraiment d'une même citation, obtenue dans un cas via le TLF. On rencontre également des récoltes effectuées automatiquement sur le Web à quelques mois d'intervalle et qui ne diffèrent que par la date de récolte. C'est le cas des attestations 3026 et 4181, par exemple, ou encore de l'exemple suivant :

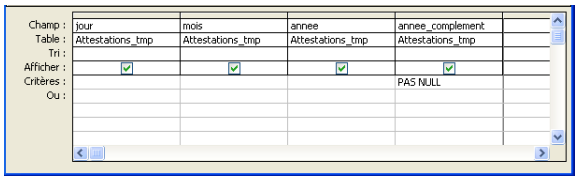
93



3.3.3. Cohérence des datations

Lorsqu'on extrait les attributs jour, mois, annee et annee_complement :

94



on constate que ce dernier attribut, dans les 88 fois où il ne correspond pas à la marque **NULL**, a des valeurs très variées :

av. (60 o. + 14 variantes av) ce qui signifie probablement que l'attestation est antérieure à la date formée par les trois autres attributs (il s'agit d'ailleurs le plus souvent d'un simple millésime, jour et mois ayant alors pour valeur 0, signe conventionnel de non réponse) ;

v. (6 o.) sans doute comme vers, comme marque d'approximation ;

? (2 o.) le doute est peut-être plus fort que dans le cas précédent ;

s. (2 o.) associé à un nombre comme 16 ou 17 comme valeur d'annee renvoie possiblement à siècle (mais on trouve également 17e s. comme valeur d'annee_complement) ;

web enfin (2 o.) sans que le sens visé ici soit clair.

Cet attribut serait à recoder de manière plus cohérente.

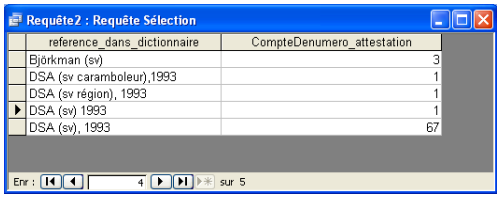
3.3.4. « Origine » des attestations

L'attribut origine est un attribut assez souvent non atomique, comme dans 'Frantext, DDL 35, Björkman 35'. Sans doute faudrait-il remplacer cet attribut non atomique par une relation 1 à n d'une attestation vers les bases des données et usuels dans lesquels elle figure.

3.3.5. Références

L'attribut reference_dans_dictionnaire a deux fonctions seulement comme le montre le résultat d'une requête utilisant des regroupements :

95

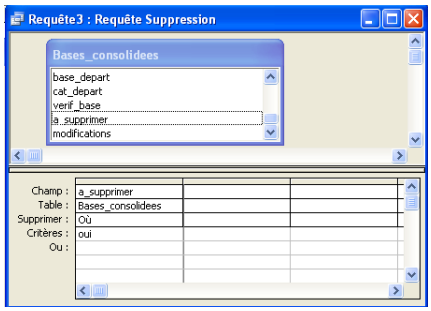


La première fonction est un renvoi à un inventaire (Björkman), la seconde indique que l'attestation figure également dans le DSA (Dictionnaire de San-Antonio). Il serait plus logique de supprimer cet attribut et de redistribuer les rares informations qu'il consigne dans l'attribut origine dûment transformé (cf. supra).

4. Consolidation de la base de données

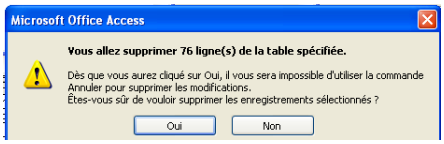
Une première étape consiste à créer une copie de la table Bases_tmp, sous le nom de Bases_consolidees. Une première requête suppression permet d'éliminer toutes les lignes pour lesquelles l'attribut a_supprimer vaut oui :

96



ce qui aboutit à détruire 76 lignes :

97



On modifie alors la structure de cette table pour élaguer les colonnes qui ne sont pas pertinentes : base_depart, cat_depart, verif_base, a_supprimer, modifications.

On opère de la même manière avec une table Derives_consolides, copie au départ de Derives_tmp :

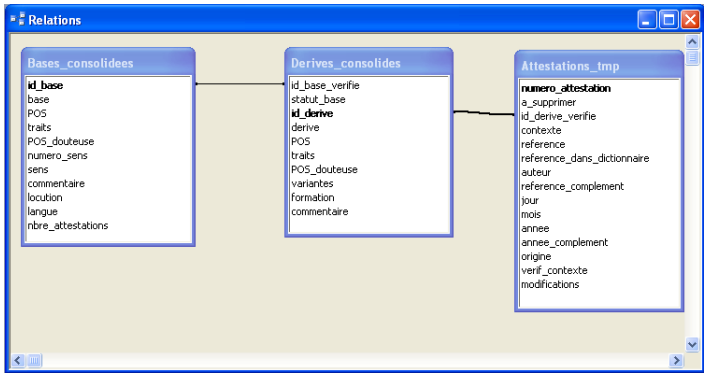
98

99

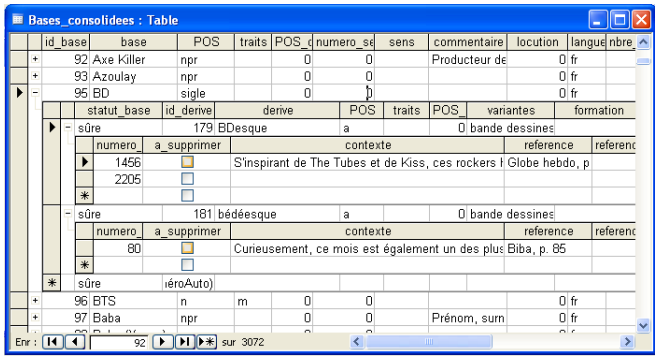
On élague les attributs a_supprimer, modification, cat_depart, sens et verif.

Il ne reste plus qu'à établir les relations pertinentes entre tables :

100



Même si la base de donnée EsqueConsolidee.mdb n'est pas une version entièrement remaniée de Esque.mdb, loin s'en faut, on entrevoit cependant, en 101, le résultat visé. Dans le cas présent, la base *BD* a deux dérivés, *BDesque* et *bédéesque*, qui ont chacun une attestation.



5. Bilan

Une série de constats à l'issue de cette restructuration des données de la table esque pour progresser vers une véritable base de données, articulant des tables consacrées aux différentes informations pertinentes pour le domaine.

En premier lieu, reprendre et réorganiser une base de données est une opération consommatrice en temps et en énergie. Les manipulations de ce chapitre représentent au bas mot deux semaines de travail, en particulier en raison des nombreuses vérifications manuelles nécessaires.

Par ailleurs, les résultats sont à tout le moins hasardeux. Il reste très certainement de nombreuses scories dans les tables de la nouvelle base de données. En outre, à partir du moment où cette restructuration a été effectuée sans consultation des auteurs de la table initiale esque, les incompréhensions et erreurs d'interprétation dans les attributs comme dans leurs valeurs ne sont pas à exclure.

On a constaté, au fil du travail, de mauvaises décisions initiales qui ont compliqué les étapes ultérieures. C'est le cas de la décision de garder ? comme une des catégories possibles des bases ou des dérivés.

Pour finir sur une touche d'humour, autant dire, au vu du coût en temps et de la qualité incertaine de la réorganisation d'une base de données pré-existante, qu'il vaut nettement mieux commencer par analyser soigneusement le problème, en chercher une modélisation correcte en termes d'entités et d'associations et ensuite en fournir une réalisation effective pour un SGBD déterminé. . .

CHAPITRE XIII

⊕ IMPORTER – EXPORTER– REMODELER

Une base de données est un maillon dans une chaîne de traitements possibles. On est donc amené à échanger des informations entre une base de données et d'autres logiciels (éventuellement d'autres SGBD que celui employé pour la base de donnée manipulée). Il est donc nécessaire d'importer des données ou d'en exporter.

Ces échanges entre logiciels supposent un « accord » sur les formats à employer. Par ailleurs, ils nécessitent souvent de remodeler les données pour faciliter leur usage par les logiciels visés.

1. Jeux de caractères

La manipulation de données électroniques continue à réserver de mauvaises surprises, par exemple lorsque des courriels présentent, sans raison apparente, de curieux caractères, comme dans l'exemple suivant : *Question de reprÃ©sentativitÃ© de corpus*. La relecture sous Unix/Linux de la fiche 62 de Préma : « Petit bébé très calme, passif » de la même fiche composée sous MacOS donnerait : « Petit b♯b♯ tr♯s calme, passif » (le caractère ♯ symbolise un caractère non visualisable sous Unix/Linux). On constate que les caractères accentués ne sont pas bien représentés. De tels décalages s'expliquent par un désaccord sur le codage de l'information. Unix/Linux et MacOS n'utilisent pas exactement le même **jeu de caractères**. Les lettres non accentuées sont les mêmes, mais les lettres accentuées et la signalisation d'un passage à la ligne diffèrent. La représentation HMTL de la même fiche est : « Petit b&eacute ;b&eacute ; tr&egrave ;s calme, passif ». Les caractères accentués sont représentés par des noms symboliques ou **entités** (`egrave` pour un e avec un accent grave, eacute pour un e avec un accent aigu), noms repérables à leur encadrement par l'esperluète (`&`) et le point-virgule (`;`). La situation était naguère encore plus délicate quand il s'agissait de faire coexister dans un même document des systèmes d'écriture différents (langue morte comme le grec, Alphabet Phonétique International – API, etc.). Il fallait recourir à des polices spécifiques et liées à une plateforme

donnée, ce qui fragilisait les exports et les partages. L'évolution des jeux de caractères change la donne.

Au niveau le plus élémentaire en effet, des données sont une suite de caractères. Ces caractères sont eux-mêmes codés par une suite de 0 ou 1, ou **bits** (*binary digit*). Une suite de 7 bits permet de coder 2^7 caractères, soit 128 caractères, une suite de 8 bits (un **octet**) 2^8 256 caractères et ainsi de suite. L'entrée dans l'ère numérique d'un nombre croissant de langues, vivantes ou mortes, a conduit à augmenter progressivement le nombre de caractères à encoder, et partant, la taille nécessaire à cet encodage.

L'origine anglo-saxonne de l'informatique – l'absence de caractères accentués en anglais – a amené à utiliser au départ un ensemble de caractères limité (128 positions – les 32 premiers caractères sont des caractères de contrôle, non imprimables) : l'ASCII (*American Standard Code for Information Interchange*), devenu en 1963 la norme ISO 646. ISO (*International Organization for Standardization*) est l'institution qui fixe des normes internationales pour faciliter les échanges intellectuels, scientifiques, techniques et économiques entre pays. Il regroupe les organismes de normalisation nationaux (pour la France, l'AFNOR, *Association Française de Normalisation*).

L'étape suivante a été le développement d'un ensemble de normes sur 8 bits (256 positions) pour rendre compte des différentes langues européennes. Pour l'Europe de l'Ouest, il s'agit d'ISO-LATIN-1 (ou ISO-8959-1). Pour rester compatible avec l'ASCII, les 128 premiers caractères d'ISO-LATIN-1 sont identiques à ceux de l'ASCII (qui en est donc un sous-ensemble). C'est le jeu de caractères par défaut pour Windows. Œ, œ et Ÿ n'y figurent pas, si bien qu'ils n'occupent pas la même position sous Unix/Linux et sous Windows.

L'étape actuelle, depuis 1991, est celle d'Unicode. Cette nouvelle norme vise à disposer d'un encodage unique couvrant toutes les langues et tous les systèmes d'écriture, anciens ou modernes (l'Alphabet Phonétique International est intégré à Unicode). Unicode est devenu l'encodage interne de nombreux langages de programmation (Java, Perl, Python) ainsi que l'encodage par défaut des documents XML. La version de 2003 (4.0) d'Unicode représente 96 382 caractères différents (dans un espace permettant d'en recevoir 1 114 112). Unicode sépare nettement la notion de **caractère**, considéré comme la plus petite unité ou composante signifiante du langage écrit (lettres, mais aussi signes de ponctuation, symboles techniques, unités de mesure, radicaux d'idéogrammes, etc.) et le **glyphe**, c'est-à-dire la représentation visuelle du caractère. Un caractère peut correspondre à plusieurs glyphes, comme pour certains caractères arabes dont la forme diffère selon la position, initiale, médiane ou finale dans le mot. Un glyphe inversement peut être partagé par des caractères différents (le glyphe ' est utilisé à la fois pour l'apostrophe et pour le guillemet simple droit) ou correspondre à plusieurs caractères, comme dans le cas des ligatures. C'est le cas aussi des émoticônes – *smileys* – dans les courriels où une combinaison de caractères, par exemple :-) ou :), donne naissance à un mini-visage souriant ou perplexe, comme :

A chaque caractère est associé un certain nombre de propriétés, par exemple la directionalité (de droite à gauche en hébreu, de haut en bas en japonais), les combinaisons, etc. Ces propriétés permettent aux programmes utilisant Unicode de trier, rechercher, substituer, etc. Différentes formes d'encodage existent pour stocker Unicode : UTF-32, UTF-16 et UTF-8. En UTF-8, le code d'un caractère peut occuper un nombre variable d'octets.

La généralisation d'Unicode va diminuer progressivement les mauvaises surprises mentionnées en début de section. Elle est fondamentale pour la gestion harmonieuse du multilinguisme [1]. Il reste nécessaire en attendant de pouvoir transcoder d'un jeu de caractères plus ancien vers un autre, en particulier Unicode.

2. Importer des fichiers « délimités »

Résultat de près d'un demi-siècle de saisies, corrections et transformations, la base textuelle Frantext rassemble près de 4 000 textes complets, étiquetés morpho-syntaxiquement (mais non lemmatisés) pour plus de la moitié. Au total, près de 220 millions de mots (près de 130 millions de mots étiquetés), un millier d'auteurs. Cette base a au départ été construite pour permettre l'écriture des 17 volumes du *Trésor de la langue française (*TLF), désormais informatisé : TLFI. La langue représentée est surtout littéraire (80% d'œuvres littéraires, 20% d'œuvres techniques), du $16^{ème}$ siècle au $20^{ème}$.

La base Lexique, disponible en ligne, a été conçue au départ pour venir en appui à des expériences en psycholinguistique. Elle fournit ainsi des fréquences des mots et des lemmes à partir de plusieurs corpus pour pouvoir examiner la corrélation entre ces fréquences et des tâches de décision lexicale : un sujet voit des chaînes de caractères présentées à l'écran et il doit décider le plus rapidement possible pour chaque chaîne si elle relève des mots ou si elle ressortit aux non-mots (comme *reveneure* dans *Il était reveneure ; les slictueux toves / Sur l'allouinde gyraient et vriblaient* du poème *Jabberwocky* dans *De l'autre côté du miroir*). Le temps de réaction est enregistré à chaque fois. On peut alors regarder la corrélation entre les temps de réaction et les fréquences dont on dispose. L'hypothèse est que plus un mot est fréquent, plus sa reconnaissance comme un mot est rapide, voire que les fréquences à l'oral sont plus pertinentes que les fréquences pour l'écrit. Lexique comprend les fréquences fournies par l'ATILF sur une partie récente de Frantext (1950–2000), soit 31 millions de mots, ainsi que les fréquences issues des sous-titres de 3 000 films ou épisodes de séries télévisées. Le corpus de sous-titres a été rassemblé pour donner une estimation de fréquences à l'oral, tandis que Frantext visait l'écrit. Lexique fournit 130 000 formes relevant de 55 000 lemmes. Ressource libre, Lexique est déchargeable, assorti d'outils de manipulation, mais également consultable en ligne.

On va utiliser un des éléments de la base Lexique, le fichier FreqFrant.txt, pour confronter les dérivés présents dans la table esque avec les mots en *-esque* repertoriés dans une base textuelle volumineuse correspondant à une période récente.

2.1. Format délimité

Le fichier FreqFrant.txt présente les 246 127 mots de la partie utilisée de Frantext. Ci-dessous un extrait de 10 lignes dont la première contient un mot en *esque* :

alibabesque 1 0.03

alibert 17 0.54

alibi 120 3.82

alibis 24 0.76

alibise 1 0.03

aliboron 2 0.06

[1] M. Jacobson, 2004, « Corpus oraux en linguistique de terrain », *TAL*, vol. 45, n°2, p. 63–88.

alibour 2 0.06

alicante 10 0.32

alice 316 10.07

alicia 11 0.35

Les trois colonnes perceptibles contiennent les informations suivantes :

1. les mots présents dans Frantext ;
2. leur fréquence brute dans Frantext (textes de Frantext publié après 1950 interrogés en avril 2000 ce qui constitue un corpus de 31,39 millions de mots) ;
3. leur fréquence par million d'occurences (fréquence brute divisée par 31,39).

Si l'on visualise le même extrait sous Word en demandant la visualisation des caractères invisibles, on obtient un résultat proche de :

alibabesque→1→0.03¶

alibert→17→0.54¶

alibi→120→3.82¶

alibis→24→0.76¶

alibise→1→0.03¶

aliboron→2→0.06¶

alibour→2→0.06¶

alicante→10→0.32¶

alice→316→10.07¶

alicia→11→0.35¶

La flèche vers la droite symbolise un caractère qui autrement ne se distingue pas visuellement de l'espace, la tabulation. La marque ¶ symbolise le changement de ligne. △

Pour des raisons historiques, le changement de ligne n'est pas réalisé de la même manière selon les systèmes d'exploitation. Il utilise ou combine les deux caractères qui correspondent à deux opérations distinctes sur les machines à écrire mécaniques :

1. *newline*, abrégé en \n, l'obtention d'une nouvelle ligne par mouvement du rouleau sur lui-même ;
2. *carriage-return*, abrégé en \r, ou retour-chariot, le retour du chariot vers la gauche.

La situation est la suivante :

Système d'exploitation	*marquage du changement de ligne*
Linux/Unix	\n
Mac OS	\r
Windows	\r\n

Word « montre » d'ailleurs cette marque ¶ et en même temps l'interprète, c'est-à-dire effectue ce que cette marque symbolise : un changement de ligne. En réalité, dans le fichier correspondant, les caractères sont à la queue-le-leu, les uns derrière les autres :

. . .alibabesque→1→0.03\r\n alibert→17→0.54\r\n alibi→120→3.82\r\n

alibis→24→0.76\r\n alibise→1→0.03\r\n aliboron→2→0.06\r\n

alibour→2→0.06\r\n alicante→10→0.32\r\n alice→316→10.07\r\n

alicia→11→0.35\r\n...

Le fichier en question relève de ce qui est appelé un **format délimité**, ce qui abrège format délimité par des caractères spécifiques. Cela signifie que les colonnes sont séparées par un caractère donné, qui ne peut donc figurer dans les valeurs des colonnes. Le caractère tabulation est un bon candidat comme délimiteur, puisqu'il n'a aucune raison a priori de figurer dans les valeurs d'une colonne. Les lignes sont également séparées, en général par le ou les caractère(s) qui indiquent un changement de ligne.

△

On rencontre parfois la mention *fichier csv* pour *comma separated values*, c'est-à-dire pour un fichier dont les valeurs sont séparées par des virgules. Les fichiers csv étaient souvent employés pour des données comptables dans lesquelles les valeurs numériques ne comportaient pas de virgule mais employaient le point comme séparateur décimal. Par extension, et abus de langage, l'expression *format csv* est parfois employée comme synonyme de *format délimité*, alors qu'elle en constitue une spécialisation (un hyponyme).

2.2. Importation

Pour importer un fichier délimité, on commence par créer la table destinée à accueillir ces informations. Elle doit comporter autant de colonnes que le fichier délimité, les types de ces colonnes doivent être compatibles avec ceux du fichier. Dans l'exemple choisi, on crée par exemple la table suivante :

```
CREATE TABLE FrantextLexique (
      graphie VARCHAR(100) BINARY NOT NULL DEFAULT "",
      frequence INT NOT NULL DEFAULT 0,
      proportion DECIMAL(4,2) NOT NULL DEFAULT 0,
      PRIMARY KEY (graphie)) ;
```

Une instruction spécifique lit le fichier délimité ligne à ligne et crée à chaque fois une nouvelle ligne dans la table, dont les valeurs sont celles de la ligne correspondante du fichier délimité. Elle est de la forme :

```
LOAD DATA LOCAL INFILE
"<fichier␣délimité>"
INTO TABLE <table d'accueil>␣;
```

Par défaut, les colonnes sont délimitées par la tabulation, les lignes par le caractère \n et la contre-oblique (\) permet d'échapper les caractères spéciaux.

Pour l'exemple choisi, on doit indiquer que le changement de ligne est celui de Windows, la suite du caractère \r et du caractère \r, d'où l'ajout de la mention **LINES**... :

```
LOAD DATA LOCAL INFILE
"/home/habert/FrantextAvr.txt"
INTO TABLE FrantextLexique
LINES TERMINATED BY '\r\n' ;
```

On obtient un message sur le nombre de lignes lues (Records), de lignes créées (rows affected) et d'avertissements (warnings) éventuels quand la ligne à intégrer pose des problèmes (incompatibilités de type, « débordements » de taille). Dans le cas présent, 104 avertissements ont été émis, et 3 lignes problématiques ont été laissées de côté (skipped) :

Query OK, 246125 rows affected, 104 warnings (6.12 sec) Records : 246128 Deleted : 0 Skipped : 3 Warnings : 104

TABLEAU 135 – ESQUE : intrus dans les mots en *-esque* de Frantext

graphie	*frequence*	*proportion*
aveclesquels	1	0.03
desquelles	360	11.47
desquellesles	1	0.03
desquels	431	13.73
desquelz	1	0.03
desqueyroux	2	0.06
englesqueville	1	0.03
esquerre	1	0.03
esquevin	2	0.06
lesqueles	1	0.03
lesquell'	1	0.03
lesquelle	1	0.03
lesquelles	1908	60.78
lesquels	1970	62.76
lesquelss'	1	0.03
lesquereux	2	0.06
romanesquerie	1	0.03
*esquerre	1	0.03
*esquevin	2	0.06
*lesquereux	2	0.06

2.3. Élagage de la table des lemmes de Frantext

On peut dans un premier temps détruire toutes les lignes qui ne sauraient être des occurrences d'un mot en *-esque*, c'est-à-dire qui ne répondent pas au motif 'esque' :

```
DELETE FROM FrantextLexique
WHERE graphie NOT REGEXP 'esque' ;
```

Un premier examen des 192 lignes restant dans la table FrantextLexique montre qu'il reste des instrus et que l'expression régulière ayant servi à élaguer était trop laxiste. Les mots qui commencent avec une majuscule commencent avec "*". Les chaînes commençant par "%" indiquent les symboles.

On repère par ailleurs d'une part des adverbes formés sur des adjectifs en *-esque*, d'autre part que cette liste est une liste d'occurrences et non de lemmes : figurent les singuliers et les pluriels. On isole d'abord les lignes dont les graphies ne se terminent pas par esque, esques ou esquement :

```
SELECT *
FROM FrantextLexique
WHERE graphie NOT REGEXP 'esques?$'
    AND graphie NOT REGEXP 'esquement$';
```

Le résultat figure dans le tableau 135 p. 442. On constate qu'il s'agit bien d'intrus à éliminer, d'où la requête :

```
DELETE FROM FrantextLexique
WHERE graphie NOT REGEXP 'esques?$'
    AND graphie NOT REGEXP 'esquement$';
```

L'examen des 172 lignes restants fait apparaître des formes douteuses. On vérifie leur présence dans la table esque :

```
SELECT *
FROM esque
WHERE derive IN ('besque', 'esques', 'fiesque', 'fresque', 'fresques', 'kesque', '
    nesque', 'perplesque', 'poesque', 'presque', 'quesques', 'vesque', '*vesque', '
    lebesque', 'levesque', 'lévesque') ;
```

Seule *poesque*, formé sur Edgar Poe, apparaît. On supprime donc les autres :

```
DELETE FROM FrantextLexique
WHERE graphie IN ('besque', 'esques', 'fiesque', 'fresque', 'fresques', 'kesque', '
    nesque', 'perplesque', 'presque', 'quesques', 'vesque', '*vesque', 'lebesque', '
    levesque', 'lévesque') ;
```


2.4. Rapprochement avec la table esque

La table esque contenant des lemmes de dérivés, et non des flexions, si l'on veut mettre les deux tables en relation via une jointure, il faut lemmatiser la table FrantextLexique. C'est le but de la requête :

```
SELECT
     CASE
          WHEN graphie REGEXP 'esques$'
               THEN SUBSTRING(graphie, 1, LENGTH(graphie) - 1)
          WHEN graphie LIKE '*%' THEN REPLACE(graphie, '*', '')
          ELSE graphie
      END AS lemme,
      graphie,
      frequence,
      proportion
FROM FrantextLexique  ;
```

Quand la graphie s'achève par *esques*, on ne garde que les caractères jusqu'avant le *s* final (premier **WHEN**), lorsqu'il y a une étoile en début de graphie, on la remplace par la chaîne vide (second **WHEN**). Dans les autres cas, on laisse la graphie telle quelle (**ELSE**). Le tableau 136 p. 444 donne un aperçu des résultats. Les lemmes différant de la forme de départ figurent sur une ligne grisée.

On utilise un principe similaire pour créer à la volée une table des lemmes, avec fréquence et proportion. On regroupe sur les lemmes produits et on ajoute les fréquences et proportions au sein d'un regroupement (résultat au tableau 137 p. 445) :

```
CREATE TABLE lemmeFrantext_frequence_proportion
     (SELECT
          CASE
               WHEN graphie REGEXP 'esques$'
                    THEN SUBSTRING(graphie, 1, LENGTH(graphie) - 1)
               WHEN graphie LIKE '*%' THEN REPLACE(graphie, '*', '')
               ELSE graphie
          END AS lemme,
      COUNT(*) AS 'formes'
      SUM(frequence) AS 'frequence',
      SUM(proportion)  AS 'proportion'
      FROM FrantextLexique
      GROUP BY
           CASE
                WHEN graphie REGEXP 'esques$' THEN SUBSTRING(graphie, 1, LENGTH(
                    graphie) - 1)
```

TABLEAU 136 – ESQUE : lemmatisation des formes de Frantext

lemme	*graphie*	*frequence*	*proportion*
gracianesque	gracianesque	2	0.06
picaresque	picaresque	8	0.25
churrigueresque	churrigueresque	1	0.03
fallacesque	fallacesque	1	0.03
clownesque	clownesques	1	0.03
grotesquement	grotesquement	18	0.57
goeringesque	goeringesque	1	0.03
tintamarresque	tintamarresque	5	0.16
vaticanesque	vaticanesque	1	0.03
simonesque	simonesque	1	0.03
dantonesque	dantonesque	1	0.03
donquichottesque	donquichottesques	1	0.03
chewinggumesque	chewinggumesque	1	0.03
arabesque	arabesques	99	3.15
diablesque	diablesque	1	0.03
hugolesque	hugolesque	1	0.03
zombiesque	zombiesque	1	0.03
demillesque	*demillesque	1	0.03
karabesque	karabesque	1	0.03
zigantesque	zigantesque	1	0.03
babelesque	babelesque	1	0.03

```
    WHEN graphie LIKE '*%' THEN REPLACE(graphie, '*', '')
    ELSE graphie
END) ;
```

2.5. Les mots en *-esque* au regard de Frantext

La mise en relation, par jointure entre les tables esque et lemmeFrantext_frequence_proportion (tableau 138 p. 446), s'effectue par la requête :

```
SELECT derive AS 'Dérivé',
     COUNT(*) AS 'ctxtes',
     frequence AS 'fréq.',
     proportion AS 'prop.'
FROM esque,
     lemmeFrantext_frequence_proportion
WHERE derive = lemme
GROUP BY derive
ORDER BY frequence DESC ;
```

Elle montre que les lemmes les plus fréquents dans Frantext ne sont pas pour autant fortement représentés dans la table esque. On entrevoit le dessein qui a présidé à la constitution de la table esque : multiplier les types de dérivés distincts, plutôt que les occurrences d'un même dérivé, pour pouvoir tirer des conclusions sur les modes de formation à partir d'un éventail le plus ouvert possible. L'écart de taille entre les deux tables est en tant que tel éloquent : 115 dérivés différents pour lemmeFrantext_frequence_proportion et près de 3 500 pour esque.

Une jointure externe donne accès aux lemmes qui ne figurent pas parmi les dérivés de la table esque :

TABLEAU 137 – ESQUE : table des lemmes de Frantext

graphie	*frequence*	*proportion*
alibabesque	1	0.03
animalesque	1	0.03
animalesques	1	0.03
arabesque	88	2.80
arabesques	99	3.15
aubignesque	1	0.03
aviationnesque	1	0.03
babelesque	1	0.03
barbaresque	8	0.25
barbaresques	20	0.64
barytonesques	1	0.03

⇓

lemme	*formes*	*frequence*	*proportion*
alibabesque	1	1	0.03
animalesque	2	2	0.06
arabesque	2	187	5.95
aubignesque	1	1	0.03
aviationnesque	1	1	0.03
babelesque	1	1	0.03
barbaresque	2	28	0.89
barytonesque	1	1	0.03

```
SELECT lemme,
       frequence,
       proportion
FROM lemmeFrantext_frequence_proportion
LEFT OUTER JOIN esque
ON lemme = derive
WHERE derive IS NULL ;
```

On constate que tous les lemmes de Frantext ne sont pas présents dans esque. On note dans les résultats du tableau 139 p. 447 des dérivés de deuxième niveau, des adverbes sur des adjectifs en *-esque* : *chevaleresquement*, *grotesquement*, *pittoresquement*, mais également des formes douteuses : *fiesque*, *levesque*, voire *latudesque*.

On notera que dans les deux requêtes précédentes, il y a un décalage entre les deux tables, puisque que les dérivés de la table esque comportent une catégorie tandis que ceux de la table lemmeFrantext_frequence_proportion non.

3. Echanger des données structurées : XML

Les formats délimités de type CSV présentent l'inconvénient de reposer sur des conventions implicites pour la structuration des données :

- le jeu de caractères employé n'est pas indiqué ;
- le nom des colonnes n'est pas forcément fourni ;

TABLEAU 138 – ESQUE : lemmes de Frantext présents dans la table esque

Dérivé	*ctxtes*	*fréq.*	*prop.*
gigantesque	4	461	14.69
romanesque	8	319	10.17
pittoresque	4	311	9.91
grotesque	4	281	8.95
arabesque	4	187	5.95
burlesque	2	77	2.46
mauresque	5	61	1.95
chevaleresque	7	50	1.59
livresque	3	46	1.47
barbaresque	4	28	0.89
simiesque	6	20	0.64
soldatesque	5	19	0.61
titanesque	6	17	0.54
picaresque	2	13	0.41
rocambolesque	5	8	0.25
dantesque	7	8	0.25
carnavalesque	5	8	0.26
clownesque	5	8	0.25
gargantuesque	3	7	0.22
pédantesque	3	7	0.22
tudesque	2	7	0.23
donjuanesque	2	7	0.22
cauchemardesque	6	6	0.19
tintamarresque	4	6	0.19
éléphantesque	7	5	0.16
moliéresque	3	5	0.16
prudhommesque	3	4	0.13
plateresque	2	4	0.12
pharaonesque	5	4	0.13
ubuesque	3	4	0.13
goyesque	3	3	0.09
grottesque	2	3	0.09
donquichottesque	1	3	0.09
funambulesque	5	3	0.10
ingresque	3	3	0.09
truandesque	4	3	0.09
animalesque	3	2	0.06
canularesque	3	2	0.06
cassandresque	2	2	0.06
faunesque	5	2	0.06
fourmillesque	3	2	0.06
vaudevillesque	4	2	0.06
caravagesque	4	2	0.06
courtelinesque	3	2	0.06
moresque	3	2	0.06
gracianesque	2	2	0.06
guignolesque	4	2	0.06
botticellesque	2	2	0.06
chaplinesque	6	2	0.06
aviationnesque	1	1	0.03
barytonesque	1	1	0.03

Dérivé	*ctxtes*	*fréq.*	*prop.*
cafardesque	2	1	0.03
charlatanesque	4	1	0.03
chichinesque	1	1	0.03
crapaudesque	2	1	0.03
gionesque	3	1	0.03
himalayesque	5	1	0.03
hugolesque	3	1	0.03
léonardesque	1	1	0.03
mirlitonesque	3	1	0.03
palestiniesque	1	1	0.03
perrichonesque	1	1	0.03
pluchesque	1	1	0.03
sardanapalesque	3	1	0.03
seicentesque	1	1	0.03
taglionesque	1	1	0.03
turlupinesque	3	1	0.03
vampiresque	3	1	0.03
zurbaranesque	1	1	0.03
raphaélesque	2	1	0.03
dantonesque	3	1	0.03
fantômastesque	1	1	0.03
kagébesque	1	1	0.03
zingaresque	1	1	0.03
alibabesque	1	1	0.03
babelesque	1	1	0.03
bignolesque	1	1	0.03
canovesque	1	1	0.03
diablesque	1	1	0.03
farcesque	4	1	0.03
footballesque	3	1	0.03
gendarmesque	5	1	0.03
giottesque	4	1	0.03
hippopotamesque	2	1	0.03
poesque	1	1	0.03
rembranesque	3	1	0.03
sauvagesque	1	1	0.03
statuesque	3	1	0.03
vaticanesque	5	1	0.03
zombiesque	5	1	0.03
churrigueresque	3	1	0.03
giorgionesque	2	1	0.03
boulardesque	1	1	0.03
simonesque	3	1	0.03
cyranesque	4	1	0.03
chewinggumesque	1	1	0.03
goeringesque	1	1	0.03
loyolesque	1	1	0.03
cornichonnesque	1	1	0.03
clochemerlesque	3	1	0.03
scoutesque	1	1	0.03

TABLEAU 139 – ESQUE : lemmes de Frantext absents de la table esque

lemme	*frequence*	*proportion*
aubignesque	1	0.03
chevaleresquement	1	0.03
demillesque	2	0.06
fallacesque	2	0.06
fiesque	1	0.03
grotesquement	18	0.57
karabesque	1	0.03
latudesque	1	0.03
levesque	1	0.03
mahonesque	1	0.03
pittoresquement	2	0.06
prudhonesque	1	0.03
rafinesque	4	0.12
zigantesque	1	0.03

– le caractère de séparation entre les colonnes est arbitraire (hormis le fait qu'il ne doit bien sûr pas figurer dans le contenu des colonnes) ;
– le passage d'une ligne à l'autre dépend des choix du système d'exploitation sous-jacent et diffère donc selon le système d'exploitation.

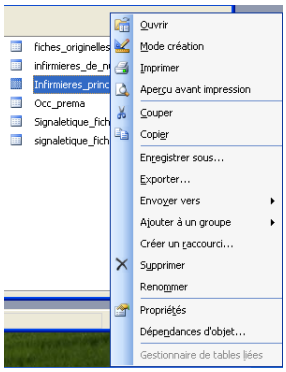
Autant dire que la transmission par de tels formats est intrinsèquement « fragile » : rien n'empêche un décalage entre l'export et l'import et donc des difficultés dans la réutilisation des données.

L'utilisation de formats structurés relevant de XML a pour objectif de disposer de conventions plus fermes et plus aisément échangeables. On va les voir à l'oeuvre dans l'export de la table Infirmieres_princeps vers XML, sous Access comme sous MySQL.

3.1. Exporter à partir d'Access

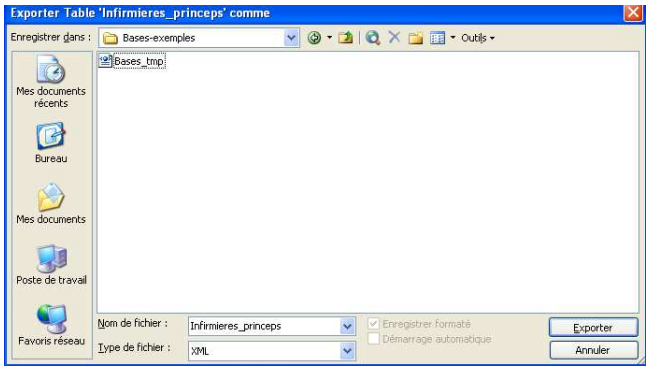
Lorsqu'on sélectionne une table, ici Infirmieres_princeps, dans l'onglet [Tables] de la fenêtre de navigation dans la table, avec le bouton droit, on peut choisir d'exporter :

1



On choisit alors à la fois le dossier et le nom de destination et le format cible, en l'occurrence XML.

2



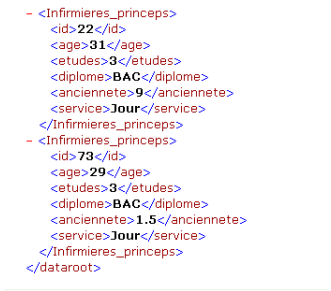
On doit déterminer si l'on veut sous format XML les données seulement, ou bien également le modèle de document, voire les indications de présentation. On s'en tient ici aux deux premiers choix, qui sont les choix par défaut :

3

Les écrans 4 et 5 montrent le début et la fin la table Infirmieres_princeps au format XML.

4

5



En XML (*Extensible Markup Language*), un texte est fondamentalement considéré comme un ensemble d'**éléments** qui peuvent s'imbriquer, c'est-à-dire comme un arbre. La figure 7 p. 449 correspond à cette représentation. Chaque attribut constitue une « boîte » dont le nom est le nom de l'attribut et qui contient la valeur associée. C'est le cas par exemple de id et de la valeur 41 en bas à gauche. Les attributs sont regroupés dans une boîte Infirmieres_princeps.

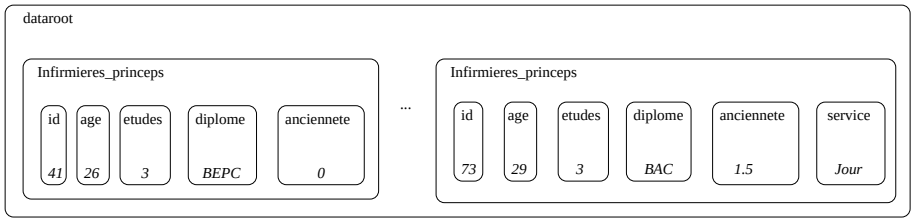


FIGURE 7 – Table infirmieres_princeps et enchâssements

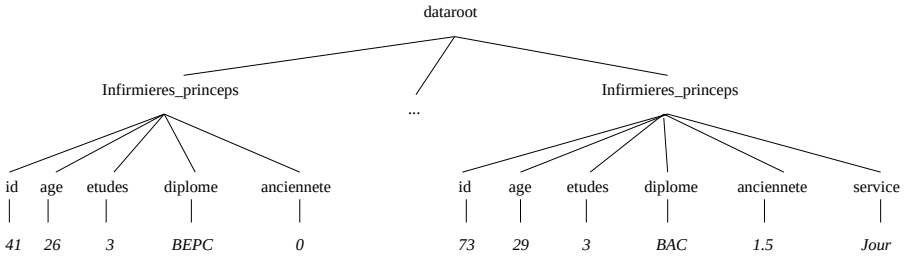


FIGURE 8 – La table infirmieres_princeps comme arbre

Les boîtes Infirmieres_princeps entrent elles-mêmes dans une boîte dataroot. La figure 8 p. 449 manifeste l'équivalence entre la représentation en boîtes et une représentation arborescente. Les écrans 4 et 5 donnent la manière de noter en XML une telle arborescence. Le marquage du début d'un sous-arbre s'effectue par une **balise ouvrante** – le nom du sous-arbre entre chevrons – et la fin par une **balise fermante** : le nom du sous-arbre après </ et avant le chevron fermant. Ainsi l'identifiant de la première infirmière constitue un sous-arbre id, à la ligne 4, délimité par <id> au début et </id> à la fin. Les cinq sous-arbres correspondant à la première infirmière sont regroupés en un arbre de plus haut niveau Infirmieres_princeps allant de la ligne 3 à la ligne 9, etc. La première ligne indique que le document obéit à la version 1.0 du standard XML et qu'il utilise l'encodage UTF-8 pour les caractères.

On reconstitue aisément la logique de transformation de la table en un arbre. Une cellule devient un arbre de profondeur 1 ayant pour racine le nom de l'attribut et pour feuille sa valeur. Une ligne devient un arbre de profondeur 2 qui enchâsse les arbres de profondeur 1 correspondant à ses cellules. La table elle-même est l'arbre de profondeur 3 qui regroupe les arbres de profondeur 2 traduisant les lignes.

La représentation obtenue explicite la structure de la table. Elle ne repose pas sur des conventions fragiles pour la délimitation des attributs ou des lignes. Elle intègre les noms des colonnes.

3.1.1. Modèle de document

Le document XML correspondant à l'export de la table Infirmieres_princeps fait référence à un modèle de document, que voici :

<?xml version="1.0" encoding="UTF-8" ?>

<xsd :schema xmlns :xsd="http ://www.w3.org/2001/XMLSchema" xmlns :od="urn :schemas-microsoft-com :officedata">

```
<xsd :element name="dataroot">
  <xsd :complexType>
    <xsd :sequence>
      <xsd :element ref="Infirmieres_princeps" minOccurs="0" maxOccurs="unbounded"/>
    </xsd :sequence>
    <xsd :attribute name="generated" type="xsd :dateTime"/>
  </xsd :complexType>
</xsd :element>
<xsd :element name="Infirmieres_princeps">
  <xsd :annotation>
    <xsd :appinfo>
    <od :index index-name="PrimaryKey" index-key="id " primary="yes" unique="yes" clustered="no"/>
    </xsd :appinfo>
  </xsd :annotation>
  <xsd :complexType>
    <xsd :sequence>
     <xsd :element name="id" minOccurs="1" od :jetType="integer" od :sqlSType="smallint" od :non-
Nullable="yes" type="xsd :short"/>
    <xsd :element name="age" minOccurs="0" od :jetType="longinteger" od :sqlSType="int" type="xsd :int"/>
       <xsd :element name="etudes" minOccurs="0" od :jetType="longinteger" od :sqlSType="int"
type="xsd :int"/>
      <xsd :element name="diplome" minOccurs="0" od :jetType="text" od :sqlSType="nvarchar">
        <xsd :simpleType>
          <xsd :restriction base="xsd :string">
          <xsd :maxLength value="255"/>
          </xsd :restriction>
        </xsd :simpleType>
      </xsd :element>
       <xsd :element name="anciennete" minOccurs="0" od :jetType="single" od :sqlSType="real"
type="xsd :float"/>
      <xsd :element name="service" minOccurs="0" od :jetType="text" od :sqlSType="nvarchar">
        <xsd :simpleType>
          <xsd :restriction base="xsd :string">
          <xsd :maxLength value="255"/>
          </xsd :restriction>
        </xsd :simpleType>
      </xsd :element>
    </xsd :sequence>
  </xsd :complexType>
</xsd :element>
```

</xsd :schema>

Ce modèle donne les règles de « bonne formation » du document Infirmieres_princeps.xml. Il y a plusieurs manières de fournir de telles règles Ray (2001)(Fitzgerald, 2004). La plus ancienne consiste à définir une DTD (*Définition de type de document*). Plus récemment, ont été introduits les schémas. Le modèle ci-dessus est un schéma, comme l'indique la balise de la deuxième ligne. À partir du moment où un document XML est un arbre, les règles formulent une « grammaire », c'est-à-dire une manière à la fois d'enchâsser les éléments (quel élément peut dominer tel ou tel élément ?) et de les enchaîner (quel élément peut suivre tel élément), en indiquant éventuellement qu'un élément est optionnel ou au contraire qu'il est répétable. Ainsi dataroot est-il défini comme une séquence d'éléments de type Infirmieres_princeps. Cette séquence peut être vide, n'avoir aucune occurrence d'éléments Infirmieres_princeps (cf. minOccurs='0') ou en avoir un nombre indéfini : maxOccurs='unbounded'. De la même manière, un élément Infirmieres_princeps domine un élément id, un élément age, un élément etudes, un élément diplome, un élément anciennete et un élément service. Sauf id, qui est la clé primaire et à ce titre forcément présent (cf. minOccurs='1'), tous les autres éléments peuvent ne pas être présents.

On pourrait reformuler cette « grammaire » ainsi :

dataroot → Infirmieres_princeps*

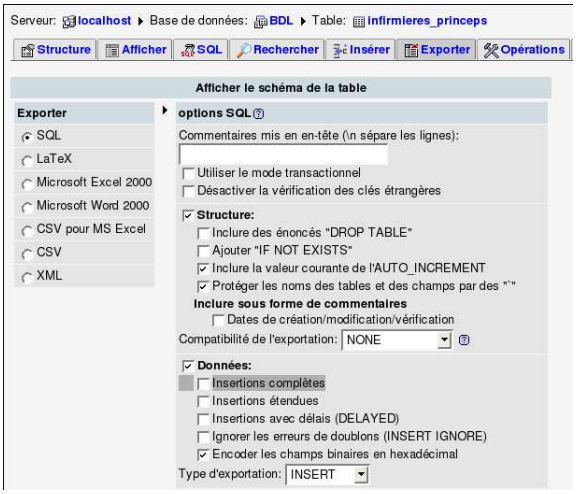
Infirmieres_princeps→ id age ? etudes ? diplome ? anciennete ? service ?

où l'étoile signifie 0 ou n occurrences de l'élément qui précède et ? 1 ou n.

Une DTD permet de formuler une telle grammaire. Un schéma comme celui supra ajoute des contraintes supplémentaires, par exemple sur les valeurs admissibles en un endroit donné. Il est ainsi précisé qu'id doit être un entier, age et etudes un entier long, anciennete un reel, diplome et service une chaîne de 255 caractères maximum.

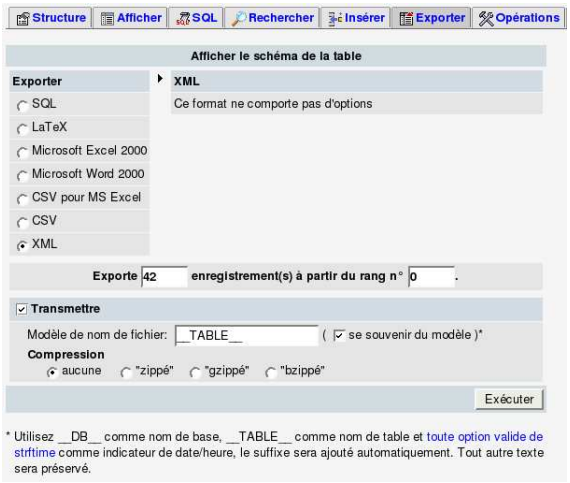
3.2. Exporter à partir de MySQL

Sous Phpmyadmin, on sélectionne la table que l'on souhaite exporter. On doit alors choisir entre différents formats d'exportation, dont des formats délimités (CSV). On choisit XML.



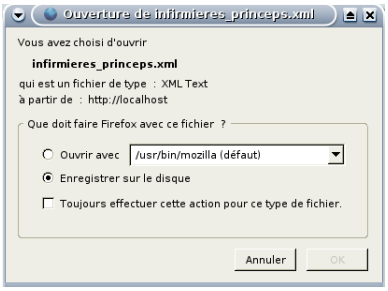
Cocher [Transmettre] aboutit dans les faits à sauvegarder dans un fichier. Il y a transmission parce que Phpmyadmin transmet effectivement les requêtes à la base de données et leurs résultats.

7



On se voit demander si l'on souhaite ouvrir le fichier résultant ou au contraire l'ouvrir avec une application.

8



Le fichier résultant est de la forme :

```
<?xml version="1.0" encoding="utf-8" ?>
<!–
-
- phpMyAdmin XML Dump
- version 2.6.2-Debian-3sarge1
- http ://www.phpmyadmin.net
-
- Serveur : localhost
- GÃ©nÃ©rÃ© le : Samedi 17 Novembre 2007 Ã  17 :02
- Version du serveur : 4.1.11
- Version de PHP : 4.3.10-16
–>
<!–
- Base de donnÃ©es : 'BDL'
–>
<BDL>
```

```
<!– Table infirmieres_princeps –>
  <infirmieres_princeps>
    <id>41</id>
    <age>26</age>
    <etudes>3</etudes>
    <diplome>BEPC</diplome>
    <anciennete>0.00</anciennete>
  </infirmieres_princeps>
...
  <infirmieres_princeps>
    <id>73</id>
    <age>29</age>
    <etudes>3</etudes>
    <diplome>BAC</diplome>
    <anciennete>1.50</anciennete>
    <service>Jour</service>
  </infirmieres_princeps>
</BDL>
```

On remarque la proximité avec l'export vers XML sous Access. La première ligne indique, de la même manière, que ce qui suit est un document XML répondant à la version 1.0 de ce standard et que les caractères relèvent du jeu UTF-8. La racine du document porte ici le nom de la base de données dont elle est extraite (BDL). Chaque ligne a pour racine le nom de la table, infirmieres_princeps. Chaque colonne devient un sous-arbre ayant le nom de la colonne et dominant sa valeur. Pour l'infirmière 41, la valeur **NULL** pour la colonne service n'est pas représentée, le sous-arbre correspondant ne figure pas dans les « enfants » du noeud infirmieres_princeps en cause.

En dehors du nom différent pour les éléments (infirmieres_princeps *vs.* Infirmieres_princeps) et pour la racine de l'arbre (BDL *vs.* dataroot), on note que la version Phpmyadmin ajoute des commentaires entre la première ligne du document et la racine de l'arbre. Ces commentaires, délimités respectivement par <!– et –> donnent des renseignements sur l'engendrement de cette table (version de MySQL employée, date, etc.). On remarquera des caractères « bizarres », liés au décalage entre le jeu de caractères de Phpmyadmin (UTF-8) et celui utilisé pour ce livre (ISO-Latin1).

3.3. Importer sous Access des données XML

Sous MySQL, on a exporté en XML avec deux versions distinctes de Phpmyadmin la table signaletique_fiches, avec des jeux de caractères distincts : ISO-Latin1 et UTF-8, comme le rappelent les noms choisis : signaletique_fiches-Latin1.xml et signaletique_fiches-utf8.xml.

La première ligne de chaque fichier indique le jeu de caractères employé :

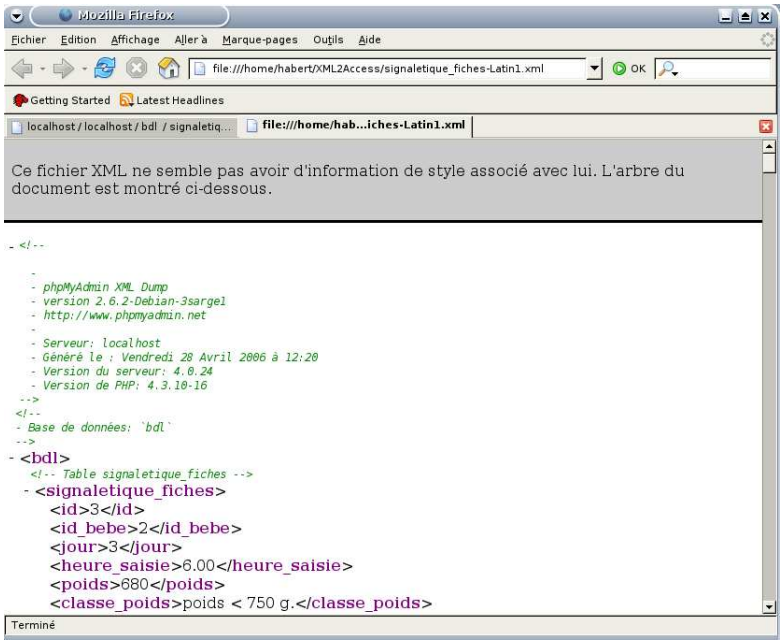
```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

et

```
<?xml version="1.0" encoding="iso-8859-1" ?>
```

Un navigateur comme Firefox permet de visualiser signaletique_fiches-Latin1.xml.

9



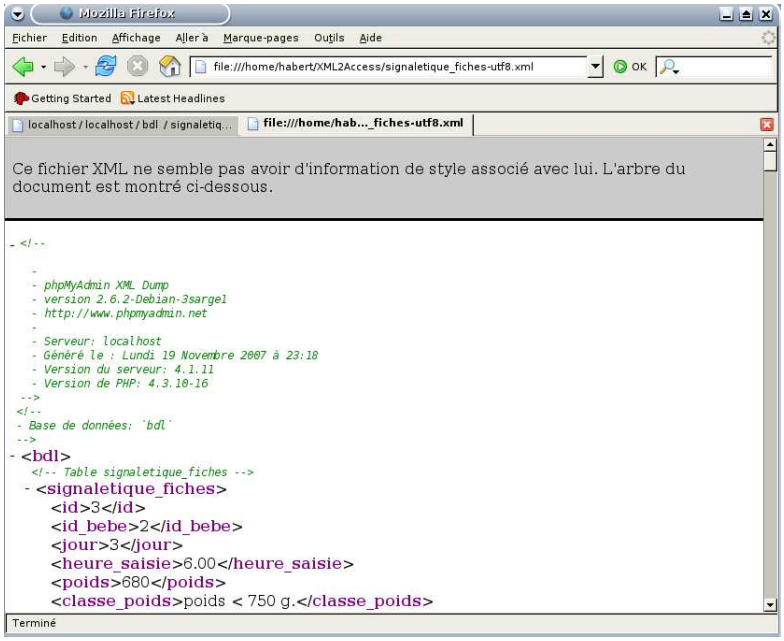
On constate que les caractères accentués de la première fiche sont correctement rendus.

10

```
- <signaletique_fiches>
    <id>3</id>
    <id_bebe>2</id_bebe>
    <jour>3</jour>
    <heure_saisie>6.00</heure_saisie>
    <poids>680</poids>
    <classe_poids>poids < 750 g.</classe_poids>
    <position>plat dos</position>
    <classe_position>dos</classe_position>
    <sedation>non</sedation>
    <ventilation>intubé</ventilation>
    <id_infirmiere>47</id_infirmiere>
    <frequence_occupation>non réponse</frequence_occupation>
    <moral_infirmiere>plutôt bien</moral_infirmiere>
    <pronostic_infirmiere>plutôt bon</pronostic_infirmiere>
    <relation_infirmiere_parents>je ne connais pas les
    parents</relation_infirmiere_parents>
    <relation_mere_bebe>je ne sais pas</relation_mere_bebe>
    <frequence_visites_parents>non réponse</frequence_visites_parents>
  </signaletique_fiches>
```

La visualisation toujours avec Firefox de signaletique_fiches-utf8.xml donne des résultats similaires.

11



Les caractères accentués sont là encore bien rendus.

12

```
- <signaletique_fiches>
    <id>3</id>
    <id_bebe>2</id_bebe>
    <jour>3</jour>
    <heure_saisie>6.00</heure_saisie>
    <poids>680</poids>
    <classe_poids>poids < 750 g.</classe_poids>
    <position>plat dos</position>
    <classe_position>dos</classe_position>
    <sedation>non</sedation>
    <ventilation>intubé</ventilation>
    <id_infirmiere>47</id_infirmiere>
    <frequence_occupation>non réponse</frequence_occupation>
    <moral_infirmiere>plutôt bien</moral_infirmiere>
    <pronostic_infirmiere>plutôt bon</pronostic_infirmiere>
    <relation_infirmiere_parents>je ne connais pas les
    parents</relation_infirmiere_parents>
    <relation_mere_bebe>je ne sais pas</relation_mere_bebe>
    <frequence_visites_parents>non réponse</frequence_visites_parents>
  </signaletique_fiches>
```

Ce résultat rassurant est moins évident qu'il n'y paraît. Si l'on examine par exemple l'un des élements de signaletique_fiches-Latin1.xml, on obtient :

<moral_infirmiere>plutôt bien</moral_infirmiere>

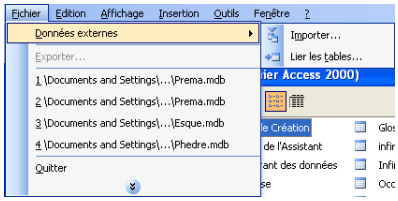
Pour son correspondant de signaletique_fiches-utf8.xml, on obtient :

<moral_infirmiere>plutÃ´t bien</moral_infirmiere>

où le ô a un rendu curieux. On constate que l'éditeur utilisé pour composer ces pages utilise le jeu de caractères Latin1 et qu'il ne sait pas rendre les caractères accentués du jeu de caractères UTF-8. A contrario, le navigateur Firefox utilise les indications de la première ligne du document XML pour représenter les caractères en fonction du jeu dont ils relèvent.

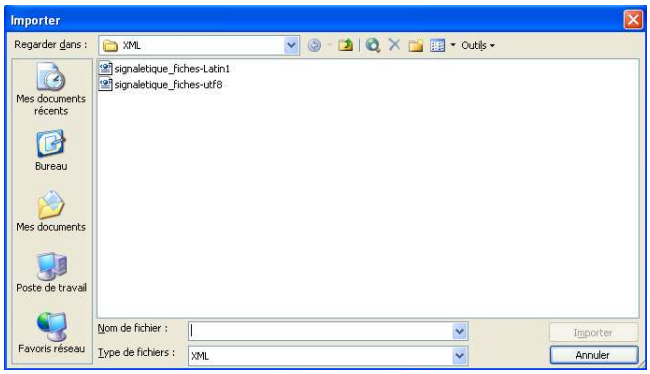
L'import de chacune de ces deux tables en XML sous Access suppose de choisir ⌈Fichier | Données externes | exporter⌉ :

13



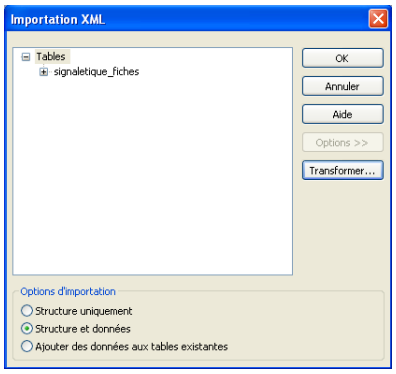
On choisit de considérer les fichiers XML en jouant sur le menu déroulant permettant de choisir les types de fichiers à importer :

14



Après avoir choisi signaletique_fiches-Latin1.xml, un menu spécifique apparaît, dont les options permettent de choisir d'importer structures et données seulement (choix par défaut), ou la structure seule, ou d'importer dans une table existante :

15

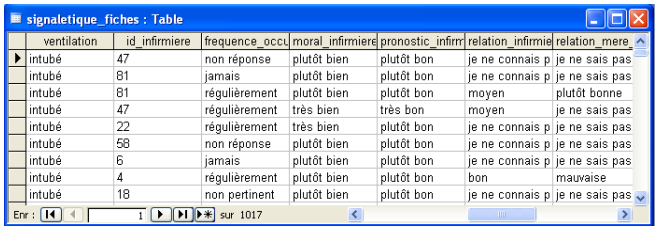


L'onglet [Tables] de la fenêtre de navigation dans la base montre que la table a été importée. Elle porte par défaut le nom des éléments enchâssant les colonnes, c'est-à-dire le nom de la table de départ, ici signaletique_fiches :

16

On constate en 17 que les caractères de signaletique_fiches-Latin1.xml, en ISO-Latin1, ont été correctement interprétés.

17

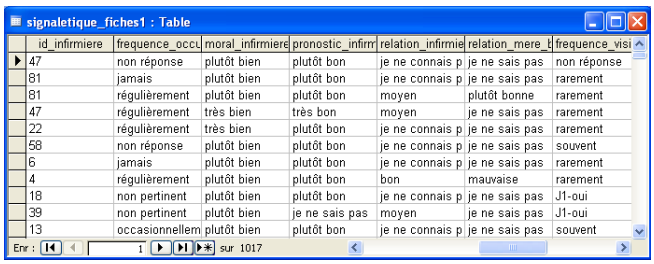


On opère de la même manière pour signaletique_fiches-utf8.xml. Une fois l'import effectué, on constate qu'Access a engendré un nom de table qui diffère du précédent par l'ajout d'un numéro d'ordre, pour éviter les confusions.

18

Lorsqu'on examine cette table, on voit que les caractères, au départ en UTF-8, ont également été bien interprétés et donc qu'Access, tout comme Firefox, utilise la première ligne du document XML pour savoir comment interpréter les caractères du document.

19



4. Remodeler des informations peu structurées

4.1. Le dictionnaire électronique DELA

Maurice Gross a impulsé le développement des lexiques-grammaires. La description fine d'un mot consiste, dans ce cadre, à examiner son comportement pour le maximum de propriétés syntaxiques pertinentes pour sa catégorie grammaticale (environ 400 pour les verbes). Par exemple, pour un verbe : se passive-t-il ? accepte-t-il une construction en *se* (*Il vend bien ses livres* vs. *ses livres se vendent bien*) ?, etc. La démarche a été progressivement étendue aux mots en plusieurs mots, ce qui a permis d'aboutir aux répertoires les plus complets qui existent pour le français (même si ces inventaires ne sauraient être considérés comme terminés). Elle prend en compte les constructions à verbe support (*il a froid* et *il fait un somme*, dans *Dormeur*), où le verbe apporte les indications de temps et d'aspect, tandis que la majeure partie du sémantisme provient de l'adjectif ou du nom.

Les lexiques-grammaires des mots simples (680 000) et des mots en plusieurs mots (100 000) sont incorporés à deux logiciels : Intex/Nooj et Unitex, offrant ainsi des dictionnaires lexico-syntaxiques sans précédents tant par leur couverture que par leur finesse d'analyse.

C'est le dictionnaire DELA, disponible sous licence GPL dans le cadre du logiciel d'étiquetage-lemmatisation Unitex, qui est utilisé dans cette section. Il comprend, dans la version mise à contribution ici, 792 260 entrées, en autant de lignes. C'est un dictionnaire de formes et non pas de lemmes. On trouve donc les singuliers et les pluriels des noms, les flexions des verbes, etc. Le tableau 140 p. 459 fournit les lignes comprenant le motif '.*prématu.*'. On note la présence de « mots en plusieurs mots », comme *enfant prématuré*, *vieillissement prématuré*. Les lignes se découpent en :

<forme>,.<informations catégorielles>

comme dans :

prématurissime,.N :ms :fs

ou encore en :

<forme>,<lemme>.<informations catégorielles>

comme dans :

prématurissimes,prématurissime.N :mp :fp

Les informations catégorielles pour *prématurissime* indiquent qu'il s'agit d'un nom, soit masculin singulier (ms) soit féminin singulier (fs). Pour *prématurissimes*, le lemme est *prématurissime* et c'est un nom soit masculin pluriel soit féminin pluriel.

L'objectif est dans un premier temps d'insérer toutes les lignes du DELA dans une table provisoire, puis d'éliminer toutes les formes qui ne contiennent pas le motif '.*esque.*' et enfin d'essayer d'extraire des entrées restantes les informations nécessaires pour rapprocher la table finale des tables attendues pour mémoriser les attestations lexicographiques de mots en *-esque*.

MySQL

4.2. Importation

Une première étape consiste à créer une table provisoire, `dela_tmp`, destinée à accueillir les entrées du DELA :

TABLEAU 140 – PRÉMA : formes ressemblant à '.*prématu.*' dans le dictionnaire DELA

entree
accouchement prématuré,.N+NA+z1 :ms
enfant prématuré,.N+NA+Hum+z1 :ms
enfants prématurés,enfant prématuré.N+NA+Hum+z1 :mp
fin prématurée,.N+NA+E01+z1 :fs
mort prématurée,.N+NA+z2 :fs
prématuration,.N :fs
prématurations,prématuration.N :fp
prématurissime,.N :ms :fs
prématurissimes,prématurissime.N :mp :fp
prématurité,.N+z2 :fs
prématurités,prématurité.N+z2 :fp
prématuré,.N+z1 :ms
prématuré,.A+z1 :ms
prématurée,prématuré.N+z1 :fs
prématurée,prématuré.A+z1 :fs
prématurées,prématuré.N+z1 :fp
prématurées,prématuré.A+z1 :fp
prématurément,.ADV+VADV+z1
prématurément,.ADV+PADV+z1
prématurément,.ADV+z1
prématurés,prématuré.N+z1 :mp
prématurés,prématuré.A+z1 :mp
vieillissement prématuré,.N+NA+z1 :ms

```
CREATE TABLE dela_tmp (
      entree_depart VARCHAR(255) BINARY
) ;
```

Peupler cette table à partir du fichier dela-fr-public.dic s'effectue par la commande :

```
LOAD DATA LOCAL INFILE
"/home/habert/dela-fr-public.dic"
INTO TABLE dela_tmp ;
```

4.3. Élagage des entrées du dictionnaire DELA

Après avoir examiné les entrées comprenant le motif '.*esque.*' et une espace (tableau 141 p. 460), via la requête :

```
SELECT *
FROM dela_tmp
WHERE entree_depart REGEXP 'esque'
      AND entree_depart REGEXP '␣' ;
```

on détruit toutes les entrées qui ne contiennent pas le motif '.*esque.*' ou qui contiennent une espace, soit 792 081, par la requête :

```
DELETE FROM dela_tmp
WHERE entree_depart NOT REGEXP 'esque'
      OR entree_depart  REGEXP '␣' ;
```

On remarque au passage que ces entrées supprimées comprennent probablement des adjectifs en *-esque* qui ne sont pas des réalisations éphémères mais qui sont au contraire entrés dans l'usage : *barbaresque*, *mauresque*, *romanesque*, *livresque*, *burlesque*, *rocambolesque*,

TABLEAU 141 – ESQUE : « mots en plusieurs mots » avec *esque* dans le dictionnaire DELA

entree_depart
Etat barbaresque,.N+NA+HumColl :ms
amour romanesque,.N+NA+z1 :ms
aventure rocambolesque,.N+NA+E01+z2 :fs
aventures rocambolesques,aventure rocambolesque.N+NA+E01+z2 :fp
création romanesque,.N+NA+z2 :fs
culture livresque,.N+NA+z1 :fs
cultures livresques,culture livresque.N+NA+z1 :fp
cycle romanesque,.N+NA+z2 :ms
cycles romanesques,cycle romanesque.N+NA+z2 :mp
dossier rocambolesque,.N+NA+Conc+E01 :ms
esprits romanesques,esprit romanesque.N+NA+z2 :mp
fenêtre mauresque,.N+NA+Conc :fs
fiction romanesque,.N+NA+z1 :fs
fictions romanesques,fiction romanesque.N+NA+z1 :fp
film burlesque,.N+NA+Conc+z2 :ms
forme romanesque,.N+NA+z1 :fs
formes romanesques,forme romanesque.N+NA+z1 :fp
fresque historique,.N+NA+Conc+z1 :fs
fresque murale,.N+NA+Conc+z2 :fs
fresques historiques,fresque historique.N+NA+Conc+z1 :fp
genre romanesque,.N+NA+z1 :ms
grande fresque,.N+AN :fs
grandes fresques,grande fresque.N+AN :fp
histoire rocambolesque,.N+NA+E01+z1 :fs
histoires rocambolesques,histoire rocambolesque.N+NA+E01+z1 :fp
illusion romanesque,.N+NA :fs
langage romanesque,.N+NA :ms
littérature chevaleresque,.N+NA :fs
littérature romanesque,.N+NA+z2 :fs
oeuvre romanesque,.N+NA+z1 :fs
oeuvres romanesques,oeuvre romanesque.N+NA+z1 :fp
ou presque,.ADV+PJC+z1
parodie burlesque,.N+NA+z2 :fs
peintre de fresques,.N+NDN+Hum :ms :fs
personnage romanesque,.N+NA+Hum+z2 :ms
pirate barbaresque,.N+NA+Hum :ms
pirates barbaresques,pirate barbaresque.N+NA+Hum :mp
pour presque rien,.ADV+PAC+z1
poésie burlesque,.N+NA :fs
production romanesque,.N+NA+z2 :fs
repas gargantuesque,.N+NA+E01+z1 :ms
roman picaresque,.N+NA+Conc :ms
scène romanesque,.N+NA+z2 :fs
scènes romanesques,scène romanesque.N+NA+z2 :fp
structure romanesque,.N+NA+z2 :fs
style burlesque,.N+NA :ms
style mauresque,.N+NA+z2 :ms
technique romanesque,.N+NA+z2 :fs
temps romanesque,.N+NA+z2 :ms
texte romanesque,.N+NA+Conc+z2 :ms
théâtre burlesque,.N+NA :ms
tortues mauresques,tortue mauresque.N+NA+Anl :fp
univers romanesque,.N+NA+z1 :ms
vaste fresque,.N+AN :fs
écriture romanesque,.N+NA+z2 :fs
érudits livresques,érudit livresque.N+NA+Hum :mp

chevaleresque, *gargantuesque*, *picaresque*. Ces adjectifs comme les noms qu'ils modifient renvoient souvent à l'art.

Les 179 entrées restantes comprennent du bruit :

entree_depart
desquelles,duquel.PREPPRO+z1 :fp
desquels,duquel.PREPPRO+z1 :mp
fresque,.N+z1 :fs
fresques,fresque.N+z1 :fp
lesquelles,lequel.PRO+z1 :fp
lesquelles,lequel.DET+z1 :fp
lesquels,lequel.PRO+z1 :mp
lesquels,lequel.DET+z1 :mp
nesquehonite,.N :fs
nesquehonites,nesquehonite.N :fp
presqu,presque.ADV+z1
presque,.ADV+z1

Exercice n°1 Supprimer les entrées en *esque* restantes qui correspondent à du bruit.

4.4. Rapprochement avec la table esque

On commence par ajouter les attributs qui correspondent à ceux de la table entrees_dictionnairiques :

```
ALTER TABLE dela_tmp
ADD COLUMN lemme VARCHAR(60) BINARY NOT NULL ;

ALTER TABLE dela_tmp
ADD COLUMN categorie VARCHAR(60) BINARY NOT NULL ;

ALTER TABLE dela_tmp
ADD COLUMN forme VARCHAR(60) BINARY NOT NULL ;

ALTER TABLE dela_tmp
ADD COLUMN dictionnaire VARCHAR(60) BINARY NOT NULL DEFAULT 'DELA' ;
```

Les entrées où la forme est également le lemme sont celles qui répondent aux conditions de restriction suivantes :

```
... WHERE entree_depart REGEXP ',\\.' ;

... WHERE entree_depart LIKE '%,.%' ;
```

Il s'agit alors, pour ces entrées, de donner aux attributs lemme et forme la sous-chaîne de l'attribut entree_depart qui précède la virgule, via la requête :

```
UPDATE dela_tmp
    SET
        lemme =
        SUBSTRING(entree_depart, 1, LOCATE(",", entree_depart) - 1),
        forme =
        SUBSTRING(entree_depart, 1, LOCATE(",", entree_depart) - 1)
WHERE entree_depart LIKE '%,.%' ;
```

Le lemme ou la forme est la sous-chaîne de l'entrée qui part du début de l'entrée et qui s'arrête 1 caractère avant l'endroit où se trouve la virgule.

Quand l'entrée relève du motif :

<forme>,<lemme>.<informations catégorielles>

on donne à l'attribut forme la sous-chaîne de l'attribut entree_depart qui précède la virgule et à l'attribut lemme la sous-forme de l'attribut entree_depart qui commence après la virgule et s'arrête avant le point, grâce à la requête :

```
UPDATE dela_tmp
    SET
        forme =
        SUBSTRING(entree_depart, 1, LOCATE(",", entree_depart) - 1),
        lemme =
        SUBSTRING(entree_depart, LOCATE(",", entree_depart) + 1, LOCATE(".",
            entree_depart) - LOCATE(",", entree_depart) - 1)
WHERE entree_depart NOT LIKE '%,.%' ;
```

Les entrées de départ en *esque* du DELA ont été analysées pour extraire la forme et le lemme. Il reste à leur attribuer des catégories identiques, si possible, à celles de ESQUE. Une première étape est celle du repérage du jeu d'étiquettes employé pour cet ensemble d'entrées, via la requête :

```
SELECT SUBSTRING(entree_depart, LOCATE(".", entree_depart) + 1, LENGTH(entree_depart
    ) - LOCATE(".", entree_depart)) AS 'Étiquette',
     COUNT(*) AS 'o.'
FROM dela_tmp
GROUP BY SUBSTRING(entree_depart, LOCATE(".", entree_depart) + 1, LENGTH(
    entree_depart) - LOCATE(".", entree_depart)) ;
```

Le résultat figure au tableau 142 p. 463. La comparaison avec les étiquettes des dérivés dans la table esque :

Catégorie du dérivé	*o*
	134
?	3
a	4143
adv	13
n	20
n?	2
nf	27
nfpl	1
nm	38
npl	1
v	1
vintr	1

via la requête :

```
SELECT cat_derive AS 'Catégorie du dérivé',
     COUNT(*) AS 'o'
FROM esque
GROUP by cat_derive ;
```

montre que 3 catégories sont pertinentes : a(djectif), adv(erbe) et n(om). Les mises à jour et les conditions de restriction sont aisées à formuler :

TABLEAU 142 – ESQUE : étiquettes des mots en *esque* dans le DELA

Étiquette	*o.*
A+XA :mp :fp	2
A+XA :ms :fs	1
A+z1 :mp :fp	19
A+z1 :ms :fs	19
A+z2 :fp	2
A+z2 :fs	2
A+z2 :mp :fp	19
A+z2 :ms :fs	19
A :mp :fp	22
A :ms :fs	22
ADV	5
ADV+z1	3
ADV+z2	5
N+z1 :fp	2
N+z1 :fs	2
N+z1 :mp	1
N+z1 :mp :fp	1
N+z1 :ms	5
N+z1 :ms :fs	1
N+z2 :fp	3
N+z2 :fs	3
N+z2 :mp :fp	1
N+z2 :ms :fs	1
N :fp	2
N :fs	2
N :mp	1
N :mp :fp	1
N :ms :fs	1

```
UPDATE dela_tmp
     SET
          categorie = 'adv'
WHERE entree_depart REGEXP '[.]ADV' ;

UPDATE dela_tmp
     SET
          categorie = 'a'
WHERE entree_depart REGEXP '[.]A[:+]' ;

UPDATE dela_tmp
     SET
          categorie = 'n'
WHERE entree_depart REGEXP '[.]N[:+]' ;
```

Le tableau 143 p. 464 le montre via un extrait, les entrées en *esque* du DELA ont été reformatées par des combinaisons relativement simples de motifs et d'opérations sur les chaînes de caractères de manière à être rapprochées autant que nécessaire de la représentation employée par ESQUE.

Access

La table esque contenant des lemmes et non des flexions de dérivés, on regroupe les formes d'un même lemme dans une nouvelle table :

TABLEAU 143 – ESQUE : entrées du DELA reformatées pour la base

entree_depart	*lemme*	*categorie*	*forme*	*dictionnaire*
Moresque,More.N+z2 :fs	More	n	Moresque	DELA
Moresques,More.N+z2 :fp	More	n	Moresques	DELA
arabesque,.N+z1 :fs	arabesque	n	arabesque	DELA
arabesques,arabesque.N+z1 :fp	arabesque	n	arabesques	DELA
aristophanesque,.A :ms :fs	aristophanesque	a	aristophanesque	DELA
aristophanesques,aristophanesque.A :mp :fp	aristophanesque	a	aristophanesques	DELA
barbaresque,.N+z2 :ms :fs	barbaresque	n	barbaresque	DELA
barbaresque,.A+z2 :ms :fs	barbaresque	a	barbaresque	DELA
barbaresques,barbaresque.N+z2 :mp :fp	barbaresque	n	barbaresques	DELA
barbaresques,barbaresque.A+z2 :mp :fp	barbaresque	a	barbaresques	DELA
burlesque,.N+z1 :ms	burlesque	n	burlesque	DELA
burlesque,.A+z1 :ms :fs	burlesque	a	burlesque	DELA
burlesquement,.ADV+z2	burlesquement	adv	burlesquement	DELA
burlesques,burlesque.N+z1 :mp	burlesque	n	burlesques	DELA
burlesques,burlesque.A+z1 :mp :fp	burlesque	a	burlesques	DELA
calembouresque,.A+z1 :ms :fs	calembouresque	a	calembouresque	DELA
calembouresques,calembouresque.A+z1 :mp :fp	calembouresque	a	calembouresques	DELA
caméléonesque,.A+z2 :ms :fs	caméléonesque	a	caméléonesque	DELA
caméléonesques,caméléonesque.A+z2 :mp :fp	caméléonesque	a	caméléonesques	DELA
cannibalesque,.A+z2 :ms :fs	cannibalesque	a	cannibalesque	DELA

```
CREATE TABLE lemmeDELA_categorie_nombreFormes
      (SELECT lemme,
            categorie,
            COUNT(*) AS 'formes'
FROM dela_tmp
GROUP BY lemme, categorie) ;
```

Les 167 formes en *-esque* du DELA donnent naissance à 92 lignes dans la nouvelle table.

4.5. Les mots en *-esque* au regard du dictionnaire DELA

69 des 92 formes en *-esque* du DELA figurent dans la table esque (tableau 144 p. 466), comme le montre la requête :

```
SELECT derive AS 'Dérivé',
      derive_POS AS 'POS',
      COUNT(*) AS 'ctxtes',
      formes AS 'DELA'
FROM esque_tmp AS e,
      lemmeDELA_categorie_nombreFormes AS d
WHERE derive = lemme
      AND derive_POS = categorie ;
```

Une jointure externe permet d'isoler les 23 formes en *-esque* du DELA qui ne figurent pas dans la table esque :

```
SELECT lemme AS 'Lemme',
      categorie AS 'POS',
      formes AS 'DELA'
FROM lemmeDELA_categorie_nombreFormes
LEFT OUTER JOIN esque_tmp
ON derive = lemme
      AND derive_POS = categorie
WHERE derive IS NULL
ORDER BY LOWER(lemme) ;
```

L'examen du résultat (tableau 145 p. 467) est instructif. On note d'abord l'importance des dérivés « de deuxième niveau », les adverbes construits sur des adjectifs en *-esque* : 13 lemmes sur 23. Par ailleurs, les formes *maure* et *more*, avec éventuellement une majuscule, adjectif et nom, inclus dans un autre adjectif (*hispano-moresque*) regroupent 5 lemmes. Pour le DELA, *maure* et *more* sont les lemmes respectivement de *mauresque* et *moresque*.

4.6. Convergences/divergences entre Frantext et DELA

Puisqu'on dispose de deux tables pour les mots en *-esque* de la base Lexique et pour ceux du DELA, on peut répérer les 33 lemmes communs (tableau 146 p. 468) :

```
SELECT DISTINCT d.lemme
FROM lemmeDELA_categorie_nombreFormes AS d
NATURAL JOIN lemmeFrantext_frequence_proportion ;
```

On peut par ailleurs, là encore via des jointures externes, isoler les mots propres à chacune des tables (tableau 147 p. 469) :

R^2_{120}

TABLEAU 144 – ESQUE : mots du DELA présents dans la table esque

Dérivé	*POS*	*ctxtes*	*DELA*
arabesque	n	2	2
aristophanesque	a	4	2
barbaresque	a	3	2
burlesque	a	1	2
burlesque	n	1	2
calembouresque	a	1	2
caméléonesque	a	5	2
cannibalesque	a	4	2
canularesque	a	3	2
caravagesque	a	3	2
carnavalesque	a	4	2
cauchemardesque	a	6	2
chaplinesque	a	6	2
charlatanesque	a	4	2
chevaleresque	a	6	2
churrigueresque	a	2	2
clownesque	a	5	2
dantesque	a	7	2
donjuanesque	a	2	2
farcesque	a	4	2
faunesque	a	4	2
feuilletonesque	a	2	2
feuilletonnesque	a	1	2
flandresque	a	1	2
funambulesque	a	4	2
gagesque	a	1	2
gargantuesque	a	3	2
gigantesque	a	3	2
gigantesque	n	1	1
giottesque	a	2	2
giottesque	n	2	2
grand-guignolesque	a	3	2
grotesque	a	2	2
grotesque	n	2	3
guignolesque	a	4	2

Dérivé	*POS*	*ctxtes*	*DELA*
hippopotamesque	a	2	2
humoresque	n	2	2
ingresque	a	3	2
livresque	a	3	2
madrigalesque	a	4	2
mauresque	a	4	2
mauresque	n	1	2
moliéresque	a	3	2
moresque	a	1	2
moresque	n	2	2
niagaresque	a	4	2
nirvanesque	a	2	2
pittoresque	a	3	2
pittoresque	n	1	1
plateresque	a	2	2
prudhommesque	a	3	2
pédantesque	a	3	2
péruginesque	a	1	2
raphaélesque	a	2	2
rembranesque	a	2	2
rocambolesque	a	5	2
romanesque	a	6	2
romanesque	n	2	1
rouletabillesque	a	2	2
sardanapalesque	a	3	2
simiesque	a	6	2
soldatesque	a	3	2
soldatesque	n	2	2
somnambulesque	a	4	2
titanesque	a	5	2
tudesque	a	2	2
ubuesque	a	3	2
vaudevillesque	a	4	2
éléphantesque	a	7	2

TABLEAU 145 – ESQUE : mots du DELA absents de la table esque

Lemme	*POS*	*DELA*
barbaresque	n	2
burlesquement	adv	1
chevaleresquement	adv	1
gigantesquement	adv	1
grotesquement	adv	1
grotesques	n	1
hispano-moresque	a	1
livresquement	adv	1
maure	a	2
maure	n	2
mauresquement	adv	1
michélangelesque	a	2
monalisesque	a	2
More	n	2
more	a	2
picaresque	a	2
pittoresquement	adv	1
prudhommesquement	adv	1
pédantesquement	adv	1
rocambolesquement	adv	1
romanesquement	adv	1
soldatesquement	adv	1
sultanesquement	adv	1


```
SELECT DISTINCT d.lemme
FROM lemmeDELA_categorie_nombreFormes AS d
LEFT OUTER JOIN lemmeFrantext_frequence_proportion AS f
ON d.lemme = f.lemme
WHERE f.lemme IS NULL ;
```

R^2_{121}

```
SELECT DISTINCT f.lemme
FROM lemmeFrantext_frequence_proportion AS f
LEFT OUTER JOIN lemmeDELA_categorie_nombreFormes AS d
ON d.lemme = f.lemme
WHERE d.lemme IS NULL ;
```

Le décalage en faveur des mots en *-esque* venant du corpus Frantext par rapport à ce qui figure dans le DELA, un rapport de 2 à un, souligne la sous-estimation de ce type de mots dans les dictionnaires de langue, et sa productivité dans l'usage.

5. Remodeler des informations structurées

Un document XML est un arbre. Un arbre organise les informations de manière simple, puisque deux relations suffisent à le décrire : la relation de dominance (un noeud domine un ou plusieurs autres noeuds et/ou des « feuilles », c'est-à-dire des noeuds sans descendance) et la relation de précédence (un noeud dominé par un noeud donné précède ou est précédé par les autres « enfants » du noeud dominant). Il y a par ailleurs un chemin unique pour aller de la racine de l'arbre à l'un de ses descendants. L'uniformité de cette organisation offre la voie à des outils de transformation systématique plus puissants que ceux qui ont été présentés

TABLEAU 146 – ESQUE : mots en *-esque* partagés par Frantext (Lexique) et DELA

lemme
arabesque
barbaresque
burlesque
caravagesque
carnavalesque
cauchemardesque
chevaleresque
chevaleresquement
clownesque
dantesque
faunesque
gargantuesque
gigantesque
grotesque
grotesquement
guignolesque
ingresque

lemme
livresque
mauresque
moliéresque
picaresque
pittoresque
pittoresquement
plateresque
prudhommesque
pédantesque
rocambolesque
romanesque
simiesque
titanesque
tudesque
vaudevillesque
éléphantesque

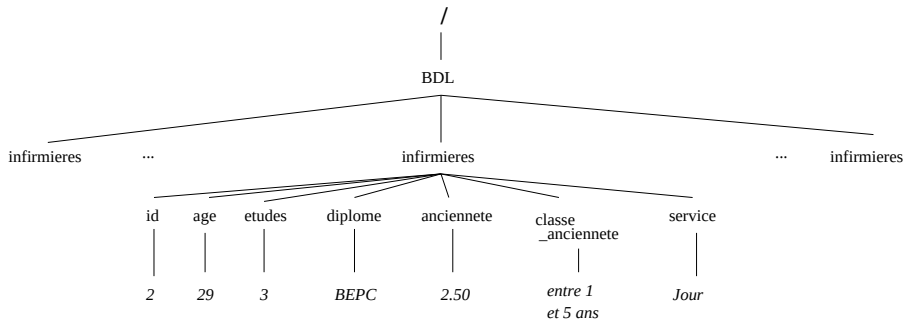


FIGURE 9 – Table infirmieres en XML

à la section précédente qui se limitaient à la réorganisation de chaînes de caractères. On peut en effet, étant donné un arbre, le manipuler comme un objet structuré. Les remodelages possibles prennent alors une autre ampleur.

XSLT constitue un langage de transformations des arbres que sont des documents XML. Son nom (*eXensible* Stylesheet *Language Transformations*) peut donner le sentiment qu'il s'agit de pouvoir associer aux documents des styles, comme on le fait dans les traitements de textes. XSLT permet en fait de remodeler en profondeur des arbres, comme nous allons le voir. Nous n'allons pas présenter dans le détail XSLT. On se reportera plutôt à (Amann & Rigaux, 2002) et à (Mangano, 2003). Nous essayons de donner une idée des possibilités offertes par ce langage.

5.1. Transformer des arbres

L'export en XML de la table infirmieres donne naissance au document que voici :

```
<?xml version="1.0" encoding="utf-8" ?>
<BDL>
```

TABLEAU 147 – ESQUE : discordances Frantext (Lexique) et DELA

DELA pas Frantext
R^2_{120} p. 465

lemme
More
aristophanesque
burlesquement
calembouresque
caméléonesque
cannibalesque
feuilletonesque
feuilletonnesque
flandresque
gagesque
gigantesquement
grand-guignolesque
grotesques
hispano-moresque
humoresque
livresquement
madrigalesque
maure
mauresquement
michélangelesque
monalisesque
more
niagaresque
nirvanesque
prudhommesquement
pédantesquement
péruginesque
rocambolesquement
romanesquement
rouletabillesque
soldatesquement
somnambulesque
sultanesquement

Frantext pas DELA
R^2_{121} p. 467

lemme
alibabesque
animalesque
aubignesque
aviationnesque
babelesque
barytonesque
bignolesque
botticellesque
boulardesque
cafardesque
canovesque
cassandresque
chewinggumesque
chichinesque
clochemerlesque
cornichonnesque
courtelinesque
crapaudesque
cyranesque
dantonesque
demillesque
diablesque
donquichottesque
fallacesque
fantômastesque
fiesque
footballesque
fourmillesque
gendarmesque
gionesque
giorgionesque
goeringesque
goyesque
gracianesque

lemme
grottesque
himalayesque
hugolesque
kagébesque
karabesque
latudesque
levesque
loyolesque
léonardesque
mahonesque
mirlitonesque
palestiniesque
perrichonesque
pharaonesque
pluchesque
poesque
prudhonesque
rafinesque
sauvagesque
scoutesque
seicentesque
simonesque
statuesque
taglionesque
tintamarresque
truandesque
turlupinesque
vampiresque
vaticanesque
zigantesque
zingaresque
zombiesque
zurbaranesque

```
...
   <infirmieres>
        <id>2</id>
        <age>29</age>
        <etudes>3</etudes>
        <diplome>BEPC</diplome>
        <anciennete>2.50</anciennete>
        <classe_anciennete>entre 1 et 5 ans</classe_anciennete>
        <service>Jour</service>
    </infirmieres>
...
</BDL>
```


La figure 9 p. 468 visualise l'arbre noté en XML par les enchâssements d'éléments, eux-mêmes délimités par des balises ouvrante, entre chevrons, et fermante, où le nom de l'élément est précédé d'un chevron ouvrant et d'une oblique, et suivi d'un chevron fermant. L'élément BDL domine des éléments infirmieres. L'élément infirmieres de l'exemple domine lui-même 7 éléments : id, age, etudes, diplome, anciennete, classe_anciennete et service. L'élément id précède immédiatement l'élément age et précède, immédiatement ou non, les éléments age, etudes, diplome, anciennete, classe_anciennete et service.

On notera qu'un document traité par XSLT comporte une racine additionnelle qui domine l'élément racine du document XML originel. La racine du document XML est l'élément BDL. Il se trouve dominé dans la figure 9 p. 468 par une racine supplémentaire, de nom / (oblique). Cette racine supplémentaire se justifie par la nécessité de pouvoir traiter un arbre XML même lorsqu'on ne connaît pas le nom de son élément racine. Il faut en quelque sorte pouvoir « tenir cet arbre par la racine ».

Notre première utilisation de XSLT vise à produire, à partir du document XML issu de la table infirmieres, un tableau HTML, visualisable avec un navigateur quelconque.

Les règles XSLT ci-dessous sont groupées en un document XML de racine xsl:stylesheet. Chaque règle constitue un élément xsl:template dont l'attribut match indique la cible, c'est-à-dire la partie de l'arbre XML qu'elle transforme.

La première règle a pour cible la racine, l'oblique, du document XML traité par XSLT. Lorsque cet élément est rencontré, la structure d'ensemble d'une page HTML est engendrée : la racine html domine un noeud body. Cet élément body domine à son tour un noeud h1, c'est-à-dire un titre de niveau 1 et un élément table, c'est-à-dire un tableau HTML. Un tableau HTML est constitué d'éléments tr (pour *table row*, ligne de tableau), pour chaque ligne, qui dominent eux-mêmes des éléments td, pour chaque cellule. Lorsqu'est rencontrée la racine du document, est engendrée une table au sein d'un document HTML. Cette table comprend une première ligne, où chaque cellule représente un en-tête de colonne, en italiques, dans la mesure où le texte au sein de la cellule est dominé par un élément i. Après cette première ligne, cette première règle fait appel à une instruction XSLT xsl:apply–templates, qui indique qu'il faut appliquer à la descendance du noeud courant toutes les règles pertinentes et intégrer leur résultat.

La deuxième règle indique que la rencontre d'un noeud infirmieres doit déclencher l'engendrement d'une nouvelle ligne, c'est-à-dire d'un élément HTML tr, qui va dominer le résultat de l'application des règles pertinentes « en dessous ».

Les trois dernières règles s'appliquent aux éléments dominés par un noeud infirmieres dans le document XML.

La règle 3 s'applique à un élément id et engendre une cellule qui contient, en gras, l'identifiant de l'infirmière.

La règle 4 s'applique à un noeud age ou diplome ou classe_anciennete ou service. Elle engendre une cellule contenant ce que ce noeud domine dans l'arbre XML traité.

La dernière règle fait la même chose pour les éléments etude ou anciennete mais indique que le texte de la cellule doit être aligné à droite, dans la mesure où il s'agit de mettre en page des nombres.

Ces règles comportent des fragments de type &#<nombre>; qui sont une des manières de noter pour HTML les caractères accentués.

La figure 10 p. 472 donne une représentation graphique de ces règles. Pour chaque règle, on trouve à gauche de la flèche de réécriture en gras, l'arbre-cible dans le document XML, et à droite, le fragment d'arbre HTML engendré.

```
<?xml version="1.0" encoding="utf-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="utf-8"/>



<!-- R1 -->
<xsl:template match="/">
    <html>
    <body>
    <h1 align="center">Infirmi&#232;res</h1>
    <table border="1">
    <tr>
     <td><i>Identifiant</i></td>
     <td><i>&#226;ge</i></td>
     <td><i>ann&#233;es d'&#233;tudes</i></td>
␣␣␣␣<td><i>dipl&#244;me</i></td>
␣␣␣␣<td><i>anciennet&#233;</i></td>
␣␣␣␣<td><i>classe␣d'anciennet&#233;</i></td>
     <td><i>service</i></td>
    </tr>
    <xsl:apply-templates/>
    </table>
    </body>
    </html>
  </xsl:template>


 <!-- R2 -->
 <xsl:template match="infirmieres">
     <tr><xsl:apply-templates/></tr>
  </xsl:template>



<!-- R3 -->
  <xsl:template match="id">
     <td><b><xsl:value-of select="."/></b></td>
  </xsl:template>
```

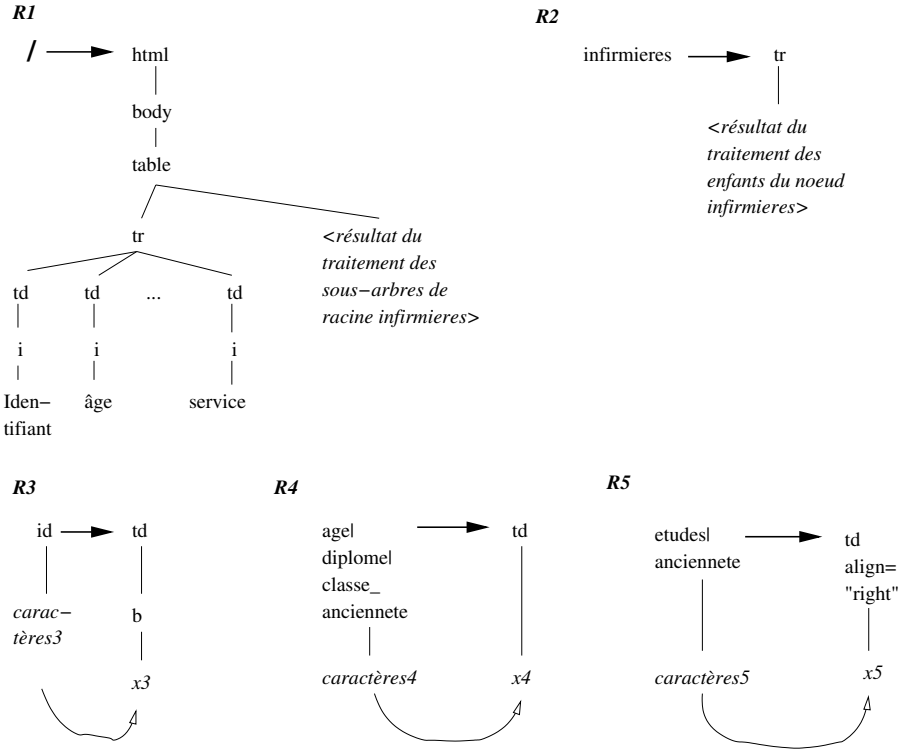


FIGURE 10 – Table infirmieres : règles XSLT de transformation en tableau HTML


```
<!-- R4 -->
  <xsl:template match="age|diplome|classe_anciennete|service">
     <td><xsl:value-of select="."/></td>
  </xsl:template>


 <!-- R4 -->
 <xsl:template match="etudes|anciennete">
     <td align="right"><xsl:value-of select="."/></td>
 </xsl:template>


</xsl:stylesheet>
```

Un certain nombre de programmes prennent en entrée les règles XSLT et le document XML et appliquent les règles à ce document. On se reportera par exemple à (Fitzgerald, 2004) pour un aperçu selon les environnements. C'est le cas, sous Unix/Linux, de xsltproc :

xsltproc TableInfirmieres1.xsl infirmieres.xml > TableInfirmieres1.html

avec TableInfirmieres1.xsl pour les règles XSLT qui viennent d'être présentées et infirmieres.xml, document XML export de la table de même nom. Le résultat de l'application des règles figure dans le fichier HTML TableInfirmieres1.html, dont est fourni un extrait :

```
<html>
```

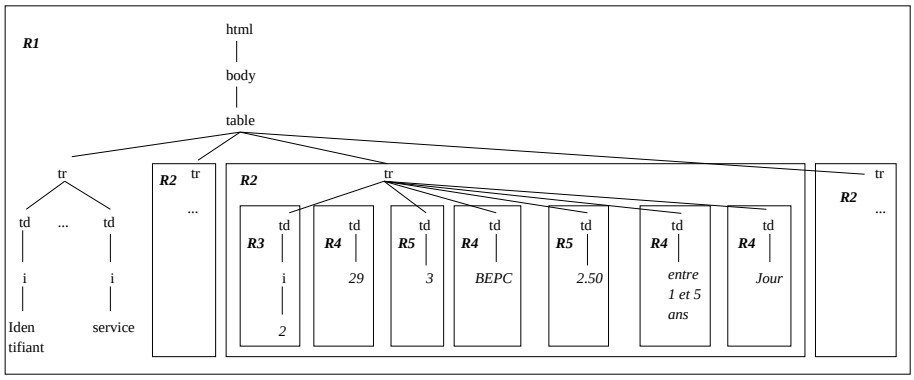


FIGURE 11 – Table infirmieres : transformation en tableau HTML via XSLT


```
<body>
<h1>Infirmières</h1>
<table border="1">
<tr>
<td><i>Identifiant</i></td>
<td><i>âge</i></td>
<td><i>années d'études</i></td>
<td><i>diplôme</i></td>
<td><i>ancienneté</i></td>
<td><i>classe d'ancienneté</i></td>
<td><i>service</i></td>
</tr>
...
    <tr>
        <td><b>2</b></td>
        <td>29</td>
        <td align="right">3</td>
        <td>BEPC</td>
        <td align="right">2.50</td>
        <td>entre 1 et 5 ans</td>
        <td>Jour</td>
    </tr>
...
</table>
</body>
</html>
```

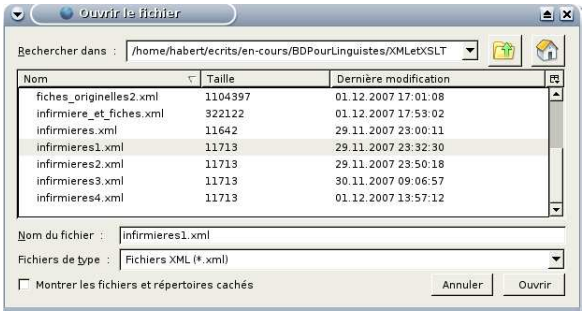
La figure 11 p. 473 est l'équivalent graphique de ce fichier et facilite la compréhension de la transformation de l'arbre XML de la figure 9 p. 468 via les règles de la figure 10 p. 472. Un encadré avec le nom de la règle en haut à gauche matérialise l'action d'une règle donnée.

Les navigateurs actuels peuvent effectuer « à la volée » la transformation d'un arbre XML au moyen de règles XSLT. Il suffit alors d'ajouter au fichier contenant le document XML, ici infirmieres1.xml, une ligne indiquant au navigateur qu'il doit employer le fichier de règles dont le nom est fourni pour produire une page HTML visualisable et navigable. On constate en 20 qu'on ouvre un fichier XML, celui dont le début figure ci-dessous. Le résultat, en 21 est le tableau HTML produit grâce à l'application, par le navigateur, des règles du fichier TableInfirmieres1.xsl au fichier infirmieres1.xml. On remarque que l'identifiant a effectivement été

mis en gras (R4) et les nombres issus des colonnes etudes et anciennete alignés à droite (R5).

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="application/xml" href="TableInfirmieres1.xsl"?>
<BDL>

...
</BDL>
```

20



21

Infirmières

Identifiant	*âge*	*années d'études*	*diplôme*	*ancienneté*	*classe d'ancienneté*	*service*
41	26	3	BEPC	0.00	moins de 1 an	
2	29	3	BEPC	2.50	entre 1 et 5 ans	Jour
4	31	3	BEPC	7.00	plus de 5 ans	Jour
6	30	4	BEPC	7.00	plus de 5 ans	Jour
9	27	3	BEPC	3.00	entre 1 et 5 ans	Jour
13	34	3	BEPC	3.00	entre 1 et 5 ans	Jour
18	27	4	BEPC	2.50	entre 1 et 5 ans	Jour
19	28	3	BEPC	5.00	entre 1 et 5 ans	Jour
21	31	4	BEPC	8.00	plus de 5 ans	Jour
24	26	4	BEPC	0.00	moins de 1 an	Jour
32	26	3	BEPC	0.00	moins de 1 an	Jour
34	31	4	BEPC	0.00	moins de 1 an	Jour
36	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
39	33	4	BEPC	3.00	entre 1 et 5 ans	Jour
43	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
46	21	3	BEPC	0.00	moins de 1 an	Jour
61	26	4	BEPC	3.00	entre 1 et 5 ans	Jour

5.2. Contrôler les transformations d'arbres

XSLT offre des moyens de contrôler finement la transformation d'un arbre XML donné. Les règles ci-dessous précisent le traitement de certains noeuds du document infirmieres.xml. Sans entrer dans le détail de la syntaxe de XSLT, indiquons simplement que la règle l. 10 ayant pour cible les éléments de type service produit une cellule HTML (td) dans tous les cas, mais elle comprend une condition (l. 11-18) : la chaîne représentant le service de l'infirmière est en italiques quand il s'agit de 'Nuit' (l. 15-18) et en gras quand il s'agit de 'Jour' (l. 12-14). De manière similaire, l. 22, pour les éléments de type classe_anciennete, la valeur 'moins de un an' est dans une taille de caractères plus petite (<font size='-1'>) que la taille normale des caractères du document HTML produit, la valeur 'plus de 5 ans' en caractères plus gros (<font size='+1'>).

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

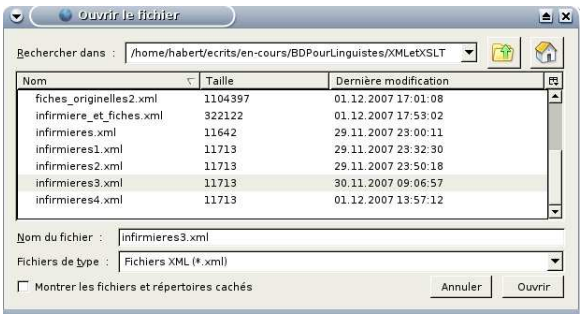
```
<xsl:output method="html" encoding="utf-8"/>
...
  <xsl:template match="age|diplome">
     <td><xsl:value-of select="."/></td>
  </xsl:template>


  <xsl:template match="service">
     <xsl:choose>
       <xsl:when test = ".= 'Jour'">
          <td><b><xsl:value-of select="."/></b></td>
       </xsl:when>
       <xsl:otherwise>
          <td><i><xsl:value-of select="."/></i></td>
       </xsl:otherwise>
     </xsl:choose>
   </xsl:template>


  <xsl:template match="classe_ancienne">
     <xsl:choose>
       <xsl:when test = ".= 'moins de 1 an'">
          <td><font size='-1'><xsl:value-of select="."/></font></td>
       </xsl:when>
       <xsl:when test = ".= 'plus de 5 ans'">
          <td><font size='+1'><xsl:value-of select="."/></font></td>
       </xsl:when>
       <xsl:otherwise>
          <td><xsl:value-of select="."/></td>
       </xsl:otherwise>
     </xsl:choose>
   </xsl:template>
...
</xsl:stylesheet>
```

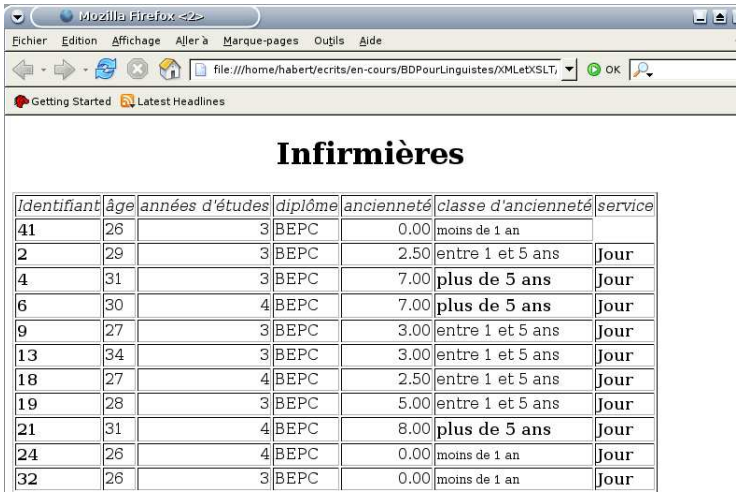
On « attache » au document XML infirmieres3.xml ces règles :

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="application/xml" href="TableInfirmieres3.xsl"?>
```

Lorsqu'on ouvre avec un navigateur ce fichier XML 22, on obtient un tableau 23 qui manifeste l'application de ces règles qui comportent des conditions.

22





Infirmières

Identifiant	*âge*	*années d'études*	*diplôme*	*ancienneté*	*classe d'ancienneté*	*service*
41	26	3	BEPC	0.00	moins de 1 an	
2	29	3	BEPC	2.50	entre 1 et 5 ans	Jour
4	31	3	BEPC	7.00	plus de 5 ans	Jour
6	30	4	BEPC	7.00	plus de 5 ans	Jour
9	27	3	BEPC	3.00	entre 1 et 5 ans	Jour
13	34	3	BEPC	3.00	entre 1 et 5 ans	Jour
18	27	4	BEPC	2.50	entre 1 et 5 ans	Jour
19	28	3	BEPC	5.00	entre 1 et 5 ans	Jour
21	31	4	BEPC	8.00	plus de 5 ans	Jour
24	26	4	BEPC	0.00	moins de 1 an	Jour
32	26	3	BEPC	0.00	moins de 1 an	Jour

Le fichier HTML engendré « à la volée » est de la forme :

```
<html>
<body>
<h1>Infirmières</h1>
<table border="1">
<tr>
<td><i>Identifiant</i></td>
<td><i>âge</i></td>
<td><i>années d'études</i></td>
<td><i>diplôme</i></td>
<td><i>ancienneté</i></td>
<td><i>classe d'ancienneté</i></td>
<td><i>service</i></td>
</tr>
...

    <tr>
        <td><b>6</b></td>
        <td>30</td>
        <td align="right">4</td>
        <td>BEPC</td>
        <td align="right">7.00</td>
        <td><font size="+1">plus de 5 ans</font></td>
        <td><b>Jour</b></td>
    </tr> ...
</table>
</body>
</html>
```

XSLT permet également d'effectuer des tris. La règle ci-dessous stipule l. 20-21 qu'il faut trier les éléments de type infirmieres d'abord par ordre croissant du contenu (textuel) des éléments service puis par ordre croissant du contenu (numérique) des éléments anciennete avant d'appeler des règles (**<xsl:apply–templates>** - l. 23) sur les éléments dominés par les noeuds infirmieres.

```
<?xml version="1.0" encoding="utf–8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <
    xsl:output method="html" encoding="iso–8859–1"/>
```

```
    <xsl:template match="/BDL">
     <html>
     <body>
     <h1>Infirmi&#232;res</h1>
     <table border="1">
     <tr>
        <td><i>Identifiant</i></td>
        <td><i>&#226;ge</i></td>
        <td><i>ann&#233;es d'&#233;tudes</i></td>
␣␣␣␣␣␣␣<td><i>dipl&#244;me</i></td>
␣␣␣␣␣␣␣<td><i>anciennet&#233;</i></td>
␣␣␣␣␣␣␣<td><i>classe␣d'anciennet&#233;</i></td>
        <td><i>service</i></td>
       </tr>
     <xsl:for-each select="infirmieres">
        <xsl:sort select="./service" data-type="text" order="ascending"/>
        <xsl:sort select="./anciennete" data-type="number" order="ascending"/>
        <tr>
           <xsl:apply-templates/>
        </tr>
     </xsl:for-each>
     </table>
     </body>
     </html>
  </xsl:template>
...
</xsl:stylesheet>
```

On attache de la même manière au fichier infirmieres4.xml les règles constituant le fichier TableInfirmieres4.xsl :

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="application/xml" href="TableInfirmieres4.xsl"?>
```

Les deux écrans qui suivent montrent le début et la fin du résultat par rapport à l'écran 23. On constate que les tris successifs sont effectués. Les mises en valeur différentes des valeurs de service et d'anciennete associées aux tris montrent par exemple, dans le deuxième écran, la part prépondérante des infirmières ayant plus de 5 ans d'ancienneté au sein des infirmières de nuit.

24

Infirmières

Identifiant	*âge*	*années d'études*	*diplôme*	*ancienneté*	*classe d'ancienneté*	*service*
41	26	3	BEPC	0.00	moins de 1 an	
24	26	4	BEPC	0.00	moins de 1 an	**Jour**
32	26	3	BEPC	0.00	moins de 1 an	**Jour**
34	31	4	BEPC	0.00	moins de 1 an	**Jour**
46	21	3	BEPC	0.00	moins de 1 an	**Jour**
67	25	4	BEPC	0.00	moins de 1 an	**Jour**
68	24	4	BEPC	0.00	moins de 1 an	**Jour**
70	29	3	BEPC	0.00	moins de 1 an	**Jour**
81	25	3	BEPC	0.00	moins de 1 an	**Jour**
97	25	3	BEPC	0.00	moins de 1 an	**Jour**
33	34	4	BAC	0.00	moins de 1 an	**Jour**
65	21	3	BEPC	0.50	moins de 1 an	**Jour**
62	33	3	BEPC	1.00	entre 1 et 5 ans	**Jour**
73	29	3	BAC	1.50	entre 1 et 5 ans	**Jour**
36	26	4	BEPC	2.00	entre 1 et 5 ans	**Jour**
43	26	4	BEPC	2.00	entre 1 et 5 ans	**Jour**
2	29	3	BEPC	2.50	entre 1 et 5 ans	**Jour**

25

19	28	3	BEPC	5.00	entre 1 et 5 ans	**Jour**
4	31	3	BEPC	7.00	**plus de 5 ans**	**Jour**
6	30	4	BEPC	7.00	**plus de 5 ans**	**Jour**
21	31	4	BEPC	8.00	**plus de 5 ans**	**Jour**
99	31	4	BEPC	8.50	**plus de 5 ans**	**Jour**
22	31	3	BAC	9.00	**plus de 5 ans**	**Jour**
17	38	4	BAC	14.00	**plus de 5 ans**	**Jour**
8	44	3	BEPC	19.00	**plus de 5 ans**	**Jour**
1	23	4	BEPC	0.00	moins de 1 an	*Nuit*
44	29	3	BEPC	2.00	entre 1 et 5 ans	*Nuit*
47	30	3	BEPC	3.00	entre 1 et 5 ans	*Nuit*
3	34	3	BEPC	5.00	entre 1 et 5 ans	*Nuit*
20	28	3	BEPC	6.00	**plus de 5 ans**	*Nuit*
16	30	4	BEPC	6.50	**plus de 5 ans**	*Nuit*
58	31	3	BEPC	8.00	**plus de 5 ans**	*Nuit*
10	35	3	BEPC	11.00	**plus de 5 ans**	*Nuit*
14	40	3	BEPC	14.00	**plus de 5 ans**	*Nuit*
28	38	3	BEPC	17.00	**plus de 5 ans**	*Nuit*
7	39	4	BEPC	19.00	**plus de 5 ans**	*Nuit*
12	40	3	BEPC	20.00	**plus de 5 ans**	*Nuit*

5.3. Relier des arbres transformés

On vient de le constater, un document XML muni de règles XSLT appropriées « devient » pour le navigateur qui sert à l'ouvrir un document HTML comme un autre. On a ainsi obtenu des tables HTML avec jeu sur les tailles de caractères et sur l'opposition maigre/gras et droit/italiques.

Rien n'empêche dès lors d'engendrer avec XSLT à partir de documents XML des documents HTML navigables, c'est-à-dire munis d'une part de liens hypertextuels vers d'autres documents et d'autre part d'ancres, c'est-à-dire de points d'arrivée spécifiques d'un lien hypertextuel.

Rappelons qu'en HTML, un élément de type :

<a name="X"**>** . . . ****

place une ancre de nom X au sein d'un document HTML. Par exemple, au sein du document bebe_et_fiches.xml :

<a name="*bb3*">...</**a**>

crée une ancre de nom bb3.

Par ailleurs, en HTML, un élément de type :

<a href="*Y*">...</**a**>

crée un lien hypertextuel (manifesté souvent par un soulignement) vers le document Y. Ainsi :

<a href="*bebe_et_fiches.xml*">...</**a**>

crée-t-il un lien vers le document bebe_et_fiches.xml tandis que :

<a href="*bebe_et_fiches.xml#bb3*">...</**a**>

installe un lien dont le point d'arrivée est l'ancre bb3 dans le document bebe_et_fiches.xml. Le caractère # sépare le document cible de l'ancre, le point précis d'arrivée du lien dans ce document cible.

Les quatre écrans qui suivent montrent la navigation entre trois documents XML via des règles XSLT. Le premier écran est issu de l'application des règles TableBebes1.xsl au fichier bebe_et_fiches.xml, le second et le troisième de celle des règles TableFiches1.xsl au fichier fiches_originelles2.xml, le quatrième de celle des règles TableInfirmieres5.xsl au fichier infirmiere_et_fiches.xml.

Dans le premier écran, pour chaque bébé sont fournies les informations signalétiques le concernant et des liens sur les fiches idoines. Si l'on clique sur l'un de ces liens, ici par exemple sur celui correspondant à la fiche 16 concernant le bébé 3, on arrive dans une visualisation HTML du fichier fiches_originelles2.xml, précisément à l'endroit concernant la fiche 16.

Les deuxième et troisième écran montrent la visualisation de cette fiche. On note en particulier un lien bb3 permettant d'aller dans la visualisation du document bebe_et_fiches.xml à l'endroit des informations sur le bébé 3 tandis que le lien inf43 a le même rôle, mais pour retrouver l'infirmière 43 dans la visualisation du fichier infirmiere_et_fiches.xml. La visualisation de ce dernier fichier (quatrième écran) est similaire à celle de bebe_et_fiches.xml : signalétique de chaque infirmière et liens sur les fiches qu'elle a rédigées.

L'application de règles XSLT aux trois fichiers XML en fait un ensemble de documents HTML reliés de manière précise entre eux, ce qui permet d'aller d'un bébé ou d'une infirmière aux fiches correspondantes ou d'une fiche à l'infirmière ou au bébé en cause.

26

Bébés

Bébé 2

Sexe Garçon, accouchement voie basse, naissance non locale, poids 745, classe poids < 750 g., terme 27.29, terme normalisé 27, classe >= 26 et < 28 SA, CRIB 4, classe 3 <= CRIB <= 6, jour 5399, date 1, SM1 12, classe 11 <= SM <=20, SM3 11, classe 11 <= SM <=20, SM7 6, classe <= 10, SM15 8, classe <= 10, J1 Centre Hospitalier X, J3 Centre Hospitalier X, J7 Centre Hospitalier X, J15 Centre Hospitalier X
Fiches 1 2 3 4 5 6 7 8 9 10 11 12

Bébé 3

Sexe Fille, accouchement voie basse, naissance non locale, poids 1010, classe poids > 1 000 g., terme 27.00, terme normalisé 27, classe >= 26 et < 28 SA, CRIB 2, classe <= 2, jour 6399, date 2, SM1 19, classe 11 <= SM <=20, SM3 16, classe 11 <= SM <=20, SM7 14, classe 11 <= SM <=20, SM15 12, classe 11 <= SM <=20, J1 Centre Hospitalier X, J3 Centre Hospitalier X, J7 Centre Hospitalier X, J15 Centre Hospitalier X
Fiches 13 14 15 16 17 18 19 20 21 22 23

TABLEAU 148 – PRÉMA : signalétique et fiches pour bébés 2 et 3

bebe	*fiche*	*sexe*	*accouchement*	*lieu_naissance*	*poids_naissance*	...	*lieu_jour15*
2	1	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
2	2	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
2	3	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
2	4	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
2	5	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
2	6	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
2	7	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
2	8	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
2	9	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
2	10	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
2	11	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
2	12	Garçon	voie basse	non locale	745	...	Centre Hospitalier X
3	13	Fille	voie basse	non locale	1010	...	Centre Hospitalier X
3	14	Fille	voie basse	non locale	1010	...	Centre Hospitalier X
3	15	Fille	voie basse	non locale	1010	...	Centre Hospitalier X
3	16	Fille	voie basse	non locale	1010	...	Centre Hospitalier X
3	17	Fille	voie basse	non locale	1010	...	Centre Hospitalier X
3	18	Fille	voie basse	non locale	1010	...	Centre Hospitalier X
3	19	Fille	voie basse	non locale	1010	...	Centre Hospitalier X
3	20	Fille	voie basse	non locale	1010	...	Centre Hospitalier X
3	21	Fille	voie basse	non locale	1010	...	Centre Hospitalier X
3	22	Fille	voie basse	non locale	1010	...	Centre Hospitalier X
3	23	Fille	voie basse	non locale	1010	...	Centre Hospitalier X

27

16	Bébé très mignonne, réactive pendant les soins malgré une sédation. Bouge les bras et les jambes. Fait des grimaces - Garde les yeux fermés	bb3	3	11.00	1010	poids > 1 000 g.	plat dos	dos	hypnovel ou fentanyl	

28

', réactive es soins ne Bouge t les ait des - Garde ermés	bb3	3	11.00	1010	poids > 1 000 g.	plat dos	dos	hypnovel ou fentanyl	intubé	inf43

29

Infirmière 43

âge 26 études 4 diplôme BEPC ancienneté 2.00 service Jour
Fiches 16 21 139 143 148 201 205 215 289 296 363 386

Pour obtenir la visualisation souhaitée d'un bébé, on doit connaître les fiches qui lui correspondent. Si l'on revient à la base de données PRÉMA, c'est ce que permet une jointure entre la table bebes et la table signaletique_fiches. De la table signaletique_fiches, on ne garde dans cette jointure que le numéro de la fiche. On obtient le tableau 148 p. 480. Cette table est « redondante » : elle donne toutes les fiches concernant un bébé donné au prix de la répétition à chaque fois de l'ensemble des informations signalétiques concernant ce bébé.

On mémorise dans la table bebe_et_fiches le résultat de cette requête :

```
CREATE TABLE bebe_et_fiches
     SELECT b.id AS 'bebe',
            f.id AS 'fiche',
            sexe,
            accouchement,
            lieu_naissance,
            poids_naissance,
            classe_poids_naissance,
            terme,
            terme_normalise,
            classe_terme,
            CRIB,
            classe_CRIB,
            b.jour,
            mois,
            annee,
            jour_experience,
            SM1,
            classe_SM1,
            SM3,
            classe_SM3,
            SM7,
            classe_SM7,
            SM15,
            classe_SM15,
            lieu_jour1,
            lieu_jour3,
            lieu_jour7,
            lieu_jour15
       FROM bebes AS b,
          signaletique_fiches AS f
       WHERE b.id = id_bebe
       ORDER BY b.id, f.id ;
```

On exporte en XML cette table dans le fichier bebe_et_fiches.xml :

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="application/xml" href="TableBebes1.xsl"?>
<bdl>
    <bebe_et_fiches>
        <bebe>2</bebe>
        <fiche>1</fiche>
        <sexe>GarÃ§on</sexe>
        <accouchement>voie basse</accouchement>
        <lieu_naissance>non locale</lieu_naissance>
        <poids_naissance>745</poids_naissance>
        <classe_poids_naissance>poids &lt; 750 g.</classe_poids_naissance>
        <terme>27.29</terme>
        <terme_normalise>27</terme_normalise>
        <classe_terme>&gt;= 26 et &lt; 28 SA</classe_terme>
        <CRIB>4</CRIB>
        <classe_CRIB>3 &lt;= CRIB &lt;= 6</classe_CRIB>
        <jour>5</jour>
        <mois>3</mois>
        <annee>99</annee>
        <jour_experience>1</jour_experience>
        <SM1>12</SM1>
```

```
        <classe_SM1>11 &lt;= SM &lt;=20</classe_SM1>
        <SM3>11</SM3>
        <classe_SM3>11 &lt;= SM &lt;=20</classe_SM3>
        <SM7>6</SM7>
        <classe_SM7>&lt;= 10</classe_SM7>
        <SM15>8</SM15>
        <classe_SM15>&lt;= 10</classe_SM15>
        <lieu_jour1>Centre Hospitalier X</lieu_jour1>
        <lieu_jour3>Centre Hospitalier X</lieu_jour3>
        <lieu_jour7>Centre Hospitalier X</lieu_jour7>
        <lieu_jour15>Centre Hospitalier X</lieu_jour15>
    </bebe_et_fiches>
    <bebe_et_fiches>
        <bebe>2</bebe>
        <fiche>2</fiche>
        <sexe>GarÃ§on</sexe>
        <accouchement>voie basse</accouchement>
        <lieu_naissance>non locale</lieu_naissance>
        <poids_naissance>745</poids_naissance>
        <classe_poids_naissance>poids &lt; 750 g.</classe_poids_naissance>
        <terme>27.29</terme>
        <terme_normalise>27</terme_normalise>
        <classe_terme>&gt;= 26 et &lt; 28 SA</classe_terme>
        <CRIB>4</CRIB>
        <classe_CRIB>3 &lt;= CRIB &lt;= 6</classe_CRIB>
        <jour>5</jour>
        <mois>3</mois>
        <annee>99</annee>
        <jour_experience>1</jour_experience>
        <SM1>12</SM1>
        <classe_SM1>11 &lt;= SM &lt;=20</classe_SM1>
        <SM3>11</SM3>
        <classe_SM3>11 &lt;= SM &lt;=20</classe_SM3>
        <SM7>6</SM7>
        <classe_SM7>&lt;= 10</classe_SM7>
        <SM15>8</SM15>
        <classe_SM15>&lt;= 10</classe_SM15>
        <lieu_jour1>Centre Hospitalier X</lieu_jour1>
        <lieu_jour3>Centre Hospitalier X</lieu_jour3>
        <lieu_jour7>Centre Hospitalier X</lieu_jour7>
        <lieu_jour15>Centre Hospitalier X</lieu_jour15>
    </bebe_et_fiches> ...
</bdl>
```

Il n'y a qu'une seule règle de transformation. Cette règle examine chaque élément de type bebe_et_fiches successivement. Si l'identifiant du bébé, c'est-à-dire la valeur de l'élément bebe, diffère de l'identifiant de l'élément bebe_et_fiches précédent (l. 11), on a affaire à un bébé non encore rencontré. On engendre une ancre de la forme : b<*numéro du bébé*> pour ce bébé (l. 13-18). On affiche les élements signalétiques de ce bébé, on passe à la ligne (l. 21), on écrit Fiches (l. 22) et on crée un lien (l. 23-29), de la forme : fiches_originelles2.xml#f<numéro de fiche>, vers la fiche dont le numéro figure dans l'élément fiche. Si l'identifiant du bébé est le même que celui de l'élément bebe_et_fiches précédent, on ne répéte pas les informations signalétiques. On se contente d'ajouter un lien vers la fiche dont le numéro figure dans l'élément fiche (l. 30-37). On le voit, cette règle, outre l'ancre et les liens qu'elle positionne, élimine les redondances du fichier XML.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <
    xsl:output method="html" encoding="utf-8"/>


   <xsl:template match="/bdl">
    <html>
    <body>
    <h1 align="center">B&#233;b&#233;s</h1>
    <xsl:for-each select="bebe_et_fiches">
       <xsl:choose>
         <xsl:when test="not(preceding-sibling::bebe_et_fiches/bebe=./bebe)">
           <h2>
              <xsl:element name="a">
                <xsl:attribute name="name">
                   <xsl:text>b</xsl:text><xsl:value-of select="./bebe"/>
                </xsl:attribute>
                B&#233;b&#233; <xsl:value-of select="./bebe"/>
              </xsl:element>
           </h2>
           Sexe <xsl:value-of select="./sexe"/>, accouchement <xsl:value-of select="
               ./accouchement"/>, naissance <xsl:value-of select="./lieu_naissance"/
               >, poids <xsl:value-of select="./poids_naissance"/>, classe <
               xsl:value-of select="./classe_poids_naissance"/>, terme <xsl:value-of
                select="./terme"/>, terme normalis&#233; <xsl:value-of select="./
               terme_normalise"/>, classe <xsl:value-of select="./classe_terme"/>,
               CRIB <xsl:value-of select="./CRIB"/>, classe <xsl:value-of select="./
               classe_CRIB"/>, jour <xsl:value-of select="./jour"/> <xsl:value-of
               select="./mois"/> <xsl:value-of select="./annee"/>, date  <
               xsl:value-of select="./jour_experience"/>, SM1 <xsl:value-of select="
               ./SM1"/>, classe <xsl:value-of select="./classe_SM1"/>, SM3 <
               xsl:value-of select="./SM3"/>, classe <xsl:value-of select="./
               classe_SM3"/>, SM7 <xsl:value-of select="./SM7"/>, classe  <
               xsl:value-of select="./classe_SM7"/>, SM15  <xsl:value-of select="./
               SM15"/>, classe  <xsl:value-of select="./classe_SM15"/>, J1 <
               xsl:value-of select="./lieu_jour1"/>, J3 <xsl:value-of select="./
               lieu_jour3"/>, J7 <xsl:value-of select="./lieu_jour7"/>, J15 <
               xsl:value-of select="./lieu_jour15"/>
<br/>
Fiches
<xsl:element name="a">
               <xsl:attribute name="href">
                  <xsl:text>fiches_originelles2.xml#f</xsl:text><xsl:value-of select
                      ="./fiche"/>
                </xsl:attribute>
                <xsl:text> </xsl:text><xsl:value-of select="./fiche"/>
           </xsl:element>
         </xsl:when>
        <xsl:otherwise>
           <xsl:element name="a">
               <xsl:attribute name="href">
                  <xsl:text>fiches_originelles2.xml#f</xsl:text><xsl:value-of select
                      ="./fiche"/>
                </xsl:attribute>
                <xsl:text> </xsl:text><xsl:value-of select="./fiche"/>
           </xsl:element>
        </xsl:otherwise>
```

```
        </xsl:choose>
              </xsl:for-each>
        </body>
        </html>
42    </xsl:template>


</xsl:stylesheet>
```

Si l'on engendre effectivement un fichier HTML au moyen de ces règles, au lieu de se reposer sur un navigateur pour qu'il l'effectue à la volée, on obtient (cf. écran 26) :

```
 1  <html>
    <body>
    <h1 align="center">BÃ©bÃ©s</h1>
    <h2><a name="b2">                    BÃ©bÃ© 2</a></h2>
              Sexe GarÃ§on, accouchement voie basse, naissance non locale, poids 745,
                  classe poids &lt; 750 g., terme 27.29, terme normalisÃ© 27, classe &
                  gt;= 26 et &lt; 28 SA, CRIB 4, classe 3 &lt;= CRIB &lt;= 6, jour
                  5399, date  1, SM1 12, classe 11  &lt;= SM &lt;=20, SM3 11, classe 11
                   &lt;= SM &lt;=20, SM7 6, classe  &lt;= 10, SM15  8, classe  &lt;=
                  10, J1 Centre Hospita lier X, J3 Centre Hospitalier X, J7 Centre
                  Hospitalier X, J15 Centre Hospitalier X<br>
 6  Fiches              <a href="fiches_originelles2.xml#f1"> 1</a><a href="
       fiches_originelles2.xml#f2"> 2</a><a href="fiches_originelles␣2.xml#f3"> 3</a><a
        href="fiches_originelles2.xml#f4"> 4</a><a href="fiches_originelles2.xml#f5"> 5
       </a><a href="fiches_origin␣elles2.xml#f6"> 6</a><a href="fiches_originelles2.xml
       #f7"> 7</a><a href="fiches_originelles2.xml#f8"> 8</a><a href="fiches_o␣
       riginelles2.xml#f9"> 9</a><a href="fiches_originelles2.xml#f10"> 10</a><a href="
       fiches_originelles2.xml#f11"> 11</a><a href= "fiches_originelles2.xml#f12"> 12</
       a>
    <h2><a name="b3">                    BÃ©bÃ© 3</a></h2>
              Sexe Fille, accouchement voie basse, naissance non locale, poids 1010,
                  classe poids &gt; 1 000 g., terme 27.00, terme normalisÃ© 27, classe
                  &gt;= 26 et &lt; 28 SA, CRIB 2, classe &lt;= 2, jour 6399, date  2,
                  SM1 19, classe 11 &lt;= SM &l t;=20, SM3 16, classe 11 &lt;= SM &lt
                  ;=20, SM7 14, classe  11 &lt;= SM &lt;=20, SM15  12, classe  11 &lt;=
                   SM &lt;=20, J1 Centre Hospitalier X, J3 Centre Hospitalier X, J7
                  Centre Hospitalier X, J15 Centre Hospitalier X<br>
    Fiches              <a href="fiches_originelles2.xml#f13"> 13</a><a href="
       fiches_originelles2.xml#f14"> 14</a><a href="fiches_origine␣lles2.xml#f15"> 15</
       a><a href="fiches_originelles2.xml#f16"> 16</a><a href="fiches_originelles2.xml#
       f17"> 17</a><a href="fic␣hes_originelles2.xml#f18"> 18</a><a href="
       fiches_originelles2.xml#f19"> 19</a><a href="fiches_originelles.xml#f20"> 20</a>
       < a href="fiches_originelles2.xml#f21"> 21</a><a href="fiches_originelles2.xml#
       f22"> 22</a><a href="fiches_originelles2.xml#f2␣3"> 23</a><h2><a name="b4">
    ...
11  </body>
    </html>
```

On note des caractères « bizarres » dans le HTML engendré ci-dessus : ils relèvent du jeu de caractères UTF-8 alors que le présent ouvrage est composé en ISO-Latin1 (cf. § 1, p. 437).

La démarche est similaire pour les infirmières. Le tableau 149 p. 485 donne la jointure entre infirmieres et signaletique_fiches en ne gardant de cette table que le numéro de fiche. On crée la table infirmiere_et_fiches :

TABLEAU 149 – PRÉMA : signalétique et fiches pour l'infirmière 43

inf	*fiche*	*age*	*etudes*	*diplome*	*anciennete*	*classe_anciennete*	*service*
43	16	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
43	21	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
43	139	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
43	143	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
43	148	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
43	201	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
43	205	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
43	215	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
43	289	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
43	296	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
43	363	26	4	BEPC	2.00	entre 1 et 5 ans	Jour
43	386	26	4	BEPC	2.00	entre 1 et 5 ans	Jour


```
CREATE TABLE infirmiere_et_fiches
    select i.id AS 'inf',
           f.id AS 'fiche',
           age,
           etudes,
           diplome,
           anciennete,
           classe_anciennete,
           service
     FROM infirmieres AS i,
           signaletique_fiches AS f
     WHERE i.id = id_infirmiere
     ORDER BY i.id, id_bebe, jour ;
```

L'export XML, infirmiere_et_fiches.xml, est de la forme :

```
<?xml version="1.0" encoding="utf-8" ?>

<?xml-stylesheet type="application/xml" href="TableInfirmieres5.xsl"?>
<bdl>
    <infirmiere_et_fiches>
        <inf>1</inf>
        <fiche>588</fiche>
        <age>23</age>
        <etudes>4</etudes>
        <diplome>BEPC</diplome>
        <anciennete>0.00</anciennete>
        <classe_anciennete>moins de 1 an</classe_anciennete>
        <service>Nuit</service>
    </infirmiere_et_fiches>
    <infirmiere_et_fiches>
        <inf>1</inf>
        <fiche>636</fiche>
        <age>23</age>
        <etudes>4</etudes>
        <diplome>BEPC</diplome>
        <anciennete>0.00</anciennete>
        <classe_anciennete>moins de 1 an</classe_anciennete>
        <service>Nuit</service>
    </infirmiere_et_fiches>
```

```
...
</bdl>
```

Les règles XSTL placent une ancre sur chaque numéro d'infirmière, de la forme : i<*numéro d'infirmière*>. Elles engendrent des liens pour chacune des fiches de l'infirmière et factorisent par contre la signalétique.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <
    xsl:output method="html" encoding="utf-8"/>


    <xsl:template match="/bdl">
     <html>
     <body>
     <h1 align="center">Infirmi&#232;res</h1>
     <xsl:for-each select="infirmiere_et_fiches">
        <xsl:choose>
          <xsl:when test="not(preceding-sibling::infirmiere_et_fiches/inf=./inf)">
            <h2>
               <xsl:element name="a">
                 <xsl:attribute name="name">
                    <xsl:text>i</xsl:text><xsl:value-of select="./inf"/>
                 </xsl:attribute>
                 Infirmi&#232;re <xsl:value-of select="./inf"/>
               </xsl:element>
            </h2>
            &#226;ge <xsl:value-of select="./age"/> &#233;tudes <xsl:value-of select=
                "./etudes"/> dipl&#244;me <xsl:value-of select="./diplome"/>
                anciennet&#233; <xsl:value-of select="./anciennete"/> service <
                xsl:value-of select="./service"/>
<br/>
Fiches
          <xsl:element name="a">
                <xsl:attribute name="href">
                   <xsl:text>fiches_originelles2.xml#f</xsl:text><xsl:value-of select
                       ="./fiche"/>
                 </xsl:attribute>
                 <xsl:text> </xsl:text><xsl:value-of select="./fiche"/>
            </xsl:element>
         </xsl:when>
        <xsl:otherwise>
            <xsl:element name="a">
                <xsl:attribute name="href">
                   <xsl:text>fiches_originelles2.xml#f</xsl:text><xsl:value-of select
                       ="./fiche"/>
                 </xsl:attribute>
                 <xsl:text>
</xsl:text> <xsl:value-of select="./fiche"/>
            </xsl:element>
       </xsl:otherwise>
      </xsl:choose>
            </xsl:for-each>
     </body>
     </html>
   </xsl:template>
```

```
45
</xsl:stylesheet>
```

Le fichier HTML correspondant serait (cf. écran 29) :

```
<html>
<body>
<h1 align="center">InfirmiÃ¨res</h1>
...
5 <h2><a name="i43">                    InfirmiÃ¨re 43</a></h2>
          Ãge 26 Ã©tudes 4 diplÃ´me BEPC ancienneté 2.00 service Jour<br>
Fiches          <a href="fiches_originelles2.xml#f16"> 16</a><a href="
    fiches_originelles2.xml#f21"> 21</a><a href="fiches_origine␣lles2.xml#f139"> 139
    </a><a href="fiches_originelles2.xml#f143"> 143</a><a href="fiches_originelles2.
    xml#f148"> 148</a><a hre f="fiches_originelles2.xml#f201"> 201</a><a href="
    fiches_originelles2.xml#f205"> 205</a><a href="fiches_originelles2.xml#f21␣5">
    215</a><a href="fiches_originelles2.xml#f289"> 289</a><a href="
    fiches_originelles2.xml#f296"> 296</a><a href="fiches_ori␣ginelles2.xml#f363">
    363</a><a href="fiches_originelles2.xml#f386"> 386</a><h2><a name="i44">

...
10 </body>

</html>
```

Pour les fiches originelles, la démarche est plus simple. L'export fiches_originelles.xml est de la forme :

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="application/xml" href="TableFiches1.xsl"?>
<bdl>
    <fiches_originelles>
        <id>3</id>
        <id_bebe>2</id_bebe>
        <jour>3</jour>
        <heure_saisie>6.00</heure_saisie>
        <poids>680</poids>
        <classe_poids>poids &lt; 750 g.</classe_poids>
        <position>plat dos</position>
        <classe_position>dos</classe_position>
        <sedation>non</sedation>
        <ventilation>intubÃ©</ventilation>
        <id_infirmiere>47</id_infirmiere>
        <frequence_occupation>non rÃ©ponse</frequence_occupation>
        <moral_infirmiere>plutÃ´t bien</moral_infirmiere>
        <pronostic_infirmiere>plutÃ´t bon</pronostic_infirmiere>
        <relation_infirmiere_parents>je ne connais pas les parents</
            relation_infirmiere_parents>
        <relation_mere_bebe>je ne sais pas</relation_mere_bebe>
        <frequence_visites_parents>non rÃ©ponse</frequence_visites_parents>
        <texte>BB tonique pendant que l'on␣s'en occupe. Reaction normale aux soins
            douloureux ou d&#233;sagrable. BB qui veut s'en␣sortir␣–␣combatif␣pour␣
            vivre.</texte>
␣␣␣␣</fiches_originelles>
...
</bdl>
```

Les règles XSLT transforment l'arbre XML en un tableau HTML. Dans ce tableau, une ancre est créée pour chaque fiche, de la forme : f<*numéro de la fiche*>. Deux liens sont engendrés, l'un de contenu bb<*numéro du bébé*> vers l'adresse bebe_et_fiches.xml#b<*numéro du bébé*> et l'autre de contenu inf<*numéro de l'infirmière*> vers l'adresse infirmiere_et_fiches#i<*numéro de l'infirmière*>.

```
<?xml version="1.0" encoding="utf-8"?> <xsl:stylesheet version="1.0" xmlns:xsl="
    http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="utf-8"/>


  <xsl:template match="/">
   <html>
   <body>
   <h1>Fiches</h1>
   <table border="1">
   <tr>
    <td><i>id</i></td>
    <td><i>texte</i></td>
    <td><i>b&#233;b&#233;</i></td>
    <td><i>jour</i></td>
    <td><i>heure saisie</i></td>
    <td><i>poids</i></td>
    <td><i>classe poids</i></td>
    <td><i>position</i></td>
    <td><i>classe position</i></td>
    <td><i>s&#233;dation</i></td>
    <td><i>ventilation</i></td>
    <td><i>infirmi&#232;re</i></td>
    <td><i>fr&#233;quence occupation</i></td>
    <td><i>moral inf.</i></td>
    <td><i>pronostic inf.</i></td>
    <td><i>relation inf./parents</i></td>
    <td><i>relation m&#232;re b&#233;b&#233;</i></td>
    <td><i>fr&#233;quence visites parents</i></td>
   </tr>     <xsl:apply-templates/>
   </table>
   </body>
   </html>
  </xsl:template>


  <xsl:template match="fiches_originelles">
     <tr>
        <xsl:apply-templates select="id"/>
        <xsl:apply-templates select="texte"/>
        <xsl:apply-templates select="id_bebe"/>
        <xsl:apply-templates select="jour"/>
        <xsl:apply-templates select="heure_saisie"/>
        <xsl:apply-templates select="poids"/>
        <xsl:apply-templates select="classe_poids"/>
        <xsl:apply-templates select="position"/>
        <xsl:apply-templates select="classe_position"/>
        <xsl:apply-templates select="sedation"/>
        <xsl:apply-templates select="ventilation"/>
        <xsl:apply-templates select="id_infirmiere"/>
        <xsl:apply-templates select="frequence_occupation"/>
```

```
51          <xsl:apply-templates select="moral_infirmiere"/>
            <xsl:apply-templates select="pronostic_infirmiere"/>
            <xsl:apply-templates select="relation_infirmiere_parents"/>
            <xsl:apply-templates select="relation_mere_bebe"/>
            <xsl:apply-templates select="frequence_visites_parents"/>
56       </tr>
      </xsl:template>


      <xsl:template match="id">
61       <td>
            <xsl:element name="a">
             <xsl:attribute name="name">
                  <xsl:text>f</xsl:text><xsl:value-of select="."/>
             </xsl:attribute>
66           <b><xsl:value-of select="."/></b>
           </xsl:element>
         </td>
      </xsl:template>

71

      <xsl:template match="id_infirmiere">
         <td>
            <xsl:element name="a">
             <xsl:attribute name="href">
76                <xsl:text>infirmiere_et_fiches.xml#i</xsl:text><xsl:value-of select=
                      "."/>
             </xsl:attribute>            inf<xsl:value-of select="."/>
           </xsl:element>
         </td>
      </xsl:template>
81

      <xsl:template match="id_bebe">
         <td>
            <xsl:element name="a">
86           <xsl:attribute name="href">
                  <xsl:text>bebe_et_fiches.xml#b</xsl:text><xsl:value-of select="."/>
             </xsl:attribute>            bb<xsl:value-of select="."/>
           </xsl:element>
         </td>
91    </xsl:template>


      <xsl:template match="classe_poids|position|classe_position|sedation|ventilation|
          frequence_occupation|moral_infirmiere|pronostic_infirmiere|
          relation_infirmiere_parents|relation_mere_bebe|frequence_visites_parents|texte
          ">
         <td><xsl:value-of select="."/></td>
96    </xsl:template>


      <xsl:template match="jour|heure_saisie|poids">
         <td align="right"><xsl:value-of select="."/></td>
101   </xsl:template>
```

```
</xsl:stylesheet>
```

L'arbre HTML engendré par ces règles est de la forme (cf. écrans 27 et 28) :

```
<html>
<body>
<h1>Fiches</h1>
<table border="1">
<tr>
<td><i>id</i></td>
<td><i>texte</i></td>
<td><i>bÃ©bÃ©</i></td>
<td><i>jour</i></td>
<td><i>heure saisie</i></td>
<td><i>poids</i></td>
<td><i>classe poids</i></td>
<td><i>position</i></td>
<td><i>classe position</i></td>
<td><i>sÃ©dation</i></td>
<td><i>ventilation</i></td>
<td><i>infirmiÃ¨re</i></td>
<td><i>frÃ©quence occupation</i></td>
<td><i>moral inf.</i></td>
<td><i>pronostic inf.</i></td>
<td><i>relation inf./parents</i></td>
<td><i>relation mÃ¨re bÃ©bÃ©</i></td>
<td><i>frÃ©quence visites parents</i></td>
</tr>
...
    <tr>
<td><a name="f16"><b>16</b></a></td>
<td>bÃ©bÃ© trÃ¨s mignonne, rÃ©active pendant les soins malgrÃ© une sÃ©dation. Bouge
    les bras et les jambes. Fait des grimace s – Garde les yeux fermÃ©s </td>
<td><a href="bebe_et_fiches.xml#b3">              bb3</a></td>
<td align="right">3</td> <td align="right">11.00</td>
<td align="right">1010</td> <td>poids &gt; 1 000 g.</td>
<td>plat dos</td> <td>dos</td>
<td>hypnovel ou fentanyl</td>
<td>intubÃ©</td>
<td><a href="infirmiere_et_fiches.xml#i43">              inf43</a></td>
<td>rÃ©guliÃ¨rement</td> <td>plutÃ´t bien</td>
<td>plutÃ´t bon</td>
<td>bon</td>
<td>plutÃ´t bonne</td>
<td>trÃ¨s souvent</td>
</tr>
...
</body>
</html>
```

6. Solutions

Solution de l'exercice n°1 p. 461 12 entrées sont supprimées via la requête :

```
DELETE FROM dela_tmp
WHERE entree_depart REGEXP 'duquel'
      OR entree_depart REGEXP 'lequel'
      OR entree_depart REGEXP 'presque'
      OR entree_depart REGEXP 'fresque'
      OR entree_depart REGEXP 'nesquehonite' ;
```

CHAPITRE XIV

⊕ INSTALLATIONS ET MISES EN ROUTE

Ce chapitre indique les étapes nécessaires à l'installation ou à l'utilisation des SGBD utilisés : MySQL et Access.

Pour chaque environnement, il montre comment mettre en place les bases de données exemples à partir des différentes versions fournies dans l'archive en ligne.

L'utilisation de ces trois bases de données est libre à condition d'indiquer à chaque fois l'origine et les publications de références (sans oublier le présent ouvrage) :

- ESQUE Marc Plénat aidé par Nicole Serna (Plénat, 1997) ;
- PHÈDRE Valérie Beaudouin pour ce qui concerne les données dramatiques et métriques (Beaudouin, 2002), Valérie Beaudouin et moi-même pour le (re)découpage en mots graphiques et phonétiques ;
- PRÉMA une équipe de réanimation néonatale. On ne peut en dire plus pour préserver l'anonymat des infirmières comme des bébés concernés. Soulignons qu'un certain nombre de modifications ont été apportées aux données pour rendre l'identification des personnes concernées moins facile.

1. MySQL

1.1. EasyPHP sous Windows

Une application, EasyPHP, fournit d'un coup plusieurs fonctionnalités autour de MySQL :

1. un « noyau » MySQL et la possibilité de formuler directement des requêtes en MySQL ;
2. une interface relativement simple sous forme de pages HTML locales, accessibles donc via le navigateur que l'on préfère ;

3. la possibilité de programmer des applications qui interrogent une base de données et produisent des pages HTML mettant en forme les informations extraites et qui ajoutent et modifient les informations d'une base via des formulaires HTML.

La présentation qui suit se centre sur les deux premiers volets. Le deuxième et le troisième reposent par ailleurs sur la transformation en serveur local de l'ordinateur sur lequel est installé EasyPHP, ce qui permet le traitement de formulaires HTML et l'engendrement de pages HTML dynamiques, c'est-à-dire dont le contenu n'est pas prédéfini une fois pour toutes, mais varie en fonction des informations présentes dans une base de données et de celles fournies par l'utilisateur (via un formulaire). Ces pages HTML dynamiques sont engendrées grâce à un langage de programmation également fourni : PHP.

Un des intérêts d'EasyPHP par rapport à Access est que l'interface graphique sert de la même manière à engendrer des requêtes SQL mais que la requête SQL est fournie en même temps que son résultat, alors qu'avec Access, il faut faire appel à un mode particulier. C'est faciliter l'apprentissage progressif du langage de requêtes. C'est pouvoir modifier une requête SQL sur un point précis sans les lourdeurs du passage par l'interface. △

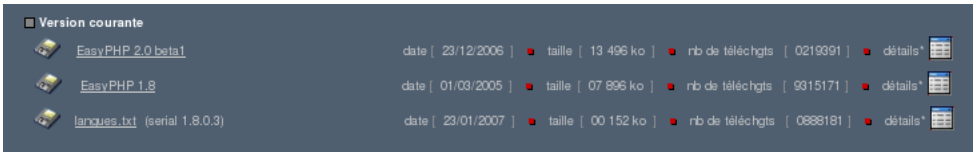
1.1.1. Téléchargement d'EasyPHP

On commence par télécharger le logiciel à l'URL : `http://www.easyphp.org`, dont la page d'accueil est en 1.

1

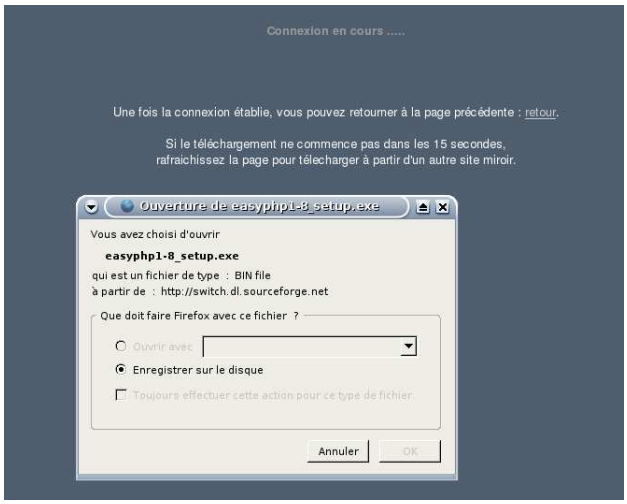
On choisit la version courante en 2.

2



La version beta est en général la dernière version en développement : elle comprend éventuellement de nouvelles fonctionnalités mais peut se révéler moins stable. C'est pourquoi c'est la version non beta qui est en général déchargée : 3.

3

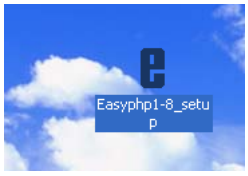


Le fichier récupéré est un fichier d'installation qui comprend le logiciel et les utilitaires permettant de l'installer sur la machine courante.

1.1.2. Installation d'EasyPHP

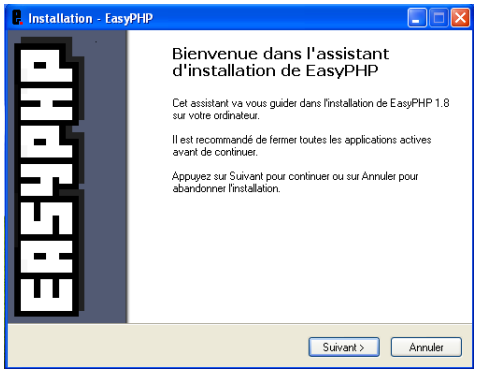
Double cliquer sur l'icône du fichier téléchargé 4 démarre l'installation. On choisit d'abord la langue d'installation, ici le français.

4



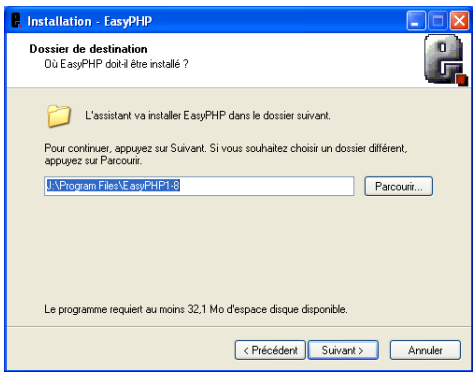
L'installation proprement dite commence : 5, avec une suite d'étapes dont certaines demandent des choix. Il faut ainsi d'abord accepter la licence proposée. Des informations sont fournies ensuite sur la version que l'on installe (numéro de version, nouvelles caractéristiques, etc.).

5



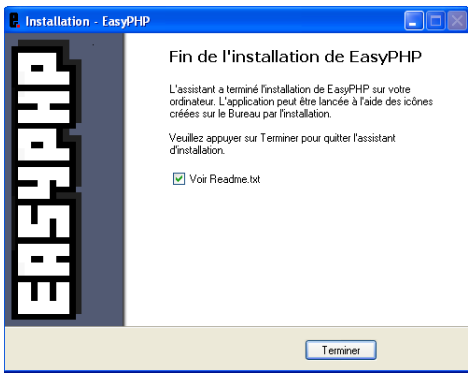
On choisit l'endroit où se trouvera la version installée : 6. L'emplacement proposé par défaut est d'ailleurs en général satisfaisant. Est déterminée la localisation, par défaut dans le menu Démarrer, des raccourcis permettant de démarrer aisément EasyPHP. Ces choix effectués et confirmés, l'utilitaire d'installation décompresse les composants du logiciel et les met en place.

6



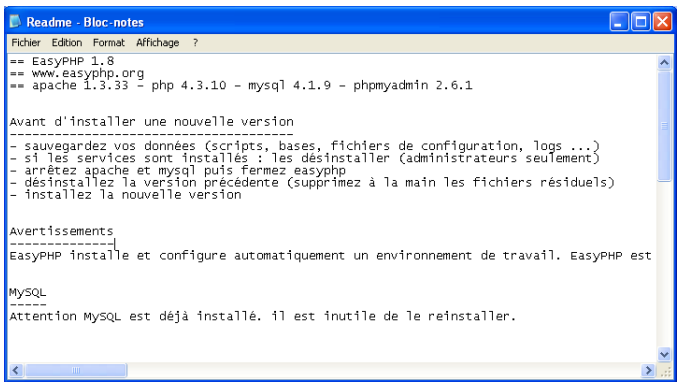
L'indication que l'installation est achevée est alors fournie : 7.

7



Les informations du fichier Readme.txt 8 indiquent la version d'EasyPHP comme « enveloppe » et celles des composants : MySQL, l'environnement graphique (PhpMyAdmin), le serveur local (Apache) et le langage de programmation utilisé (PHP).

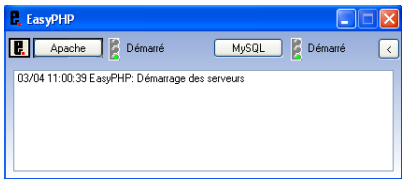
8



1.1.3. Utilisation d'EasyPHP

Le démarrage d'EasyPHP indique que le serveur local Apache et MySQL sont lancés : 9 (les petits feux peuvent mettre quelques instants à passer au vert).

9

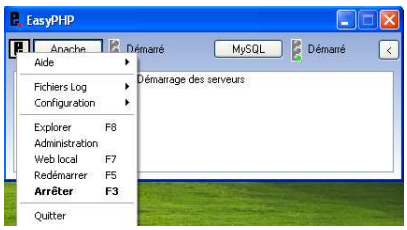


L'icône est un e avec un point rouge. Elle apparaît alors à droite dans la barre de menu du bas et permet d'accéder à l'application : 10.

10

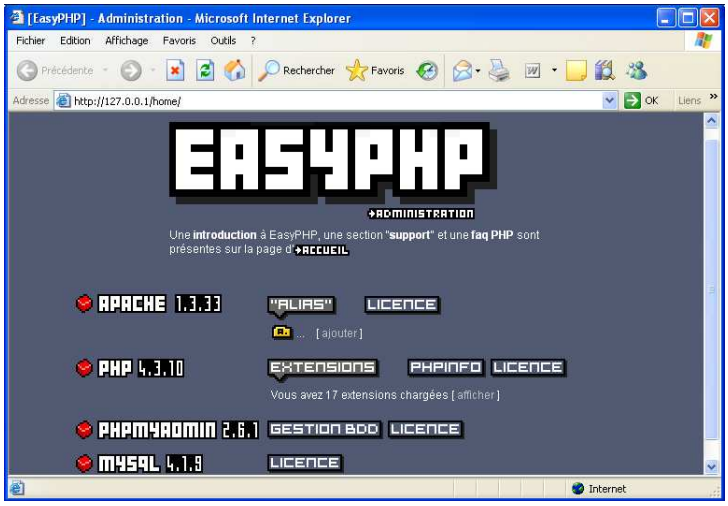
Cliquer le e dans un carré blanc à gauche en dessous de la barre de menu du haut ouvre un menu : 11.

11



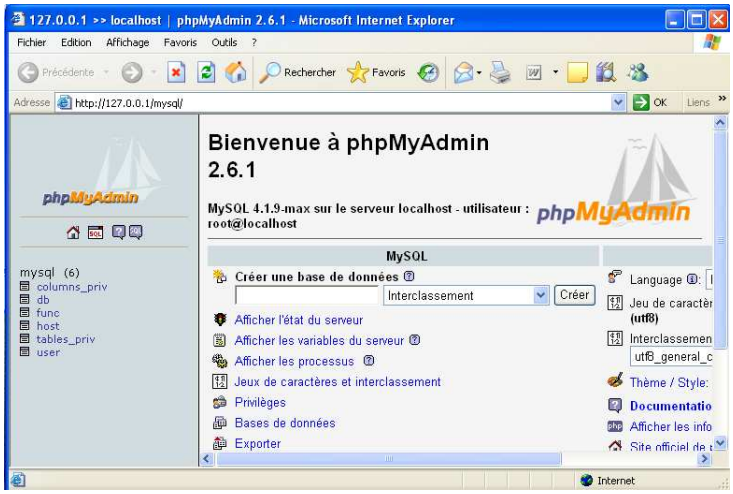
Choisir [Administration] ouvre un navigateur (Internet Explorer dans l'exemple) sur le serveur local, dont l'adresse est `http://127.0.0.1/`, avec les différents volets de l'application : 12.

12



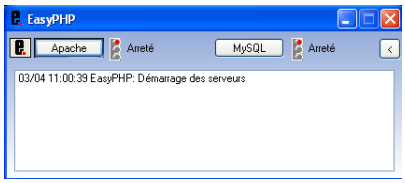
Le choix de [PhpMyAdmin | Gestion BDD] est celui d'une interface graphique simple avec MySQL doublée de la possibilité de requêtes directement en MySQL : 13. On voit en particulier sur cet écran que l'on peut créer une base de données, en fournissant son nom.

13



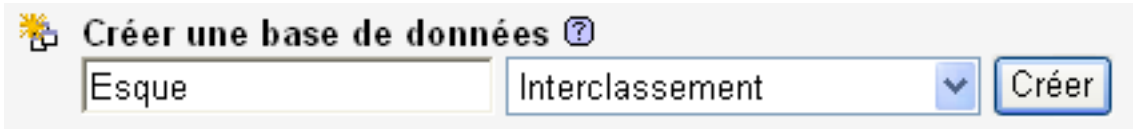
Ce dernier écran montre l'arrêt d'EasyPHP : les deux feux passent au rouge, parfois après un certain temps d'attente :

14



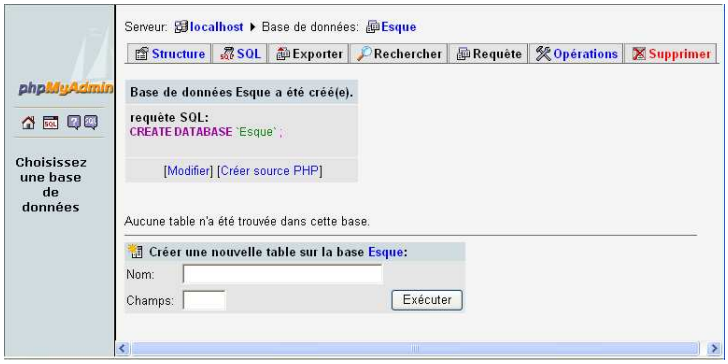
Création d'une base de données À partir de l'écran 13, on fournit un nom : 15 et on clique sur le bouton [Créer].

15



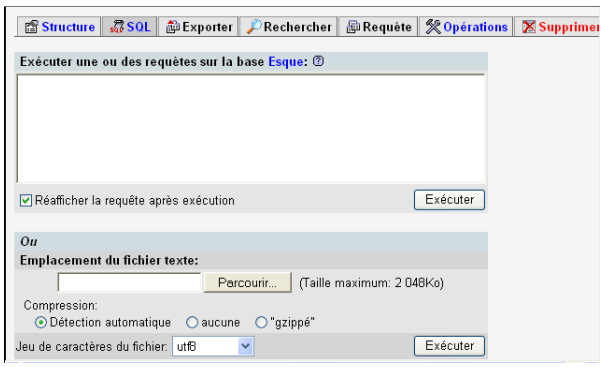
La base se trouve créée : 16 et devient la base courante. La requête MySQL équivalent est fournie en même temps que l'indication de succès de la commande.

16



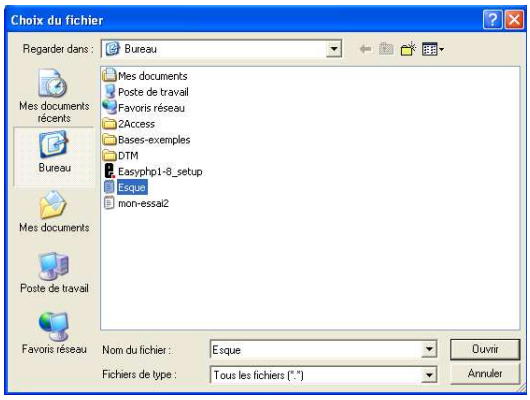
Le choix de [SQL] donne accès à une fenêtre en 17 où l'on peut soit écrire une requête MySQL soit charger un fichier comprenant une ou plusieurs instructions SQL. C'est ce dernier choix qui est utilisé pour lire le fichier contenant les instructions de création et de peuplement de la table esque. La taille d'un tel fichier d'instruction est limitée (2Mo), mais le fichier peut être compressé (bas de l'écran).

17



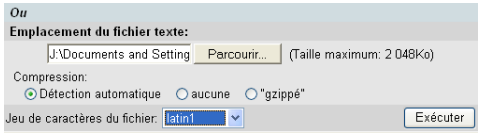
Le bouton [Parcourir] permet d'explorer les disques et dossiers et de choisir le fichier souhaité : 18.

18



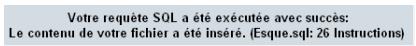
Il ne reste plus, en 19, qu'à exécuter les instructions contenues dans le fichier.

19



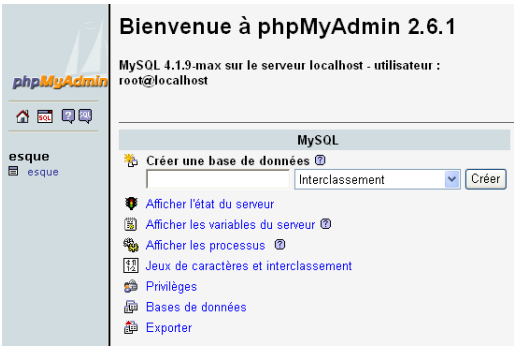
Un message de succès est émis : 20.

20



On voit apparaître dans le bandeau de droite, sous le nom de la base courante, Esque, le nom de la table qui vient d'être créée : 21.

21



En 22, dans le bandeau de gauche, on peut revenir à l'accueil en cliquant sur la petite maison à gauche, ou en dessous, choisir la base de données sur laquelle on veut travailler. Le bandeau de droite, via [Bases de données] permet aussi de choisir la base de données à utiliser.

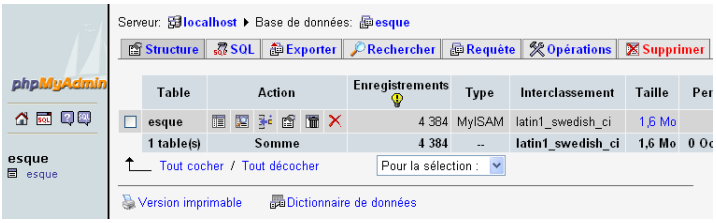
22

△

La base mysql est une base utilisée par le logiciel pour la gestion des bases des utilisateurs. On s'interdira donc de la modifier, sous peine de surprises désagréables.

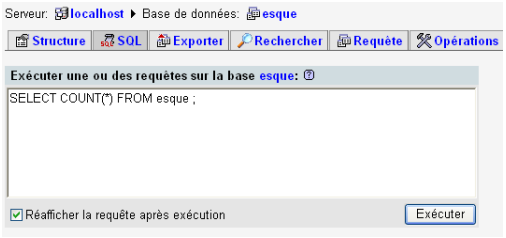
Le choix de la table esque au sein de la base du même nom est illustré en 23.

23



Cliquer sur [SQL] ouvre une fenêtre où formuler une requête SQL, ici le décompte des lignes de la table : 24.

24

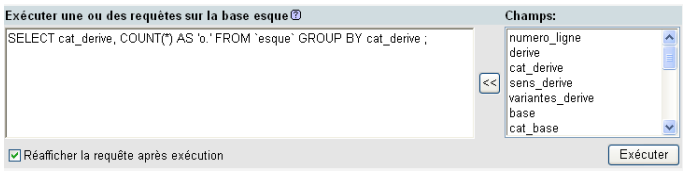


Le résultat 25 comprend la reprise de la requête qui vient d'être exécutée. Elle peut ainsi être reprise, amendée, etc.

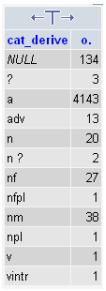
25

En 26, une requête un peu plus complexe, dont le résultat figure en 27.

26



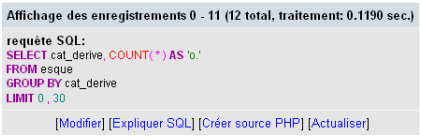
27



cat_derive	o.
NULL	134
?	3
a	4143
adv	13
n	20
n ?	2
nf	27
nfpl	1
nm	38
npl	1
v	1
vintr	1

On constate en 28 que la requête s'est vue adjoindre automatiquement **LIMIT** 0,30 qui entraîne l'affichage de 0 à 30 lignes maximum. Si 30 lignes sont affichées et que le résultat en comprend d'autres, un lien permet d'afficher les lignes suivantes, par tranches de 30 lignes au plus : c'est tenir compte des contraintes de l'interface HTML qui rechigne aux trop longs tableaux qui supposent de manipuler l'ascenseur latéral.

28



1.2. Sous Linux

Unix/Linux constitue un environnement multi-utilisateurs. Chaque utilisateur possède des droits spécifiques. Un utilisateur privilégié, le super-utilisateur, dénommé *root*, administre le système, crée les comptes des utilisateurs, attribue et modifie leurs droits, etc.

Le monde MySQL s'organise aussi en un super-utilisateur MySQL (ou administrateur base de données) et des utilisateurs « normaux ». Le super-administrateur MySQL peut d'ailleurs ne pas être le super-utilisateur du système Unix/Linux. Dans la suite, *super-utilisateur* sans plus de précisions renvoie à *super-utilisateur MySQL*.

1.2.1. Installation de MySQL

L'installation d'un nouveau logiciel, comme MySQL, doit donc être effectuée par le super-utilisateur système. C'est également ce dernier qui détermine qui est le super-utilisateur MySQL et qui lui permet de choisir un mot de passe.

Deux bases sont créées automatiquement lors de l'installation de MySQL :

1. une base de test, dénommée... test ;
2. une base nommée mysql qui comprend précisément les informations nécessaires à la gestion de MySQL : création et destruction de bases, ajout d'utilisateurs et indications de leurs droits, etc. On ne manipulera pas cette base.

C'est le super-utilisateur MySQL par contre qui crée les bases de données et qui attribue aux utilisateurs les droits leur permettant d'accéder à une base déterminée.

1.2.2. Créer une base de données

Pour créer une base de données, c'est-à-dire un réceptacle vide au départ, dans lequel insérer des tables, etc., il faut d'abord lancer MySQL en tant que super-utilisateur MySQL, par une instruction du type :

```
mysql −u <super−utilisateur MySQL> −p
```

par exemple :

```
mysql −u bdadmin −p
```

qui signifie que le super-utilisateur MySQL (-u), de nom bdadmin, veut se connecter à MySQL après avoir fourni un mot de passe (*password*, d'où l'option -p).

L'examen des bases de données existantes découle de la requête :

```
SHOW DATABASES ;
```

dont le résultat est le listage des bases existantes, de la forme :

Database
...
mysql
phedre
prema
test

L'ajout d'une nouvelle base s'effectue par la requête :

```
CREATE DATABASE <nom de la nouvelle base> ;
```

par exemple :

```
CREATE DATABASE esque ;
```

Si l'on répète la requête de listage ci-dessus, on obtient maintenant un résultat de la forme :

Database
...
esque
mysql
phedre
prema
test

1.2.3. Ajouter des utilisateurs à une base

Le super-utilisateur, une fois une base créée, ajoute des utilisateurs à cette base par deux requêtes :

```
GRANT ALL PRIVILEGES ON <base>.*
TO <utilisateur>@"%" IDENTIFIED BY '<mot de passe>'
WITH GRANT OPTION ;

GRANT ALL PRIVILEGES ON <base>.*
TO <utilisateur>@"localhost" IDENTIFIED BY '<mot de passe>'
WITH GRANT OPTION ;
```

Par exemple :

```
GRANT ALL PRIVILEGES ON esque.*
TO habert@"%" IDENTIFIED BY 'Cadou51'
WITH GRANT OPTION ;

GRANT ALL PRIVILEGES ON esque.*
TO habert@"localhost" IDENTIFIED BY 'Cadou51'
WITH GRANT OPTION ;
```

On ne rentrera pas dans le détail des droits qui peuvent être accordés ou refusés. On se reportera à la documentation de MySQL. Notons tout de même que MySQL permet des gestions très fines des droits (lire une table mais sans pouvoir la modifier, etc.). Tous les SGBD professionnels permettent d'ailleurs une telle gestion. Il importe en effet de distinguer l'administrateur d'une base et ses utilisateurs mais également de pouvoir déterminer, au sein de ces derniers, des sous-ensembles qui vont accéder à des parties et à des opérations éventuellement distinctes. Tout n'est pas pertinent pour chaque utilisateur. En outre, il faut pouvoir se prémunir contre les dégâts accidentels et contre la malveillance.

1.3. Bases de données exemples

Leur installation suppose la réalisation des trois étapes précédentes. MySQL doit être installé (éventuellement comme composant d'EasyPHP), les bases de données doivent avoir été créées, l'utilisateur qui veut installer les bases-exemples doit avoir les droits sur ces trois bases.

1.3.1. Installation sous EasyPHP

On se reportera à la section précédente p. 497 qui montre l'installation et l'utilisation de la base ESQUE.

1.3.2. Installation directement sous MySQL

On prendra l'exemple de la création et du peuplement de la base PHÈDRE à partir des données fournies dans l'archive en ligne.

Création de la base de données et d'un utilisateur L'administrateur MySQL crée la base phedre et l'utilisateur habert. Il donne à ce dernier les droits nécessaires (l. 12-16). La base phedre est vide au départ comme le montre le résultat de l'instruction **SHOW** TABLES (l. 19-20).

```
habert@harris:~$ su
Password:
harris:/home/habert#    mysql
Welcome to the MySQL monitor.  Commands end with ; or \g. Your MySQL connection id
    is 6 to server version: 4.1.11-Debian_4sarge5-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.


mysql> create database phedre ;
Query OK, 1 row affected (0.15 sec)


mysql> mysql> GRANT ALL PRIVILEGES ON phedre.* TO habert@"%" IDENTIFIED BY 'Cadou51'
    WITH GRANT OPTION ;
Query OK, 0 rows affected (0.19 sec)

mysql> GRANT ALL PRIVILEGES ON phedre.* TO habert@"localhost" IDENTIFIED BY 'Cadou51
    ' WITH GRANT OPTION ;
Query OK, 0 rows affected (0.00 sec)


mysql> show tables ;
Empty set (0.00 sec)


mysql> quit
Bye
harris:/home/habert#
```

Exécution des instructions de création des tables et de leur peuplement Le fichier Phedre.sql dans l'archive en ligne contient à la fois les instructions **CREATE TABLE** permettant de créer les trois tables de la base correspondante et les instructions **INSERT** permettant de les peupler. L'utilisateur habert se connecte alors à la base phedre en exécutant les instructions de ce fichier (l. 1). Cette connexion se limite à l'exécution de ces instructions. La connexion suivante montre que les tables ont été créées (l. 10-18) et qu'elles ont été peuplées (l. 36-41).

```
habert@harris:~$  mysql -u habert -p phedre < /Bases/MySQL/Dumps/Phedre.sql
Enter password:
harris:/home/habert# mysql -u habert -p phedre
Enter password:
Reading table information for completion of table and column names You can turn off
    this feature to get a quicker startup with -A
Welcome to the MySQL monitor.  Commands end with ; or \g. Your MySQL connection id
    is 9 to server version: 4.1.11-Debian_4sarge5-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.


mysql> show tables ;
+--------------------+
| Tables_in_phedre |
+--------------------+
| occurrences        |
```

```
| positions          |
| vers               |
+--------------------+
3 rows in set (0.00 sec)

  mysql> DESCRIBE vers ;
+-------------------+--------------+------+-----+---------+--------+
| Field             | Type         | Null | Key | Default | Extra  |
+-------------------+--------------+------+-----+---------+--------+
| acte              | varchar(5)   |      |     |         |        |
| scene             | varchar(5)   |      |     |         |        |
| numero_vers       | int(11)      |      | PRI | 1       |        |
| personnage_s      | varchar(255) |      |     |         |        |
| partage_en        | int(11)      |      |     | 1       |        |
| vers              | varchar(200) |      | UNI |         |        |
| vers_phonetique   | varchar(250) |      |     |         |        |
+-------------------+--------------+------+-----+---------+--------+
7 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM vers ;
+----------+
| COUNT(*) |
+----------+
|     1654 |
+----------+
1 row in set (0.04 sec)

mysql> quit
Bye
habert@harris:~$
```

Exécution des instructions de création des tables puis peuplement sous MySQL On reprend juste après la création de la base phedre par le super-utilisateur MySQL et l'octroi des droits nécessaires à l'utilisateur habert :

```
habert@harris:~$ mysql -u habert -p phedre
Enter password:
...

mysql> show tables ;
Empty set (0.00 sec)


mysql> quit
Bye
habert@harris:~$
```

La première étape consiste à créer la structure des tables de la base :

```
habert@harris:~$ mysql -u habert -p phedre < /Bases/MySQL/Struct/Phedre_Structure.sql
Enter password:
```

Le fichier Phedre_Structure.sql dans l'archive en ligne contient seulement les instructions **CREATE TABLE** permettant de créer les trois tables de la base correspondante.

La seconde étape, sous MySQL, revient à remplir les tables dûment créées (mais vides cf. l. 5-20) grâce aux fichiers délimités (de type CSV) disponibles également dans l'archive en ligne (l. 23) :

```
habert@harris:~$ mysql -u habert -p phedre
Enter password:
...

mysql> show tables ;
+--------------------+
| Tables_in_phedre |
+--------------------+
| occurrences      |
| positions        |
| vers             |
+--------------------+
3 rows in set (0.00 sec)


mysql> SELECT COUNT(*) FROM vers ;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.00 sec)
mysql> LOAD DATA LOCAL INFILE "/Bases/CVS/Phedre/vers.tab" INTO TABLE vers LINES
    TERMINATED BY '\r\n' ;

Query OK, 1655 rows affected, 2 warnings (0.11 sec)

Records: 1655  Deleted: 0  Skipped: 0  Warnings: 2

mysql> SELECT COUNT(*) FROM vers ;
+----------+
| COUNT(*) |
+----------+
|     1655 |
+----------+
1 row in set (0.00 sec)
```

L'instruction **LOAD DATA** permet d'insérer dans une table existante (ici vers) des lignes d'un fichier délimité dont le nom est fourni. Remarquer qu'on spécifie que les changements de ligne sont marqués dans ce fichier par la suite de deux caractères propre à Windows : un retour-chariot (\r) suivi d'un passage à la ligne (\n).

2. Access

2.1. Logiciel

Les bases de données supposent des opérations conceptuelles plus complexes que celles des outils bureautiques les plus courants : traitement de texte, tableur et gestionnaire de présentations visuelles. C'est pourquoi Access ne fait pas partie de la distribution la plus courante de MS Office, l'éventail Microsoft d'outils bureautiques.

2.2. Bases de données-exemples

2.2.1. Installation des bases en format Access

Il suffit de copier les 3 fichiers :

- prema.mdb
- phedre.mdb
- esque.mdb

de l'archive en ligne sur le disque dur dans un répertoire approprié.

2.2.2. Reconstitution des bases de données à partir du format délimité

Pour les tables d'une base de données, on utilise les versions dont les colonnes sont délimitées par des tabulations (\t) et les lignes par un retour-chariot (\r) suivi d'un passage à la ligne (\n), suite qui délimite conventionnellement les lignes sous Windows. Ces versions sont disponibles dans le répertoire CSV de l'archive en ligne. La première ligne de ces versions est constituée des noms de colonnes de la tables.

La démarche de reconstitution des tables d'une base de données à partir d'un format délimité est détaillée au ch. XIII, § 2, p. 439.

CHAPITRE XV

⊕ MYSQL, ACCESS ET SQL SERVER

1. Terminologies comparées

La terminologie choisie par Access ne facilite pas toujours la compréhension du modèle sous-jacent. Access appelle en effet *relation* une jointure « persistante » entre deux tables alors que la notion de relation en algèbre relationnelle correspond à une table dont toutes les lignes sont distinctes et qui constitue donc un ensemble. Par ailleurs, Access conserve des termes provenant des origines des bases de données et de leur réalisation physique (Gardarin, 2005, ch. III). C'est le cas d'**enregistrement** (*record*) qui renvoie à la réalité physique de l'écriture sur disque d'un ensemble structuré d'informations liées entre elles. C'est le cas aussi de **champ** (*field*), partie d'enregistrement consacrée à une de ses facettes.

Par cohérence avec l'évolution des bases de données qui vise à « faire oublier » autant que faire se peut leur réalisation physique, nous n'employons donc pas une terminologie qui rappelle cette dernière. Le tableau 150 p. 508 met en correspondance les différentes terminologies et fournit en italiques les correspondants en anglais. △

Rappelons qu'un SGBD manipule des tables qui ne sont pas forcément des relations au sens strict, c'est-à-dire des ensembles de tuples tous différents : elles peuvent contenir des doublons (des lignes identiques). On parle alors plus précisément de **multi-ensembles**. Une restriction qui prend en entrée une relation en entrée produit également une relation. La sélection d'un sous-ensemble de colonnes peut produire ou non une relation. C'est l'opposition dans le livre entre PROJECTION AVEC DOUBLONS et PROJECTION (sans doublons).

2. De la notation abstraite à SQL, MySQL, Access et SQL Server

Dans les pages qui suivent :

TABLEAU 150 – Terminologies comparées : informelle, relationnelle, Access...

« *Physique* »	*relationnelle*	*Access*	*informelle*
	relation	table, feuille de données	table
enregistrement *record*	n-uplet *tuple*	enregistrement	ligne *row*
champ *field*	attribut	champ	colonne *column*
	jointure	relation	lien
	cardinalité d'une relation		nombre de lignes
	degré, arité d'une relation		nombre de colonnes
	identifiant (unique), clé primaire *primary key*		
	clé étrangère *foreign key*		
	contrainte référentielle		
	type, domaine		

- la mention SQL renvoie à une réalisation d'une opération abstraite valide en MySQL comme en SQL Server, la version de SQL sous-jacente aux opérations d'Access ;
- la mention MySQL à une réalisation spécifique de ce SGBD ;
- la mention Access réfère à ce qu'il faut indiquer dans l'interface et qui peut différer de la requête SQL Server engendrée automatiquement ou écrite directement. Ainsi, les arguments de certaines fonctions dans l'interface d'Access sont séparés par des points-virgules, alors que la fonction SQL Server correspondante utilise des virgules. On sera attentif à ces distinguos sources d'erreurs.

Sont présentées les différences dialectales entre MySQL et SQL Server uniquement pour les fonctionnalités exemplifiées dans l'ouvrage. (Gennick, 2006) montre les opérateurs disponibles pour des SGBD très répandus : Oracle, IBM DB2, mais aussi MS (MicroSoft) SQL Server et MySQL. On s'en servira donc de référence.

On sera attentif au caractère servant à séparer les arguments d'une fonction :

- MySQL et SQL Server : une virgule ;
- Access : le plus souvent, un point virgule.

2.1. Opérations sur les regroupements

- COMPTE(<valeurs d'une colonne>)
  - SQL : **COUNT**(<valeurs d'une colonne>)
  - Access : **Compte**(<valeurs d'une colonne>)
- SOMME(<valeurs d'une colonne>)
  - SQL : **SUM**(<valeurs d'une colonne>)

- Access : **Somme**(<valeurs d'une colonne>)

2.2. Sélection de valeurs

- Comparaison : = > >= <= <> ou !=
- PARMI (<valeur$_1$>, . . ., <valeur$_n$>)
  - SQL : **IN**(<valeur$_1$>, . . ., <valeur$_n$>)
  - Access : *id.* mais le séparateur est le point-virgule et non la virgule.
- Intervalle (les bornes sont incluses)
  - SQL : **BETWEEN** <valeur$_1$> **AND** <valeur$_2$>
  - Access : **ENTRE** <valeur$_1$> **ET** <valeur$_2$>

2.3. Opérateurs logiques (booléens)

- ET
  - SQL : **AND**
  - Access : ET
- OU
  - SQL : **OR**
  - Access : OU
- PAS
  - SQL : **NOT**
  - Access : PAS
- SI
  - MySQL : **IF**(<condition>, <valeur si condition remplie>, <valeur sinon>)
  - SQL Server : **IIF**(<condition>, <valeur si condition remplie>, <valeur sinon>)
  - Access : **VraiFaux**(<condition> ; <valeur si condition remplie> ; <valeur sinon>)

2.4. Traitement de la marque NULL

- EST NULL
  - SQL : **IS NULL**
  - Access : EST **NULL**
- EST PAS NULL
  - SQL : **IS NOT NULL**
  - Access : EST PAS **NULL**

2.5. Fonctions sur les chaînes de caractères

- Mise en minuscules
  - MySQL : **LOWER**(<chaîne>)
  - SQL Server : **LCASE**(<chaîne>)
  - Access : **MINUSCULE**(<chaîne>)
- Longueur
  - MySQL : **LENGTH**(<chaîne>)
  - SQL Server : **LEN**(<chaîne>)
  - Access : **NbCar**(<chaîne>)
- Extraction des k caractères à gauche
  - MySQL : **SUBSTRING**(<chaîne>, 1, k)
  - SQL Server : **LEFT**(<chaîne>, k)

TABLEAU 151 – Opérateurs d'approximation : MySQL *vs.* Access

MySQL		*Access*
Rôle	*opérateur*	*opérateur*
Ensembles de caractères		
tout car.	• (point)	?
chiffre	[0-9]	#
0 ou n car.	•*	*
ensemble énuméré	[<car.$_1$>...<car.$_n$>]	*id.* ou virgule séparatrice
ensemble borné	[<car. début>-<car. fin>]	*id.*
complémentaire d'un ensemble	[^<ensemble de car.>]	[!<ensemble de car.>]
nombre d'occurrences d'un sous-motif		
0 ou n fois	*	*absent*
1 ou n fois	+	*absent*
0 ou 1 fois	?	*absent*
ancrage d'un motif		
en début de colonne	^	*non nécessaire*
en fin de colonne	$	*non nécessaire*
combinaison de motifs		
disjonction	| (barre verticale)	*absent*
parenthésage	( )	*absent*

  - Access : **Gauche**(<chaîne> ; k)
- Inversion
  - MySQL : **REVERSE**(<chaîne>)
  - SQL Server : **StReverse**(<chaîne>)

2.6. Opérateurs d'approximation sur les chaînes de caractères

Le tableau 151 p. 510 résume les opérateurs disponibles sous MySQL et sous Access.

2.7. Fonctions numériques

- Choix du nombre de chiffres décimaux
  - MySQL : **FORMAT**(<nombre à décimales>, <nombre de décimales retenu>)
  - SQL Server : **FORMATNUMBER**(<nombre à décimales>, <nombre de décimales retenu>)

2.8. Tris, limitation des résultats

- Sélection des k premiers (%) résultats
  - MySQL : **LIMIT** k (en toute fin de requête)
  - SQL Server : **TOP** k ou **TOP** k **PERCENT** juste après **SELECT**
- Tri aléatoire
  - MySQL : **ORDER BY RAND**()
  - SQL Server : **ORDER BY RND**()

3. Types d'attributs disponibles

3.1. Valeurs textuelles

Une valeur textuelle peut comprendre uniquement des chiffres, mais dans ce cas-là, il s'agit de numéros d'identification ne permettant pas des calculs (comme un numéro de sécurité sociale ou un numéro de téléphone).

3.1.1. MySQL

MySQL oppose des attributs faisant la distinction de casse (majuscules / minuscules) dans le tri comme dans la comparaison et la recherche. Cette distinction pour les types CHAR et VARCHAR se marque par un modifieur (BINARY), pour les autres par une opposition entre les types de suffixe TEXT, insensibles à la casse, et les types de suffixe BLOB, sensibles à la casse.

Pour les valeurs limitées, MySQL offre les types suivants :

CHAR(<taille>) La taille va de 0 à 255 caractères ;

VARCHAR(<taille>) La taille varie également de 0 à 255 caractères, mais un attribut de ce type permet de stocker du texte de taille variable, allant jusqu'à la <taille> déclarée. Ainsi un attribut déclaré comme VARCHAR(4) pourra stocker des valeurs de 0 (chaîne vide) à 4 caractères ;

Tinytext / Tinyblob de 0 à 255 caractères (*tiny* = minuscule).

Pour les valeurs plus importantes, sont disponibles les types :

Text / Blob de 0 à 65 535 caractères.

Mediumtext / Mediumblob de 0 à 16 777 215 caractères ;

Longtext / Longblob de 0 à 4 294 967 295 caractères.

3.1.2. Access

Access dispose de deux types d'attributs à valeur textuelle :

Texte un attribut de type Texte a une longueur maximale de 255 caractères ;

Mémo Ce type est adapté aux textes « longs », jusqu'à 64 000 caractères au plus (soit environ 8 000 mots ou 16 pages « ordinaires »). On notera que certaines requêtes sur du texte sont moins aisées voire impossibles avec des attributs de type Mémo.

3.2. Valeurs numériques

3.2.1. MySQL

Par défaut les nombres sont positifs ou négatifs. Le modifieur UNSIGNED (non signé) indique que la colonne correspondante ne peut contenir que des nombres positifs.

Pour les entiers (*int* pour *integer*), MySQL fournit les types suivants, par taille croissante des nombres à stocker :

Tinyint nombre compris entre -128 et 127 ou entre 0 et 255 s'il n'est pas signé ;

Smallint nombre compris entre -32 768 et 32 767 (non signé : entre 0 et 65 535) ;

Mediumint nombre compris entre -8 388 608 et 3 388 607 (non signé : entre 0 et 16 777 215) ;

Int entre -2 147 483 648 et 1 147 483 647 (non signé : entre 0 et 4 294 967 295) ;

Bigint entre -9 223 372 036 854 775 808 et 9 223 372 036 854 775 807 (non signé : entre 0 et 18 446 744 073 709 551 615).

Pour les nombres décimaux, les types suivants correspondent à des précisions croissantes :

Decimal de la forme Decimal(<nombre de chiffres significatifs>, <dont chiffres significatifs après le séparateur décimal>). Ainsi Decimal(4,2) permet d'aller de -99,99 à 99,99 ;

Float ;

Double .

3.2.2. Access

Trois types sont disponibles pour les valeurs entières, selon la taille des nombres à mémoriser :

Octet nombre positif de 0 à 255 ;

Entier nombre positif ou négatif de -32 768 à 32 767 ;

Entier long nombre positif ou négatif de -2 147 483 647 à 2 147 483 447.

Trois types sont également disponibles pour les nombres décimaux, par ordre de précision croissante :

Réel simple (précision décimale : 7) ;

Réel double (précision décimale : 15) ;

Décimal (précision décimale : 28).

3.3. Valeurs temporelles

Pour des historiens, les valeurs disponibles peuvent se révéler insatisfaisantes (pour des datations au tout début de notre ère, par exemple) et il faut alors se rabattre sur d'autres types. △

3.3.1. MySQL

Year année au format AAAA (ex. : 2007) entre 1900 et 2155 ;

Time moment au format hh :mm :ss (ex. : '17 :15 :02' pour 17h 15 minutes et 2 secondes) ;

Date au format AAAA-MM-JJ (ex. : '2006-06-17' pour le 17 juin 2006), entre le 1^{er} janvier 1000 et 31 décembre 9999 dans le calendrier grégorien ;

Datetime au format AAAA-MM-JJ hh :mm :ss (ex. : '2006-06-17 17 :15 :02'), entre le 1^{er} janvier 1000 à minuit et 31 décembre 9999 à 23h59 et 59 s. dans le calendrier grégorien ;

Timestamp au format AAAAMMJJhhmmss (ex. : '20060617171502') entre le 1^{er} janvier 1970 à minuit et le 31 décembre 2037, une seconde avant minuit. C'est un tampon temporel, comme son nom l'indique en anglais, qui sert en particulier à dater les modifications d'une table. La modification d'une ligne comprenant une colonne TIMESTAMP entraîne la mise à jour de celle-ci à la date et l'heure de la modification qui vient d'être effectuée.

3.3.2. Access

Date/Heure Le paramétrage d'un attribut de ce type permet de préciser le mode d'affichage.

3.4. Valeur auto-incrémentée

Un attribut à valeur auto-incrémentée est le moyen de donner automatiquement à chacune des lignes d'une table un numéro d'ordre arbitraire tel qu'une nouvelle ligne ait un numéro d'ordre plus grand que tous les numéros d'ordre déjà existants. Chaque nouvelle ligne ajoutée comprenant un tel attribut voit cet attribut prendre la valeur suivante de la dernière valeur donnée automatiquement.

3.4.1. MySQL

Auto_increment Cette caractéristique s'applique au plus à une colonne de type entier d'une table. Il n'y a donc pas de type auto_increment analogue au type NuméroAuto d'Access mais le moyen de transformer une seule colonne d'une table pour qu'à chaque ajout d'une ligne, sa valeur soit incrémentée automatiquement par rapport à la dernière valeur attribuée.

3.4.2. Access

NuméroAuto

3.5. Types propres à MySQL

Enum(<valeur$_1$>,. . .,<valeur$_n$>) permet de stocker une des valeurs appartenant à une liste prédéfinie de chaînes de caractères (de 1 à 255, voire 65 535 valeurs) ;

Set(<valeur$_1$>,. . .,<valeur$_n$>) permet d'indiquer 0, une ou plusieurs des valeurs spécifiées.

3.6. Types propres à Access

Assistant Liste de choix On peut faire en sorte qu'un attribut de ce type prenne ses valeurs dans un menu déroulant, dont les valeurs possibles sont celles d'une autre table de la base de données ou d'une liste fournie explicitement. Quand un attribut peut prendre un nombre limité et pré-défini de valeurs, le choix de ce type diminue les risques d'erreur de saisie et accélère l'entrée des informations.

Objet OLE Un attribut de ce type permet d'incorporer ou de renvoyer à un objet créé avec un programme comme Microsoft Word ou Microsoft Excel, ou encore à une image, du son ou des données binaires.

Oui/Non Sert à représenter les choix logiques (Oui/Non, ou vrai/faux) ;

Lien hypertexte De tels liens peuvent être opératoires soit dans les formulaires de saisie soit dans les états (sorties).

Monétaire Les nombres de ce type d'attribut peuvent aller jusqu'à 15 chiffres avant le séparateur décimal et 4 après. On peut choisir le symbole monétaire adéquat dans les sorties (états).

CONCLUSION

1. ⊕ Angles morts conscients

Si l'ouvrage s'écarte d'un guide de l'utilisateur pour un ou plusieurs logiciels et s'il présente les principes généraux des systèmes de gestion de bases de données (relationnelles), il n'approfondit cependant pas trois aspects plus techniques et moins fondamentaux pour les applications visées dans ces pages :

1. la réalisation matérielle sous-jacente en termes de fichiers, etc. Les SGBD ont d'ailleurs pour fonction explicite de protéger l'utilisateur des détails du niveau matériel.
2. les facilités qu'offrent les SGBD pour permettre à plusieurs utilisateurs de travailler sur des données partagées sans créer d'incohérences (si nous avons réservé une place dans un train, nous entendons bien qu'elle ne soit pas louée en parallèle à quelqu'un d'autre). Les SGBD, généralement multi-utilisateurs, sont d'ailleurs prévus pour permettre à chaque utilisateur de se comporter comme si le système était mono-utilisateur.
3. les normalisations de données conseillées, qui ne sont traitées que partiellement.

2. ⊕ Données pas données

La généralisation des bases de données aboutit à la présence d'*entrepôts de données* qui ont pour corrolaire la *fouille de données* (*data-mining*), chargée de repérer des régularités significatives dans des ensembles arbitrairement volumineux de données plus ou moins structurées.

Dans ces trois dénominations, le mot *données* semble avoir une connotation relativement passive. Les données, en première analyse, c'est ce qui se donne, ce qui est là.

Au contraire, comme le rappelle F. Rastier, les données, c'est ce qu'on se donne, c'est-à-dire ce que l'on dégage explicitement du magma perceptuel et conceptuel qui nous entoure. C'est le résultat d'une abstraction et d'une modélisation préalables et continuées.

Par ailleurs, les données ne sont jamais immédiatement « propres ». Les trois bases de données utilisées par l'ouvrage, bien qu'elles soient le fruit chacun d'un travail important de collation et de mise au propre, ont été l'occasion de constater la nécessité de vérifier la justesse et la cohérence des données :

- dans PRÉMA :
  - existence d'infirmières « fantômes » dont les fiches en fait ont dû être rattachées à d'autres infirmières ;
  - fiche saisie au jour 2, ce qui déroge aux choix de l'enquête ;
  - inadéquation du codage de certaines variables (le 0 pour indiquer l'absence du poids, par exemple) ;
  - lemmes inadéquats pour certains verbes ;
  - lemmes multiples sans raison pour une même forme normalisée ;
- dans PHÈDRE :
  - absence de majuscules en début de phrase ;
  - transcriptions phonétiques erronées ;
- dans ESQUE :
  - représentation gênante de l'absence d'information (**NULL**) pour la base d'un nombre non négligeable de dérivés en *-esque* ;
- ...

En somme, les bases de données engagent à une démarche pénélopesque, où il faut sans cesse sur le métier remettre l'ouvrage. Reste que les SGBD constituent l'outil privilégié d'amélioration de la qualité des données.

△

Les SGBD permettent une amélioration continue de la qualité des données. Pour autant, une base de données est le reflet, à un moment précis, des informations qui y sont mémorisées, de leur structure et de leurs relations. Par contre, une base de données ne donne pas directement accès à l'historique de la constitution de ces informations et à leur évolution. Ainsi, la fiche rédigée un 2^{e} jour de vie d'un prématurée et qui a été jugée ne répondant pas au protocole de l'étude a disparu pour toujours dans les oubliettes électroniques. L'« épaisseur » de la constitution des données échappe. Si l'on souhaite en conserver la trace, il faut s'en donner les moyens, en rajoutant par exemple une table dans la base associant à une date, et à un utilisateur, la description textuelle des modifications faites, ou de manière plus fine, en associant à chaque table susceptible d'évoluer une table-mémoire, qui conserve les états précédents des lignes modifiées.

3. ⊕ Bases de données et schémas classificatoires

3.1. ⊕ Penser, classer, vivre

Les deux citations qui suivent témoignent de l'importance des schémas classificatoires. La première, via un exemple emprunté à la littérature, souligne la difficulté qu'il y a à comprendre les motivations sous-jacentes aux subdivisions choisies pour un domaine, tant la diversité des points de vue permet des partitionnements et hiérarchies distincts. L'humour de la seconde citation laisse transparaître l'importance vitale des classements adoptés : classer, c'est agir, et mal classer, c'est prendre des risques.

Ce livre a son lieu de naissance dans la lecture d'un texte de Borges. [...] Ce texte cite « une certaine encyclopédie chinoise » où il est écrit que « les animaux se divisent en : a) appartenant à l'Empereur, b) embaumés, c) apprivoisés, d) cochons de lait, e) sirènes, f) fabuleux, g) chiens en liberté, h) inclus dans la présente classification, i) qui s'agitent comme des fous, j) innombrables, k) dessinés avec un pinceau très fin en poils de chameau, l) et caetera, m) qui viennent de casser la cruche, n) qui de loin semblent des mouches.

Michel Foucault *Les mots et les choses*, Gallimard, 1966, p. 7.

Dans notre édition d'hier, une légère erreur technique nous a fait imprimer les noms des champignons vénéneux sous les photos des champignons comestibles, et vice-versa.

Nos lecteurs survivants auront rectifié d'eux-mêmes.

Pierre Desproges *Fonds de tiroir*, Le Seuil, 1990.

4. ⊕ Les bases de données : un instrument pour classer

Il est frappant de voir converger, à un quart de siècle de distance, Maurice Gross et Geoffrey Sampson, un des tenants actuels d'une « linguistique empirique » liée à l'annotation automatique de corpus, dans le soutien à une démarche explicitement « linnéenne » en linguistique. Ils indiquent tous deux que la description botanique a progressé grâce à l'acceptation d'un schéma classificatoire imparfait, et connu comme tel, mais offrant un langage commun favorisant la convergence des efforts taxonomiques. Le premier (« Présentation », *in* J.-P. Boons, A. Guillet, C. Leclère *La structure des phrases simples en français*, Droz, Genève, 1976, p. 16–17) plaide pour une phase préalable, taxonomique, de classification des phrases : « [. . .] la classification des formes syntaxiques semble encore loin d'avoir trouvé son Linné, son Tournefort ou ses Jussieu, encore moins son Adenson [1] [. . .] il n'exist[e] pas d'autre voie possible à la construction de théories, ce terme étant pris au sens des sciences naturelles et non pas dans son sens métaphorique de plus en plus répandu et appliqué aux sciences humaines et sociales. De telles classifications constituent la seule base empirique praticable aujourd'hui. À partir de telles constructions pourraient éventuellement se dégager des théories permettant des vérifications expérimentales et constituant des explications dont les caractérisations seraient analogues à celles des 'sciences dures'. » Les termes du second font écho très exactement à ceux du premier (G. Sampson, *Empirical Linguistics*, Continuum, Londres, 2001, p. 84) : « Il est peu probable qu'une science se trouve en position de développer des théories profondes pour expliquer ses données avant qu'il existe un cadre faisant consensus (*agreed scheme*) sur la manière d'identifier et de noter ces données », y compris dans la référence à Linné (*ibid.*, p. 84 et p. 93, note 3).

Dans cette optique, les bases de données constituent un apport fondamental dans une démarche classificatoire, en linguistique mais aussi plus largement en sciences humaines. En premier lieu, la modélisation du domaine d'application en termes d'entités et de relations et en fonction d'un point de vue déterminé débouche sur une ontologie possible de ce domaine selon le point de vue choisi. En second lieu, la « mise en tables » des entités du domaine permet a posteriori un retour sur la modélisation. En particulier, les opérations offertes par les SGBD facilitent l'examen de la cohérence des choix classificatoires effectués. On l'a vu par exemple pour ce qui regarde les catégories des bases ou des dérivés de ESQUE. En troisième lieu, les bases de données permettent de « cumuler » les informations venant de sources distinctes et de les confronter. Par exemple, pour Esque, aussi bien des attestations issues de livres, de journaux, de la Toile que des indications sur la présence dans les différents dictionnaires.

5. ⊕ Savoir ce qui peut se dire

Les **langues** dites **naturelles** comme le français, l'anglais, etc. permettent de « tout dire », au prix éventuellement de périphrases. Par opposition les **langages artificiels**, comme les langages de programmation, comme SQL, restreignent ce qui est exprimable. On l'a constaté

[1] Voir F. Dagognet, *Le catalogue de la vie*, PUF, 1970, pour le tournant taxonomique en botanique et en zoologie des 18^{e} et 19^{e} siècle.

au ch. VII § 6, les SGBD relationnels ne permettent pas de traiter des relations transitives comme « rimer avec ». On peut, par auto-jointure, rapprocher les vers ou les mots qui riment deux à deux. On ne peut pas rassembler l'ensemble des vers ou des mots tels qu'au sein de cet ensemble, chaque élément rime avec un autre et que de proche en proche chaque élément rime avec tous les autres. De la même manière, on l'a vu, les SGBD relationnels offrent une prise limitée sur les phénomènes séquentiels. Là encore, l'auto-jointure permet d'isoler le contexte gauche de *puce*, mais on pourrait difficilement généraliser cette démarche pour des contextes arbitrairement plus larges. Nous touchons du doigt, sur deux points, les limites du **pouvoir expressif** des bases de données relationnels. D'autres types de bases de données, les bases de données déductives, savent utiliser les relations transitives.

L'utilisation systématique de MySQL et de SQL Server dans la partie en ligne de l'ouvrage rend manifeste là encore des différences de pouvoir expressif au sein même des bases de données relationnelles. Il n'y a pas d'équivalent immédiat en SQL Server de NOMBRE DE VALEURS DISTINCTES(<attribut>), qui s'exprime sous MySQL via :

COUNT(**DISTINCT**(<attribut>)

Il faut user de « périphrases » pour obtenir cette fonctionnalité. En ce qui concerne les motifs, le langage d'expressions régulières disponible sous MySQL est plus « fin » que les caractères génériques d'Access (chapitre III § 8.3 et tableau 151 p. 510). À l'inverse, les requêtes croisées d'Access sont réalisables en MySQL, mais de manière plus contournée.

Dans tous les cas, il est crucial de savoir aussi précisément que possible ce qui peut se dire dans le langage artificiel dont on dispose. Ainsi, l'obtention des tables fiches_depart, fiches_normalisees, etc., de la base PRÉMA, n'est pas possible en utilisant uniquement les ressources de SQL. Il a fallu recourir à un langage de programmation autre (en l'occurrence PHP) pour lire la table occ_prema et peupler à partir des informations qui y figurent les tables dévolues à un état donné du texte. Savoir par avance ce qui peut se dire, c'est à la fois utiliser au mieux le langage artificiel dont on dispose, en reconnaître les limites et déterminer quand il est nécessaire de recourir à d'autres langages artificiels.

La réalisation des requêtes du premier tome comme les exercices ont souvent montré plusieurs manières, y compris pour un SGBD donné (MySQL *vs.* Access) d'arriver à un résultat donné. On a souligné, de ci de là, les bizarreries de chaque SGBD, pour mettre en garde et faire sentir l'écart qui existe entre le modèle relationnel et les systèmes qui le mettent en oeuvre.

△

On ne confondra pas le pouvoir d'expression et la facilité avec laquelle on peut exprimer telle ou telle opération. L'interface que fournit Access a l'avantage de fournir de nombreux « filets » : les opérations potentiellement dangereuses sont précédées de demandes de confirmation. L'utilisation directe de MySQL peut déboucher sur des dégâts, en cas d'inattention. L'utilisation de l'interface dans Access par rapport à l'utilisation directe de SQL Server peut par contre limiter ce qui peut être dit. Certaines opérations ne se formulent pas bien ou pas du tout via l'interface d'Access.

⊕ Remerciements « complétés » et crédits

Les exemples, les exercices et leurs solutions ont été mis au point et testés :

- pour MySQL avec la version 4.1.11 sous Linux (Debian) ;
- pour Access, avec la version 2003 sous Windows XP Professional, via l'émulation de Windows vmware.

Michel Lastes (LIMSI) a installé les versions de logiciels (MySQL, Access, vmware) qui m'ont permis de fabriquer les exemples, les exercices et leurs solutions. Sammy Debaggi (LIMSI) m'a permis de comprendre comment obtenir en Access l'équivalent de NOMBRE DE VALEURS DISTINCTEs (<colonne>).

Le prêt à clicher de la version papier et de son complément en ligne ont été composés en LaTeX sous Linux (Debian), en utilisant l'éditeur LaTeX Lyx (version 1.3.4). Bernard Desgraupes (université Paris X – Nanterre), auteur en particulier de *LaTeX : apprentissage, guide et référence* (Vuibert, Paris, 2003), a créé la feuille de style LaTeX correspondant à la collection *L'essentiel français* et m'a aidé à composer en LaTeX le prêt à clicher du premier tome et le second tome. Pierre Zweigenbaum (LIMSI) a également contribué aux choix de mise en page et de notation abstraite des requêtes.

Les tableaux ont été engendrés par un script PHP transmettant la requête MySQL à la base de données exemple en cause et utilisant la réponse pour créer les instructions LaTeX correspondant à la table-résultat.

Les packages LaTeX suivants ont été mis à contribution :

- colortbl pour disposer de colonnes ou de lignes grisées dans les tableaux ;
- listings (version 1.3) pour la mise en page des requêtes SQL ;
- hyperref : pour obtenir un PDF navigable du tome 2 ;
- xr : pour les références croisées entre les deux tomes du livre.

Les figures ont été réalisées avec Xfig.

Je remercie très vivement celles et ceux qui m'ont fourni les trois bases de données qui structurent l'exposé et qui fournissent des exemples réalistes et riches :

ESQUE Marc Plénat et Nicole Sernat (ERSS – université de Toulouse-le-Mirail) ;

PHÈDRE Valérie Beaudouin (FT R&D) ;

PRÉMA une équipe de réanimation néonatale.

Le travail continu depuis plusieurs années avec chacun(e) m'a aidé à « entrer » dans les trois « mondes » modélisés par ces bases de données et à entrevoir les questions pertinentes. M'ont également apporté de nouvelles pistes les projets en 2000 et 2001 du DEA de Sciences cognitives de Paris-Sud sur PRÉMA et, à l'université Paris X–Nanterre, le mémoire de master 1 sur *-esque* de Sandra Petel (2005) ainsi que les mémoires de master 1 et de master 2 sur PRÉMA de Sandie Morel (2004) et Roberte de la Taille (2006) respectivement.

Je remercie aussi les étudiant(e)s qui ont suivi depuis 2002–2003 les cours de maîtrise et DESS (devenus respectivement master 1 et 2) que j'ai consacrés à l'université Paris X aux bases de données et à leur utilisation. Particulièrement Anne Garcia-Fernandez pour m'avoir aidé dans l'un de ces cours et, pour le travail de création d'une base de données en ligne dans le cadre d'un projet de master professionnel : Christelle Ayache, Ségolène Baron, Aurélie Cousseau, Marion Crapet, Bérangère Decout, Sara El Ayarri, Virginie Heyd, Alexandra Jacomin, Anne Kuhn, Delphine Lagarde, Élise Lemaire, Claire Mierziak, Lise Pâris et Delphine Tribout. Deux collègues, Carole Tisset (IUFM de Versailles) et Dominique Fattier (université de Cergy-Pontoise), ont suivi l'un de ces cours et y ont contribué en fournissant deux études de cas : analyse de rédactions d'élèves, étude d'un morphème de créole haïtien.

Plusieurs collègues et ami(e)s m'ont aidé par leurs relectures et leurs idées : Valérie Beaudouin (FT R&D) ; Didier Bourigault (ERSS) ; Sammy Debaggi (LIMSI) ; Sara El Ayarri (LIMSI) ; Cécile Fabre (ERSS) ; Serge Fleury (SYLED) ; Anne Garcia-Fernandez (LIMSI) ; Marie Guégan (LIMSI) ; Sylvain Loiseau (LIMSI) ; Natacha Lubtchansky (université de Tours) ; Pierre Zweigenbaum (LIMSI) ; Carole Tisset (IUFM de Versailles) et Marie-Paule Péry-Woodley (ERSS).

⊕ LIENS

au 20 juillet 2009

Logiciels

Microsoft Access

`http://office.microsoft.com/fr-fr/FX010857911036.aspx`

MySQL `http://www.mysql.fr/` et sa documentation `http://dev.mysql.com/doc/` en version HTML ou PDF (A4) [plus de 1 600 pages]

OpenOffice Base `http://fr.openoffice.org/`

Cours et tutoriels

Tutoriels en français `http://sgbd.developpez.com/cours/`

(Georges Gardarin) `http://www.prism.uvsq.fr/~gardarin/home.html` (en particulier les transparents en anglais `http://www.prism.uvsq.fr/~gardarin/index.htm`)

Tutoriels en anglais de difficulté graduelle `http://www.sql.org/`

(Nicole Bidoit) `http://www.lri.fr/~bidoit/` (ouverture sur les bases de données déductives) Suppose de pouvoir afficher à l'écran des fichiers PostScript (cf. Ghostscript `http://www.cs.wisc.edu/~ghost/`)

⊕ ORIENTATIONS BIBLIOGRAPHIQUES

Les guides d'utilisation des gestionnaires de bases de données foisonnent. On utilisera autant que faire se peut les dernières éditions, y compris les rééditions quand elles existent pour les ouvrages cités ci-dessous : un changement de version d'un logiciel peut s'accompagner de remaniements importants. Ainsi le format de fichier a-t-il changé entre Access97 et Access 2000.

Pour les ouvrages d'informatique, on se fiera surtout à ceux des éditions Eyrolles, et, plus encore, à ceux des éditions O'Reilly, largement mises à contribution ci-dessous.

Bases de données : fondements

L'article historique, qui fonde le modèle relationnel est (Codd, 1970). Le modèle Entité-Association a été popularisé par (Chen, 1976).

Bases de données : approfondissements et et mise en œuvre

(Flory & Laforest, 2005) est centré sur la présentation du modèle relationnel. SQL est introduit précisément sous cet angle. Les mises en forme normale sont détaillées. Le modèle entité-relation également.

(Larrousse, 2006), complémentaire de (Hainaut, 2002) ou de (Akoka & Comyn-Wattiau, 2001), se cantonne cependant à la démarche qui conduit de l'analyse du domaine d'application à une réalisation SQL via une modélisation entité-association. De nombreux exercices corrigés complètent l'ouvrage.

Recherche approximative et expressions régulières

Les données en sciences humaines et sociales sont très souvent textuelles. Il importe donc de maîtriser parfaitement les mécanismes de recherche approximative, par expressions régulières. (Fourmond, 2005) en fournit une présentation précise et complète et néanmoins concise, tout en montrant la place des expressions régulières dans les langages de programmation récents comme Java et dans les langages « à tout faire » que sont Perl, PHP, Python.

Constitution et utilisation de corpus

La portée de (Baude, 2006) ne se limite pas aux corpus oraux, comme son titre pourrait le laisser penser. Ce livre aborde en particulier les aspects juridiques de la constitution d'un corpus : consentement éclairé des locuteurs enregistrés, etc. Pour une mise en perspective plus large, on se reportera à (Habert *et al.*, 1997) ou, mieux, à (Kennedy, 1998).

SQL et dialectes SQL

Malgré la collection dans laquelle il figure (. . . *pour les nuls*), (Taylor, 2001) fournit une introduction à la fois large et précise à SQL. Il aborde des caractéristiques avancées comme les requêtes imbriquées, la protection d'une base de données et celle des données elles-mêmes (volet transactionnel). Il présente la norme SQL 1999.

(Celko, 1999) se situe à l'autre extrémité du spectre. Même s'il correspond à des versions anciennes de la norme SQL (1989 et 1992), on y trouvera des développements précis sur des sujets comme le statut et le traitement de **`NULL`** ou encore les différents types de jointures. L'ouvrage indique également comment représenter en SQL des structures de données qui a priori ne s'y prêtent pas : les arbres et les graphes.

(Gennick, 2006) montre les opérateurs disponibles pour les versions SQL des SGBD très répandus : Oracle, IBM DB2, mais aussi et surtout les deux dialectes utilisés dans l'ouvrage, MS (MicroSoft) SQL Server, sous-jacent à Access, et MySQL.

(Brouard & Soutou, 2005) met l'accent sur la norme SQL (1999 et 2003). Il détaille les types de données, en particulier ceux introduits par ces versions de la norme SQL. Un chapitre est consacré aux différents types de jointure et aux opérations ensemblistes sur les tables (union, intersection, etc.). L'ouvrage, comme tous ceux de la collection, comprend de nombreux exercices corrigés.

MySQL

(Reese, 2004) constitue un aide-mémoire pratique pour les opérations courantes. (Metayer, 2001) fournit une présentation plus étendue, en particulier sur les types de données et fonctions associées. On y trouvera également des indications précieuses sur l'optimisation d'une base MySQL et sur son administration (gestion de droits, sauvegardes). (Rivereau & Pichot, 2002), qui avoisine les 1000 pages, relève des guides de référence, en particulier sur les aspects avancés (types de tables, sécurité, API). La documentation en ligne (et en français) de MySQL reste toutefois la source d'information la plus fiable, la plus complète et la plus à jour.

Bases de données en ligne

Un des usages majeurs des bases de données est aujourd'hui leur accès en ligne : réservation de places de transport ou de spectacle, commandes de livres, de films ou de disques, etc. La base de données contient l'état des stocks, celui des commandes et des clients. On se connecte à une de ces bases par un formulaire internet. Par ce biais, on choisit ce que l'on souhaite commander et l'on fournit ses coordonnées. Le formulaire est traité au fur et à mesure par des programmes qui interrogent la base de données pour vérifier la disponibilité de ce qui est commandé et qui en retour indiquent au client la suite donnée à sa commande. Les outils de gestion collaborative comme les wikis ou les blogues reposent aussi sur des bases de données et sur une interface de mise à jour.

Cet usage majeur sort lui aussi des limites du présent ouvrage. De nombreux ouvrages y sont consacrés. MySQL est très souvent mobilisé dans de telles architectures. C'est ainsi MySQL qui sert de socle au wiki, mediawiki, utilisé par les différentes versions de Wikipedia, ce qui témoigne indirectement de la fiabilité de MySQL pour gérer des bases de données très volumineuses et soumises à de très nombreuses requêtes. MySQL est souvent couplé dans ce cadre au langage de scripts PHP (Lerdorf, 2000) qui permet d'engendrer facilement les formulaires d'envoi de requêtes à la base de données et des programmes de traitement de ces requêtes. (Metayer, 2001) fournit un chapitre d'introduction à l'articulation PHP et MySQL. Des ouvrages comme (Nocton, 2001) vont plus loin, en reprenant les protocoles sous-jacents (HTTP) et en présentant les notions HTML nécessaires à la compréhension de l'engendrement de formulaires ou de pages présentant les résultats de requêtes à une base de données. Cependant, on se reportera de préférence à (Rigaux, 2001), dans la dernière édition disponible. Le lien est fait avec XML. En outre, la méthodologie de développement d'une base de données en ligne est présentée via un exemple suivi « réaliste ».

Access

Un certain nombre de guides de l'utilisateur pour Access consacrent une portion tout à fait congrue aux notions des SGBD. Ils visent plutôt le repérage des manipulations usuelles et de leurs variantes. Les copies d'écran, éventuellement incrustées d'éléments de repérage additionnels, facilitent la prise en main de l'interface graphique. C'est le cas par exemple de (Maran, 1999). Dans le même esprit, mais sous une forme plus concise, (Israel, 1997) comprend par contre un chapitre relativement substantiel et bien organisé sur les requêtes ainsi qu'un autre, précieux, sur l'administration et la sécurité. Dans cette famille de guides, (Nashe, 2000) offre un grand volume d'information, en particulier sur les relations et sur les tris, filtrages et requêtes (dont le recours à des motifs), auxquels trois chapitres sont consacrés. (Spona, 2001) comprend un chapitre très détaillé sur les formulaires et états.

Le parti pris de (Aubert, 2004) est diamétralement opposé : c'est une introduction aux SGBD et à leurs opérations et notions fondamentales, qui prend appui sur Access et un cas utilisé tout du long (la modélisation de recettes de cuisines). Le modèle Entités/Associations est présenté en détail, ainsi que les concepts connexes (dépendances fonctionnelles, normalisation, intégrité référentielle). En même temps, cet ouvrage constitue une bonne introduction aux aspects pratiques d'Access, qu'il s'agisse des requêtes, des formulaires ou des états. Une annexe juridique est consacrée aux liens entre bases de données et protection de la vie privée.

(Mata-Toledo & Cushman, 2002) s'appuie également sur Access pour ses exemples, mais ne développe pas un cas suivi. Même si l'ouvrage contient de nombreux exercices corrigés, la démarche privilégie les principes formels des bases relationnelles et détaille les dépendances fonctionnelles ainsi que le processus de normalisation. Curieusement, c'est *in fine* seulement qu'est abordé le modèle entité-relation.

(Bluttman, 2005) vise un public sachant et voulant programmer, désireux de construire des applications complexes sur Access (mode multi-utilisateur, contrôle fin de la saisie, protection des informations précieuses, etc.). Le chapitre sur les requêtes fait le lien entre interface et réalisation en SQL Server, il détaille la formulation de motifs. L'ouvrage traite particulièrement les relations entre Access et d'autres logiciels, dont Excel et Word, donc les fonctionnalités d'import et d'export (y compris en XML).

Importer, exporter, remodeler

(Ray, 2001) fournit une introduction lumineuse à XML et aborde XSLT.(Fitzgerald, 2004),

déjà un peu ancien, donne un aperçu de l'éventail large des traitements disponibles autour de XML et des logiciels correspondants.

(Amann & Rigaux, 2002) fournit une présentation d'ensemble de XSLT. Ce livre aborde également le lien avec les bases de données via XSLT. (Mangano, 2003), comme les autres ouvrages de la même collection « en action » chez O'Reilly, montre des solutions en XSLT à de nombreux problèmes « classiques » (utiliser XML pour produire du texte, des pages HTML, du XML transformé, etc.). Il détaille les parcours possibles dans les arbres des documents traités.

BIBLIOGRAPHIE

AKOKA J. & COMYN-WATTIAU I. (2001). *Conception des bases de données relationnelles en pratique*. Informatique. Paris : Vuibert.

AMANN B. & RIGAUX P. (2002). *Comprendre XSLT*. Paris : O'Reilly. Edition originale.

AUBERT J. (2004). *Informatique de gestion – Bases de données – Implémentation avec Access*. Technosup. Paris : Ellipses.

O. BAUDE, Rédacteur (2006). *Corpus oraux – Guide des bonnes pratiques 2006*. Paris : Presses universitaires d'Orléans & CNRS Éditions.

BEAUDOUIN V. (2002). *Mètre et rythmes du vers classique : Corneille et Racine*. Lettres numériques. Paris : Honoré Champion.

BLUTTMAN K. (2005). *Access à 200% : 100 techniques pour tirer le meilleur parti de vos données*. Paris : O'Reilly. Traduction d'Hervé Soulard.

BROUARD F. & SOUTOU C. (2005). *SQL*. Synthex. Paris : Pearson Education.

CELKO J. (1999). *SQL avancé*. Informatique. Paris : Vuibert. traduction de Martine Chalmond.

CHEN P. P. (1976). The Entity-Relationship Model – Towards a Unified View of Data. *ACM TODS*, **1**(1), 9–36.

CODD E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, **13**(6).

DATE C. J. (2000). *Introduction aux bases de données*. Informatique. Vuibert, 7^{e} edition. Traduction de Martine Chamond, Nora et Frédéric Cuppens.

FITZGERALD M. (2004). *XML hacks : 100 industrial-strength tips & tools*. Cambridge : O'Reilly.

FLORY A. & LAFOREST F. (2005). *Les bases de données relationnelles*. Gestion. Paris : Economica, 3ème edition.

FOURMOND V. (2005). *Les expressions régulières par l'exemple*. Technique & Pratique. Paris : H&K.

GARDARIN G. (2005). *Bases de données*. Best of. Paris : Eyrolles. 6ème tirage.

GENNICK J. (2006). *SQL précis & concis*. Paris : O'Reilly. Traduction de Éric Jacoboni.

HABERT B. (2005). *Instruments et ressources électroniques pour le français*. L'essentiel français. Gap/Paris : Ophrys.

HABERT B., NAZARENKO A. & SALEM A. (1997). *Les linguistiques de corpus*. U Linguistique. Paris : Armand Colin/Masson.

HAINAUT J.-L. (2002). *Bases de données et modèles de calcul. Outils et méthodes pour l'utilisateur*. Sciences Sup. Paris : Dunod, 3ème edition.

ISRAEL M. (1997). *Access 97*. Aide-mémoire. Paris : Dunod.

KENNEDY G. (1998). *An introduction to corpus linguistics*. Studies in language and linguistics. London : Longman.

LARROUSSE N. (2006). *Création de bases de données*. Synthex. Paris : Pearson Education.

LERDORF R. (2000). *PHP précis & concis*. Paris : O'Reilly. Traduction de James Guérin.

MANGANO S. (2003). *XSLT en action*. Paris : O'Reilly. Traduction d'Éric Jacoboni et Alain Ketterlin.

MARAN R. (1999). *Poche Visuel Access 2000*. 3-D Visuel. Paris : First Interactive.

MATA-TOLEDO R. A. & CUSHMAN P. K. (2002). *Introduction aux bases de données relationnelles*. Schaum's. Paris : EdiSciences/Dunod. Traduit de l'américain par Jean-Claude de Vos.

METAYER F. (2001). *MySQL*. PC Poche. Paris : Micro Application.

NASHE J. (2000). *Access 2000*. clic&zap. Paris : First Interactive.

NOCTON C. (2001). *PHP 4 et MySQL en ligne*. PC Poche. Paris : Micro Application.

PLÉNAT M. (1997). Analyse morpho-phonologique d'un corpus d'adjectifs dérivés en *-esque*. *French Language Studies*, (7), 163–179.

RAY E. T. (2001). *Introduction à XML*. Paris : O'Reilly. Traduction d'Alain Ketterlin.

REESE G. (2004). *MySQL précis & concis*. Paris : O'Reilly. Traduction de Guillaume Merck.

RIGAUX P. (2001). *Pratique de MySQL et PHP*. Paris : O'Reilly.

RIVEREAU N. & PICHOT A. (2002). *MySQL*. Référence. Paris : Micro Application.

SPONA H. (2001). *Access 2002*. PC Poche. Paris : Micro Application. Traduction de Hassina Abbasbhay et Pierre Wolf.

TAYLOR A. G. (2001). *SQL*. Pour les nuls. Paris : First Interactive. traduction de Geneviève Vassaux et Denis Duplan.

⊕ GLOSSAIRE : COMPLÉMENTS

Les définitions figurent directement dans le texte de la partie-papier. Elles sont également accessibles *via* la page en gras italique à l'entrée correspondante de l'index.

Annotation débarquée Les données primaires (le texte, le son, l'image, la vidéo) sont munies de coordonnées : par exemple, les caractères pour le texte, le déroulement du temps en milli-secondes pour le son. Les annotations sont conservées à part. Chaque annotation indique les références, dans le système de coordonnées, du segment de données primaires sur lequel elle porte.

Clause Composant d'une requête SQL : clause **SELECT**, clause **FROM**, clause **WHERE**, etc.

Écart-type L'écart-type mesure la dispersion des valeurs par rapport à la moyenne. C'est la racine carrée du carré des écarts de chaque valeur à la moyenne. Prendre le carré des écarts à la moyenne accroît la contribution des « grands écarts ».

Formes contractées Issues historiquement d'une préposition suivie d'un article ou d'un pronom relatif. Ainsi *des*, *au*, *desquels* proviennent de : *de les*, *à le*, *de lesquels.*

Invite Marque conventionnelle qui indique à l'utilisateur qu'il doit entrer une commande.

Langage artificiel Langage développé pour réaliser une tâche sur un ordinateur (programmation, gestion de bases de données) et conçu pour empêcher les ambiguïtés : une « phrase » d'un tel langage a une interprétation et une seule.

Langue naturelle Langue qui sert à la communication d'une population humaine donnée (français, québécois, espagnol, etc.). Elle se caractérise par l'ambiguïté et par la variation (phonétique, lexicale, syntaxique) des réalisations d'un « sens » donné.

Monde fermé (hypothèse du) On considère que la base de données comprend l'ensemble des assertions pertinentes du domaine d'application. L'absence d'une assertion équivaut au fait que cette assertion est considérée comme fausse.

Multi-ensembles Voir Tables multi-ensembles.

Norme homologation explicite d'un langage par une instance de normalisation, nationale, comme l'AFNOR (*Association Française de Normalisation*), ou internationale, comme ISO (*International Organization for Standardization*). Le métalangage de description et d'échange de documents, SGML, ancêtre d'XML, a été adopté comme une norme par ISO en 1986.

Pouvoir expressif Ensemble des opérations qui peuvent s'exprimer dans un langage artificiel donné.

Prompt (anglais) Voir Invite.

Question fermée (PRÉMA) Question dont la réponse possible figure dans une liste pré-définie

Question ouverte (PRÉMA) Question dont la réponse est libre.

Standard homologation implicite d'un langage par une communauté d'utilisateurs. XML, métalangage de description et d'échange de documents qui est le « descendant » de SGML, est une proposition de standard international, promue en particulier par le consortium qui gère le Web, W3C. Standard de fait, XML est engagé dans le processus permettant d'en faire une norme.

SGBD voir Système de gestion de bases de données

Tables multi-ensembles Tables contenant des doublons, des lignes identiques et qui ne constituent donc pas des relations.

INDEX

Nombre de requêtes : 121

Nombre d'exercices : 71

Nombre de captures d'écran : 669