Contact avec l'auteur – Corrigés des exercices

Si vous avez des remarques à formuler sur le contenu de cet ouvrage, n'hésitez pas à m'écrire à l'adresse `soutou@iut-blagnac.fr`. Ne me demandez pas de déboguer votre procédure cataloguée ou d'optimiser une de vos requêtes… Seules les remarques relatives à l'ouvrage trouveront une réponse.

Par ailleurs, un site d'accompagnement de l'ouvrage (*errata*, corrigés des exercices, source des exemples et compléments) est en ligne et accessible via www.editions-eyrolles.com.

- Ils offrent la possibilité de stocker des informations non structurées (comme le texte, l'image, etc.) dans des champs appelés LOB (*Large Object Binary*).

Nous étudierons les principales instructions SQL de MySQL qui sont classifiées dans le tableau suivant :

Tableau 0-1 Classification des ordres SQL

Ordres SQL	Aspect du langage
`CREATE – ALTER – DROP - RENAME – TRUNCATE`	Définition des données (LDD)
`INSERT – UPDATE – DELETE - LOCK TABLE`	Manipulation des données (LMD)
`SELECT`	Interrogation des données (LID)
`GRANT – REVOKE – COMMIT – ROLLBACK – SAVEPOINT - SET TRANSACTION`	Contrôle des données (LCD)

Modèle de données

Le modèle de données relationnelles repose sur une théorie rigoureuse bien qu'adoptant des principes simples. La table relationnelle (*relational table*) est la structure de données de base qui contient des enregistrements appelés aussi « lignes » (*rows*). Une table est composée de colonnes (*columns*) qui décrivent les enregistrements.

Tables et données

Considérons la figure suivante qui présente deux tables relationnelles permettant de stocker des compagnies, des pilotes et le fait qu'un pilote soit embauché par une compagnie :

Figure 0-1 *Deux tables*

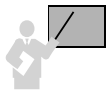
`Compagnie`

comp	nrue	rue	ville	nomComp
AF	10	Gambetta	Paris	Air France
SING	7	Camparols	Singapour	Singapore AL

`Pilote`

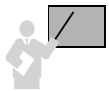
brevet	nom	nbHVol	*compa*
PL-1	Louise Ente	450	AF
PL-2	Jules Ente	900	AF
PL-3	Paul Soutou	1000	SING

Les clés



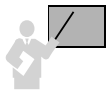
La clé primaire (*primary key*) d'une table est l'ensemble minimal de colonnes qui permet d'identifier de manière unique chaque enregistrement.

Dans la figure précédente, les colonnes « clés primaires » sont notées en gras. La colonne `comp` représente le code de la compagnie et la colonne `brevet` décrit le numéro du brevet.



Une clé est dite « candidate » (*candidate key*) si elle peut se substituer à la clé primaire à tout instant. Une table peut contenir plusieurs clés candidates ou aucune.

Dans notre exemple, les colonnes `nomComp` et `nom` peuvent être des clés candidates si on suppose qu'aucun homonyme n'est permis.



Une clé étrangère (*foreign key*) référence dans la majorité des cas une clé primaire d'une autre table (sinon une clé candidate sur laquelle un index unique aura été défini). Une clé étrangère est composée d'une ou de plusieurs colonnes. Une table peut contenir plusieurs clés étrangères ou aucune.

Dans notre exemple, la colonne `comp` (notée en italique dans la figure) est une clé étrangère, car elle permet de référencer un enregistrement unique de la table `Compagnie` via la clé primaire `comp`.

Le modèle relationnel est ainsi fondamentalement basé sur les valeurs. Les associations entre tables sont toujours binaires et assurées par les clés étrangères. Les théoriciens considèrent celles-ci comme des pointeurs logiques. Les clés primaires et étrangères seront définies dans les tables en SQL à l'aide de contraintes.

MySQL

MySQL est à la fois le nom du SGBD et le nom de la société (qui se nomme en fait MySQL AB, décrite sur http://www.mysql.com) dont le siège se trouve en Suède à Uppsala – compter une cinquantaine de kilomètres au nord de Stockholm. Selon leurs dires, leur serveur de données, qui est écrit en C et C++, serait installé de façon opérationnelle sur plus de six millions de sites. D'un point de vue coût, l'utilisation du SGBD sur des projets importants (entre 250 000 € et 500 000 €) ferait économiser 90 % sur le prix des licences du serveur, 60 % sur les ressources système, 70 % sur le prix du matériel, 50 % sur les tâches d'administration et de support.

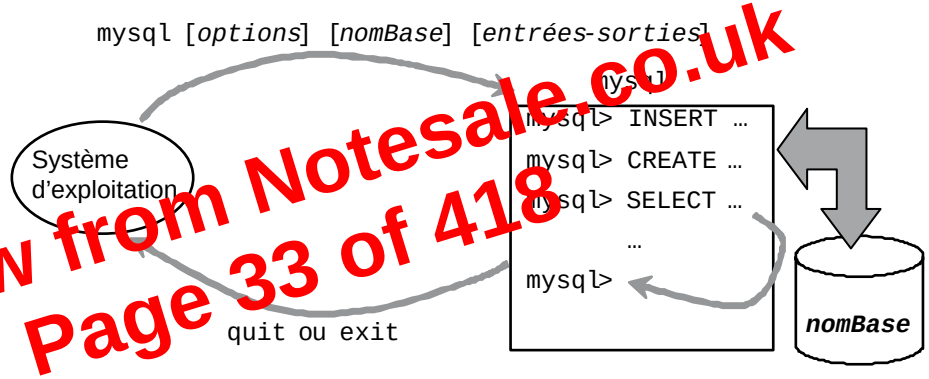
(c'est le mode le plus courant), le résultat des extractions est présenté sous une forme tabulaire au format ASCII.

Vous verrez qu'il est notamment possible :

- d'exécuter des instructions SQL (créér des tables, manipuler des données, extraire des informations, etc.) ;
- de compiler des procédures cataloguées et des déclencheurs ;
- de réaliser des tâches d'administration (création d'utilisateurs, attribution de privilèges, etc.).

Le principe général de l'interface est le suivant : après une connexion locale ou distante, des instructions sont saisies et envoyées à la base qui retourne des résultats affichés dans la même fenêtre de commande.

Figure 0-5 *Principe général de l'interface de commande en ligne*



N'ayez pas honte de bien maîtriser cette interface au lieu de connaître toutes les options d'un outil graphique (comme *PhpMyAdmin*, *MySQL Administrator* ou autre). Il vous sera toujours plus facile de vous adapter aux différents boutons et menus, tout en connaissant les instructions SQL, que l'inverse.

Imaginez-vous un jour à Singapour sur une machine ne disposant pas d'outils graphiques, que le client vous demande la réduction que vous pouvez lui faire sur la vente d'une piscine intérieure d'un Airbus A380 et que vous devez interroger (ou mettre à jour) une table sur le serveur du siège social à Blagnac. Vous ne savez pas vous servir de l'interface en ligne : vous n'êtes pas un vrai informaticien !

Création d'un utilisateur

Vous allez maintenant créer un utilisateur MySQL. Ouvrez le fichier `premierPas.sql` qui se trouve dans le répertoire `Introduction`, à l'aide du bloc-notes (ou d'un éditeur de texte de votre

Types des colonnes

Pour décrire les colonnes d'une table, MySQL fournit les types prédéfinis suivants (*built-in datatypes*) :

- caractères (`CHAR`, `VARCHAR`, `TINYTEXT`, `TEXT`, `MEDIUMTEXT`, `LONGTEXT`) ;
- valeurs numériques (`TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, `INTEGER`, `BIGINT`, `FLOAT`, `DOUBLE`, `REAL`, `DECIMAL`, `NUMERIC`, et `BIT`) ;
- date/heure (`DATE`, `DATETIME`, `TIME`, `YEAR`, `TIMESTAMP`) ;
- données binaires (`BLOB`, `TINYBLOB`, `MEDIUMBLOB`, `LONGBLOB`) ;
- énumérations (`ENUM`, `SET`).

Détaillons à présent ces types. Nous verrons comment utiliser les plus courants au chapitre 2 et les autres au fil de l'ouvrage.

Caractères

Le type `CHAR` permet de stocker des chaînes de caractères de taille fixe. Les valeurs sont stockées en ajoutant, s'il le faut, des espaces (*trailing spaces*) à concurrence de la taille définie. Ces espaces ne seront pas considérés après extraction à partir de la table.

Le type `VARCHAR` permet de stocker des chaînes de caractères de taille variable. Les valeurs sont stockées sans l'ajout d'espaces à concurrence de la taille définie. Depuis la version 5.0.3 de MySQL, les éventuels espaces de fin de chaîne seront stockés et extraits en conformité avec la norme SQL. Des caractères Unicode (méthode de codage universelle qui fournit une valeur de code unique pour chaque caractère quels que soient la plate-forme, le programme ou la langue) peuvent aussi être stockés.

Tableau 1-4 Types de données caractères

Type	Description	Commentaire pour une colonne
`CHAR(`*n*`) [BINARY | ASCII | UNICODE]`	Chaîne fixe de *n* octets ou caractères.	Taille fixe (maximum de 255 caractères).
`VARCHAR(`*n*`) [BINARY]`	Chaîne variable de *n* caractères ou octets.	Taille variable (maximum de 65 535 caractères).
`BINARY(`*n*`)`	Chaîne fixe de *n* octets.	Taille fixe (maximum de 255 octets).
`VARBINARY(`*n*`)`	Chaîne variable de *n* octets.	Taille variable (maximum de 255 octets).
`TINYTEXT(`*n*`)`	Flot de *n* octets.	Taille fixe (maximum de 255 octets).
`TEXT(`*n*`)`	Flot de *n* octets.	Taille fixe (maximum de 65 535 octets).
`MEDIUMTEXT(`*n*`)`	Flot de *n* octets.	Taille fixe (maximum de 16 mégaoctets).
`LONGTEXT(`*n*`)`	Flot de *n* octets.	Taille fixe (maximum de 4,29 gigaoctets).

Remplacement d'un enregistrement

L'instruction REPLACE consiste, comme son nom l'indique, à remplacer un enregistrement dans sa totalité (toutes ses colonnes). Il faut avoir les privilèges INSERT et DELETE sur la table. C'est selon la valeur de la clé primaire ou celle d'un index unique que l'enregistrement sera remplacé.

Si la table ne dispose pas d'une contrainte PRIMARY KEY ou UNIQUE, l'utilisation de REPLACE n'a pas de sens et devient équivalente à INSERT.

La syntaxe simplifiée de l'instruction UPDATE est la suivante :

```
REPLACE [LOW_PRIORITY | DELAYED]
      [INTO] [nomBase.] nomTable [(colonne1,...)]
      VALUES ({expression1 | DEFAULT},...) [,(...),...]
```

- LOW_PRIORITY et DELAYED ont la même signification que pour INSERT et UPDATE.
- VALUES contient les valeurs de remplacement.

L'instruction suivante remplace l'enregistrement relatif à la compagnie de code 'AN1' (voir figure 2-6) :

```
REPLACE INTO Compagnie VALUES ('AN1', 24, 'Salas', 'Ramonville',
'AL- RENATO');
```


Suppressions d'enregistrements

Les instructions DELETE et TRUNCATE permettent de supprimer un ou plusieurs enregistrements d'une table. Pour pouvoir supprimer des enregistrements dans une table, il faut que cette dernière soit dans votre base ou que vous ayez reçu le privilège DELETE sur la table.

Instruction DELETE

La syntaxe simplifiée de l'instruction DELETE est la suivante :

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM [nomBase.] nomTable
      [WHERE (condition)]
      [ORDER BY listeColonnes]
      [LIMIT nbreLimite]
```

- LOW_PRIORITY, IGNORE et LIMIT ont la même signification que pour UPDATE.

- `QUICK` (pour les tables de type `MyISAM`) ne met pas à jour les index associés pour accélérer le traitement.
- La condition du `WHERE` sélectionne les lignes à supprimer dans la table. Si aucune condition n'est précisée, toutes les lignes seront détruites. Si la condition ne sélectionne aucune ligne, aucun enregistrement ne sera supprimé.
- `ORDER BY` réalise un tri des enregistrements qui seront effacés dans cet ordre.

Détaillons les possibilités de cette instruction en considérant les différentes tables précédemment définies. La première commande supprime tous les pilotes de la compagnie de code 'AF', la seconde, avec une autre écriture, détruit la compagnie de code 'AF'.


```
DELETE FROM Pilote WHERE compa = 'AF';

DELETE FROM Compagnie WHERE comp = 'AF';
```

Tentons de supprimer une compagnie qui est référencée par un pilote à l'aide d'une clé étrangère. Il s'affiche une erreur qui sera expliquée dans la section *Intégrité référentielle*.

```
mysql> DELETE FROM Compagnie WHERE comp = 'SING';
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign
key constraint fails (`bd_boutou/pilote`, CONSTRAINT `fk_Pil_compa_
Comp` FOREIGN KEY (`compa`) REFERENCES `compagnie` (`comp`))
```

Détruisons enfin les deux premières compagnies triées par ordre croissant de clé, ici 'AC' et 'AN1' qui ne sont référencées par aucun pilote.

```
DELETE FROM Compagnie LIMIT 2;
```


Instruction TRUNCATE

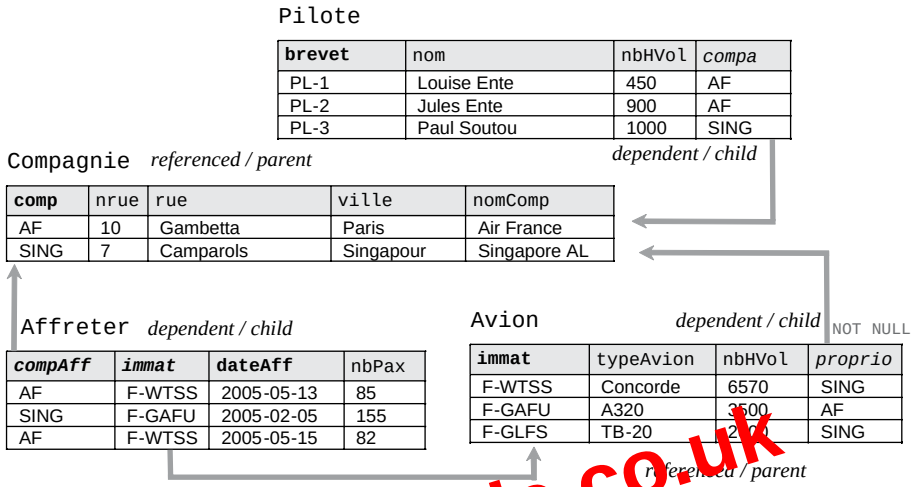
La commande `TRUNCATE` est une extension de SQL qui a été proposée par Oracle et reprise par MySQL. Cette commande supprime tous les enregistrements d'une table et libère éventuellement l'espace de stockage utilisé par la table. La syntaxe est la suivante :

```
TRUNCATE [TABLE] [nomBase.] nomTable;
```

Avec le moteur InnoDB, l'opération est programmée en `DELETE`. Pour les autres moteurs, l'opération diffère de `DELETE` de la manière suivante :

- La table est supprimée (`DROP`) puis recréée (`CREATE`), ce qui est plus rapide que de détruire les enregistrements un à un.
- L'opération peut être interrompue si une transaction active utilise la table (ou si un verrou est posé).
- Le nombre d'enregistrements supprimés n'est pas retourné.
- L'éventuelle dernière valeur d'une colonne `AUTO_INCREMENT` n'est pas mémorisée.

Figure 2-11 *Intégrité référentielle*



Deux types de problèmes sont automatiquement résolus par MySQL pour assurer l'intégrité référentielle :

- La cohérence du « fils » vers le « père » : on ne doit pas pouvoir insérer un enregistrement « fils » (ou modifier sa clé étrangère) rattaché à un enregistrement « père » inexistant. Il est cependant possible d'insérer un « fils » (ou de modifier sa clé étrangère) sans rattacher d'enregistrement « père », à la condition qu'il n'existe pas de contrainte `NOT NULL` au niveau de la clé étrangère.
- La cohérence du « père » vers le « fils » : on ne doit pas pouvoir supprimer un enregistrement « père » si un enregistrement « fils » y est encore rattaché. Il est possible de supprimer les « fils » associés (`DELETE CASCADE`), d'affecter la valeur nulle aux clés étrangères des « fils » associés (`DELETE SET NULL`) ou de répercuter une modification de la clé primaire du père (`UPDATE CASCADE` et `UPDATE SET NULL`).

Déclarons à présent ces contraintes sous MySQL en détaillant les options disponibles.

Contraintes côté « père »

Le tableau suivant illustre les deux possibilités (clé primaire ou candidate) dans le cas de la table `Compagnie`. Notons que pour le cas de la clé candidate, une clé primaire peut être définie par ailleurs (sur `nomComp` par exemple).

Tableau 2-13 Écritures des contraintes de la table « père »

Clé primaire	Clé candidate
`CREATE TABLE Compagnie (comp CHAR(4), nrue INTEGER(3), rue CHAR(20), ville CHAR(15), nomComp CHAR(15), CONSTRAINT pk_Compagnie PRIMARY KEY(comp));`	`CREATE TABLE Compagnie (comp CHAR(4) NOT NULL, nrue INTEGER(3), rue CHAR(20), ville CHAR(15), nomComp CHAR(15), CONSTRAINT un_Compagnie UNIQUE(comp), CONSTRAINT pk_Compagnie PRIMARY KEY(nomComp));`

Contraintes côté « fils »

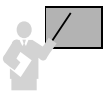
Indépendamment de l'écriture de la table « père », plusieurs écritures sont possibles au niveau de la table « fils » selon qu'on créé les index ou qu'on laisse MySQL le faire, et selon qu'on nomme ou pas la contrainte de clé étrangère.

La première écriture nomme la contrainte, mais ne précise rien à propos de l'index. La deuxième ne nomme pas la contrainte, mais définit l'index (sans option toutefois). L'écriture à adopter est un mélange des deux, à savoir nommer les contraintes et décrire les index.

Tableau 2-14 Écritures des contraintes de la table « fils »

Contrainte nommée sans index	Contrainte pas nommée et index
`CREATE TABLE Pilote (brevet CHAR(6), nom CHAR(15), nbHVol DECIMAL(7,2), compa CHAR(4), CONSTRAINT pk_Pilote PRIMARY KEY(brevet), CONSTRAINT fk_Pil_compa_Comp FOREIGN KEY (compa) REFERENCES Compagnie(comp));`	`CREATE TABLE Pilote (brevet CHAR(6), nom CHAR(15), nbHVol DECIMAL(7,2), compa CHAR(4), CONSTRAINT pk_Pilote PRIMARY KEY(brevet), INDEX (compa), FOREIGN KEY (compa) REFERENCES Compagnie(comp));`

Clés composites et nulles



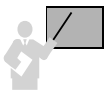
Les clés étrangères ou primaires peuvent être définies sur plusieurs colonnes (16 au maximum), on parle de *composite keys*.

Des clés étrangères peuvent être nulles (si elles ne font pas partie d'une clé primaire) si aucune contrainte `NOT NULL` n'est déclarée.

Décrivons à présent le script SQL qui convient à notre exemple (la syntaxe de création des deux premières tables a été discutée plus haut) et étudions ensuite les mécanismes programmés par ces contraintes. Décidons qu'un avion aura toujours un propriétaire (`NOT NULL`).

```
CREATE TABLE Avion
 (immat CHAR(6), typeAvion CHAR(15), nbhVol DECIMAL(10,2),
  proprio CHAR(4) NOT NULL, CONSTRAINT pk_Avion PRIMARY KEY(immat),
  INDEX (proprio),
  CONSTRAINT fk_Avion_comp_Compag
      FOREIGN KEY(proprio) REFERENCES Compagnie(comp));
CREATE TABLE Affreter
 (compAff CHAR(4), immat CHAR(6), dateAff DATE, nbPax INTEGER(3),
  CONSTRAINT pk_Affreter PRIMARY KEY (compAff, immat, dateAff),
  INDEX (immat),
  CONSTRAINT fk_Aff_na_Avion
      FOREIGN KEY(immat) REFERENCES Avion(immat),
 INDEX (compAff),
 CONSTRAINT fk_Aff_comp_Compag
     FOREIGN KEY(compAff) REFERENCES Compagnie(comp));
```

Cohérence du fils vers le père



Si la clé étrangère est déclarée NOT NULL, l'insertion d'un enregistrement « fils » n'est possible que s'il est rattaché à un enregistrement « père » existant. Dans le cas inverse, l'insertion d'un enregistrement « fils » rattaché à aucun « père » est possible.

Le tableau suivant décrit des insertions correctes et une incorrecte. Le message d'erreur est ici en anglais (il vous question de ne pouvoir ajouter un enregistrement « fils »).

Tableau 2-15 Insertions correctes et incorrectes

Web

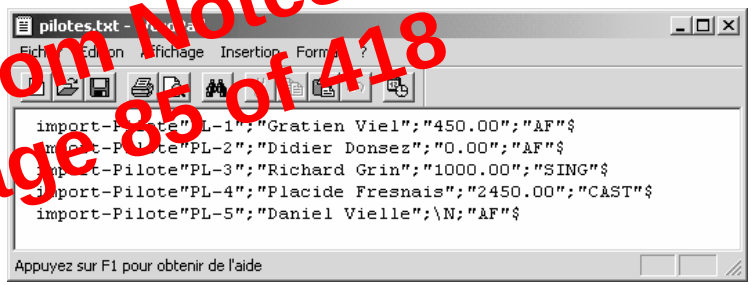
Insertions correctes	Insertion incorrecte
`-- fils avec père` `INSERT INTO Pilote VALUES` `('PL-3', 'Paul Soutou', 1000, 'SING');` `-- fils sans père` `INSERT INTO Pilote VALUES` `('PL-4', 'Un Connu', 0, NULL);` `-- fils avec pères` `INSERT INTO Avion VALUES` `('F-WTSS', 'Concorde', 6570, 'SING');` `INSERT INTO Affreter VALUES` `('AF', 'F-WTSS', '15-05-2003', 82)`	`-- avec père inconnu` `mysql> INSERT INTO Pilote VALUES` `('PL-5', 'Pb de Compagnie', 0, '?');` `ERROR 1452 (23000): Cannot add or update` `a child row: a foreign key constraint` `fails (`bdsoutou/pilote`, CONSTRAINT` `fk_Pil_compa_Comp` FOREIGN KEY (`compa`)` `REFERENCES `compagnie` (`comp`))`

Pour insérer un affrètement, il faut donc avoir ajouté au préalable au moins une compagnie et un avion. Le chargement de la base de données est conditionné par la hiérarchie des contraintes référentielles. Ici il faut insérer d'abord les compagnies, puis les pilotes (ou les avions), enfin les affrètements.

- `FIELDS` décrit comment sont formatées dans le fichier les valeur à insérer dans la table. En l'absence de cette clause, `TERMINATED BY` vaut `'\t'`, `ENCLOSED BY` vaut `''` et `ESCAPED BY` vaut `'\\'`.
  - `FIELDS TERMINATED BY` décrit le caractère qui sépare deux valeurs de colonnes.
  - `FIELDS ENCLOSED BY` permet de contrôler le caractère qui encadrera chaque valeur de colonne.
  - `FIELDS ESCAPED BY` permet de contrôler les caractères spéciaux.
- `LINES` décrit comment seront écrites dans le fichier les lignes extraites de(s) table(s). En l'absence de cette clause, `TERMINATED BY` vaut `'\n'` et `STARTING BY` vaut `''`.
- `IGNORE` permet de ne pas importer les *nb* premières lignes du fichier (contenant des éventuelles déclarations).

Lisons le fichier « `pilotes.txt` » situé dans le répertoire « `D:\dev` » (ouvert à l'aide du WordPad dans la figure suivante), en important la totalité des données dans la table `Pilote2` créée à cet effet (`brevet VARCHAR(6), nom VARCHAR(16), nbHVol DECIMAL(7,2), compa CHAR(4) et PRIMARY KEY(brevet)`). Notez l'utilisation du double « \ » pour désigner une arborescence Windows. Le caractère `NULL` est importé par le caractère « \N ».

***Figure 2-12** Importation de données*




```
LOAD DATA INFILE 'D:\\dev\\pilotes.txt'  REPLACE INTO TABLE Pilote2
        FIELDS TERMINATED BY ';'  ENCLOSED BY '"'
        LINES STARTING BY 'import -Pilote' TERMINATED BY '$ \n';
```

Une fois la table créée, il est possible de l'interroger :

```
mysql> SELECT * FROM Pilote2 ORDER BY compa, nom;
+--------+-----------------+---------+-------+
| brevet | nom             | nbHVol  | compa |
+--------+-----------------+---------+-------+
| PL-5   | Daniel Vielle   |    NULL | AF    |
| PL-2   | Didier Donsez   |    0.00 | AF    |
| PL-1   | Gratien Viel    |  450.00 | AF    |
| PL-4   | Placide Fresnais | 2450.00 | CAST  |
| PL-3   | Richard Grin    | 1000.00 | SING  |
+--------+-----------------+---------+-------+
```

Exercices

Les objectifs de ces exercices sont :

- d'insérer des données dans les tables du schéma *Parc Informatique* ;
- de créer une séquence et d'insérer des données en utilisant une séquence ;
- de modifier des données.

Exercice 2.1 Insertion de données

Écrire puis exécuter le script SQL (que vous appellerez `insParc.sql`) afin d'insérer les données dans les tables suivantes :

Tableau 2-18 Données des tables

Table	Données
Segment	**INDIP** NOMSEGMENT ETAGE 130.120.80 Brin RDC 130.120.81 Brin 1er étage 130.120.82 Brin 2e étage
Salle	**NSALLE** NOMSALLE NBPOSTE INDIP s01 Salle 1 3 130.120.80 s02 Salle 2 2 130.120.80 s03 Salle 3 2 130.120.80 s11 Salle 11 2 130.120.81 s12 Salle 12 1 130.120.81 s21 Salle 21 2 130.120.82 s22 Salle 22 0 130.120.83 s23 Salle 23 0 130.120.83
Poste	**NPOSTE** NOMPOSTE INDIP AD TYPEPOSTE NSALLE p1 Poste 1 130.120.80 01 TX s01 p2 Poste 2 130.120.80 02 UNIX s01 p3 Poste 3 130.120.80 03 TX s01 p4 Poste 4 130.120.80 04 PCWS s02 p5 Poste 5 130.120.80 05 PCWS s02 p6 Poste 6 130.120.80 06 UNIX s03 p7 Poste 7 130.120.80 07 TX s03 p8 Poste 8 130.120.81 01 UNIX s11 p9 Poste 9 130.120.81 02 TX s11 p10 Poste 10 130.120.81 03 UNIX s12 p11 Poste 11 130.120.82 01 PCNT s21 p12 Poste 12 130.120.82 02 PCWS s21

Figure 3-7 *Avant la désactivation de contraintes*

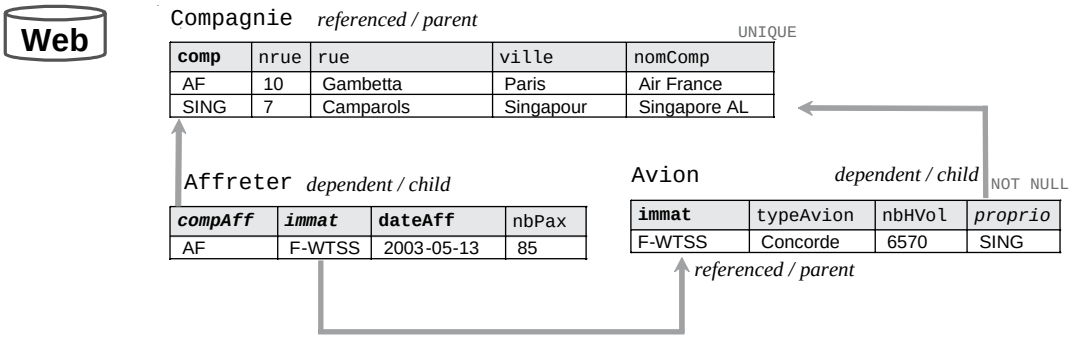


Tableau 3-3 Insertions après la désactivation de l'intégrité référentielle

Instructions valides	Instructions non valides
mysql> **SET FOREIGN_KEY_CHECKS=0;**	
mysql> INSERT INTO Avion VALUES ('F-GLFS', 'TB-22', 50, **'Toto'**); Query OK, 1 row affected (0.04 sec) mysql> INSERT INTO Avion VALUES ('**Bidon1**', 'TB-21', 100, 'AF'); Query OK, 1 row affected (0.10 sec) mysql> INSERT INTO Affreter VALUES ('AF', '**Toto**', '2005-05-13', 0); Query OK, 1 row affected (0.10 sec) mysql> INSERT INTO Affreter VALUES ('**GTI**', 'F-WTSS', '2005-11-07',10); Query OK, 1 row affected (0.02 sec) mysql> INSERT INTO Affreter VALUES ('**GTI**', '**Toto**', '2005-11-07',40); Query OK, 1 row affected (0.03 sec)	mysql> INSERT INTO Compagnie VALUES ('GTR', 1, 'Brassens', 'Blagnac', **'Air France'**); ERROR **1062** (23000): Duplicate entry 'Air France' for key 2 mysql> INSERT INTO Avion VALUES ('Bidon1', 'TB-20', 2000, **NULL**); ERROR **1048** (23000): Column 'proprio' cannot be null mysql> INSERT INTO Avion VALUES ('F-GLFS', 'TB-21', 1000, 'AF'); ERROR **1062** (23000): Duplicate entry 'F-GLFS' for key 1

On remarque bien que seules les clés étrangères ne sont plus vérifiées. Toute autre contrainte (`UNIQUE`, `PRIMARY KEY` et `NOT NULL`) reste active. L'état de la base est désormais comme suit.

Bien qu'il semble incohérent de réactiver les contraintes sans s'occuper au préalable des valeurs ne respectant pas l'intégrité référentielle (données notées en gras), nous verrons qu'il est possible de le faire.

- La restriction qui est programmée dans le `WHERE` de la requête permet de restreindre la recherche à une ou plusieurs lignes. Dans notre exemple une restriction répond à la question : « *Quels sont les avions de type A320 ?* ».
- La projection qui est programmée dans le `SELECT` de la requête permet d'extraire une ou plusieurs colonnes. Dans notre exemple elle répond à la question : « *Quels sont les numéros de brevet et les nombres d'heures de vol de tous les pilotes ?* ».
- La jointure qui est programmée dans le `WHERE` de la requête permet d'extraire des données de différentes tables en les reliant deux à deux (le plus souvent à partir de contraintes référentielles). Dans notre exemple la première jointure répond à la question « *Quels sont les numéros de brevet et nombres d'heures de vol des pilotes de la compagnie de nom Air France ?* ». La deuxième jointure répond à la question : « *Quels sont les avions de la compagnie de nom Air France ?* ».

En combinant ces trois fonctionnalités, toute question logique devrait trouver en théorie une réponse par une ou plusieurs requêtes. Les questions trop complexes peuvent être programmées à l'aide des vues (chapitre 5) ou par traitement (programmes mélangeant requêtes et instructions procédurales).

Syntaxe (SELECT)

Pour pouvoir extraire les enregistrements d'une table, il faut que celle-ci soit dans votre base ou que vous ayez reçu le privilège `SELECT` sur la table.

La syntaxe SQL simplifiée de l'instruction `SELECT` est la suivante :

```
SELECT [ { DISTINCT | DISTINCTROW } | ALL ] listeColonnes
 FROM nomTable1 [,nomTable2]...
 [ WHERE condition ]
 [ clauseRegroupement ]
 [ HAVING condition ]
 [ clauseOrdonnancement ]
 [ LIMIT [rangDépart,] nbLignes ] ;
```

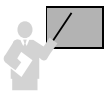
Nous détaillerons chaque option à l'aide d'exemples au cours de ce chapitre.

Pseudotable

La pseudotable est une table qui n'a pas de nom et qui est utile pour évaluer une expression de la manière suivante : « `SELECT expression;` ». Les résultats fournis seront uniques (si aucune jointure ou opérateur ensembliste ne sont employés dans l'interrogation).

Alias

Les alias permettent de renommer des colonnes à l'affichage ou des tables dans la requête. Les alias de colonnes sont utiles pour les calculs.



L'utilisation de la directive AS est facultative (pour se rendre conforme à SQL2).

Il faut préfixer les colonnes par l'alias de la table lorsqu'il existe.

Tableau 4-4 Alias (colonnes et tables)

Alias de colonnes

```
SELECT compa AS c1,
       nom AS NometPrenom, brevet c3
   FROM Pilote;

+------+------------------+------+
| c1   | NometPrenom      | c3   |
+------+------------------+------+
| AF   | Gratien Viel     | PL-1 |
| AF   | Didier Donsez    | PL-2 |
| SING | Richard Grin     | PL-3 |
| CAST | Placide Fresnais | PL-4 |
| AF   | Daniel Vielle    | PL-5 |
+------+------------------+------+
```

Alias de table

```
SELECT aliasPilotes.compa AS c1,
       aliasPilotes.nom
   FROM Pilote aliasPilotes;

+------+------------------+
| c1   | nom              |
+------+------------------+
| AF   | Gratien Viel     |
| AF   | Didier Donsez    |
| SING | Richard Grin     |
| CAST | Placide Fresnais |
| AF   | Daniel Vielle    |
+------+------------------+
```


Il semble préférable d'utiliser la directive AS afin qu'il n'y ait pas d'ambiguïté dans l'expression (oubli d'une virgule) `SELECT col1 col2 FROM nomTable` où MySQL interprétera la seconde colonne comme un alias de la première.

Duplicatas

Les directives `DISTINCT` ou `DISTINCTROW` éliminent les éventuels duplicatas. Pour la deuxième requête, les écritures « `DISTINCT compa` », « `DISTINCTROW(compa)` » et « `DISTINCTROW compa` » sont équivalentes. La notation entre parenthèses est nécessaire lorsque l'on désire éliminer des duplicatas par paires, triplets, etc.

Figure 4-3 *Table Pilote*

Pilote

brevet	nom	nbHVol	prime	compa
PL-1	Gratien Viel	450	500	AF
PL-2	Didier Donsez	0		AF
PL-3	Richard Grin	1000	90	SING
PL-4	Placide Fresnais	2450	500	CAST
PL-5	Daniel Vielle	400	600	SING
PL-6	Francoise Tort		0	CAST

Opérateurs de comparaison

Le tableau suivant décrit des requêtes pour lesquelles la clause `WHERE` contient des opérateurs de comparaison.

Les écritures « `prime=500` » et « `(prime=500)` » sont équivalentes. Les écritures « `prime<>500` » et « `NOT (prime=500)` » sont équivalentes. Les parenthèses sont utiles pour composer des conditions.

Notez l'utilisation du simple guillemet pour comparer des chaînes de caractères.

Tableau 4-11 Égalité, inégalité et comparaison

Égalité

```
SELECT brevet, nom AS "Prime 500"
       FROM Pilote WHERE prime = 500 ;
+--------+------------------+
| brevet | Prime 500        |
+--------+------------------+
| PL-1   | Gratien Viel     |
| PL-4   | Placide Fresnais |
+--------+------------------+
```

```
SELECT brevet, nom "de Air-France"
       FROM Pilote WHERE compa = 'AF';
+--------+---------------+
| brevet | de Air-France |
+--------+---------------+
| PL-1   | Gratien Viel  |
| PL-2   | Didier Donsez |
+--------+---------------+
```

Comparaison et inégalité

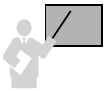
```
SELECT brevet, nom, prime
       FROM Pilote WHERE prime <= 400 ;

+--------+----------------+-------+
| brevet | nom            | prime |
+--------+----------------+-------+
| PL-3   | Richard Grin   |    90 |
| PL-6   | Francoise Tort |     0 |
+--------+----------------+-------+
```

```
SELECT brevet, nom, prime FROM Pilote
       WHERE prime <> 500 ;
+--------+----------------+-------+
| brevet | nom            | prime |
+--------+----------------+-------+
| PL-3   | Richard Grin   |    90 |
| PL-5   | Daniel Vielle  |   600 |
| PL-6   | Francoise Tort |     0 |
+--------+----------------+-------+
```

Pour preuve, le script suivant ne renvoie aucune erreur :

```
CREATE TABLE   Test (c1 DECIMAL(6,3), c2 DATE   ,c3 VARCHAR(1), c4
CHAR);
INSERT INTO Test VALUES ('548.45', '20060116', 3, 5);
```

Explicites



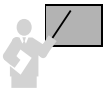
Une conversion est dite « explicite » quand on utilise une fonction à cet effet. Les fonctions de conversion les plus connues sont `CAST` et `CONVERT` (qui respectent la syntaxe de la norme SQL).

Les fonctions de conversion sont décrites dans le tableau suivant :

Tableau 4-21 Fonctions de conversion

Fonction	Conversion	Exemple
`BINARY(expr)`	L'expression en bits.	Pour le premier pilote de notre dernier exemple, le test `BINARY(brevet)` = `BINARY('pl-1')` renverra faux.
`CAST(expression AS typeMySQL)`	L'expression dans le type en paramètre (`BINARY`, `CHAR`, `DATE`, `DATETIME`, `DECIMAL`, `SIGNED`, `TIME`, `UNSIGNED`).	`CAST(2 AS CHAR)` retourne '2'.
`CONVERT(c, jeu-car)`	La chaîne *c* dans le jeu de caractères passé en paramètre.	`CONVERT('A E I Ø' USING cp850)` jeu de caractère DOS, retourne "? Ê Í ?".

Comparaisons



MySQL compare deux variables entre elles en suivant les règles suivantes :

- Si l'une des deux valeurs est `NULL`, la comparaison retourne `NULL` (sauf pour l'opérateur <=> qui renvoie vrai si les deux valeurs sont `NULL`).
- Si les deux valeurs sont des chaînes, elles sont comparées en tant que telles.
- Si les deux valeurs sont des numériques, elles sont comparées en tant que telles.
- Les valeurs hexadécimales sont traitées comme des chaînes de bits si elles ne sont pas comparées à des numériques.
- Si l'une des valeurs est `TIMESTAMP` ou `DATETIME` et si l'autre est une constante, cette dernière est convertie en `TIMESTAMP`.
- Dans les autres cas, les valeurs sont comparées comme des numériques (flottants).

Tableau 4-24 Autres fonctions *(suite)*

Fonction	Objectif	Exemple
`LEAST(`*`expression[, expression`*`]...)`	Retourne la plus petite des expressions.	`LEAST('Villepin','Sarkozy','X-Men')` retourne `'Sarkozy'`.
`NULLIF(`*`expr1,expr2`*`)`	Si *expr1* = *expr2* retourne `NULL`, sinon retourne *expr1*.	`NULLIF('Raffarine','Parafine')` retourne `'Raffarine'`.
`IFNULL(`*`expr1,expr2`*`)`	Convertit *expr1* susceptible d'être nul en une valeur réelle (*expr2*).	`IFNULL(diplome,'Aucun !')` retourne `'Aucun !'` si `diplome` est `NULL`.

Regroupements

Cette section traite à la fois des regroupements de lignes (agrégats) et des fonctions de groupe (ou multilignes). Nous étudierons les parties surlignées de l'instruction `SELECT` suivante :

```
SELECT [ { DISTINCT | DISTINCTROW } | ALL ] listeColonnes
FROM nomTable [ WHERE condition ]
 [ clauseRegroupement ]
 [ HAVING condition ]
 [ clauseOrdonnancement ]
 [ LIMIT [rangDépart,] nbLignes ] ;
```

- *listeColonnes* : peut inclure des expressions (présentes dans la clause de regroupement) ou des fonctions de groupe.
- *clauseRegroupement* : `GROUP BY (`*expression1*`[,`*expression2*`]...)` permet de regrouper des lignes selon la valeur des expressions (colonnes, fonction, constante, calcul).
- `HAVING` *condition* : pour inclure ou exclure des lignes aux groupes (la condition ne peut faire intervenir que des expressions du `GROUP BY`).
- *ClauseOrdonnancement* : déjà étudié (`ORDER BY` dans la section Projection/Ordonnancement).

Interrogeons la table suivante en composant des regroupements et en appliquant des fonctions de groupe :

Figure 4-7 *Table Pilote*

Web

`Pilote`

brevet	nom	nbHVol	prime	embauche	typeAvion	compa
PL-1	Gratien Viel	450	500	1965-02-05	A320	AF
PL-2	Didier Donsez	0		1995-05-13	A320	AF
PL-3	Richard Grin	1000		2001-09-11	A320	SING
PL-4	Placide Fresnais	2450	500	2001-09-21	A330	SING
PL-5	Daniel Vielle	400	600	1965-01-16	A340	AF
PL-6	Francoise Tort		0	2000-12-24	A340	CAST

Tableau 4-27 Exemples de fonctions de groupe avec GROUP BY *(suite)*

Fonction	Exemples

GROUP BY et HAVING	Compagnies (et nombre de leurs pilotes) ayant plus d'un pilote.
	`SELECT compa, COUNT(brevet) FROM Pilote`
	`GROUP BY(compa)`
	`HAVING COUNT(brevet)>=2;`
	`+-------+---------------+`
	`| compa | COUNT(brevet) |`
	`+-------+---------------+`
	`| AF | 3 |`
	`| SING | 2 |`
	`+-------+---------------+`

Opérateurs ensemblistes

Une des forces du modèle relationnel repose sur le fait qu'il est fondé sur une base mathématique (théorie des ensembles). Le langage SQL devrait programmer les opérations binaires (entre deux tables) suivantes :

- **intersection** qui extrait des données présentes simultanément dans les deux tables ;
- **union** par les opérateurs UNION et UNION ALL qui fusionnent des données des deux tables ;
- **différence** qui extrait des données présentes dans une table sans être présentes dans la deuxième table ;
- **produit cartésien** par le fait de disposer de deux tables dans la clause FROM, ce qui permet de composer des combinaisons à partir des données des deux tables.

Restrictions

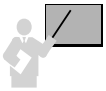
Seules des colonnes de même type (CHAR, VARCHAR, DATE, numériques, etc.) doivent être comparées avec des opérateurs ensemblistes.

L'intersection et la différence ne sont pas encore disponibles sous MySQL. La différence se programme à l'aide de DISTINCT et NOT IN, l'intersection à l'aide de DISTINCT et IN.

Attention, pour les colonnes CHAR, à veiller à ce que la taille soit identique entre les deux tables pour que la comparaison fonctionnne. Le nom des colonnes n'a pas d'importance. Il est possible de comparer plusieurs colonnes de deux tables.

Nous allons principalement étudier les deux premières écritures qui sont les plus utilisées. Nous parlerons en fin de section des deux dernières.

Jointure relationnelle



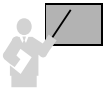
La forme la plus courante de la jointure est la jointure dite « relationnelle » (aussi appelée SQL89), caractérisée par une seule clause FROM contenant les tables et alias à mettre en jointure deux à deux. La syntaxe générale suivante décrit une jointure relationnelle :

```
SELECT [alias1.]col1, [alias2.]col2…
    FROM [nomBase.]nomTable1 [alias1], [nomBase.]nomTable2 [alias2]…
    WHERE (conditionsDeJointure);
```

Cette forme est la plus utilisée, car elle est la plus simple à écrire. Un autre avantage de ce type de jointure est qu'elle laisse le soin au SGBD d'établir la meilleure stratégie d'accès (choix du premier index à utiliser, puis du deuxième, etc.) pour optimiser les performances.

Afin d'éviter les ambiguïtés concernant le nom des colonnes, on utilise en général des alias de tables pour suffixer les tables dans la clause FROM et préfixer les colonnes dans les clauses SELECT et WHERE.

Jointures SQL2



Afin de se rendre conforme à la norme SQL2, MySQL propose aussi des directives qui permettent de programmer d'une manière plus verbale les différents types de jointures :

```
SELECT [ALL | DISTINCT | DISTINCTROW ] listeColonnes
 FROM [nomBase.]nomTable1 [{ INNER | { LEFT | RIGHT } [OUTER] }]
  JOIN [nomBase.]nomTable2{ ON condition | USING ( colonne1 [, colonne2]... )}
  | { CROSS JOIN | NATURAL [{ LEFT | RIGHT } [OUTER] ]
      JOIN [nomBase.]nomTable2 } …
[ WHERE condition ];
```

Cette écriture est moins utilisée que la syntaxe relationnelle. Bien que plus concise pour des jointures à deux tables, elle se complique pour des jointures plus complexes.

Types de jointures

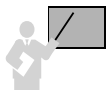
Bien que, dans le vocabulaire courant, on ne parle que de « jointures » en fonction de la nature de l'opérateur utilisé dans la requête, de la clause de jointure et des tables concernées, on distingue :

- Les jointures internes (*inner joins*) :
  - L'équijointure (*equi join*) est la plus connue, elle utilise l'opérateur d'égalité dans la clause de jointure. La jointure naturelle est conditionnée en plus par le nom des colonnes. La non équijointure utilise l'opérateur d'inégalité dans la clause de jointure.
  - L'autojointure (*self join*) est un cas particulier de l'équijointure, qui met en œuvre deux fois la même table (des alias de tables permettront de distinguer les enregistrements entre eux).
- La jointure externe (*outer join*), la plus compliquée, qui favorise une table (dite « dominante ») par rapport à l'autre (dite « subordonnée »). Les lignes de la table dominante sont retournées même si elles ne satisfont pas aux conditions de jointure.

Le tableau suivant illustre cette classification sous la forme de quelques conditions appliquées à notre exemple :

Tableau 4-34 Exemples de conditions

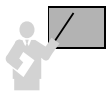
Type de jointure	Syntaxe de la condition
Équijointure	…WHERE comp = compa;
Autojointure	…WHERE alias1.chefPil = alias2.brevet;
Jointure externe	…FROM Compagnie LEFT OUTER JOIN Pilote ON comp = compa;



Pour mettre trois tables *T1*, *T2* et *T3* en jointure, il faut utiliser deux clauses de jointures (une entre *T1* et *T2* et l'autre entre *T2* et *T3*). Pour *n* tables, il faut *n*-1 clauses de jointures. Si vous oubliez une clause de jointure, un produit cartésien restreint est généré.

Étudions à présent chaque type de jointure avec les syntaxes « relationnelle » et « SQL2 ».

Équijointure



Une équijointure utilise l'opérateur d'égalité dans la clause de jointure et compare généralement des clés primaires avec des clés étrangères.

En considérant les tables suivantes, les équijointures se programment soit sur les colonnes `comp` et `compa` soit sur les colonnes `brevet` et `chefPil`. Extrayons par exemple :

Tableau 4-37 Exemples d'inéquijointures *(suite)*

Requête	Jointure relationnelle	Jointure SQL2
R6	`SELECT pil.brevet, pil.nom, pil.nbHVol, hv.titre FROM Pilote pil, HeuresVol hv WHERE pil.nbHVol BETWEEN hv.basnbHVol AND hv.hautnbHVol;`	`SELECT brevet, nom, nbHVol, titre FROM Pilote JOIN HeuresVol ON nbHVol BETWEEN basnbHVol AND hautnbHVol;`

```
+--------+------------------+---------+----------+
| brevet | nom              | nbHVol  | titre    |
+--------+------------------+---------+----------+
| PL-1   | Pierre Lamothe   |  450.00 | Débutant |
| PL-2   | Didier Linxe     |  900.00 | Initié   |
| PL-3   | Christian Soutou | 1000.00 | Initié   |
| PL-4   | Henri Alquié     | 3400.00 | Expert   |
+--------+------------------+---------+----------+
```

Jointures externes

Les jointures externes permettent d'extraire des enregistrements qui ne répondent pas aux critères de jointure. Lorsque deux tables sont en jointure externe, une table est « dominante » par rapport à l'autre (qui est dite « subordonnée »). Ce sont les enregistrements de la table dominante qui sont retournés (même s'ils ne satisfont pas aux conditions de jointure).

Comme les formules internes, les jointures externes sont généralement basées sur les clés primaires et étrangères. On distingue les jointures unilatérales qui considèrent une table dominante et une table subordonnée, et les jointures bilatérales pour lesquelles les tables jouent un rôle symétrique (pas de dominant).

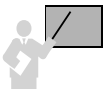
Jointures unilatérales

En considérant les tables suivantes, une jointure externe unilatérale permet d'extraire :

- la liste des compagnies et leurs pilotes, même les compagnies n'ayant pas de pilote (requête *R7*). Sans une jointure externe, la compagnie 'CAST' ne peut être extraite ;
- la liste des pilotes et leurs qualifications, même les pilotes n'ayant pas encore de qualification (requête *R8*).

La figure illustre les tables dominantes et subordonnées :

Sous-interrogation dans la clause FROM



Introduite dans SQL2, la possibilité de construire dynamiquement une table dans la clause FROM d'une requête est opérationnelle sous MySQL.

```
SELECT listeColonnes
     FROM table1 aliasTable1, (SELECT… FROM table2 WHERE…) aliasTable2
   [ WHERE (conditionsTable1etTable2) ];
```

Considérons la table suivante. Le but est d'extraire le pourcentage partiel de pilotes par compagnie. Dans notre exemple, il y a 5 pilotes dont 3 dépendent de 'AF'. Pour cette compagnie le pourcentage partiel de pilotes est de 3/5 soit 60 %.

Figure 4-18 *Table Pilote*

Pilote

brevet	nom	nbHVol	compa
PL-1	Pierre Smother	450	AF
PL-2	Didier Lixte	900	AF
PL-3	Christian Soutou	2000	SING
PL-4	Henri Alquié	3400	AF
PL-5	Michel Castafiore		

La requête suivante construit dynamiquement deux tables (alias a et b) dans la clause FROM pour répondre à cette question :

Tableau 4-43 SELECT dans un FROM

Requête et tables évaluées dans le FROM	Résultat

```
SELECT a.compa "Comp", a.nbpil/b.total*100 "%Pilote"
  FROM (SELECT compa, COUNT(*) nbpil
          FROM Pilote  GROUP BY compa) a,
       (SELECT COUNT(*) total FROM Pilote)  b;
```

a

compa	nbpil
AF	3
SING	1
	1

b

total
5

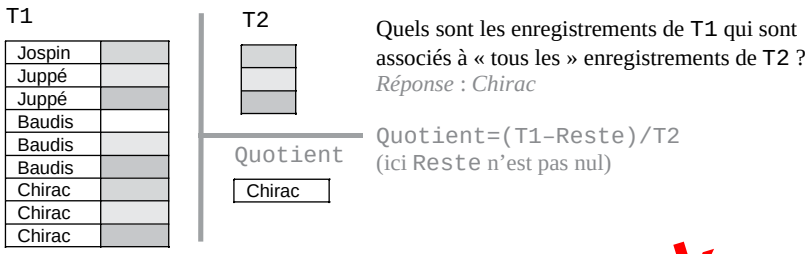
```
+------+---------+
| Comp | %Pilote |
+------+---------+
| NULL | 20.0000 |
| AF   | 60.0000 |
| SING | 20.0000 |
+------+---------+
```

Sous-interrogations synchronisées

Une sous-interrogation est synchronisée si elle manipule des colonnes d'une table du niveau supérieur. Une sous-interrogation synchronisée est exécutée une fois pour chaque enregistre-

La figure suivante illustre l'opérateur de division dans sa plus simple expression (je ne parle pas du contenu des tables, bien sûr…). Le schéma fait plus apparaître le deuxième aspect révélateur énoncé ci-avant, à savoir comparer un ensemble (la table *T1*) avec un ensemble de référence (la table *T2*).

Figure 4-21 *Division*



Définition

La division de la table *T1*[*a1*,...,*an*,*b1*,...,*bn*] par la table *T2*[*b1*,...,*bn*] (la structure de *T2* est incluse dans la structure de *T1*) donne la table *T3*[*a1*,...,*an*] qui contient les enregistrements *ti* vérifiant *ti* ∈ *T3* (de structure [*a1*,...,*an*]), *tj* ∈ *T2* (*tj* de structure [*b1*,...,*bn*]) et *ti*,*tj* ∈ *T1* (*ti*,*tj* de structure [*a1*,...,*an*,*b1*,...,*bn*]).

Classification

Considérons l'exemple suivant pour décrire la requête à construire. Il s'agit de répondre à la question : « *Quels sont les avions affrétés par* toutes *les compagnies françaises ?* ». L'ensemble de référence (*A*) est constitué des codes des compagnies françaises. L'ensemble à comparer (*B*) est formé des codes des compagnies pour chaque avion.

Deux cas sont à envisager suivant la manière de comparer les deux ensembles :

- Division inexacte (le reste n'est pas nul) : un ensemble est seulement inclus dans un autre ($A \in B$). La question à programmer serait : « *Quels sont les avions affrétés par* toutes *les compagnies françaises ?* », sans préciser si les avions ne doivent pas être aussi affrétés par des compagnies étrangères. L'avion ('A3', 'Mercure') répondrait à cette question, que la dernière ligne de la table `Affretements` soit présente ou pas.
- Division exacte (le reste est nul) : les deux ensembles sont égaux ($B=A$). La question à programmer serait : « *Quels sont les avions affrétés* exactement *(ou* uniquement*) par toutes les compagnies françaises ?* ». L'avion ('A3', 'Mercure') répondrait à cette question si la dernière ligne de la table `Affretements` était inexistante. Les lignes concernées dans les deux tables sont grisées.

Le jeu de caractères par défaut est défini dans `my.ini` à l'aide de la variable `default-character-set`. Il est donc possible de créer des bases de données associées à différents jeux de caractères au sein d'un même serveur. Le jeu de caractères d'une base définit celui des tables qui seront constituées dedans, à moins que la table ne soit combinée à un autre jeu (créé avec la directive `[DEFAULT] CHARACTER SET` *`jeu`* `[COLLATE` *`nomCollation`*`]`).

Notons enfin qu'il est même possible d'affecter un jeu de caractères à une colonne d'une table. L'exemple suivant construit la table `testChap5` dans la base `bdnouvelle2` (par défaut chinoise) en spécifiant que la colonne `col1` sera associée au jeu `cp850` : *DOS West European*, tandis que le reste de la table (pour l'instant de portée `col2`) sera appliqué au jeu `latin1` : *cp1252 West European*. Insérons une ligne.

```
CREATE TABLE bdnouvelle2.testChap5
(col CHAR(5) CHARACTER SET cp850, col2 CHAR(4))
 CHARACTER SET latin1;
INSERT INTO bdnouvelle2.testChap5 VALUES ('GTR','IUT');
```

Sélection d'une base de données (USE)

Ceux qui ont travaillé sous *Dbase* se souviennent de l'instruction `USE` qui désignait la table courante dans un programme. Pour MySQL, `USE` sélectionne une base de données qui devient active dans une session.

`USE` *`nomBase`*`;`

Si vous désirez travailler simultanément dans différentes bases de données, faites toujours préfixer le nom des tables par celui de la base par la notation pointée (*`nomBase.nomTable`*).

L'exemple suivant exécute une jointure sur deux tables situées dans deux bases distinctes :

Tableau 5-4 Sélection de bases

Instruction SQL	Résultat
`CREATE TABLE bdnouvelle.testUSE (col3 CHAR(5), col4 CHAR(4)) CHARACTER SET latin1;` `INSERT INTO bdnouvelle.testUSE VALUES ('ACTMP','IUT');`	Création d'une table dans la base. Insertion d'une ligne.
`USE bdnouvelle2;`	Sélection de la base `bdnouvelle2`.
`SELECT col, bdnouvelle.testUSE.col3 FROM testChap5, bdnouvelle.testUSE WHERE col2 = bdnouvelle.testUSE.col4;` `+------+-------+` `| col	col3 |` `+------+-------+` `| GTR	ACTMP |` `+------+-------+`	Jointure de la table `testChap5` située dans la base active (`bdnouvelle2`) avec `testUSE` située dans la base `bdnouvelle`.

Modification d'une base (ALTER DATABASE)

ALTER DATABASE vous permet de modifier le jeu de caractères par défaut d'une base de données. Pour pouvoir changer ainsi une base, vous devez avoir le privilège ALTER sur la base de données en question.

```
ALTER {DATABASE nomBase
  [ [DEFAULT] CHARACTER SET nomJeu ]
     [ [DEFAULT] COLLATE nomCollation ];
```

L'instruction suivante modifie la base « chinoise » en lui affectant le jeu de caractères de type *DOS*.

```
ALTER DATABASE bdnouvelle2 DEFAULT CHARACTER SET cp850;
```

Suppression d'une base (DROP DATABASE)

Pour pouvoir supprimer une base de données, vous devez posséder le privilège DROP sur la base (ou au niveau global pour effacer toute base). Cette commande détruit tous les objets (tables, index, etc.) et le répertoire contenant dans la base.

```
DROP {DATABASE | SCHEMA} [IF EXISTS] nomBase;
```

- IF EXISTS évite une erreur dans le cas où la base de données n'existerait pas.
- Cette instruction retourne le nombre de tables qui on été supprimées (fichiers à l'extension « .frm »).

Disons *prévert_dos* à la base « chinoise » :

```
DROP DATABASE bdnouvelle2;
```


Privilèges

Depuis le début du livre, nous avons parlé de privilèges. Il est temps à présent de préciser ce que recouvre ce terme. Un privilège (sous-entendu *utilisateur*) est un droit d'exécuter une certaine instruction SQL (on parle de privilège *système*), ou un droit relatif aux données des tables situées dans différentes bases (on parle de privilège *objet*). La connexion, par exemple, sera considérée comme un privilège système bien que n'étant pas une commande SQL.

Les privilèges système diffèrent sensiblement d'un SGBD à un autre. Chez Oracle, il y en a plus d'une centaine, MySQL est plus modeste en n'en proposant qu'une vingtaine. En revanche, on retrouvera les mêmes privilèges objet (exemple : autorisation de modifier la colonne nomComp de la table Compagnie) qui sont attribués ou retirés par les instructions GRANT et REVOKE.

Vous pouvez, par analogie, pour cet exemple et pour les suivants, découvrir les prérogatives des autres accès (ici `root` et *anonyme*).

Privilèges objet (LDD) sur toutes les bases de données

La requête suivante extrait les prérogatives à propos des instructions LDD. Pour l'instant, le caractère `'N'` étant dans toutes les colonnes, `Paul` ne peut ni créer une table ou une base (`Create_priv`), ni en supprimer (`Drop_priv`), ni créer ou supprimer un index (`Index_priv`), ni modifier la structure d'une table, la renommer ou modifier une base (`Alter_priv`), et ce quelle que soit la base de données (excepté les bases système `test` et `information_schema`).

```
SELECT Host, User, Create_priv, Drop_priv,Index_priv, Alter_priv
       FROM mysql.user;
+-----------+------+-------------+-----------+------------+------------+
| Host      | User | Create_priv | Drop_priv | Index_priv | Alter_priv |
+-----------+------+-------------+-----------+------------+------------+
| localhost | root | Y           | Y         | Y          | Y          |
| localhost |      | Y           | Y         | Y          | Y          |
| %         |      | N           | N         | N          | N          |
| localhost | Paul | N           | N         | N          | N          |
+-----------+------+-------------+-----------+------------+------------+
```


Privilèges système (LCD) sur toutes les bases de données

La requête suivante extrait les prérogatives à propos des instructions LCD. Pour l'instant, le caractère `'N'` étant dans toutes les colonnes, `Paul` ne peut ni créer un utilisateur (`Create_user_priv`), ni transmettre des droits qu'il aura lui-même reçus (`Grant_priv`), ni lister les bases de données existantes (`Show_db_priv`), et ce quelle que soit la base de données.

```
SELECT Host,User, Create_user_priv, Grant_priv, Show_db_priv FROM mysql.user;
+-----------+------+------------------+------------+--------------+
| Host      | User | Create_user_priv | Grant_priv | Show_db_priv |
+-----------+------+------------------+------------+--------------+
| localhost | root | Y                | Y          | Y            |
| localhost |      | N                | Y          | Y            |
| %         |      | N                | N          | N            |
| localhost | Paul | N                | N          | N            |
+-----------+------+------------------+------------+--------------+
```

Privilèges à propos des vues sur toutes les bases de données

La requête suivante extrait les prérogatives à propos des instructions relatives aux vues (*views* détaillées dans la section suivante). Pour l'instant, le caractère `'N'` étant dans toutes les colonnes, `Paul` ne peut ni créer une vue (`Create_view_priv`), ni lister les vues existantes (`Show_view_priv`), et ce quelle que soit la base de données.

```
SELECT CONCAT(User,'@',Host) "Compte",
       CONCAT(,'.',Routine_name,':',Routine_type) "Objet",
       Grantor, Proc_priv FROM mysql.procs_priv;
+-----------------+----------------------+----------------+---------------+
| Compte          | Objet                | Grantor        | Proc_priv     |
+-----------------+----------------------+----------------+---------------+
| Jules@localhost | bdpaul.sp1:PROCEDURE | root@localhost | Execute       |
| Jules@localhost | bdpaul.sp2:FUNCTION  | root@localhost | Execute       |
| Paul@localhost  | bdpaul.sp1:PROCEDURE | root@localhost | Alter Routine |
+-----------------+----------------------+----------------+---------------+
```

On retrouve le privilège en modification de `sp1` pour `Paul`, et les deux privilèges d'exécution de `Jules`.

Révocation de privilèges (REVOKE)

La révocation d'un ou de plusieurs privilèges est réalisée par l'instruction `REVOKE`. Pour pouvoir révoquer un privilège, vous devez détenir (avoir reçu) au préalable le même privilège avec l'option `WITH GRANT OPTION`.

Syntaxe

Dans la syntaxe suivante, les options sont les mêmes que celles sur la commande `GRANT`.

```
REVOKE privilège [ (col1 [, col2 ...])] [,privilège2 ... ]
    ON [ {TABLE | FUNCTION | PROCEDURE} ]
        {nomTable | *.* | nomBase.*}
    FROM utilisateur [,utilisateur2 ... ];
```

Exemples

Le tableau suivant décrit la révocation de certains privilèges acquis des utilisateurs `Paul` et `Jules`.

Tableau 5-8 Révocation de privilèges

Instruction faite par `root`	Explication
`REVOKE CREATE ON bdpaul.* FROM 'Paul'@'localhost';`	Privilège système *database level* : `Paul` (en accès local) ne peut plus créer de tables dans la base `bdpaul`.
`REVOKE ALTER,INSERT,UPDATE(ISBN) ON bdpaul.Livre FROM 'Paul'@'localhost';`	Privilège objet *table level* : `Paul` ne peut plus modifier la structure (ou les contraintes), insérer et modifier la colonne `ISBN` de la table `Livre` contenue dans la base `bdpaul`.
`GRANT USAGE ON bdpaul.* TO 'Jules'@'localhost' WITH MAX_QUERIES_PER_HOUR 0 MAX_UPDATES_PER_HOUR 0;`	Privilège système *database level* : `Jules` n'est plus limité en requêtes `SELECT` et `UPDATE` sur la base de données `bdpaul`. Ici c'est un `GRANT` qu'il faut faire, car il s'agit plus d'une restriction de connexion que d'une instruction SQL.

Vérifications

Une fois ces actualisations réalisées, les cinq tables de la base `mysql` contiennent un peu plus le caractère `'N'` qu'auparavant. Les colonnes `SET` des tables `mysql.tables_priv`, `mysql.columns_priv` et `mysql.procs_priv` sont également mises à jour. Ainsi, l'extraction du profil actuel de `Paul` au niveau *table* fait apparaître les deux seuls droits qu'il lui reste.

```
SELECT CONCAT(User,'@',Host) "Compte",CONCAT(Db,'.',Table_name)
"Objet",
       Grantor, Table_priv FROM mysql.tables_priv
       WHERE User='Paul' AND Host='localhost';
+----------------+-------------+----------------+---------------+
| Compte         | Objet       | Grantor        | Table_priv    |
+----------------+-------------+----------------+---------------+
| Paul@localhost | bdpaul.Livre | root@localhost | Select,Delete |
+----------------+-------------+----------------+---------------+
```

L'extraction du profil actuel de `Jules` au niveau *database* fait apparaître que les deux limitations de connexion sur les `SELECT` et `UPDATE` ont disparu.

```
SELECT Host,User, max_questions "Requetes", max_updates "Modifs" ,
       max_connections "Connexions", max_user_connections "Cx simult."
       FROM mysql.user WHERE User='Jules' AND Host='localhost';
+-----------+-------+----------+--------+------------+------------+
| Host      | User  | Requetes | Modifs | Connexions | Cx simult. |
+-----------+-------+----------+--------+------------+------------+
| localhost | Jules |        0 |      0 |          6 |          3 |
+-----------+-------+----------+--------+------------+------------+
```


Tout en une fois !

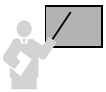
Il existe une instruction qui révoque tous les droits en une fois. Vous en avez assez d'un utilisateur qui ne cesse de vous casser les pieds, utilisez `REVOKE ALL PRIVILEGES`. Pensez quand même à sauvegarder au préalable le profil de `Jules` (`SHOW GRANT FOR`) pour pouvoir le faire travailler de nouveau quand vous serez calmé.

Selon la documentation officielle, la syntaxe suivante permet de supprimer toutes les prérogatives aux niveaux *global*, *database*, *table* et *column*. Et les privilèges *routine*, me direz-vous ? Ils ont dû l'oublier dans la documentation, mais ils sont aussi effacés, ne vous inquiétez pas, je l'ai testé.

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM utilisateur [, utilisateur2 ...] ;
```

Pour pouvoir annihiler ainsi un utilisateur, il faut détenir le privilège `CREATE USER` au niveau *global* ou le privilège `UPDATE` au niveau *database* sur la base `mysql`.

Tables protégées (key preserved tables)



Une table est dite protégée par sa clé (*key preserved*) si sa clé primaire est préservée dans la clause de jointure et se retrouve en tant que colonne de la vue multitable (elle peut jouer le rôle de clé primaire de la vue).

En considérant les données initiales pour la vue multitable `Vue_Multi_Comp_Pil`, la table préservée est la table `Pilote`, car la colonne `brevet` identifie chaque enregistrement extrait de la vue, alors que la colonne `comp` ne le fait pas.

Tableau 5-19 Vue multitable

Création de la vue

```
CREATE VIEW
Vue_Multi_Comp_Pil
 AS SELECT c.comp,
     c.nomComp,
     p.brevet, p.nom,
     p.nbHVol
 FROM  Pilote p,
       Compagnie c
 WHERE p.comp=c.comp;
```

Résultats

```
+------+--------------+--------+-------------+---------+
| comp | nomComp      | brevet | nom         | nbHVol  |
+------+--------------+--------+-------------+---------+
| AF   | Air France   | PL-1   | Louise Ente |  450.00 |
| AF   | Air France   | PL-2   | Paul Ente   |  900.00 |
| SING | Singapore AL | PL-3   | Paul Soutou | 1000.00 |
+------+--------------+--------+-------------+---------+
```


Cela ne veut pas dire pour autant que cette vue est modifiable. Étudions à présent les conditions qui régissent ces limitations.

Critères

Une vue multitable modifiable (*updatable join view* ou *modifiable join view*) est une vue qui n'est pas définie avec l'option `ALGORITHM=TEMPTABLE` et qui est telle que la requête de définition contient plusieurs tables dans la clause `FROM`.

Aucune suppression n'est possible.

Les insertions sont permises seulement en isolant toutes les colonnes d'une seule table source.

Attention aux effets de bord quand vous modifiez une colonne provenant d'une table non protégée par clé. Il est plus naturel de modifier directement la table en question.

Modifions de différentes manières la vue multitable `Vue_Multi_Comp_Pil`. La première tente une suppression, les deux suivantes modifient tantôt une colonne de la table protégée, tantôt une colonne de la table non protégée. Les deux dernières instructions insèrent dans chacune des deux tables.

Les transformations peuvent concerner toutes les parties d'une vue existante : la politique de création (ALGORITHM), la liste des colonnes, la requête, etc. Voir la section *Création d'une vue*.

Visualisation d'une vue (SHOW CREATE VIEW)

Pour pouvoir visualiser la requête de définition d'une vue, l'instruction que MySQL propose est la suivante :

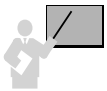
```
SHOW CREATE VIEW  [nomBase.]nomVue;
```

En arrangeant l'état de sortie, vous pouvez découvrir comment MySQL stocke la définition de la vue précédemment créée :

```
SHOW CREATE VIEW VueDesCompagniesJoursFeries;
+----------------------------+-----------------------------------------------+
| View                       | Create View                                   |
+----------------------------+-----------------------------------------------+
| VueDesCompagniesJoursFeries | CREATE ALGORITHM=UNDEFINED                    |
|                            | DEFINER=`root`@`localhost` SQL SECURITY       |
|                            | DEFINER VIEW `VueDesCompagniesJoursFeries` AS |
|                            | select sql_no_cache `Compagnie`.`comp` AS     |
|                            | `comp`,`Compagnie`.`nrue` AS                  |
|                            | `nrue`,`Compagnie`.`rue` AS                   |
|                            | `rue`,`Compagnie`.`ville` AS                  |
|                            | `ville`,`Compagnie`.`nomComp` AS `nomComp`    |
|                            | from `Compagnie` where                        |
|                            | (date_format(sysdate(),_latin1'%W') in        |
|                            | (_latin1'Sunday',_latin1'Saturday'))          |
+----------------------------+-----------------------------------------------+
```


Suppression d'une vue (DROP VIEW)

Vous devez posséder le privilège DROP sur une vue pour pouvoir la supprimer.



La suppression d'une vue n'entraîne pas la destruction des données qui résident toujours dans les tables.

La syntaxe SQL est la suivante :

```
DROP VIEW [IF EXISTS]
          [nomBase.]nomVue [,nomBase2.]nomVue2...
          [RESTRICT | CASCADE];
```

- IF EXISTS évite une erreur dans le cas où la vue n'existe pas.
- RESTRICT et CASCADE ne sont pas encore opérationnels, il concerneront probablement la répercussion de la suppression entre vues interdépendantes.

Identificateurs

Avant de parler des différents types de variables MySQL, décrivons comment il est possible de nommer les objets des sous-programmes. Un identificateur commence par une lettre (ou un chiffre). Un identificateur n'est pas limité en nombre de caractères. Les autres signes pourtant connus du langage sont interdits, comme le montre le tableau suivant :

Tableau 6-2 Identificateurs

Autorisés	Interdits
`t2`	`moi&toi` (symbole « & »)
`code_brevet`	`debit-credit` (symbole « - »)
`2nombresMysql`	`on/off` (symbole « / »)
`_t`	`code brevet` (symbole espace)

Commentaires

MySQL prend en charge deux types de commentaires : monoligne, commençant au symbole « `--` » et finissant à la fin de la ligne ; et multilignes, commençant par « `/*` » et finissant par « `*/` ». Le tableau suivant décrit quelques exemples :

Tableau 6-3 Commentaires

Sur une ligne	Sur plusieurs lignes

`-- Lecture de la table Pilote`
`SELECT nbHVol INTO v_nbHVol`
`  FROM Pilote -- Extraction heures de vol`
`  WHERE nom = 'Gratien Viel';`
`SET v_bonus := v_nbHVol*0.15; -- Calcul`
	`/* Lecture de la table Pilote */`
	`SELECT salaire INTO v_salaire`
	`  FROM Pilote`
	`  /* Extraction du salaire`
	`  pour calculer le bonus */`
	`  WHERE nom = 'Thierry Albaric';`
	`/*Calcul*/`
	`SET v_bonus := v_salaire*0.15;`

Variables

Un sous-programme est capable de manipuler des variables qui sont déclarées (et éventuellement initialisées) par la directive `DECLARE`. Ces variables permettent de transmettre des valeurs à des sous-programmes via des paramètres, ou d'afficher des états de sortie sous l'interface. Deux types de variables sont disponibles sous MySQL :

- scalaires : recevant une seule valeur d'un type SQL (ex : colonne d'une table) ;
- externes : définies dans la session et qui peuvent servir de paramètres d'entrée ou de sortie.

Web

```
delimiter $
DROP PROCEDURE sp1$
CREATE PROCEDURE sp1()
BEGIN
 SET AUTOCOMMIT = 0;
 INSERT INTO TableaVous VALUES (...);
END;
$
--appel de la transaction
CALL sp1()$
SELECT * FROM TableaVous$
```

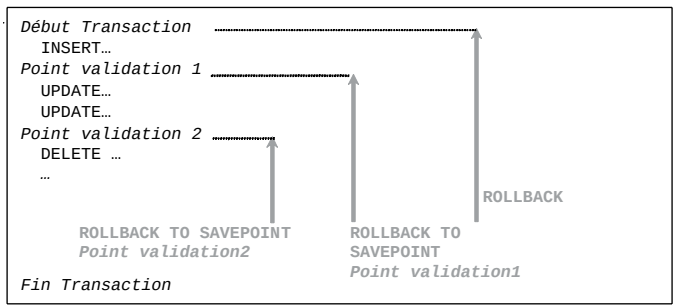
Exécutez ce bloc dans l'interface, puis déconnectez-vous soit en cassant la fenêtre (icône en haut à droite), soit proprement avec `exit`. Reconnectez-vous et constatez que l'enregistrement n'est pas présent dans votre table. Même quand la fin du programme est normale, la transaction n'est pas validée (car il manque `COMMIT`). Relancez le bloc en ajoutant cette instruction après l'insertion. Notez que l'enregistrement est présent désormais dans votre table, même après une déconnexion douce ou brutale.

Contrôle de transactions

Il est intéressant de pouvoir découper une transaction en insérant des points de validation (*savepoints*) qui rendent possible l'annulation de tout ou partie des opérations composant ladite transaction.

La figure suivante illustre une transaction découpée en trois parties. L'instruction `ROLLBACK` peut s'écrire sous différentes formes. Ainsi `ROLLBACK TO SAVEPOINT Pointvalidation1` invalidera les `UPDATE` et le `DELETE` tout en laissant la possibilité de confirmer l'instruction `INSERT` (en fonction des commandes se trouvant après ce `ROLLBACK` restreint et de la manière dont la session se terminera).

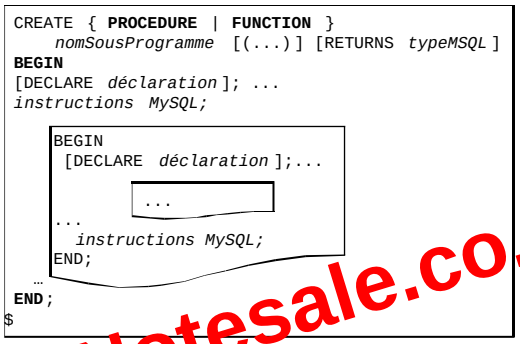
Figure 6-6 *Points de validation*



Structure d'un sous-programme

Dans une procédure, comme dans une fonction, les déclarations des variables, curseurs et exceptions suivent directement l'en-tête du bloc (après la directive `BEGIN`). La figure suivante illustre la structure d'une spécification et d'un corps d'un sous-programme MySQL. Le bloc d'instructions doit contenir au moins une instruction MySQL.

Figure 7-1 *Structure d'un sous-programme*



Exemples

Considérons la table `Pilote`. Nous allons écrire (dans la base `bdsoutou`) une fonction et une procédure :

- La fonction `ObjectifsHeure(comp,heures)` devra renvoyer le nombre de pilotes d'une compagnie donnée (premier paramètre) qui ont plus d'heures de vol que la valeur du deuxième paramètre (si aucun pilote, retourne 0). Si aucune compagnie n'est passée en paramètre (mettre `NULL`), le calcul inclut toutes les compagnies. Les éventuelles erreurs ne sont pas encore traitées (compagnie de code inexistant, par exemple).
- La procédure `PlusExperimente(comp,nom,heures)` doit retourner le nom et le nombre d'heures de vol du pilote (par l'intermédiaire des deuxième et troisième paramètres) le plus expérimenté d'une compagnie donnée (premier paramètre). Si plusieurs pilotes ont la même expérience, un message d'erreur est affiché. Si aucune compagnie n'est passée en paramètre (mettre `NULL`), la procédure retourne le nom du plus expérimenté et le code de sa compagnie (par l'intermédiaire du premier paramètre).

Remarquez que la fonction aurait pu être programmée par une procédure ayant un troisième paramètre de sortie.

Tableau 7-3 Récursivité

Web

Code MYSQL	Commentaires

`delimiter $`
`CREATE FUNCTION factorielle(n INT)`
`  RETURNS INT`
`BEGIN`
`  IF n = 1 THEN`
`  RETURN (1);`
`  ELSE`
`  RETURN (n * factorielle(n - 1));`
`  END IF;`
`END;`
`$;`
	Condition de terminaison.
	
	Appel récursif.

`SELECT factorielle(10) AS 'Factorielle de 10'$` `ERROR 1424 (HY000): Recursive stored routines are not allowed.`	Appel de la fonction !

Sous-programmes imbriqués

STOP

Il n'est pas possible de créer un sous-programme (*nested subprogram*) dans un autre sous-programme.

Cela n'est valable que pour les blocs d'instructions qui peuvent éventuellement en inclure d'autres (voir chapitre 6).

Le tableau suivant décrit la déclaration invalide du sous-programme `Mouchard` dans la procédure `imbriquee`. Ce sous-programme insérerait une ligne dans une table pour tracer l'appel de la procédure en fonction de l'utilisateur et du moment de l'exécution.

Tableau 7-4 Sous-programme imbriqué

Web

Code MySQL	Commentaires

`CREATE PROCEDURE bdsoutou.imbriquee`
`  (INOUT p1 VARCHAR(2))`
`BEGIN`
`  CREATE PROCEDURE bdsoutou.Mouchard()`
`  BEGIN`
`  INSERT INTO test.Trace VALUES`
`  (CONCAT(USER(),' a lancé imbriquee`
`  le ',SYSDATE()));`
`  END;`
`  SET p1 := 'OK';`
`  CALL bdsoutou.Mouchard();`
`END;`
`$`
	Déclaration du sous-programme.
	
	Déclaration du sous-programme imbriqué.
	
	Début du sous-programme.
	Appel du sous-programme imbriqué.

Restrictions

Les restrictions que nous mentionnons ici s'appliquent également aux déclencheurs (étudiés en fin de ce chapitre). Bien qu'il existe des restrictions qui s'appliquent seulement aux fonctions et aux déclencheurs, elles peuvent s'appliquer à une procédure si cette dernière est appelée dans le code de la fonction ou du déclencheur.

- Les instructions suivantes ne peuvent être présentes dans un sous-programme `CHECK TABLES`, `LOCK TABLES`, `UNLOCK TABLES`, `LOAD DATA`, `LOAD TABLE` et `OPTIMIZE TABLE`.
- Il n'est pas possible de programmer des instructions SQL en dynamique (`PREPARE`, `EXECUTE`, `DEALLOCATE PREPARE`) dans un déclencheur (possible dans une fonction ou une procédure).
- Il n'y a pas encore d'outil de débogage pour les sous-programmes.
- Les instructions `CALL` ne peuvent être préparées à l'avance (`CALL` *`variable`*).
- MySQL ne prend pas encore en charge la notion de paquetage (*package*) qui est un module regroupant plusieurs objets (variables, exceptions, curseurs, fonctions ou procédures) fournissant un ensemble de services (un peu comme une classe dans l'approche objet).

Curseurs

Les curseurs sont très utilisés, pour ne pas dire qu'ils sont omniprésents, dans toute application importante. Le concept, analogue au niveau de JDBC, est programmé à l'aide de la classe `ResultSet` et dans ASP de Microsoft, à l'aide de la classe `RecordSet` (appelée `DataSet` avec .Net).

MySQL n'est compatible qu'avec des curseurs en lecture seulement, non navigables, non modifiables et non dynamiques.

Généralités

Un curseur est une zone mémoire qui est générée côté serveur (mise en cache) et qui permet de traiter individuellement chaque ligne renvoyée par un `SELECT`. Un sous-programme peut travailler avec plusieurs curseurs en même temps. Un curseur, durant son existence (de l'ouverture à la fermeture), contient en permanence l'adresse de la ligne courante.

La figure suivante illustre la manipulation de base d'un curseur. Le curseur est décrit après les variables. Il est ouvert dans le code du programme ; il s'évalue alors et va se charger en extrayant les données de la base. Le programme peut parcourir tout le curseur en récupérant les lignes une par une dans une variable locale. Le curseur est ensuite fermé.

Les curseurs doivent être déclarés après les variables et avant les exceptions.

Accès concurrents (FOR UPDATE)

Si vous voulez verrouiller les lignes d'une table interrogée par un curseur dans le but de mettre à jour la table, sans qu'un autre utilisateur ne la modifie en même temps, il faut utiliser la clause FOR UPDATE. Elle s'emploie lors de la déclaration du curseur et verrouille les lignes concernées dès l'ouverture du curseur. Les verrous sont libérés à la fin de la transaction.

Il est souvent intéressant de pouvoir modifier facilement la ligne courante d'un curseur (UPDATE ou DELETE) à répercuter au niveau de la table. Il est conseillé d'utiliser un curseur FOR UPDATE pour verrouiller les lignes à actualiser.

Le tableau suivant décrit un bloc qui se sert du curseur FOR UPDATE pour :

- augmenter le nombre d'heures de 100 pour les pilotes de la compagnie de code 'AF' ;
- diminuer ce nombre de 100 pour les pilotes de la compagnie de code 'SING' ;
- supprimer les pilotes des autres compagnies.

Tableau 7-7 Curseur avec verrouillage explicite

Web

Code MySQL	Commentaires

BEGIN
  DECLARE fincurs BOOLEAN DEFAULT 0;
  DECLARE v_brevet VARCHAR(6);
  DECLARE v_nbHv DECIMAL(7,2);
  DECLARE v_comp VARCHAR(4);
  DECLARE curs **CURSOR FOR**
  SELECT brevet,nbHVol,comp
  FROM bdsoutou.Pilote **FOR UPDATE**;
	Déclaration du curseur avec verrou.

DECLARE CONTINUE HANDLER FOR NOT FOUND SET fincurs := 1; SET AUTOCOMMIT = 0; OPEN curs; FETCH curs INTO v_brevet,v_nbHv,v_comp; WHILE (NOT fincurs) DO	Chargement et parcours du curseur.

IF (v_comp='AF') THEN
  UPDATE bdsoutou.Pilote
  SET nbHVol = nbHVol + 100
  WHERE brevet = v_brevet;
	Mise à jour de la table Pilote par l'intermédiaire du curseur.

ELSEIF (v_comp='SING') THEN
  UPDATE bdsoutou.Pilote
  SET nbHVol = nbHVol - 100
  WHERE brevet = v_brevet;
  ELSE
  DELETE FROM bdsoutou.Pilote
  WHERE brevet = v_brevet;
  END IF;
  FETCH curs INTO v_brevet,v_nbHv,v_comp;
END WHILE;
CLOSE curs;
COMMIT;
	Validation de la transaction.

Restrictions

- Une validation (`COMMIT`) avant la fermeture d'un curseur `FOR UPDATE` aura des effets de bord fâcheux.
- Il n'est pas possible de déclarer un curseur `FOR UPDATE` en utilisant dans la requête les directives `DISTINCT` ou `GROUP BY`, un opérateur ensembliste, ou une fonction d'agrégat.
- Il n'existe pas encore de directive `WHERE CURRENT OF` pour modifier l'enregistrement courant d'un curseur avec verrou.
- Un curseur, comme un tableau, ne peut pas être passé en paramètre d'un sous-programme ni en entrée (`IN`), ni en sortie (`OUT`).

Exceptions

Afin d'éviter qu'un programme ne s'arrête dès la première erreur suite à une instruction SQL (`SELECT` ne retournant aucune ligne, `INSERT` ou `UPDATE` d'une valeur incorrecte, `DELETE` d'un enregistrement « père » ayant des enregistrements « fils » associés, etc.), il est indispensable de prévoir les cas potentiels d'erreurs et d'associer à chacun de ces cas la programmation d'une exception (*handler* dans le vocabulaire de MySQL).

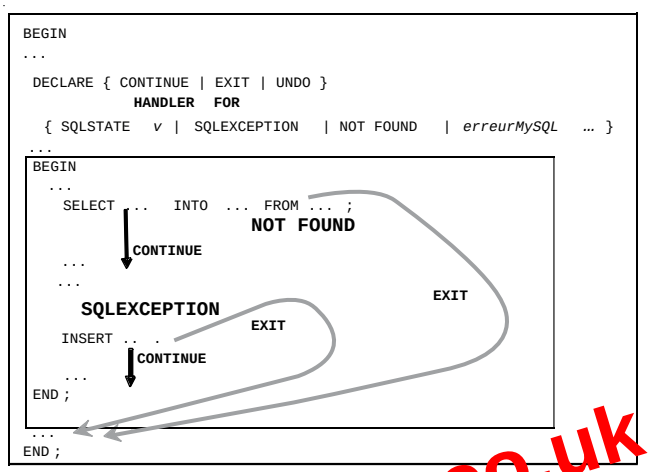
Dans le langage des informaticiens, on dit qu'on *garde la main* pendant l'exécution du programme. Le mécanisme des exceptions (*handling errors*) est largement utilisé par tous les développeurs car il est prépondérant dans la mise en œuvre des transactions. Les exceptions peuvent se paramétrer dans un sous-programme (fonction ou procédure cataloguée) ou un déclencheur.

Généralités

Une exception MySQL correspond à une condition d'erreur et peut être associée à un identificateur (exception nommée). Une exception est détectée (aussi dite « levée ») si elle est prévue dans un *handler* au cours de l'exécution d'un bloc (entre `BEGIN` et `END`). Une fois levée, elle fait continuer (ou sortir du bloc) le programme après avoir réalisé une ou plusieurs instructions que le programmeur aura explicitement spécifiées.

La figure suivante illustre les deux mécanismes qui peuvent être mis en œuvre pour gérer une exception (seuls `CONTINUE` et `EXIT` sont actuellement pris en charge par MySQL). Supposons que l'extraction ne ramène aucune ligne, on peut programmer la sortie du bloc courant ou continuer dans le bloc. Supposons que l'insertion déclenche une erreur, on peut également décider de sortir ou de poursuivre le traitement.

Figure 7-4 *Principe général des exceptions*



Si aucune erreur ne se produit, le traitement se termine ou retourne à son appelant, s'il s'agit d'un sous-programme lancé d'un programme principal. La syntaxe générale d'une exception est la suivante. Les exceptions doivent être déclarées de préférence après les variables et avant les curseurs.

```
DECLARE { CONTINUE | EXIT | UNDO }
        HANDLER FOR
  { [SQLSTATE [VALUE] 'valeur_sqlstate' | nomException | SQLWARNING
    | NOT FOUND | SQLEXCEPTION | code_erreur_mysql }
  instructions MySQL;
  [, { SQLSTATE...} ...]
```

- La directive `CONTINUE` (appelée *handler*) force à poursuivre l'exécution de programme lorsqu'il se passe un événement prévu dans la clause `FOR`.
- Le *handler* `EXIT` fait sortir l'exécution du bloc courant (entre `BEGIN` et `END`).

Le *handler* `UNDO` n'est pas encore reconnu. À son nom, on se doute de son utilité, à savoir défaire les instructions SQL qui auront été exécutées (sans avoir été validées par un `COMMIT`) avant que l'exception ne se déclenche.

- `SQLSTATE` permet de couvrir toutes les erreurs d'un état donné.
- *`nomException`* s'applique à la gestion des exceptions nommées (étudiées plus loin).
- `SQLWARNING` permet de couvrir toutes les erreurs d'état `SQLSTATE` débutant par 01.
- `NOT FOUND` permet de couvrir toutes les erreurs d'état `SQLSTATE` débutant par 02.

- `SQLEXCEPTION` gère toutes les erreurs qui ne sont ni gérées par `SQLWARNING` ni par `NOT FOUND`.
- *`instructions MySQL`* : une ou plusieurs instructions du langage de MySQL (bloc, appel possibles par `CALL` d'une fonction ou d'une procédure cataloguée).

Il est possible de grouper plusieurs déclarations d'exceptions, ainsi que de prévoir plusieurs conditions pour une même exception. En plus de pouvoir tester des erreurs pour tel ou tel état (`SQLSTATE`), nous verrons que l'on peut récupérer une erreur de code donné (paramètre `code_erreur_mysql`). Par exemple, `ERROR 1046` désigne la non sélection d'une base de données (`1046` devra être écrit en lieu et place de `code_erreur_mysql`).

Restrictions

Il n'est pas encore possible de dérouter volontairement l'exécution d'un sous-programme avec certaines conditions, par l'intermédiaire d'une instruction spécifique comme `RAISE` ou `RESIGNAL`. L'exception serait ainsi manuellement déclenchée et pourrait être définie par le programmeur (par exemple, la condition `PILOTE_TROP_JEUNE` si l'âge d'un pilote est inférieur à 20 ans).

Il n'est pas non plus permis de déclarer ses propres exceptions (par exemple, pour pouvoir dérouter le sous-programme si l'âge d'un pilote est inférieur à une valeur donnée).

Étudions à présent les différents types d'exceptions en programmant des procédures simples interrogeant la table `Pilote` illustrée à la figure 7-2.

Exceptions avec EXIT

Examinons la clause `EXIT` de l'instruction `DECLARE HANDLER` à travers un exemple.

Gestion de `NOT FOUND`

Le tableau suivant décrit une procédure qui gère une erreur : aucun pilote n'est associé à la compagnie de code passé en paramètre (`NOT FOUND`). La procédure ne se termine pas correctement si plusieurs lignes sont retournées (erreur `Result consisted of more than one row`).

Gestion d'une erreur particulière

Le tableau suivant décrit une amélioration de la précédente procédure par le fait de gérer l'erreur particulière permettant de savoir si la requête renvoie plusieurs lignes (`ERROR 1172 (42000): Result consisted of more than one row`). La procédure se termine maintenant correctement si la requête retourne une seule ligne ou plusieurs (message personnalisé en sortie de bloc).

Tableau 7-9 Exceptions 1172 et `NOT FOUND` traitées avec `EXIT`

Web

Code MySQL	Commentaires

`CREATE PROCEDURE bdsoutou.procException1`
`(IN p_comp VARCHAR(4))`
`BEGIN`
`DECLARE flagPlusDun BOOLEAN DEFAULT 0;`
`DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;`
`DECLARE var1 VARCHAR(20);`
	Déclaration de la procédure et des variables.

`BEGIN`
`DECLARE EXIT HANDLER FOR 1172`
`SET flagPlusDun :=1;`
`DECLARE EXIT HANDLER FOR NOT FOUND`
`SET flagNOTFOUND := 1;`
	Bloc qui déclare les deux exceptions.

`SELECT nom INTO var1 FROM bdsoutou.Pilote` `WHERE comp=p_comp;`	Requête pouvant déclencher l'exception prévue.

`SELECT CONCAT('Le seul pilote de la compagnie ',`
`p_comp,' est ',var1)`
`AS 'Resultat procException1';`
	Affichage du résultat.

`END;`	Fin du bloc.

`IF flagNOTFOUND THEN`
`SELECT CONCAT('Il n''y a pas de pilote pour la`
`compagnie ',p_comp)`
`AS 'Resultat procException1';`
`END IF;`
`IF flagPlusDun THEN`
`SELECT CONCAT('Il y a plusieurs pilotes pour la`
`compagnie ',p_comp)`
`AS 'Resultat procException1';`
`END IF;`
	Gestion des erreurs.

`END;`	Fin de la procédure.

La trace d'une exécution correcte de cette procédure (qui se déroute hors du bloc du fait de plusieurs lignes retournées) est la suivante :

La trace d'une exécution de cette procédure (quelle que soit la valeur du paramètre passé en entrée), suite à la suppression de la table `Pilote` dans la base `bdsoutou`, est la suivante :

```
CALL bdsoutou.procException1('AF')$
+-----------------------+
| Resultat autreErreur  |
+-----------------------+
| Erreur mais laquelle? |
+-----------------------+
```

Même erreur sur différentes instructions

Plusieurs cas de figure sont possibles suivant qu'on désire maîtriser une exception ou terminer toutes les exceptions avant la fin du sous-programme.

Gestion d'une seule exception

Le tableau suivant décrit une procédure qui gère deux fois l'erreur non trouvée (`NOT FOUND`) sur deux requêtes distinctes. La première requête extrait le nom du pilote de code passé en paramètre. La deuxième donne le nom du pilote ayant un nombre d'heures de vol égal à celui passé en paramètre. Le sous-programme se termine correctement si les deux requêtes ne retournent qu'un seul enregistrement. Les autres erreurs possibles ne sont pas prises en compte.

Le principe est d'utiliser une variable indiquant quelle est la requête qui a fait sortir du bloc par l'exception levée.

Exécutons cette procédure avec différents paramètres, on obtient :

```
CALL bdsoutou.procException2('PL-1', 1000)$
+-------------------------------------------------------+
| CONCAT('Le pilote de  code ',p_brevet,' est ',v_nom) |
+-------------------------------------------------------+
| Le pilote de  code PL-1 est Gilles Laborde           |
+-------------------------------------------------------+
+-------------------------------------------------------------------+
| CONCAT('Le pilote ayant ',p_heures,' heures de vol est ',v_nom) |
+-------------------------------------------------------------------+
| Le pilote ayant 1000 heures de vol est Florence Périssel        |
+-------------------------------------------------------------------+
CALL bdsoutou.procException2('PL-0', 2450)$
+-----------------------------------------------+
| CONCAT('Pas de pilote de brevet : ',p_brevet) |
+-----------------------------------------------+
| Pas de pilote de brevet : PL-0                |
+-----------------------------------------------+
```

Tableau 7-26 Déclencheur simulant un CHECK

Web

Déclencheur	Commentaires
`CREATE TRIGGER TrigInsGrade` `BEFORE INSERT ON Pilote FOR EACH ROW`	Déclaration de l'événement déclencheur.

`BEGIN`
`IF (NEW.grade = 'CDB' AND (NEW.nbHVol<1000)) THEN`
`SET NEW.grade := 'COPI';`
`END IF;`
`IF (NEW.grade = 'CDB' AND (NEW.nbHVol>4000)) THEN`
`SET NEW.grade := 'INST';`
`END IF;`
`IF (NEW.grade = 'COPI' AND (NEW.nbHVol>1000)) THEN`
`SET NEW.grade := 'CDB';`
`END IF;`
`IF (NEW.grade = 'INST' AND (NEW.nbHVol<3000))`
`OR (NEW.nbHVol<100) THEN`
`SET NEW.grade := NULL;`
`END IF;`
`END;`
	Corps du déclencheur.
	
	Test des conditions et mise à jour éventuelle de la nouvelle valeur à insérer au niveau de la colonne `grade`.

Si aucune condition n'est vérifiée, l'ajout se réalise sans aucun changement. Le test de ce déclencheur est le suivant. On remarque que les quatre premiers INSERT sont inchangés, alors que les deux derniers sont modifiés (mais pas annulés !).

Tableau 7-27 Test du déclencheur BEFORE INSERT

Insertions valides	Insertions non valides

`INSERT INTO Pilote VALUES`
`('PL-1','Daniel Vielle',1000,'CDB');`
`INSERT INTO Pilote VALUES`
`('PL-2','Benoit Treihlou',450,'COPI');`
`INSERT INTO Pilote VALUES`
`('PL-3','Pierre Filoux',9000,'INST');`
`INSERT INTO Pilote VALUES`
`('PL-4','Philippe Minier',1000,'COPI');`
	`INSERT INTO Pilote VALUES`
	`('PL-5','Trop jeune',100,'CDB');`
	
	`INSERT INTO Pilote VALUES`
	`('PL-6','Inexperimente',2999,'INST');`

```
SELECT * FROM Pilote;
+--------+-----------------+---------+-------+
| brevet | nom             | nbHVol  | grade |
+--------+-----------------+---------+-------+
| PL-1   | Daniel Vielle   | 1000.00 | CDB   |
| PL-2   | Benoit Treihlou |  450.00 | COPI  |
| PL-3   | Pierre Filoux   | 9000.00 | INST  |
| PL-4   | Philippe Minier | 1000.00 | COPI  |
| PL-5   | Trop jeune      |  100.00 | COPI  |
| PL-6   | Inexperimente   | 2999.00 | NULL  |
+--------+-----------------+---------+-------+
```

Tableau 7-29 Test du déclencheur BEFORE INSERT

Événement déclencheur	Résultat

`--ajout incorrect`
`INSERT INTO Qualifications`
`  VALUES`
`('PL-1','A380',:SYSDATE())$`
	`ERROR 1048 (23000): Column 'col' cannot be null`
	`-- ne fait pas l'INSERT dans Qualifications`
	`-- ni dans Trace !`

`-- ajout correct`
`INSERT INTO Qualifications`
`  VALUES`
`('PL-3','A380',SYSDATE())$`
	`Query OK, 1 row affected (0.09 sec)`
	`-- fait l'INSERT dans Qualifications et met à jour`
	`-- la table Pilote (colonne nbQualif)`

Citons le travail de Roland Bouman, un Hollandais qui a écrit une fonction (UDF *user-defined function*) en C, qui se comporte comme une fonction native (*built-in*) simulant le `RAISE_APPLICATION_ERROR`. Cette fonction permet de retourner un message personnalisé à partir du corps d'un déclencheur (http://rpbouman.blogspot.com/2005/11/using-udf-to-raise-errors-from-inside.html).

Tables mutantes

Alors qu'il n'est pas, en général, possible de manipuler la table sur laquelle se porte le déclencheur dans le corps du déclencheur lui-même. Oracle le parle de *mutating tables*, et MySQL permet d'accéder à la table en lecture. Si on tente d'y parvenir en mise à jour (`INSERT`, `UPDATE` ou `DELETE`), l'erreur est « `ERROR 1442 (HY000): Can't update table 'xxx' in stored function/trigger because it is already used by statement which invoked this stored function/trigger` ».

L'exemple suivant décrit la programmation d'un déclencheur qui compte les lignes d'une table après chaque nouvelle insertion.

Tableau 7-30 Déclencheur (table mutante)

Web

Déclencheur	Trace

`SET @vs_nombre=0$`
 	
`CREATE TRIGGER TrigMutant`
` AFTER INSERT ON Trace FOR EACH ROW`
`BEGIN`
 	
` SELECT COUNT(*) INTO @vs_nombre`
`  FROM Trace;`
`END;`
`$`
	`SELECT @vs_nombre$`
	`+------------+`
	`| @vs_nombre |`
	`+------------+`
	`| 0          |`
	`+------------+`
	`INSERT INTO Trace`
	`  VALUES ('Test TrigMutant')$`
	`SELECT @vs_nombre$`
	`+------------+`
	`| @vs_nombre |`
	`+------------+`
	`| 1          |`
	`+------------+`

```
+---------+---------+------+-----+---------+-------+
| Field   | Type    | Null | Key | Default | Extra |
+---------+---------+------+-----+---------+-------+
| immat   | char(4) | YES  |     | NULL    |       |
| turbine | int(11) | YES  |     | NULL    |       |
+---------+---------+------+-----+---------+-------+
2 rows in set (0.39 sec)
```

Exemple avec placeholder

La procédure cataloguée suivante crée dynamiquement la requête d'extraction du type et du nombre d'heures de vol (colonnes de noms passés en premier et en deuxième paramètres) de la table de nom passé en troisième paramètre, en fonction d'une condition sur une colonne (de nom passé en quatrième paramètre). Cette condition fait intervenir un paramètre (*placeholder*) valant ici 'F-GLFS'.

Tableau 7-36 Création dynamique d'une requête avec placeholder

Web

Code MySQL	Commentaires

`CREATE PROCEDURE bdsoutou.sousProg`
`(IN v_param1 CHAR(6), IN v_param2 CHAR(6),`
`IN v_param3 CHAR(5), IN v_param4 CHAR(1))`
`BEGIN`
`SET @vs_immat := 'F-GLFS';`	

`SET @vs_chaine :=`
`CONCAT('SELECT ',v_param1,',',v_param2,`
`' FROM bdsoutou.',v_param3,`
`' WHERE ',v_param4,' = ?');`
	Avec l'appel suivant, construction de la chaîne :
	'SELECT typeAv,nbHVol
	FROM bdsoutou.Avion
	WHERE immat=?'

`PREPARE etat FROM @vs_chaine;`
`EXECUTE etat USING @vs_immat;`
`DEALLOCATE PREPARE etat;`
`END;`
	Exécution de la requête paramétrée.

L'appel de cette procédure avec les paramètres suivants aura pour effet d'extraire les valeurs des deux colonnes du premier enregistrement de la table `Avion` présentée au début de cette section.

```
CALL bdsoutou.sousProg('typeAv','nbHVol','Avion','immat')$
+--------+---------+
| typeAv | nbHVol  |
+--------+---------+
| A320   | 1000.00 |
+--------+---------+
1 row in set (0.01 sec)
Query OK, 0 rows affected (0.01 sec)
```

Curseurs navigables

Un curseur `ResultSet` déclaré sans option n'est ni navigable ni modifiable. Seul un déplacement du début vers la fin (par la méthode `next`) est admis. Il est possible de rendre un curseur navigable en permettant de le parcourir en avant ou en arrière, et en autorisant l'accès direct à un enregistrement d'une manière absolue (en partant du début ou de la fin du curseur) ou relative (en partant de la position courante du curseur). On peut aussi rendre un curseur modifiable (la base pourra être changée par l'intermédiaire du curseur).

Dès l'instant où on déclare un curseur navigable, il faut aussi statuer sur le fait qu'il soit modifiable ou pas (section suivante). La nature du curseur est explicitée à l'aide d'options de la méthode `createStatement` :

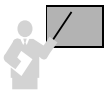
```
Statement createStatement(int typeCurseur, int modifCurseur)
```

Constantes

Les valeurs permises du premier paramètre (*typeCurseur*) et qui concernent le sens de parcours, sont présentées dans le tableau suivant :

Table 8-18 Constantes de navigation d'un curseur

Constante	Explication
`ResultSet.TYPE_FORWARD_ONLY`	Le parcours du curseur s'opère invariablement du début à la fin (non navigable).
`ResultSet.TYPE_SCROLL_INSENSITIVE`	Le curseur est navigable mais pas sensible aux modifications.
`ResultSet.TYPE_SCROLL_SENSITIVE`	Le curseur est navigable et sensible aux modifications.



Un curseur est sensible dès que des mises à jour de la table sont automatiquement répercutées au niveau du curseur durant la transaction. Lorsqu'il est déclaré insensible, les modifications de la table ne sont pas renvoyées dans le curseur.

Méthodes

Les principales méthodes que l'on peut appliquer à un curseur navigable sont les suivantes. Les deux premières sont aussi des méthodes de l'interface `Statement` qui affectent et précisent le sens de parcours pour tous les curseurs de l'état donné.

Tableau 8-19 Méthodes de navigation dans un curseur

Méthode	Fonction
`void setFetchDirection(int)`	Affecte la direction du parcours : `ResultSet.FETCH_FORWARD (1000)`, `ResultSet.FETCH_REVERSE (1001)` ou `ResultSet.FETCH_UNKNOWN (1002)`.
`int getFetchDirection()`	Extrait la direction courante (une des trois valeurs ci-dessus).
`boolean isBeforeFirst()`	Indique si le curseur est positionné avant le premier enregistrement (`false` si aucun enregistrement n'existe).
`void beforeFirst()`	Positionne le curseur avant le premier enregistrement (aucun effet si le curseur est vide).
`boolean isFirst()`	Indique si le curseur est positionné sur le premier enregistrement (`false` si aucun enregistrement n'existe).
`boolean isLast()`	Indique si le curseur est positionné sur le dernier enregistrement (`false` si aucun enregistrement n'existe).
`boolean isAfterLast()`	Indique si le curseur est positionné après le dernier enregistrement (`false` si aucun enregistrement n'existe).
`void afterLast()`	Positionne le curseur après le dernier enregistrement (aucun effet si le curseur est vide).
`boolean first()`	Positionne le curseur sur le premier enregistrement (`false` si aucun enregistrement n'existe).
`boolean previous()`	Positionne le curseur sur l'enregistrement précédent (`false` si aucun enregistrement ne précède).
`boolean last()`	Positionne le curseur sur le dernier enregistrement (`false` si aucun enregistrement n'existe).
`boolean absolute(int)`	Positionne le curseur sur le n-ième enregistrement (en partant du début si *n* est positif, ou de la fin si *n* est négatif, `false` si aucun enregistrement n'existe à cet indice).
`boolean relative(int)`	Positionne le curseur sur le n-ième enregistrement en partant de la position courante (en avant si *n* est positif, ou en arrière si *n* est négatif, `false` si aucun enregistrement n'existe à cet indice).

Connector/J de MySQL ne permet pas encore de changer le sens de parcours d'un curseur au niveau de l'état et au niveau du curseur lui-même (seule la constante `ResultSet.FETCH_FORWARD` est interprétée). Aucune erreur n'a lieu à l'exécution si vous modifiez le sens de parcours d'un curseur, la direction restera simplement inchangée (en avant toute !).

Ainsi, pour parcourir un curseur à l'envers, il faudra soit utiliser des indices négatifs (dans les méthodes `absolute` et `relative`), soit employer la méthode `previous` en partant de la fin du curseur (`afterLast`).

Tableau 8-21 Positionnements dans un curseur navigable *(suite)*

Code Java	Commentaires

`if (curseurPosJava.absolute(-2)`
`  System.out.println("absolute(-2) : "+`
`  curseurPosJava.getString(1));`
`else`
`  System.out.println("Pas d'avant dernier avion");`
	Accès à l'avant-dernier enregistrement.

`curseurPosJava.afterLast();`	Parcours du curseur en sens inverse.
`while(curseurPosJava.previous()) { … }`	
`curseurPosJava.close();`	Fermeture du curseur.
`} catch(SQLException ex) { … }`	Gestion des erreurs.

Pour définir un curseur navigable :

- Une requête ne doit pas contenir de jointure.
- Écrivez « `SELECT a.* FROM table a…` » à la place de « `SELECT * FROM table…` ».

Curseurs modifiables

Un curseur modifiable permet de mettre à jour la base de données : transformation de colonnes, suppressions et insertions d'enregistrements. Les valeurs permises du deuxième paramètre (*`modifCurseur`*) de la méthode `createStatement`, définie à la section précédente, sont présentées dans le tableau suivant :

Tableau 8-22 Constantes de modification d'un curseur

Constante	Explication
`ResultSet.CONCUR_READ_ONLY`	Le curseur ne peut être modifié.
`ResultSet.CONCUR_UPDATABLE`	Le curseur peut être modifié.

Le caractère modifiable d'un curseur est indépendant de sa navigabilité. Néanmoins, il est courant qu'un curseur modifiable soit également navigable (pour pouvoir se positionner à la demande sur un enregistrement avant d'effectuer sa mise à jour).

Pour spécifier un curseur de nature `CONCUR_UPDATABLE` :

- Une requête ne doit pas contenir de jointure ni de regroupement, elle doit seulement extraire des colonnes (les fonctions monolignes et multilignes sont interdites).
- Écrivez « `SELECT a.* FROM table a…` » à la place de « `SELECT * FROM table…` » ;

Une fois l'état créé, il faut répertorier le type des paramètres de sortie (méthode `registerOutParameter`), passer les valeurs des paramètres d'entrée, appeler le sous-programme et analyser les résultats. Le tableau suivant décrit les principales méthodes de l'interface `CallableStatement` :

Tableau 8-39 Méthodes de l'interface CallableStatement

Méthode	Description
`ResultSet executeQuery()`	Idem `PreparedStatement`.
`int executeUpdate()`	Idem `PreparedStatement`.
`boolean execute()`	Idem `PreparedStatement`.
`void registerOutParameter(int, int)`	Transfère un paramètre de sortie à un indice donné d'un type Java (classification `java.sql.Types`).
`boolean wasNull()`	Détermine si le dernier paramètre de sortie extrait est à `NULL`. Cette méthode doit être seulement invoquée après une méthode de type `getXXX`.

Exemple

Le programme JDBC suivant (`callableProcedure.java`) décrit l'appel de la procédure `leNomCompagnieEst` (avec deux paramètres). Le premier indique l'avion de la compagnie recherché, le second contient le résultat (nom de la compagnie).

```
CREATE PROCEDURE bdsoutou.leNomCompagnieEst
      (IN p_immat CHAR(6),OUT p_nomcomp VARCHAR(25))
BEGIN
 DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;
 BEGIN
   DECLARE EXIT HANDLER FOR NOT FOUND SET flagNOTFOUND :=1;
   SELECT nomComp INTO p_nomcomp FROM Compagnie
   WHERE comp = (SELECT compa FROM Avion WHERE immat = p_immat);
 END;

 IF flagNOTFOUND THEN
    SET p_nomcomp := NULL;
 END IF;
END;
```

Le tableau suivant décrit les étapes nécessaires à l'appel de cette procédure (qui ne gère pas les éventuelles erreurs) pour l'avion d'immatriculation 'F-GLFS'.

Exercice 8.3 **Appel d'un sous-programme**

Compiler, dans votre base, la procédure cataloguée `supprimeSalle(IN ns VARCHAR(7),OUT res TINYINT)` qui supprime une salle dont le numéro est passé en premier paramètre.

```
CREATE PROCEDURE supprimeSalle(IN ns VARCHAR(7),OUT res TINYINT)
BEGIN
 DECLARE ligne VARCHAR(20);
 DECLARE EXIT HANDLER FOR NOT FOUND    SET res := -1;
 DECLARE EXIT HANDLER FOR SQLEXCEPTION SET res := -2;
 SELECT nomSalle INTO ligne FROM Salle WHERE nSalle = ns;
 DELETE FROM Salle WHERE nsalle = ns;
 SET res := 0;
 COMMIT ;
END;
```

La procédure retourne en second paramètre :

- 0 si la suppression s'est déroulée correctement ;
- -1 si le code de la salle est inconnu ;
- -2 si la suppression est impossible (contraintes référentielles).

Écrire la méthode Java `int supprimeSalleSP(String)` qui appelle le sous-programme `supprimeSalle`. Essayer les différents cas d'erreurs en appelant cette méthode d'abord avec un numéro de salle référencé par un poste de travail, et ensuite avec un numéro de salle inexistant. Penser à donner à l'utilisateur le privilège en exécution sur cette procédure.

```
#Ajout pour PHP
LoadModule php5_module "c:/php/php5apache.dll"
…
# Ajout pour PHP
AddModule mod_php5.c
SEtEnv PHPRC C:/php
AddType application/x-httpd-php .php
…
DirectoryIndex index.html index.php
…
# Ajout pour MySQL : répertoire contenant les sources php (pas
d'accent dans les noms de répertoire)
DocumentRoot "D:/dev/PHP-MySQL"
```

Dans le fichier `php.ini` (se trouvant dans `C:\WINDOWS` dans mon cas), ajoutez les lignes suivantes (le « ; » désigne un commentaire) :

```
; Paths and Directories ;
extension_dir = "C:\PHP\ext"
; Dynamic Extensions ;
extension=php_mysqli.dll
```

Copiez le fichier `libmysql.dll` qui se situe dans le répertoire de PHP (`C:\PHP` dans mon cas), dans le répertoire de Windows (`C:\WINDOWS` dans mon cas).

Test d'Apache et de PHP

Écrivez le programme suivant (`index.php`) et disposez-le dans le répertoire contenant les sources PHP (`D:/dev/PHP-MySQL` dans mon cas).

```
<html> <head> <title>test Apache 1.3 PHP5 </title> </head>
<body>
Test de la configuration Apache 1.3 - PHP5 - Livre MySQL - C. Sou-
tou
<?php
 phpinfo();
?>
</body> </html>
```

Pour tester votre serveur, arrêter puis relancer Apache. Dans le navigateur, saisir l'URL de votre serveur sur le port concerné (`http://camparols:9999` dans mon cas), qui doit lancer le programme `index.php`. Vous devez voir l'affichage précédent suivi de la configuration actuelle de PHP (résultat de la fonction système PHP `phpinfo`).

Tableau 9-2 Fonctions d'analyse et d'exécution

Nom de la fonction	Paramètres
`ressource msqli_prepare(ressource connexion, string ordreSQL)`	Le premier paramètre désigne l'identifiant de la connexion. Le second contient l'ordre SQL à analyser (`SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`…).
`boolean mysqli_stmt_execute(ressource ordreSQL)`	Le paramètre désigne l'identifiant d'état à exécuter (renvoyé par *prepare*).
`boolean mysqli_stmt_fetch(ressource ordreSQL)`	Affecte aux variables de liaison PHP le résultat d'une requête préparée.

Validation

Les fonctions `mysqli_commit` et `mysqli_rollback` permettent de gérer des transactions, elles retournent `TRUE` en cas de succès, `FALSE` sinon.

Tableau 9-3 Fonctions de validation et d'annulation

Nom de la fonction	Paramètre
`boolean mysqli_commit(ressource connexion)`	Valide la transaction de la connexion en paramètre.
`boolean mysqli_rollback(ressource connexion)`	Annule la transaction de la connexion en paramètre.

Le programme suivant (`insert1.php`) insère une nouvelle compagnie (en supposant qu'aucune erreur n'est retournée de la part de la base). Nous étudierons plus loin comment passer des paramètres à une instruction (*prepared statement*) et comment récupérer, au niveau de PHP, les erreurs renvoyées par MySQL.

Tableau 9-4 Insertion d'un enregistrement

Web

Code PHP	Commentaires
`<?php`	Connexion.
`if ( ($service = mysqli_connect` `('localhost','soutou','iut','bdsoutou')) > 0)`	
`{ mysqli_autocommit($service,FALSE);`	Début de la transaction.
`$insert1 = "INSERT INTO bdsoutou.Compagnie` `VALUES('AL','Air Lib')";`	Création de l'instruction.
`$ordre = mysqli_prepare($service, $insert1);`	Prépare l'insertion.
`if ( ($res = mysqli_stmt_execute($ordre)) > 0)` `{ print "  Ajout opéré";` `mysqli_commit($service); }`	Exécute l'insertion.
`mysqli_stmt_free_result($ordre);`	Validation. Libère les ressources.
`mysqli_close($service);`	Ferme la connexion.
`}`	
`else print "  La connexion est un échec!";`	
`?>`	

```
(root@localhost) [bdsoutou] mysql> select * from avion;
+--------+-----------+----------+-------+
| immat  | typeAvion | capacite | compa |
+--------+-----------+----------+-------+
| F-GAFU | A320      |      160 | AF    |
| F-GLFS | A320      |      170 | AF    |
| F-WOWW | A380      |     NULL | AF    |
| F-WTSS | Concorde  |       90 | AF    |
+--------+-----------+----------+-------+
```

- Le *flag* vaut 16 387 pour la colonne `immat`, car cette colonne est une clé primaire (ajouter 1) et non nulle (ajouter 2), et elle fait partie d'une clé (ici primaire, ajouter 16 384). Ci-après, un extrait du fichier `mysql_conf.h`. Le *flag* vaut 16 392 pour la colonne `compa`, car elle fait partie d'une clé (ici étrangère, ajouter 16 384), et c'est une clé étrangère (ajouter 8), etc.

```
#define NOT_NULL_FLAG   1          /* Field can't be NULL */
#define PRI_KEY_FLAG    2          /* Field is part of a primary key */
#define UNIQUE_KEY_FLAG 4          /* Field is part of a unique key */
#define MULTIPLE_KEY_FLAG 8        /* Field is part of a key */
#define BLOB_FLAG       16         /* Field is a blob */
#define UNSIGNED_FLAG   32         /* Field is unsigned */
#define ZEROFILL_FLAG   64         /* Field is zerofill */
#define BINARY_FLAG     128        /* Field is binary   */
#define ENUM_FLAG       25         /* field is an enum */
#define AUTO_INCREMENT_FLAG 512    /* field is a autoincrement field */
#define TIMESTAMP_FLAG  1024       /* Field is a timestamp */
#define SET_FLAG        2048       /* field is a set */
#define NO_DEFAULT_VALUE_FLAG 4096 /* Field doesn't have default value */
#define PART_KEY_FLAG   16384      /* Intern; Part of some key */
#define NUM_FLAG        32768      /* Field is num (for clients) */
```


Fonction mysqli_stmt_result_metadata

Peu de changements par rapport au programme précédent. La fonction `mysqli_stmt_result_metadata` suppose qu'on travaille avec un état préparé. Elle sert à affecter un identifiant de résultat qui passe en paramètre de `mysqli_fetch_field`, comme vu précédemment.

```
$requete = "SELECT * FROM bdsoutou.Avion";
$ordre   = mysqli_prepare($service,$requete);
if (($res = mysqli_stmt_result_metadata($ordre) ) > 0)
{ ...
  while ($obj=mysqli_fetch_field($res) )
      { ... print "<td>$obj->name</td>"; ...
```

Tracez les différents cas d'erreurs (numéro de salle référencé par un poste de travail puis numéro de salle inexistant). Pensez à donner à l'utilisateur le privilège en exécution sur cette procédure.

Exercice 9.3 Insertion préparée

Écrire le programme PHP `exo3suite.php` qui, à partir d'une saisie des paramètres nécessaires pour enregistrer une nouvelle installation à la date du jour d'un logiciel sur un poste de travail, réalise l'insertion dans la table `Installer`. Cette saisie sera réalisée dans le programme `exo3debut.php` ci-après :

```
<html> <head> <title>Nouvelle installations de logiciel sur un
poste</title> </head>
<body>
<form action="./exo3suite.php" method="POST"><p>
 <?php
  $format='j-n-Y';
  $datej = date($format);
  print "<H4>Installation d'un logiciel à la date du $datej</H4><P>";
 ?>
 Numéro de poste : <input type="text" name="np" maxlength="7"/>
 Code du logiciel : <input type="text" name="nl" maxlength="5"/><BR>
 <input type="submit" value="Enregistrer" />
 <input type="reset"  value="Reset" />
 </form>
 </body> </html>
```


Figure 9-9 *Saisie du formulaire*

Pour tester une éventuelle erreur de compatibilité entre le type du poste et celui du logiciel (voir le déclencheur de l'exercice en fin de chapitre 7), vous afficherez le message d'erreur (avec `mysqli_stmt_error`) et le code d'erreur si l'insertion se passe mal. Tester une insertion correcte ('p10', 'log1') et une insertion incorrecte du fait des types différents ('p8', 'log7').

Figure 9-10 *Insertion correcte*

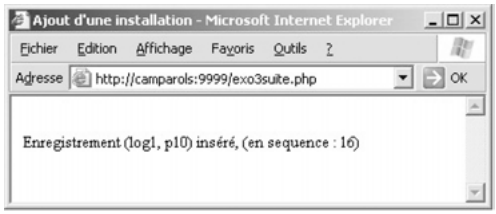


Figure 9-11 *Erreur après insertion*

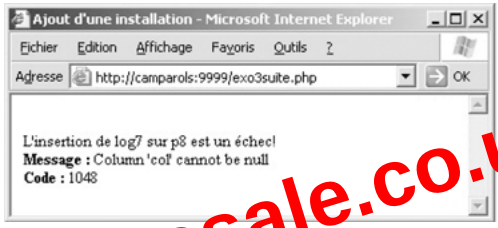
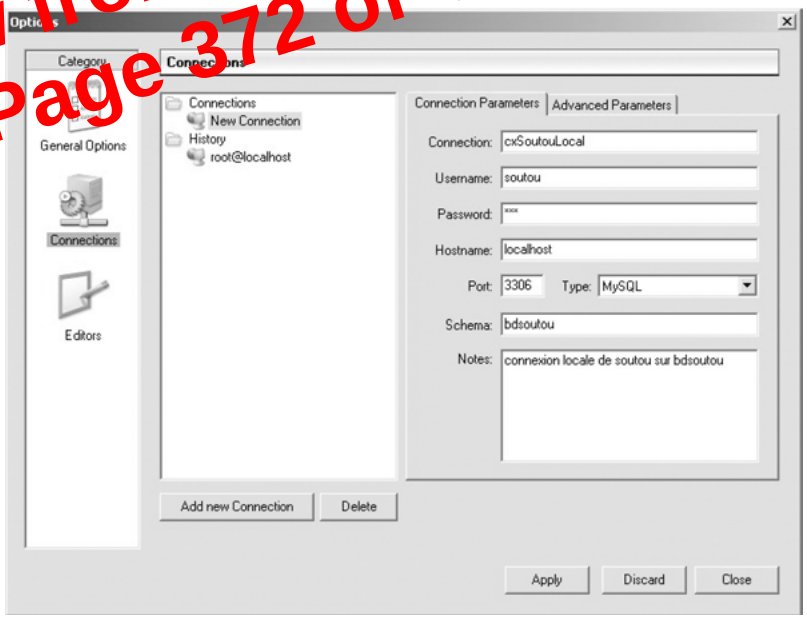


Figure 10-1 *Connexion à un serveur*

Connexion nommée

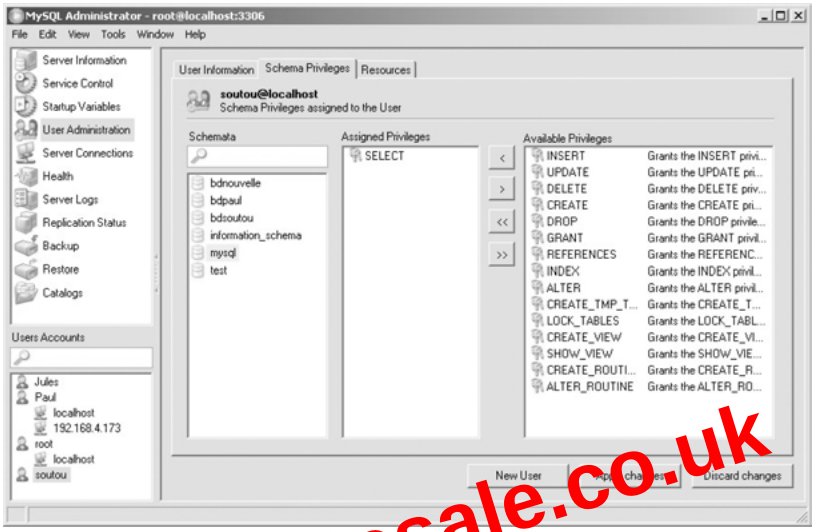
L'écran suivant décrit l'interface accessible (choix « ... » à côté de `Stored Connection`) pour prédéfinir une connexion. Ici elle se nomme `cxSoutouLocal` et correspondra à la connexion de `soutou` sur la base `bdsoutou` située sur le serveur local.

Figure 10-2 *Création d'une connexion nommée*



Une fois la connexion choisie, il faut s'identifier au niveau de la base de données cible.

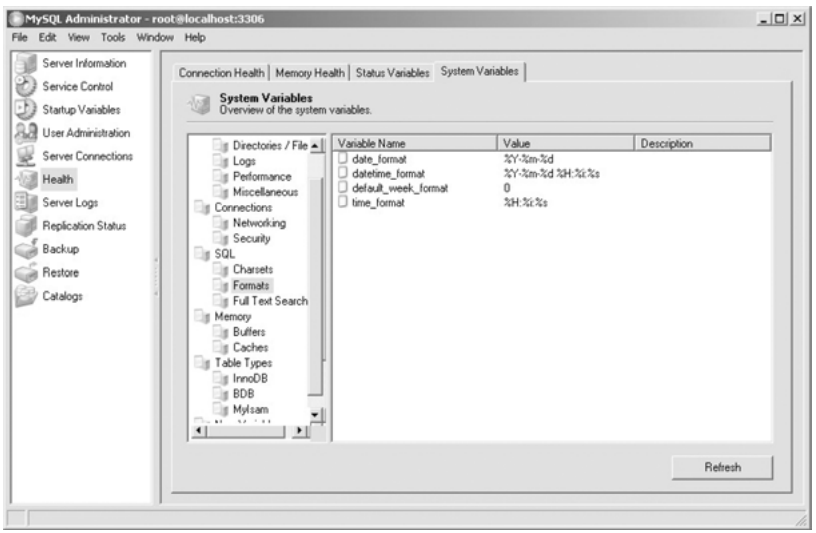
Figure 10-4 *Caractéristiques d'un utilisateur*



Caractéristiques système

Le choix Health de la fenêtre principale renseigne les occupations mémoire des process en cours de serveur MySQL, ainsi que la valeur des variables système. L'écran suivant affiche par exemple le format par défaut de représentation des colonnes de type date-heure.

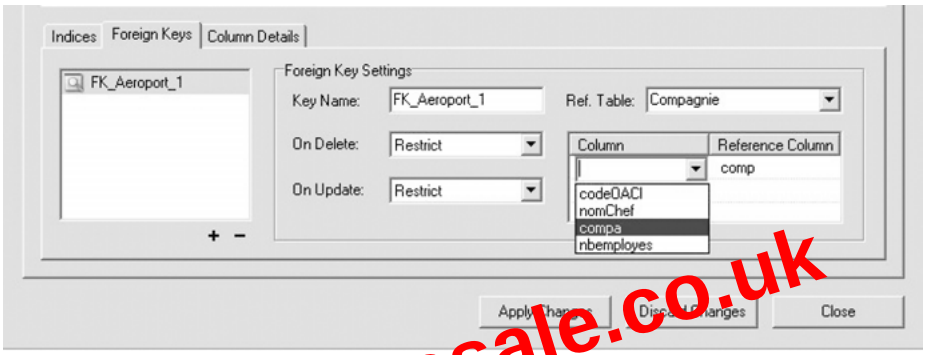
Figure 10-5 *Variables système*



Ajout contrainte

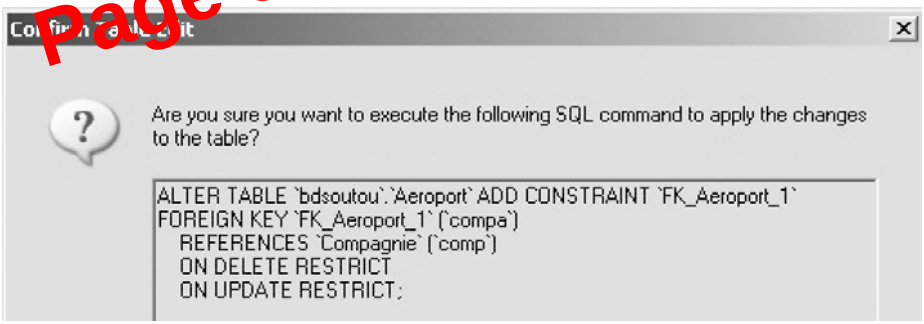
Insérons à présent la clé étrangère reliant la table `Aeroport` à la table `Compagnie` (« père »). Dans l'ordre, ajouter le nom de la contrainte « + », choisir la table cible puis la colonne de la table « fils » (ici `compa`). Enfin, vous pouvez restreindre les actions en cascade.

Figure 10-12 *Ajout d'une clé étrangère*



À l'issue de cette modification arrive l'écran qui décrit la syntaxe SQL générée. Si vous désirez conserver une trace de ce script, pensez à copier-coller le contenu de la fenêtre.

Figure 10-13 *Script SQL d'ajout d'une clé étrangère*



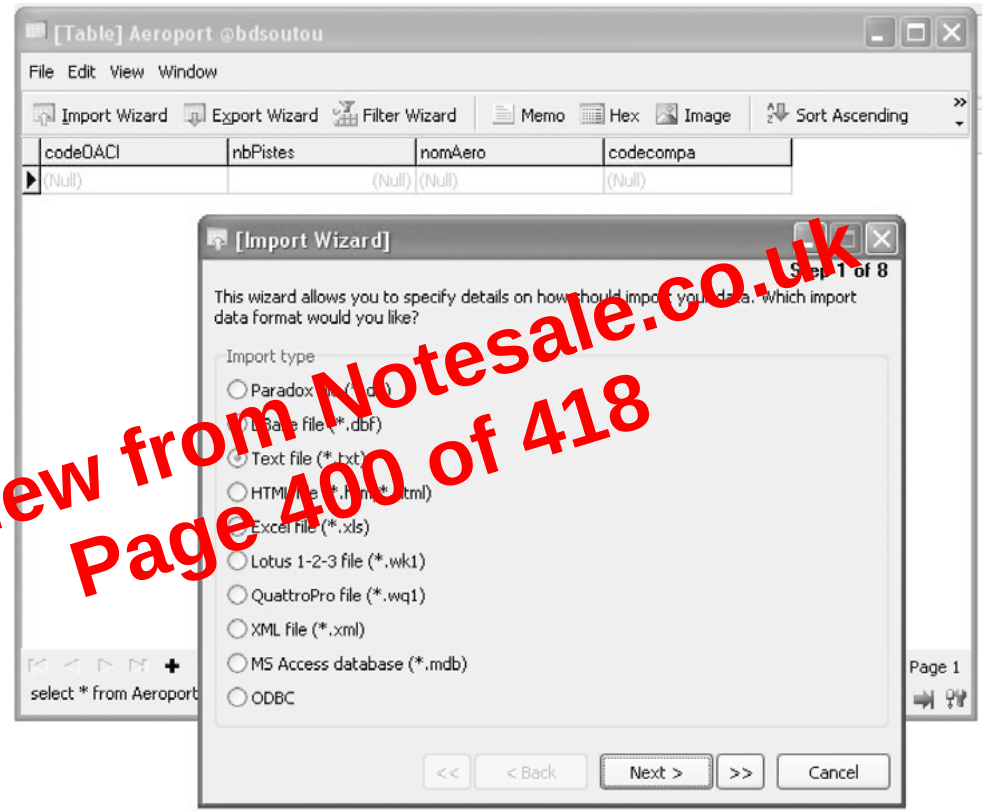
Restriction

Pour toute modification dans quelque fenêtre que ce soit, il n'est pas possible d'extraire la commande SQL générée automatiquement (sauf pour la création et la modification de tables). Cette option est très précieuse pour les administrateurs qui désirent archiver les sources de toutes leurs opérations pour les réutiliser à la demande, si nécessaire.

Importation

Au niveau d'une table d'une base de données, l'onglet `Import Wizard` lance un assistant qui comporte huit étapes et permet de charger la table en enregistrements pouvant provenir de dix formats différents.

Figure 10-38 *Importation de données*



L'assistant d'exportation comporte cinq étapes et reconnaît une vingtaine de formats de données cibles.

Gestion des utilisateurs

Le choix `Manage Users` offre une manière simple et efficace pour créer, modifier, supprimer des accès utilisateurs et les privilèges associés à tous les niveaux (*global*, *database*, *table*, *column* et *procedure*).

F

G

H

I

J

K

L

M