

Christian Soutou

Apprendre SQL avec MySQL

Avec 40 exercices corrigés

Avec 40 exercices corrigés

EYROLLES

CHEZ LE MÊME ÉDITEUR —————

Du même auteur —————

C. SOUTOU. – **De UML à SQL.**
N°11098, 2002, 500 pages.

C. SOUTOU. – **SQL pour Oracle – 2^e édition.**
N°11697, 2005, 480 pages.

Autour de SQL et MySQL —————

P. J. PRATT – **Initiation à SQL.**
N°9285, 2001, 328 pages.

A.G. TAYLOR. – **SQL Web Training.**
N°25413, 2002, 428 pages.

R. LENTZNER. – **300 astuces pour SQL et MySQL.**
N°25359, 2001, 254 pages.

M. KOFLER. – **MySQL 5. Guide de l'administrateur et du développeur.**
N°11633, 2005, 672 pages.

J.-M. AQUILINA. – **Aide-mémoire MySQL.**
N°25451, 2002, 384 pages.

J.-M. DEFRENCE. – **PHP/MySQL avec Flash MX 2004.**
N°11468, 2005, 710 pages.

P. CHALÉAT, D. CHARNAI et J.-R. ROUET. – **Les Cahiers du programmeur PHP/MySQL et JavaScript.**
N°11678, 2005, 212 pages.

J.-M. DEFRENCE. – **PHP/MySQL avec Dreamweaver 2004 (best of).**
N°11709, 2005, 550 pages.

Christian Soutou

Apprendre SQL avec MySQL

Avec 40 exercices corrigés

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com



Le code de la propriété intellectuelle du 1^{er} juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée notamment dans les établissements d'enseignement, provoquant une baisse brutale des achats de livres, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée.

En application de la loi du 11 mars 1957, il est interdit de reproduire intégralement ou partiellement le présent ouvrage, sur quelque support que ce soit, sans autorisation de l'éditeur ou du Centre Français d'Exploitation du Droit de Copie, 20, rue des Grands-Augustins, 75006 Paris.

© Groupe Eyrolles, 2006, ISBN : 2-212-11915-1

Pour Aurélia, mon chat aux yeux si bleus, si verts, si gris.

Pour René et Lydie, Jean et Denise qui sont devenus mes parents.

Remerciements
Avant-propos
Guide de lecture
Première partie : SQL de base
Deuxième partie : programmation procédurale
Troisième partie : langages et outils
Annexe
Typographie
Contact avec l'auteur – Corrigés des exercices
Introduction
SQL, une norme, un succès
Modèle de données
Tables et données
Les clés
MySQL
Un peu d'histoire
Offre du moment
Licences
Et la concurrence ?
Notion de schéma (database)
Notion d'hôte
Aspects étudiés
Mise en œuvre de MySQL (sous Windows)
Installation
Désinstallation
Reconfiguration
Premiers pas
L'interface de commande
Création d'un utilisateur
Connexion au serveur
Vérification de la version
Options de base
Batch
Votre prompt, et vite !
Commandes de base

1 Définition des données . . .**Tables relationnelles . . .**

- Création d'une table (CREATE TABLE)
- Délimiteurs
- Sensibilité à la casse
- Commentaires
- Premier exemple
- Contraintes
- Conventions recommandées
- Types des colonnes
- Structure d'une table (DESCRIBE)
- Restrictions

Index

- Arbres balancés
- Création d'un index (CREATE INDEX)
- Bilan

Destruction d'un schéma

- Suppression d'une table (DROP TABLE)
- Ordre des suppressions

Exercices**2 Manipulation des données . . .****Insertions d'enregistrements (INSERT) . . .**

- Syntaxe
- Renseigner toutes les colonnes
- Renseigner certaines colonnes
- Plusieurs enregistrements
- Ne pas respecter des contraintes
- Données binaires
- Énumérations
- Dates et heures

Séquences

- Utilisation en tant que clé primaire
- Modification d'une séquence
- Utilisation en tant que clé étrangère

Modifications de colonnes

- Syntaxe (UPDATE)
- Modification d'une colonne
- Modification de plusieurs colonnes
- Modification de plusieurs enregistrements
- Ne pas respecter les contraintes

	Restrictions
	Dates et intervalles
Remplacement d'un enregistrement
Suppressions d'enregistrements
	Instruction DELETE
	Instruction TRUNCATE
Intégrité référentielle
	Syntaxe
	Cohérences assurées
	Contraintes côté « père »
	Contraintes côté « fils »
	Clés composites et nulles
	Cohérence du fils vers le père
	Cohérence du père vers le fils
	En résumé
Insertions à partir d'un fichier
Exercices

3 Évolution d'un schéma

Renommer une table (RENAME)
Modifications structurelles (ALTER TABLE)
	Ajout de colonnes
	Renommer des colonnes
	Modifier le type des colonnes
	Valeurs par défaut
	Supprimer des colonnes
Modifications comportementales
	Ajout de contraintes
	Suppression de contraintes
	Désactivation des contraintes
	Réactivation des contraintes
	Contraintes différées
Exercices

4 Interrogation des données

Généralités
	Syntaxe (SELECT)
	Pseudotable
Projection (éléments du SELECT)
	Extraction de toutes les colonnes
	Extraction de certaines colonnes
	Alias
	Duplicatas

Expressions et valeurs nulles	8
Ordonnancement	8
Concaténation	8
Insertion multiligne	9
Limitation du nombre de lignes	9
Restriction (WHERE)	9
Opérateurs de comparaison	9
Opérateurs logiques	9
Opérateurs intégrés	9
Alias	9
Fonctions	9
Caractères	9
Numériques	9
Fonction pour les bits	10
Dates	10
Conversions	10
Comparaisons	10
Énumérations	10
Autres fonctions	10
Regroupements	10
Fonctions de groupe	11
Étude du GROUP BY et HAVING	11
Opérateurs ensemblistes	11
Restrictions	11
Exemple	11
Intersection	11
Opérateurs UNION et UNION ALL	11
Différence	11
Ordonner les résultats	11
Produit cartésien	11
Bilan	12
Jointures	12
Classification	12
Jointure relationnelle	12
Jointures SQL2	12
Types de jointures	12
Équijointure	12
Autojointure	12
Inéquijointure	12
Jointures externes	12
Jointures procédurales	13
Jointures mixtes	13
Sous-interrogations synchronisées	13
Autres directives SQL2	14

Division
Définition
Classification
Division inexacte
Division exacte
Résultats en HTML ou XML
Écriture dans un fichier
Exercices
5 Contrôle des données
Gestion des utilisateurs
Classification
Création d'un utilisateur (CREATE USER)
Modification d'un utilisateur
Renommer un utilisateur (RENAME USER)
Suppression d'un utilisateur (DROP USER)
Gestion des bases de données
Création d'une base (CREATE DATABASE)
Sélection d'une base de données (USE)
Modification d'une base (ALTER DATABASE)
Suppression d'une base (DROP DATABASE)
Privilèges
Niveaux de privilèges
Tables de la base mysql
Table mysql.user
Attribution de privilèges (GRANT)
Table mysql.db
Table mysql.host
Table mysql.tables_priv
Table mysql.columns_priv
Table mysql.procs_priv
Révocation de privilèges (REVOKE)
Attributions et révocations « sauvages »
Accès distants
Connexion par l'interface de commande
Table mysql.host
Vues
Création d'une vue (CREATE VIEW)
Classification
Vues monatables
Vues complexes
Autres utilisations de vues
Transmission de droits
Modification d'une vue (ALTER VIEW)

Visualisation d'une vue (SHOW CREATE VIEW)	18
Suppression d'une vue (DROP VIEW)	18
Dictionnaire des données	19
Constitution	19
Modèle graphique du dictionnaire des données	19
Démarche à suivre	19
Classification des vues	19
Bases de données du serveur	19
Composition d'une base	19
Détail de stockage d'une base	19
Structure d'une table	19
Recherche des contraintes d'une table	19
Composition des contraintes d'une table	19
Recherche du code source d'un sous-programme	20
Privilèges des utilisateurs d'une base de données	20
Commande SHOW	20
Exercices	20
Partie II Programmation procédurale	20
6 Bases du langage de programmation	20
Généralités	20
Environnement client-serveur	20
Avantages	21
Structure d'un bloc	21
Portée des objets	21
Casse et lisibilité	21
Identificateurs	21
Commentaires	21
Variables	21
Variables scalaires	21
Affectations	21
Restrictions	21
Résolution de noms	21
Opérateurs	21
Variables de session	21
Conventions recommandées	21
Test des exemples	21
Structures de contrôle	21
Structures conditionnelles	21
Structures répétitives	21
Interactions avec la base	22
Extraire des données	22
Manipuler des données	22

Transactions
Caractéristiques
Début et fin d'une transaction
Mode de validation
Votre première transaction
Contrôle des transactions
Transactions imbriquées
Exercices
7 Programmation avancée.
Sous-programmes
Généralités
Procédures cataloguées
Fonctions cataloguées
Structure d'un sous-programme
Exemples
Fonction n'interagissant pas avec la base
Compilation
Appel d'un sous-programme
Récursivité
Sous-programmes imbriqués
Modification d'un sous-programme
Destruction d'un sous-programme
Restrictions
Curseurs
Généralités
Instructions
Parcours d'un curseur
Accès concurrents (FOR UPDATE)
Restrictions
Exceptions
Généralités
Restrictions
Exceptions avec EXIT
Exceptions avec CONTINUE
Gestion des autres erreurs (SOLEXCEPTION)
Même erreur sur différentes instructions
Exceptions nommées
Déclencheurs
Généralités
À quoi sert un déclencheur ?
Mécanisme général
Syntaxe
Déclencheurs LMD (de lignes)

Appel de sous-programmes	27
Dictionnaire des données	27
Programmation d'une contrainte de vérification	27
Invalidation dans le déclencheur	27
Tables mutantes	27
Restrictions	27
Suppression d'un déclencheur	27
SQL dynamique	27
Syntaxe	27
Exemples	28
Restrictions	28
Exercices	28
Partie III Langages et outils	28
8 Utilisation avec Java	28
JDBC avec Connector/J	28
Classification des pilotes (drivers)	29
Le paquetage <code>java.sql</code>	29
Structure d'un programme	29
Test de votre configuration	29
Connexion à une base	29
Base Access	29
Base MySQL	29
Interface Connection	29
États d'une connexion	29
Interfaces disponibles	29
Méthodes génériques pour les paramètres	29
États simples (interface Statement)	29
Méthodes à utiliser	29
Correspondances de types	29
Manipulations avec la base	30
Suppression de données	30
Ajout d'enregistrements	30
Modification d'enregistrements	30
Extraction de données	30
Curseurs statiques	30
Curseurs navigables	30
Curseurs modifiables	30
Suppressions	30
Modifications	31
Insertions	31

Gestion des séquences
Méthode getGeneratedKeys
Curseur modifiable
Interface ResultSetMetaData
Interface DatabaseMetaData
Instructions paramétrées (PreparedStatement)
Extraction de données (executeQuery)
Mises à jour (executeUpdate)
Instruction LDD (execute)
Procédures cataloguées
Exemple
Transactions
Points de validation
Traitement des exceptions
Affichage des erreurs
Traitement des erreurs
Exercices

9 Utilisation avec PHP

Configuration adoptée
Logiciels
Fichiers de configuration
Test d'Apache et de PHP
Test d'Apache, de PHP et de MySQL
API de PHP pour MySQL
Connexion
Interactions avec la base
Extractions
Instructions paramétrées
Gestion des séquences
Traitement des erreurs
Procédures cataloguées
Métadonnées
Exercices

10 Outils graphiques

MySQL Administrator
Connexion
Connexion nommée
Liste des accès utilisateur
Gestion des privilèges
Caractéristiques système
Options scripts SQL
Composition d'une base

Composition d'une table	35
Composition des index	35
Modification d'un schéma	35
Restriction	35
MySQL Query Browser	35
Fenêtre principale	35
Extraction	36
Modification	36
phpMyAdmin	36
Composition de la base	36
Structure d'une table	36
Administrer une table	36
Extractions	36
Rechercher	36
Exporter	36
Utilisateurs	36
TOAD for MySQL	36
Administration	36
Instructions en ligne	37
Développement	37
Création d'objets	37
Recherche d'objets	37
Exportations	37
Navicat	37
Création d'une table	37
Définition d'une contrainte	37
Création d'une requête	37
Importation	37
Gestion des utilisateurs	37
MySQL Manager	37
Création d'une table	38
Création d'une requête	38
Procédures cataloguées	38
Gestion des utilisateurs	38
Gestion des priviléges	38
Bilan	38
Annexe : bibliographie et webographie	38
Index	38

Remerciements

Je n'ai que deux personnes à remercier. Il s'agit de deux jeunes informaticiens rencontrés hasard d'un forum. Ils sont talentueux et désintéressés, ce qui devient tellement rare dans le monde d'individualisme exacerbé. Le premier recherchait un emploi à la période de la révolution, souhaitons qu'il trouve clavier à ses mains. Le second dirige la rubrique MySQL du site Developpez.com.

Merci, Pierre Caboche, pour la lecture de la première moitié de l'ouvrage, pour tes remarques que j'ai (presque) toutes prises en compte, pour les compléments en ligne à propos des auteurs ensemblistes dans les requêtes.

Merci, Guillaume Lebur, pour tous tes commentaires de qualité et pour tes corrections propos de la programmation sous MySQL. Merci pour ton activité et tes tutoriels mis en ligne sur le site de Developpez.com.

Avant-propos

Nombre d'ouvrages traitent de SQL et de MySQL ; certains résultent d'une traduction hâtive et sans vocation pédagogique, d'autres ressemblent à des Bottin téléphoniques provenant de la traduction de la documentation officielle en moins bien. Les survivants sont peut-être plus vraiment à jour.

Ce livre a été rédigé avec une volonté de concision et de progression dans sa démarche ; illustré par ailleurs de nombreux exemples et figures. Bien que la source principale d'informations fût la documentation officielle de MySQL (<http://dev.mysql.com/doc>), l'ouvrage ne constitue pas un condensé de commandes SQL. Chaque notion importante est introduite avec un exemple simple et que j'espère démonstratif. En fin de chaque chapitre des exercices permettront de tester vos connaissances.

La documentation en ligne de MySQL (*MySQL 5 Reference Manual*) représente une dizaine de mégaoctets au format HTML. Tous les concepts s'y trouvant ne pourraient pas être décemment expliqués, sauf peut-être si cet ouvrage ressemblait à un annuaire. J'ai tenté d'en extraire seulement les aspects fondamentaux sous la forme d'une synthèse.

Vous n'y trouverez donc pas des considérations à propos d'aspects avancés du langage ou de l'administration d'un serveur comme l'optimisation de requêtes, la restauration d'une base, la réPLICATION, la veillance du serveur à partir de laquelle telle ou telle fonction est apparue, etc.

Ce livre résulte de mon expérience de l'enseignement dans le domaine des bases de données (en tant qu'assistant de recherche, en premier, deuxième et troisième cycles universitaires dans des cursus d'informatique et d'ingénierie, et en formation professionnelle (IUT, licences et masters professionnels)).

Cet ouvrage s'adresse principalement aux novices désireux de découvrir SQL en programmant sous MySQL.

- Les étudiants et enseignants trouveront des exemples pédagogiques pour chaque concept abordé, ainsi que des exercices thématiques.
- Les développeurs PHP ou Java découvriront des moyens de stocker leurs données.

Ce livre s'organise autour de trois parties distinctes mais complémentaires. La première intéressera le lecteur débutant en la matière, car elle concerne les instructions SQL et les notions de base de MySQL. La deuxième partie décrit la programmation avec le langage procédural de MySQL. La troisième partie attirera l'attention des programmeurs qui envisagent d'utiliser MySQL à l'aide d'outils natifs, ou tout en programmant avec des langages évolués ou via des interfaces Web (PHP ou Java).

Première partie : SQL de base

Cette partie présente les différents aspects du langage SQL de MySQL, en étudiant en détail les instructions de base. À partir d'exemples, j'explique notamment comment déclarer, manipuler, faire évoluer et interroger des tables avec leurs différentes caractéristiques et leurs éléments associés (contraintes, index, vues, séquences). Nous étudions aussi SQL dans un contexte multi-utilisateur (droits d'accès), et au niveau du dictionnaire de données.

Deuxième partie : programmation procédurale

Cette partie décrit les caractéristiques du langage procédural de MySQL. Le chapitre 6 traite des éléments de base (structure d'un programme, variables, structures de contrôle, interaction avec la base et transactions). Le chapitre 7 traite des sous-programmes, des curseurs, de la gestion des exceptions, des déclencheurs et de l'utilisation du SQL dynamique.

Troisième partie : langages et outils

Cette partie intéressera les programmeurs qui envisagent d'exploiter une base MySQL en utilisant un langage de programmation. Le chapitre 8 détaille l'API JDBC 3.0 qui permet de manipuler une base MySQL 5 par l'intermédiaire d'un programme Java. Le chapitre 9 décrit les principales fonctions de l'API *mysqli* qui permet d'interfacer un programme PHP 5 avec une base MySQL 5.

Le chapitre 10 synthétise les fonctionnalités de plusieurs outils graphiques tels que *MySQL Administrator*, *MySQL Query Browser* et *phpMyAdmin*. D'autres consoles graphiques d'administration sont étudiées, à savoir *Toad for MySQL*, *Navicat* et *EMS SQL Manager*.

Annexe

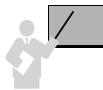
L'annexe contient une bibliographie et des adresses Web.

La police **courrier** est utilisée pour souligner les instructions SQL, noms de types, ta contraintes, etc. (ex : `SELECT nom FROM Pilote`).

Les majuscules sont employées pour les directives SQL, et les minuscules pour les éléments. Les noms des tables, index, vues, fonctions, procédures... sont précédés d'une majuscule (exemple : la table `CompagnieAerienne` contient la colonne `nomComp`).

Les termes de MySQL (bien souvent traduits littéralement de l'anglais) sont notés en italique, exemple : *trigger, table, column, etc.*

Dans une instruction SQL, les symboles { } désignent une liste d'éléments, et le symbole | un choix (exemple `CREATE { TABLE | VIEW }`). Les symboles « [» et «] » précisent le caractère optionnel d'une directive au sein d'une commande (exemple : `CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name [USING index_type] ON table_name (index_col_name,...)`).



Ce sigle introduit une définition, un concept ou une remarque importante. Il apparaît soit dans une partie théorique soit dans une partie technique pour souligner des instructions importantes ou la marche à suivre avec SQL.



Ce sigle annonce soit une impossibilité de mise en œuvre d'un concept soit une mise en garde. Il est principalement utilisé dans la partie consacrée à SQL.

J'en profite pour faire passer le message suivant : si vous travaillez en version 4 de MySQL, certaines instructions décrites dans ce livre ne fonctionneront pas. Cet ouvrage n'est pas un guide de référence ! Vous trouverez sur le Web des ressources pour connaître la compatibilité de telle ou telle fonction SQL.



Ce sigle indique que le code source est téléchargeable à partir du site des éditions Eyrolles (www.eyrolles.com). Cela est valable pour les exercices corrigés mais aussi pour tous les exemples du livre.



Ce sigle signale une astuce ou un conseil personnel.

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, sources des exemples et compléments) est en ligne et accessible via www.editions-eyrolles.com.

Dans cette introduction, nous présentons, tout d'abord, le cadre général dans lequel cet ouvrage se positionne (SQL, le modèle de données et l'offre MySQL). Nous décrivons, enfin, la procédure d'installation de MySQL sous Windows et l'utilisation de l'interface de ligne de commande pour que vous puissiez programmer en SQL dès le chapitre 1.

SQL, une norme, un succès

C'est IBM, à tout seigneur tout honneur, qui, avec System-R, a implanté le modèle relationnel au travers du langage SEQUEL (*Structured English as QUERy Language*), rebaptisé plus tard SQL (*Structured Query Language*).

La première norme (SQL1) date de 1987. Elle était le résultat de compromis entre constructeurs, mais elle était fortement influencée par le dialecte d'IBM. SQL2 a été normalisé en 1992. Elle définit quatre niveaux de conformité : le niveau d'entrée (*entry level*), les niveaux intermédiaires (*transitional et intermediate levels*) et le niveau supérieur (*full level*). Les langages SQL des principaux éditeurs sont tous conformes au premier niveau et ont beaucoup de caractéristiques relevant des niveaux supérieurs. Depuis 1999, la norme est appelée SQL3. Elle comporte de nombreuses parties (concepts objets, entrepôts de données, séries temporelles, accès à des sources non SQL, réPLICATION des données, etc.).

Le succès que connaissent les éditeurs de SGBD relationnels a plusieurs origines et raisons, notamment sur SQL :

- Le langage est une norme depuis 1986, qui s'enrichit au fil du temps.
- SQL peut s'interfacer avec des langages de troisième génération comme C ou Cobol, mais aussi avec des langages plus évolués comme C++, Java ou C#. Certains considèrent que le langage SQL n'est pas assez complet (le dialogue entre la base et l'interface n'est pas direct), et la littérature parle de « défaut d'impédance » (*impedance mismatch*).
- Les SGBD rendent indépendants programmes et données (la modification d'une structure de données n'entraîne pas forcément une importante refonte des programmes d'application).
- Ces systèmes sont bien adaptés aux grandes applications informatiques de gestion (architectures type client-serveur et Internet) et ont acquis une maturité sur le plan de la fiabilité et des performances.
- Ils intègrent des outils de développement comme les précompilateurs, les générateurs de code, d'états, de formulaires.

- 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 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

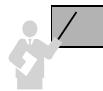
Compagnie

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 un instant. Une table peut contenir plusieurs clés candidates ou aucune.

Dans notre exemple, les colonnes **nomComp** et **nom** peuvent être des clés candidates mais il 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 **compa** (notée en italique dans la figure) est une clé étrangère car elle permet de référencer un enregistrement unique de la table **Compagnie** via sa 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. Elle emploie environ 1 000 personnes et compte plus de cinquante bureaux dans le monde entier. MySQL a commencé à compter une cinquantaine de kilomètres au nord de Stockholm. Selon leurs dires, leur serveur MySQL est capable de traiter jusqu'à 100 000 requêtes simultanées. MySQL est un système de gestion de bases de données, qui est écrit en C et C++, serait installé de façon opérationnelle sur plus de 10 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.

Par analogie avec les systèmes d'exploitation, depuis 1999, MySQL connaît le succès de Linux. Téléchargée près d'un million de fois en trois semaines (en octobre 2005), la version *production* de MySQL doit sa popularité du fait de son caractère *open source*, de ses fonctionnalités de plus en plus riches, de ses performances, de son ouverture à tous les principaux langages du marché, de son fonctionnement sur les systèmes les plus courants (les distributions classiques de Linux, Windows, Mac OS, BSD, Novell et les dérivés d'Unix) et de sa facilité d'utilisation pour des applications Web de taille moyenne.

Un peu d'histoire

Le tableau suivant résume l'historique des fonctionnalités importantes du serveur de données MySQL. Une version majeure comme 3.23 se décline au fil des mois en différentes sous-versions, en fonction des diverses phases de développement :

- *alpha* correspond à la phase active, de nouvelles fonctionnalités sont ajoutées.
- *bêta* correspond à l'implémentation des nouvelles fonctionnalités de la phase *alpha*, sans apport important de code.
- *gamma* correspond au terme *release candidate* (après résolution des bugs de la version *bêta*).
- *production* est l'étape suivante aussi appelée GA (*Generally Available*). Les bugs sont résolus s'ils ne modifient pas le comportement général du serveur.
- *old* correspond à la précédente version de production (par rapport à la courante).

Ainsi, en septembre 2005, la version 4.1.14 était mise en production et en octobre 2005, la version 5.0 l'était aussi.

Tableau 0-2 Dates importantes pour MySQL

Année – versions	Caractéristiques principales
1999 – 3.23.x	RéPLICATION – Recherches textuelles – Transactions et Intégrité référentielle tables InnoDB (2002 – 3.23.44)
2001 – 4.0.x	Cache de requêtes – Sécurisation SSL – Sauvegarde à chaud
2003 – 4.1.x	Support de Unicode – Données géographiques – SQL dynamique
2004 – 5.0.x	Vues – Curseurs – Procédures cataloguées – Déclencheurs – Dictionnaire des données – Transactions distribuées (XA)
À venir – 5.1.x	Jointure externe – Contraintes CHECK – Sauvegarde à chaud et Intégrité référentielle tables MyISAM

Ce qui est planifié à moyen terme concerne un enrichissement général des commandes SQL, la prise en compte des types manquants de SQL2 et de ODBC3, la programmation des requêtes hiérarchiques (interrogation de structures en arbres) en s'inspirant de ce qu'a fait Oracle dans la matière (directive CONNECT BY dans un SELECT).

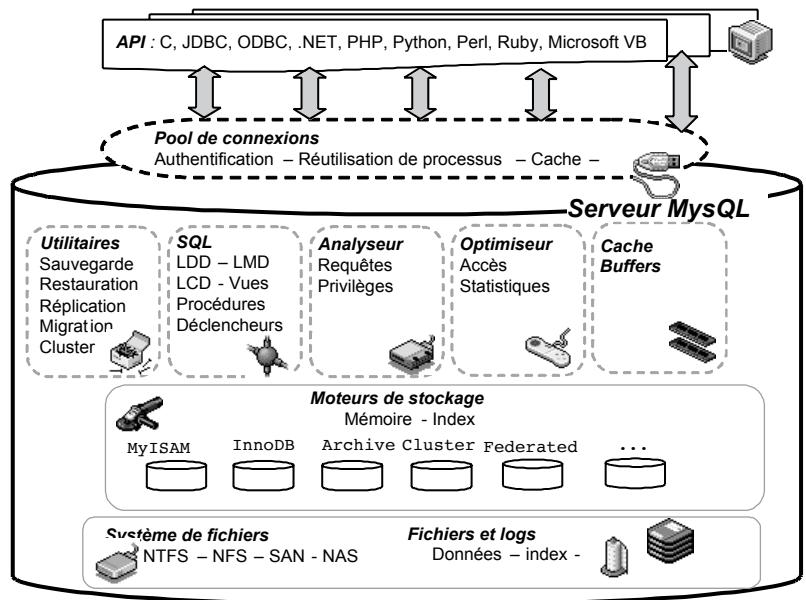
Ce qui n'est pas encore planifié reste la prise en charge du stockage de données au XML et les extensions objets de SQL3 (principalement les types abstraits, les héritage méthodes).

Les changements opérés sont présentés en détail et par version à l'adresse <http://dev.mysql.com/doc/refman/x.y/en/news.html> (x.y étant le numéro de version majeure exemple actuellement 5.1)

Offre du moment

La figure suivante (merci au passage à <http://www.iconarchive.com>) présente la majorité des fonctionnalités de MySQL qui se positionnent au sein du serveur de données (SGBD).

Figure 0-2 Offre MySQL



Les API permettent d'intégrer SQL dans des programmes de différents langages. Le langage SQL sera utilisé par tous ceux (manuellement ou par un outil) travaillant sur la base de données (administrateur, développeur, utilisateur). Le langage procédural de MySQL permet d'incorporer nativement tout ordre SQL dans un programme.

Concrètement, une fois téléchargé et installé, vous avez accès à un SGBD, un client en ligne de texte (interface de commande). Les pilotes ODBC, JDBC, API pour les langages C et C++, les outils *MySQL Administrator*, *MySQL Query Browser*, et *MySQL Migration Toolkit* sont à télécharger puis à installer séparément.

Deux types de licences sont proposés par MySQL : commerciale et GPL. Dans le cadre d'un développement d'application entièrement sous licence GPL, MySQL est gratuit. Il en va de même si l'il n'est pas copié, modifié, distribué ou employé pour une utilisation en combinaison avec un serveur Web (si vous développez l'application Web vous-même).

Dans tous les autres cas, il est nécessaire d'obtenir une licence commerciale. Par exemple, si vous incluez un serveur MySQL ou des pilotes MySQL dans une application non *open source*.

Et la concurrence ?

Elle fait rage. Deux types de concurrents : ceux qui sont dans le domaine de l'*open source* et ceux qui engrangent des dollars à tour de bras.

Dans la première catégorie, citons principalement Interbase 6 et Firebird de Borland, PostgreSQL et Berkeley DB.

Dans la seconde, on trouvera 4D, Filemaker, IBM (*DB2, UDB*), Informix, Microsoft (*SQL Server, Access*), Oracle et Sybase (*Adaptive Server, SQL Anywhere Studio*). Depuis peu, les grands éditeurs s'ouvrent à la distribution gratuite. Ces produits nécessitent de solides configurations, ce que n'exigent pas les éditeurs de la première catégorie :

- Fin 2004, Microsoft proposait *SQL Server 2005 Express Edition* qui est limité d'un seul processeur, 1 Go de mémoire et 4 Go de données.
- Fin 2004, Sybase offrait *ASE Express Edition*. Plusieurs instances peuvent cohabiter sur une machine tout en n'utilisant pas plus d'un processeur, moins de 2 Go de RAM et pas plus de 5 Go de données.
- Oracle annonçait, en novembre 2005, une version gratuite avec *Oracle Database XE (Express Edition)*. Version toutefois « limitée » à un processeur, moins de 1 Go de RAM et une seule instance par système et pas plus de 4 Go de données.
- Peu de temps avant, IBM avait aussi annoncé une licence gratuite de *DB2 Express*. Pas plus de deux processeurs et moins de 4 Go de RAM.

Tout le monde est donc sur les rangs. La guerre des versions n'est donc pas finie. Bien malin qui pourra prédire qui gagnera sur ce terrain.

Notion de schéma [database]

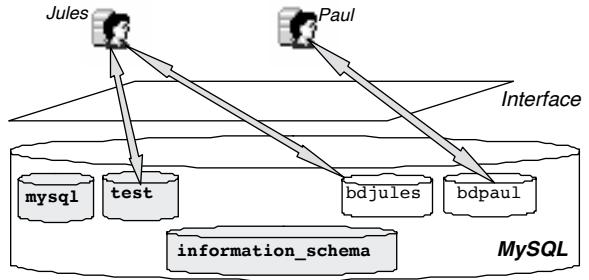
MySQL appelle *database* un regroupement logique d'objets (tables, index, vues, déclencheurs, procédures cataloguées, etc.) pouvant être stockés à différents endroits de l'espace disque. Je ferai donc souvent référence au terme « base de données » pour parler de cette notion.

On peut aussi assimiler ce concept à la notion de schéma, pour ceux qui connaissent Oracle. Là où MySQL et d'autres SGBD diffèrent, c'est sur la notion d'utilisateur (*user*).

- Pour tous, un utilisateur sera associé à un mot de passe pour pouvoir se connecter et manipuler des tables (s'il en a le droit, bien sûr).
- Pour MySQL, il n'y a pas de notion d'appartenance d'un objet (table, index, etc.) à un utilisateur. Un objet appartient à son schéma (*database*). Ainsi, deux utilisateurs distincts (Jules et Paul) se connectant sur la même base (*database*) ne pourront pas créer une table de nom **Compagnie**. S'ils doivent le faire, ce sera dans deux bases différentes (**bdjules** et **bdpaul**).
- Pour Oracle ou d'autres SGBD, chaque objet appartient à un schéma (*user*). Ainsi, deux utilisateurs distincts Jules et Paul se connectant à la base (qui est un ensemble de mas) pourront créer chacun une table de nom **Compagnie** (la première sera référencée **Jules.Compagnie**, la seconde **Paul.Compagnie**).

La figure suivante illustre deux utilisateurs travaillant sur différentes bases par une interface qui peut être la fenêtre de commande en ligne (dans la majeure partie des enseignements) ou un langage de programmation comme C, Java ou PHP (utilisation d'une API). Notez l'existence de trois bases initiales (**mysql**, **test** et **information_schema**) que nous détaillerons au chapitre 5.

Figure 0-3 Notions de base et d'utilisateur MySQL



Notion d'hôte

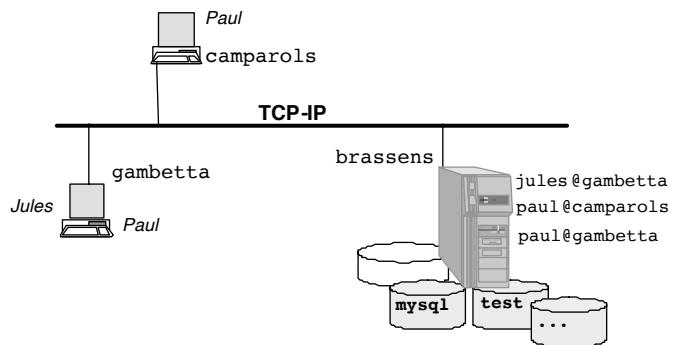
MySQL dénomme *host* la machine hébergeant le SGBD. MySQL diffère aussi à ce niveau des autres SGBD, car il est possible de distinguer des accès d'un même utilisateur suivant qu'il se connecte à partir d'une machine ou d'une autre. La notion d'identité est basée sur le combinaison du nom d'utilisateur MySQL (*user*) côté serveur, machine cliente.

Identités

Ainsi l'utilisateur **Paul**, se connectant depuis la machine **camparols**, peut ne pas être le même que celui se connectant depuis la machine **gambetta**. S'il s'agit du même, il faudra au niveau du serveur, éventuellement composer un ensemble de prérogatives équivalent pour les deux accès (voir le chapitre 5). S'il s'agit de deux personnes différentes, il faudra distinguer

les attributions des différents droits. La figure suivante illustre le fait que deux utilisateurs peuvent se connecter par deux accès différents. Trois identités seront donc à prévoir côté serveur.

Figure 0-4 Notion d'hôte MySQL



Nous verrons au chapitre 5 comment configurer le serveur et les clients. Nous étudierons au chapitre 9 des outils graphiques d'aide à l'administration.

Accès à MySQL

Une fois que vous aurez installé MySQL sur votre ordinateur, vous serez libre de choisir l'accès qui vous convient. Ce livre utilise essentiellement l'interface de commande en ligne fournie avec le SGBD, mais aussi Java via JDBC, et le navigateur Web au travers de PHP.

Aspects étudiés

Nous n'étudierons pas tous les éléments d'une base, car certains sont assez spécifiques et sortent du cadre traditionnel de l'enseignement, même supérieur. Le tableau suivant indique dans quel chapitre du livre les principaux éléments d'un schéma sont étudiés :

Tableau 0-3 Éléments d'une base MySQL

Éléments étudiés – Chapitre	Aspects non étudiés
Déclencheurs (<i>triggers</i>) – 7 Fonctions et procédures – 7 Tables et index – 1 Séquences – 2, 5 Vues (<i>views</i>) et utilisateurs – 5	Clusters – Moteurs de stockage (<i>storage engine</i>) – Partitionnement – Données spatiales

Si tout se passe bien, comptez quelques minutes pour installer MySQL. Je vous conseille toutefois de créer un point de restauration Windows pour pouvoir revenir à votre dernière bonne configuration connue. Extraire l'archive téléchargée sur le site officiel de MySQL (<http://dev.mysql.com/downloads/>) dans un répertoire temporaire (exemple : C:\Temp) et exécuter `Setup.exe`.

Installation

Le premier choix est donné pour le type d'installation : typique, complète et personnalisée. Choisir typique dans un premier temps.

Le deuxième écran vous invite à vous enregistrer – ce n'est pas obligatoire, mais conseillé toutefois pour saluer l'énorme travail fait par ces jeunes informaticiens de génie (voire pommade est passée).

La suite concerne la configuration du serveur de données (choisir la configuration détaillée pour suivre les différentes étapes et mieux comprendre le paramétrage de votre serveur).

- L'assistant propose en premier lieu de déterminer le type de votre serveur (machine de développement, serveur ou machine dédiée). J'ai opté pour le premier choix.
- Choisissez ensuite le type de base de données que vous comptez exploiter (multifonctionnel, mode transactionnel seulement ou jamais transactionnel). J'ai opté pour le premier choix.
- Déterminez ensuite le répertoire qui contiendra les données des bases *InnoDB* (c'est le seul type de tables que nous utiliserons dans cet ouvrage, car étant le seul, dans cette version, à prendre en charge les clés étrangères). J'ai opté pour le choix par défaut.
- Choisissez ensuite l'option qui convient à votre utilisation (en fonction du nombre de connexions). J'ai opté pour le premier choix (vingt connexions maximum).
- Il est ensuite possible de modifier le port UDP d'écoute (par défaut 3306) et le mode de comportemental du serveur par rapport à la syntaxe des instructions SQL. J'ai opté pour le premier choix (mode strict).
- Attention à ne pas sélectionner un jeu de caractères japonais dans l'écran qui arrive ensuite. J'ai opté pour les choix par défaut (*West European* et *latin1*).
- L'écran suivant permet de déclarer MySQL comme un service Windows (qu'on peut arrêter via le panneau de configuration plutôt qu'avec une belle commande en ligne, ouvrir un Ctrl-Alt-Suppr bien connu des bidouilleurs...). Penser aussi à inclure dans le paramètre de chemin de l'exécutable `mysql` de manière à pouvoir lancer une connexion en ligne de commande. Il vous suffit de cocher la case appropriée.
- Saisissez un mot de passe pour root ; vous pouvez aussi créer un compte anonyme (connexion fantôme sans utilisateur ni mot de passe).

Après avoir validé la demande d'exécution (bouton **Execute**), quelques secondes vont s'écouler avant qu'on vous invite à terminer l'installation.

Voilà, MySQL est installé. Si ce n'est pas le cas, la section suivante vous aidera sûrement. Dernière chose, si vous n'utilisez pas souvent MySQL, pensez à arrêter et à positionner **MySQL/.../MySQL Server Instance Wizard**.

Pour modifier une configuration existante, vous trouverez un assistant dans **Démarrer/Tous les programmes/MySQL/.../MySQL Server Instance Wizard**, choix **Remove Instance**. Cependant le répertoire installé par défaut dans **Program Files** reste sur le disque (comptant une centaine de mégaoctets). De même qu'en ce qui concerne les entrées dans le menu **Démarrer**, je ne parle pas de la base de registres qui est inchangée...

Dans *Panneau de configuration*, **Ajout/Suppression de programmes**, supprimez **MySQL Server**. Il reste encore des choses dans la base de registres. Le répertoire MySQL sous **Program Files** a diminué de taille, mais il est toujours présent. Une fois que tout est fait, redémarrez votre machine et reprenez l'installation initiale. Prudence si vous modifiez les chemins des répertoires des données entre plusieurs installations.

Désinstallation

Pour supprimer une configuration, vous trouverez un assistant dans **Démarrer/Tous les programmes/MySQL/.../MySQL Server Instance Wizard**, choix **Remove Instance**. Cependant le répertoire installé par défaut dans **Program Files** reste sur le disque (comptant une centaine de mégaoctets). De même qu'en ce qui concerne les entrées dans le menu **Démarrer**, je ne parle pas de la base de registres qui est inchangée...

Reconfiguration

En relançant une installation, il vous est donné d'ajouter des composants (option **Modify**) si vous n'avez pas fait une installation complète au début. Vous avez aussi la possibilité de « réparer » (option **Repair**) une configuration existante.

Premiers pas

La procédure suivante va guider vos premiers pas pour travailler sous cette interface d'une manière « professionnelle ». Il s'agit de stocker vos fichiers de commande qui pourront servir à différentes actions (créations de tables, de vues ou d'utilisateurs, insertions, modifications ou suppressions d'enregistrements, élaboration de requêtes, de procédures cataloguées, etc.).

L'interface de commande

L'interface de commande en ligne porte le nom du SGBD (**mysql**). Cette interface ressemble à une fenêtre DOS ou **telnet** et permet de dialoguer de la plus simple façon avec la base de données. L'utilisation peut être interactive ou « *batch* ». Quand l'utilisation est interactive,

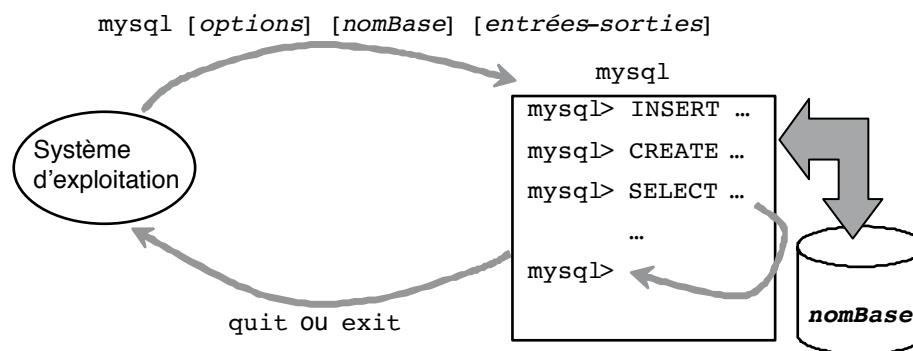
(c'est le mode le plus courant), le résultat des extractions est présenté sous une forme tabulaire au format ASCII.

Vous verrez qu'il est notamment possible :

- d'exécuter des instructions SQL (créer 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 privilégiés, etc.).

Le principe général de l'interface est le suivant : après une connexion locale ou distante, les 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 tout de même 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 et 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 située 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` que vous trouvez dans le répertoire `Introduction`, à l'aide du bloc-notes (ou d'un éditeur de texte de votre choix).

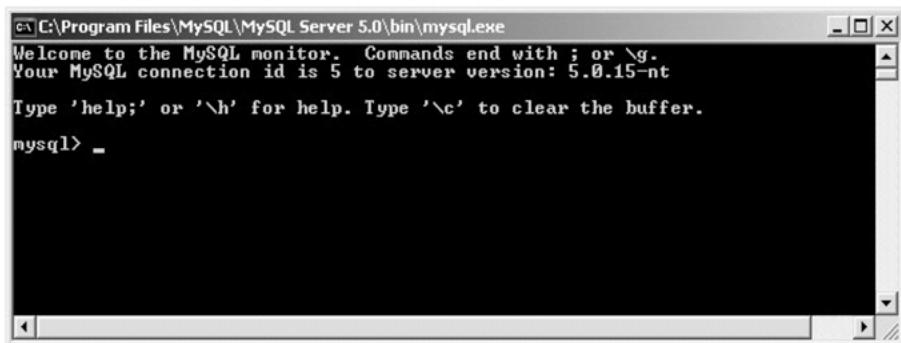
choix). Changez « util » par le nom de l'utilisateur à créer (modifiez aussi le nom de la base). Vous pouvez changer le mot de passe si vous voulez. Enregistrez ce fichier dans un de vos répertoires.

Connexion au serveur

Dans une fenêtre de commande Windows, Linux (ou autre), lancer l'interface en ligne de commande connectant l'utilisateur root avec le mot de passe que vous avez donné lors de l'installation.

```
mysql --user=root -p
```

Figure 0-6 Interface en mode ligne de commande



Une fois connecté, par copier-coller (en effectuant un clic droit dans la fenêtre de commande MySQL), exécutez une à une les différentes instructions (création de la base, de l'utilisateur, des priviléges et déconnexion de root). Nous étudierons au chapitre 5 les notions élémentaires de droits et de sécurité. Les lignes précédées de « -- » sont des commentaires.

Voilà, votre utilisateur (*util*) est créé, il peut se connecter et il possède toutes les prérogatives sur la base (*bdutil*) pour exécuter les instructions décrites dans cet ouvrage. Pour tester votre connexion, lancez la commande suivante qui se connecte au serveur sur la base *bdutil*, sous l'utilisateur *util*.

```
mysql --user=util --host=localhost -p --database=bdutil
```

Vérification de la version

Pour contrôler la version du serveur sur lequel vous êtes connecté, exécutez la connexion et la déconnexion suivante dans une fenêtre de commande Windows, Linux (ou autre).

```
mysql --version
```

Si vous êtes déjà connecté, la commande « `SELECT VERSION();` » vous renseignera également à propos de la version du SGBD. Si vous n'êtes pas en version 5, il vous sera impossible de travailler avec des procédures cataloguées, vues et déclencheurs. Pour ma part lors de la rédaction de cet ouvrage, cette commande a renvoyé le résultat suivant :

```
+-----+
| VERSION() |
+-----+
| 5.0.15-nt |
+-----+
```

Options de base

Les principales options au lancement de mysql sont résumées dans le tableau suivant :

Tableau 0-4 Principales options de la commande mysql

Option	Commentaire
--help ou -?	Affiche les options disponibles, l'état des variables d'environnement et rend la main.
--batch ou -B	Toute commande SQL peut être lancée dans la fenêtre de commande système sans pour autant voir l'invite ; les résultats (colonnes) séparés par des tabulations.
--database= <i>nomBD</i> ou -D <i>nomBD</i>	Sélection de la base de données à utiliser après la connexion.
--host= <i>nomServeur</i> ou -h <i>nomServeur</i>	Désignation du serveur.
--html ou -H	Formatte le résultat des extractions en HTML.
--one-database ou -O	Restreint les instructions à la base de données spécifiée initialement.
-p	Demande le mot de passe sans l'employer en tant que paramètre.
--password= <i>motdePasse</i>	Transmission du mot de passe de l'utilisateur à connecter. Évitez cette option et préférez la précédente...
--prompt= <i>parametre</i>	Personnalise l'invite de commande (par défaut mysql>).
--silent ou -s	Configure le mode silence pour réduire les messages de MySQL.
--skip-column-names ou -N	N'écrira aucun en-tête de colonne pour les résultats d'extraction.
--table ou -t	Formatte le résultat des extractions en tables à en-tête de colonne par défaut dans le mode interactif.
--tee= <i>cheminNomFichier</i>	Copie la trace de toute la session dans le fichier que vous indiquez.
--user= <i>utilisateur</i> ou -u <i>utilisateur</i>	Désigne l'utilisateur devant se connecter.
--verbose ou -v	Mode verbeux pour avoir davantage de messages du serveur.
--version ou -V	Affiche la version du serveur et rend la main.
--vertical ou -E	Affiche les résultats des extractions verticalement (non plus en lignes horizontales).
--xml ou -X	Formatte le résultat des extractions en XML. Les noms de balises générées sont <resultset> pour la table résultat, <row> pour chaque ligne et <field> pour les colonnes.

Ces options peuvent se combiner en les séparant simplement par un espace (exemple : `mysql --tee=D:\\dev\\sortiemysql.txt --database=bdsoutou` va se connecter anonymement à la base `bdsoutou` en inscrivant le contenu de la trace de la session dans fichier `sortiemysql.txt` situé dans le répertoire `D:\\dev`).

Le tableau suivant résume les principaux paramètres pour afficher les invites de commandes (relatives à l'option `prompt`).

Tableau 0-5 Principales options de l'invite de commandes

Option	Commentaire
<code>\v</code>	Version du serveur.
<code>\d</code>	Base de données en cours d'utilisation.
<code>\h</code>	Nom du serveur.
<code>\u</code>	Nom d'utilisateur.
<code>\U</code>	Nom d'utilisateur long (au format <code>nom@serveur</code>).
<code>_</code>	Un espace.
<code>\R</code>	Heure (0 à 23).
<code>\m</code>	Minutes.
<code>\s</code>	Secondes.
<code>\Y</code>	Année sur quatre chiffres.
<code>\D</code>	Date en cours.
<code>\c</code>	Compteur d'instructions.

Batch

Pour lancer plusieurs commandes regroupées dans un fichier à extension « `.sql` », il faut préciser le chemin du fichier et celui qui contiendra les éventuels résultats (c'est du « brut ou décoffrage » !). Ainsi, l'instruction suivante exécute dans la base `bdsoutou`, sous l'autorisation de l'utilisateur `soutou`, les commandes contenues dans le fichier `Testbatch.sql` situées dans le répertoire `D:\\dev` (notez l'utilisation du double *back-slash* pour désigner une arborescence Windows). Le résultat sera consigné dans le fichier `sortie.txt` du même répertoire.

```
mysql --user=soutou --password=iut bdsoutou
<D:\\dev\\Testbatch.sql >D:\\dev\\sortie.txt
```

Votre prompt, et vite !

L'exécution de l'instruction « `mysql --prompt="(\u@\h) [\d]>` » --user=root --password=iut dans une fenêtre de commande *shell* ou *DOS* connectera l'utilisateur root en lui demandant son mot de passe. L'invite de commande à l'affichage sera de la forme suivante :

(root@localhost) [bdsoutou]> une fois que root aura sélectionné la base bdsoutou (par la commande « use nombase; »).



Configurez votre invite de commande SQL dans le fichier de configuration my.ini située dans le répertoire C:\Program Files\MySQL\MySQL Server xx de la manière qui vous convient le plus.

Pour ma part, j'ai ajouté les deux lignes suivantes sous la section [mysql] elle-même sous l'étiquette [client].

```
#mon prompt  
prompt=(\\u@\\h) [\\d]\\_mysql>\\_
```

Une fois le serveur redémarré, en considérant que votre compte s'appelle **util**, toutes les commandes SQL devraient en principe être préfixées de la syntaxe suivante :

```
[util@localhost] [bdutil] mysql>
```

Commandes de base

Une fois connecté, vous pouvez utiliser des commandes ou faire des copier-coller d'un éditeur de texte dans l'interface mysql (ce moyen de faire correspond plus à un environnement de test qui conviendra à l'apprentissage). Le tableau suivant résume les principales instructions pour manipuler le *buffer* d'entrée de l'interface.

Tableau 0-6 Commandes de base du buffer d'entrée

Commande	Commentaire
?	Affichage des commandes disponibles.
delimiter chaîne	Modifie le délimiteur (par défaut « ; »).
use nomBase	Rend une base de données courante.
prompt chaîne	Modifie l'invite de commande avec les paramètres vus précédemment.
quit ou exit	Quitte l'interface.
source cheminNomFichier.sql	Charge et exécute dans le buffer le contenu du <i>cheminNomFichier.sql</i> (ex : source D:\\dev\\\\Testbatch.sql exécutera le script Testbatch.sql situé dans D:\\dev).
tee nomFichierSortie	Création nomFichierSortie dans le répertoire C:\\Program Files\\MySQL\\MySQL Server n.n\\bin qui contiendra la trace de la session.

La commande **source** est très utile pour éviter les copier-coller de trop nombreuses instructions.

Partie I

SQL de base

Définition des données

Ce chapitre décrit les instructions SQL qui constituent l'aspect LDD (langage de définition des données). À cet effet, nous verrons notamment comment déclarer une table avec ses différents index et contraintes.

Tables relationnelles

Une table est créée en SQL par l'instruction `CREATE TABLE`, modifiée au niveau de sa structure par l'instruction `ALTER TABLE` et supprimée par la commande `DROP TABLE`.

Création d'une table (`CREATE TABLE`)

Pour pouvoir créer une table dans votre base, il faut que vous ayez reçu le privilège `CREATE`. Le mécanisme des privilèges est décrit au chapitre 5.

La syntaxe SQL simplifiée est la suivante :

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] [nomBase.]nomTable
    (colonne1 type1
        [NOT NULL | NULL] [DEFAULT valeur1] [COMMENT 'chaine1'])
    [, colonne2 type2
        [NOT NULL | NULL] [DEFAULT valeur2] [COMMENT 'chaine2'])
    [CONSTRAINT nomContrainte1 typeContrainte1] ...)
    [ENGINE= InnoDB | MyISAM | ...];
```

- **TEMPORARY** : pour créer une table qui n'existera que durant la session courante (la sera supprimée à la déconnexion). Deux connexions peuvent ainsi créer deux tables temporaires de même nom sans risquer de conflit. Il faut posséder le privilège `CREATE TEMPORARY TABLES`.
- **IF NOT EXISTS** : permet d'éviter qu'une erreur se produise si la table existe déjà (c'est le cas, elle n'est aucunement affectée par la tentative de création).
- **nomBase** : (jusqu'à 64 caractères permis dans un nom de répertoire ou de fichier « / », « \ » et « . ») s'il est omis, il sera assimilé à la base connectée. S'il est précisé

désigne soit la base connectée soit une autre base (dans ce cas, il faut que l'utilisateur courant ait le droit de créer une table dans l'autre base). Nous aborderons ces points dans le chapitre Contrôle des données et nous considérerons jusque-là que nous travaillons dans la base courante (ce sera votre configuration la plupart du temps).

- *nomTable* : mêmes limitations que pour le nom de la base.
- *colonnei typei* : nom d'une colonne (mêmes caractéristiques que pour les noms de tables) et son type (INTEGER, CHAR, DATE...). Nous verrons quels types sont disponibles sous MySQL. La directive DEFAULT fixe une valeur par défaut. La directive NOT NULL interdit que la valeur de la colonne soit nulle.



NULL représente une valeur qu'on peut considérer comme non disponible, non affectée, inconnue ou inapplicable. Elle est différente d'un espace pour un caractère ou d'un zéro pour un nombre.

- COMMENT : (jusqu'à 60 caractères) permet de commenter une colonne. Ce texte sera ensuite automatiquement affiché à l'aide des commandes SHOW CREATE TABLE et SHOW FULL COLUMNS (voir le chapitre 5).
- *nomContraintei typeContraintei* : nom de la contrainte et son type (clé primaire, clé étrangère, etc.). Nous allons détailler dans le paragraphe suivant les différentes contraintes possibles.
- ENGINE : définit le type de table (par défaut InnoDB, bien adapté à la programmation de transactions et adopté dans cet ouvrage). MyISAM correspond au type par défaut des versions 3, parfaitement robuste, mais ne supportant pas pour l'heure l'intégrité référentielle. D'autres types existent, citons MEMORY pour les tables temporaires, ARCHIVE, etc.
- « ; » : symbole par défaut qui termine une instruction MySQL en mode ligne de commande (en l'absence d'un autre délimiteur).

Délimiteurs

En mode ligne de commande, il est possible (par la directive delimiter) de choisir le symbole qui terminera chaque instruction. Dans l'exemple suivant on choisit le dollar ; au cours de l'ouvrage nous resterons avec le symbole par défaut de MySQL à savoir « ; ».

```
delimiter $  
CREATE TABLE Test (t CHAR(8))$
```

Sensibilité à la casse

Alors que MySQL est sensible par défaut à la casse (au niveau des noms de base et de table dans la plupart des distributions Unix, il ne l'est pas pour Windows ! En revanche, concernant les noms de colonnes, index, alias de colonnes, déclencheurs et procédures cataloguées,

MySQL n'est pas sensible à la casse tous systèmes confondus. En fait, tous ces noms stockés en minuscules dans le dictionnaire de données.

La variable `lower_case_table_names` permet de forcer la sensibilité à la casse des noms des tables et des bases de données (si elle vaut 0, la sensibilité à la casse est active ; les noms sont stockés en minuscules ; 1, pas de sensibilité à la casse et les noms stockés en minuscules ; 2, pas de sensibilité à la casse et les noms sont stockés en respectant la casse).

Je vous invite à positionner cette variable à 0 de manière à homogénéiser le codage et contrôler un peu plus l'écriture de vos instructions SQL. De plus, c'est l'option par défaut sur Linux. Dans le fichier `my.ini`, sous la section serveur identifiée par `[mysqld]`, ajoutez une ligne et le commentaire suivants :

```
# Rend sensible à la CASSE les noms de tables et de database
lower_case_table_names=0
```



Refusez ce type de programmation (rendue impossible d'ailleurs si la variable `lower_case_table_names` est positionnée à 0).

```
mysql> SELECT * FROM Avion WHERE AVION.capacite > 150;
```

Par ailleurs, la casse devrait toujours avoir (quel que soit le SGBD concerné) une incidence majeure dans les expressions de comparaison entre colonnes et valeurs, que ce soit dans une instruction SQL ou un test dans un programme.



Ainsi l'expression « `nomComp='Air France'` » a la même signification que l'expression « `nomComp='AIR France'` » avec MySQL ! Horreur, oui.

Donc, si vous désirez vérifier la casse au sein même des données, il faudra utiliser la fonction `BINARY()` qui convertit en bits une expression. En effet, « `BINARY('AIR France')` » est différent de « `BINARY('Air France')` » et « `BINARY(nomComp)=BINARY('Air France')` » renverra vrai en respectant la casse.

Commentaires

Dans toute instruction SQL (déclaration, manipulation, interrogation et contrôle des données), il est possible d'inclure des retours chariot, des tabulations, espaces et commentaires (sur une ligne précédée de deux tirets « `--` », en fin de ligne à l'aide du dièse « `#` », au sein d'une ligne ou sur plusieurs lignes entre « `/*` » et « `*/` »). Les scripts suivants décrivent la déclaration d'une même table en utilisant différentes conventions :

Tableau 1-1 Différentes écritures SQL

Sans commentaire	Avec commentaires
<pre>CREATE TABLE Test(colonne DECIMAL(38,8));</pre>	<pre>CREATE TABLE -- nom de la table Test(#début de la description COLONNE DECIMAL(38,8)) -- fin, ne pas oublier le point-virgule ; CREATE TABLE Test (/* une plus grande description des colonnes */ COLONNE /* type : */ DECIMAL(38,8))</pre>
<pre>CREATE TABLE Test (colonne DECIMAL(38,8));</pre>	

Comme nous le conseillons dans l'avant-propos, il est préférable d'utiliser les conventions suivantes :



- Tous les mots-clés de SQL sont notés en majuscules.
- Les noms de tables sont notés en Minuscules (excepté la première lettre, ces noms seront quand même stockés dans le système en minuscules).
- Les noms de colonnes et de contraintes en minuscules.

L'adoption de ces conventions rendra vos requêtes, scripts et programmes plus lisibles (un peu à la mode Java).

Premier exemple

Le tableau ci-après décrit l'instruction SQL qui permet de créer la table `Compagnie` illustrée par la figure suivante, dans la base `bdsoutou` (l'absence du préfixe « `bdsoutou.` » conduira au même résultat si `bdsoutou` était la base connectée lors de l'exécution du script).

Figure 1-1 Table à créer

Compagnie

comp	nrule	rue	ville	nomComp

Tableau 1-2 Créeation d'une table et de ses contraintes

Instruction SQL	Commentaires
<pre>CREATE TABLE bdsoutou.Compagnie (comp CHAR(4), nrule INTEGER(3), rue CHAR(20), ville CHAR(15) DEFAULT 'Paris' COMMENT 'Par defaut : Paris', nomComp CHAR(15) NOT NULL);</pre>	<p>La table contient cinq colonnes (quatre chaînes de caractères et un numérique de trois chiffres). La colonne <code>ville</code> est commentée.</p> <p>La table inclut en plus deux contraintes :</p> <ul style="list-style-type: none"> • <code>DEFAULT</code> qui fixe <code>Paris</code> comme valeur par défaut de la colonne <code>ville</code> ; • <code>NOT NULL</code> qui impose une valeur non nulle pour la colonne <code>nomComp</code>.

Contraintes

Les contraintes ont pour but de programmer des règles de gestion au niveau des colonnes de tables. Elles peuvent alléger un développement côté client (si on déclare qu'une note doit être comprise entre 0 et 20, les programmes de saisie n'ont plus à tester les valeurs en entrée mais seulement le code retour après connexion à la base ; on déporte les contraintes côté serveur).

Les contraintes peuvent être déclarées de deux manières :

- En même temps que la colonne (valable pour les contraintes monocolonnes) ; ces contraintes sont dites « en ligne » (*inline constraints*). L'exemple précédent en déclare deux.
- Après que la colonne est déclarée ; ces contraintes ne sont pas limitées à une colonne et peuvent être personnalisées par un nom (*out-of-line constraints*).

Il est recommandé de déclarer les contraintes `NOT NULL` en ligne, les autres peuvent soit être déclarées en ligne, soit être nommées. Étudions à présent les types de contraintes nommées (*out-of-line*). Les quatre types de contraintes les plus utilisées sont les suivants :

```
CONSTRAINT nomContrainte
  UNIQUE (colonne1 [,colonne2]...)
  PRIMARY KEY (colonne1 [,colonne2]...)
  FOREIGN KEY (colonne1 [,colonne2]...)
  REFERENCES nomTablePere [(colonne1 [,colonne2]...)]
    [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
    [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
  CHECK (condition)
```

- La contrainte `UNIQUE` impose une valeur distincte au niveau de la table (les valeurs nulles font exception à moins que `NOT NULL` soit aussi appliquée sur les colonnes).
- La contrainte `PRIMARY KEY` déclare la clé primaire de la table. Un index est généralement automatiquement créé sur la ou les colonnes concernées. Les colonnes clés primaires ne peuvent être ni nulles ni identiques (en totalité si elles sont composées de plusieurs colonnes).

- La contrainte FOREIGN KEY déclare une clé étrangère entre une table enfant (*child*) et une table père (*parent*). Ces contraintes définissent l'intégrité référentielle que nous aborderons plus tard. Les directives ON UPDATE et ON DELETE disposent de quatre options que nous détaillerons avec les directives MATCH à la section Intégrité référentielle du chapitre 2).



La contrainte CHECK impose un domaine de valeurs ou une condition simple ou complexe entre colonnes (exemple : CHECK (note BETWEEN 0 AND 20), CHECK (grade='Copilote' OR grade='Commandant')). Cette contrainte est prise en charge au niveau de la déclaration mais n'est pas encore opérationnelle même dans la version 5.1.



Il est recommandé de ne pas définir de contraintes sans les nommer (bien que cela soit possible), car il sera difficile de les faire évoluer (désactivation, réactivation, suppression), et la lisibilité des programmes en sera affectée.

Nous verrons au chapitre 3 comment ajouter, supprimer, désactiver et réactiver des contraintes (options de la commande ALTER TABLE).

Conventions recommandées

Adoptez les conventions d'écriture suivantes pour vos contraintes :



- Préfixez par pk_ le nom d'une contrainte clé primaire, fk_ une clé étrangère, ck_ une vérification, un_ une unicité.
- Pour une contrainte clé primaire, suffixez du nom de la table la contrainte (exemple pk_Avion).
- Pour une contrainte clé étrangère, renseignez (ou abrégez) les noms de la table source, de la clé, et de la table cible (exemple fk_Pil_compa_Comp).

En respectant nos conventions, déclarons les tables de l'exemple suivant (Compagnie avec sa clé primaire et Avion avec sa clé primaire et sa clé étrangère). Du fait de l'existence de la clé étrangère, la table Compagnie est dite « parent » (ou « père ») de la table Avion « enfant » (ou « fils »). Cela résulte de la traduction d'une association *un-à-plusieurs* entre les deux tables (*De UML à SQL*, Eyrolles 2002). Nous reviendrons sur ces principes à la section Intégrité référentielle du prochain chapitre.

Figure 1-2 Deux tables reliées à créer

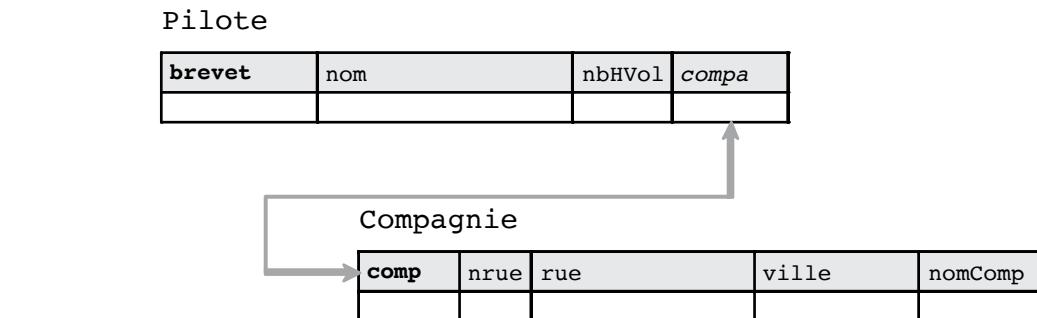


Tableau 1-3 Contraintes en ligne et nommées

Web	Tables	Contraintes
	<pre>CREATE TABLE Compagnie (comp CHAR(4), nrue INTEGER(3), rue CHAR(20), ville CHAR(15) DEFAULT 'Paris' COMMENT 'Par defaut : Paris', nomComp CHAR(15) NOT NULL, CONSTRAINT pk_Compagnie PRIMARY KEY(comp));</pre>	Deux contraintes en ligne et une contrainte nommée de clé primaire.
	<pre>CREATE TABLE Pilote (brevet CHAR(6), nom CHAR(15) NOT NULL, nbHVol DECIMAL(7,2), compa CHAR(4), CONSTRAINT pk_Pilote PRIMARY KEY(brevet), CONSTRAINT ck_nbHVol CHECK(nbHVol BETWEEN 0 AND 20000), CONSTRAINT un_nom UNIQUE (nom), CONSTRAINT fk_Pil_compa_Comp FOREIGN KEY (compa) REFERENCES Compagnie(comp));</pre>	Une contrainte en ligne et quatre contraintes nommées : <ul style="list-style-type: none"> • Clé primaire • NOT NULL • CHECK (nombre d'heures de vol compris entre 0 et 20 000) • UNIQUE (homonymes interdits) • Clé étrangère

Remarques



- L'ordre n'est pas important dans la déclaration des contraintes nommées.
- PRIMARY KEY équivaut à :UNIQUE + NOT NULL + index.
- L'ordre de création des tables est important quand on définit les contraintes en même temps que les tables (on peut différer la création ou l'activation des contraintes, voir le chapitre 10). Il faut créer d'abord les tables « pères » puis les tables « fils ». Le script de destruction des tables suit le raisonnement inverse.

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 1 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 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(<i>n</i>) [BINARY ASCII UNICODE]	Chaîne fixe de <i>n</i> octets ou caractères.	Taille fixe (maximum de 255 caractères).
VARCHAR(<i>n</i>) [BINARY]	Chaîne variable de <i>n</i> caractères ou octets.	Taille variable (maximum de 65 535 caractères).
BINARY(<i>n</i>)	Chaîne fixe de <i>n</i> octets.	Taille fixe (maximum de 255 octets).
VARBINARY(<i>n</i>)	Chaîne variable de <i>n</i> octets.	Taille variable (maximum de 255 octets).
TINYTEXT(<i>n</i>)	Flot de <i>n</i> octets.	Taille fixe (maximum de 255 octets).
TEXT(<i>n</i>)	Flot de <i>n</i> octets.	Taille fixe (maximum de 65 535 octets).
MEDIUMTEXT(<i>n</i>)	Flot de <i>n</i> octets.	Taille fixe (maximum de 16 mégaoctets).
LONGTEXT(<i>n</i>)	Flot de <i>n</i> octets.	Taille fixe (maximum de 4,29 gigaoctets).

Les types **BINARY** et **VARBINARY** sont similaires à **CHAR** et **VARCHAR**, excepté par le fait qu'ils contiennent des chaînes d'octets sans tenir compte d'un jeu de caractères en particulier.

Les quatre types permettant aussi de stocker du texte sont **TINYTEXT**, **TEXT**, **MEDIUMTEXT** et **LONGTEXT**. Ces types sont associés à un jeu de caractères. Il n'y a pas de mécanisme de suppression d'espaces de fin ni de possibilité d'y associer une contrainte **DEFAULT**.

Valeurs numériques

De nombreux types sont proposés par MySQL pour définir des valeurs exactes (entiers ou décimaux) positifs ou négatifs : **INTEGER** et **SIMILARINT**, et des valeurs à virgule fixe ou flottante (**FLOAT**, **DOUBLE** et **DECIMAL**). En plus des spécifications de la norme SQL, MySQL propose les types d'entiers restreints (**TINYINT**, **MEDIUMINT** et **BIGINT**). Le tableau suivant décrit ces types :

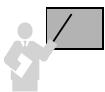
- n indique le nombre de positions de la valeur à l'affichage (le maximum est de 255). Ainsi, il est possible de déclarer une colonne **TINYINT(4)** sachant que seules 3 positions sont nécessaires en fonction du domaine de valeurs permises.

Tableau 1-5 Types de données numériques

Type	Description
BIT[(n)]	Ensemble de n bits. Taille de 1 à 64 (par défaut 1).
TINYINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur un octet) de -128 à 127 signé, 0 à 255 non signé.
BOOL et BOOLEAN	Synonymes de TINYINT(1) , la valeur zéro est considérée comme fausse. Le non-zéro est considéré comme vrai. Dans les prochaines versions, le type <i>boolean</i> , comme le préconisé par la norme SQL, sera réellement pris en charge.
SMALLINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 2 octets) de -32 768 à 32 767 signé, 0 à 65 535 non signé.
MEDIUMINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 3 octets) de -8 388 608 à 8 388 607 signé, 0 à 16 777 215 non signé.
INTEGER[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 4 octets) de -2 147 483 648 à 2 147 483 647 signé, 0 à 4 294 967 295 non signé.
BIGINT[(n)] [UNSIGNED] [ZEROFILL]	Entier (sur 8 octets) de -9 223 372 036 854 775 808 à 9 223 372 036 854 775 807 signé, 0 à 18 446 744 073 709 551 615 non signé.
FLOAT[(n[,p])] [UNSIGNED] [ZEROFILL]	Flottant (de 4 à 8 octets) p désigne la précision simple (jusqu'à 7 décimales) de $-3.4 \cdot 10^{38}$ à $-1.1 \cdot 10^{-38}$, 0, signé, et de $1.1 \cdot 10^{-38}$ à $3.4 \cdot 10^{38}$ non signé.
DOUBLE[(n[,p])] [UNSIGNED] [ZEROFILL]	Flottant (sur 8 octets) p désigne la précision double (jusqu'à 15 décimales) de $-1.7 \cdot 10^{308}$ à $-2.2 \cdot 10^{-308}$, 0, signé, et de $2.2 \cdot 10^{-308}$ à $1.7 \cdot 10^{308}$ non signé.
DECIMAL[(n[,p])] [UNSIGNED] [ZEROFILL]	Décimal à virgule fixe, p désigne la précision (nombre de chiffres après la virgule, maximum 30). Par défaut n vaut 10, p vaut 0.

- La directive `UNSIGNED` permet de considérer seulement des valeurs positives.
- La directive `ZEROFILL` complète par des zéros à gauche une valeur (par exemple : soit une colonne `INTEGER(5)` contenant 4, si `ZEROFILL` est appliquée, la valeur extraite sera « 00004 »). En déclarant une colonne `ZEROFILL`, MySQL l'affecte automatiquement aussi à `UNSIGNED`.

Synonymes et alias



- `INT` est synonyme de `INTEGER`.
- `DOUBLE PRECISION` et `REAL` sont synonymes de `DOUBLE`.
- `DEC NUMERIC` et `FIXED` sont synonymes de `DECIMAL`.
- `SERIAL` est un alias pour `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.
- Dans toute instruction SQL, écrivez la virgule avec un point (7/2 retourne 3.5).

Dates et heures

Les types suivants permettent de stocker des moments ponctuels (dates, dates et heures, années, et heures). Les fonctions `NOW()` et `SYSDATE()` retournent la date et l'heure courante. Dans une procédure ou un déclencheur `SYSDATE` est réévaluée en temps réel, alors que `NOW()` désignera toujours l'instant de début de traitement.

Tableau 1-6 Types de données dates et heures

Type	Description	Commentaire pour une colonne
<code>DATE</code>	Dates du 1 ^{er} janvier de l'an 1000 au 31 décembre 9999 après J.-C.	Sur 3 octets. L'affichage est au format 'YYYY-MM-DD'.
<code>DATETIME</code>	Dates et heures (de 0 h de la première date à 23 h 59 minutes 59 secondes de la dernière date).	Sur 8 octets. L'affichage est au format 'YYYY-MM-DD HH:MM:SS'.
<code>YEAR[(2 4)]</code>	Sur 4 positions : de 1901 à 2155 (incluant 0000). Sur 2 positions : de 70 à 69 (désignant 1970 à 2069).	Sur 1 octet ; l'année est considérée sur 2 ou 4 positions (4 par défaut). Le format d'affichage est 'YYYY'.
<code>TIME</code>	Heures de -838 h 59 minutes 59 secondes à 838 h 59 minutes 59 secondes.	L'heure au format 'HHH:MM:SS' sur 3 octets.
<code>TIMESTAMP</code>	Instants du 1 ^{er} Janvier 1970 0 h 0 minute 0 seconde à l'année 2037.	Estampille sur 4 octets (au format 'YYYY-MM-DD HH:MM:SS') ; mise à jour à chaque modification sur la table.

Données binaires

Les types BLOB (*Binary Large OBject*) permettent de stocker des données non structurées comme le multimédia (images, sons, vidéo, etc.). Les quatre types de colonnes BLOB sont TINYBLOB, BLOB, MEDIUMBLOB et LONGBLOB. Ces types sont traités comme des flots d'octets sans jeu de caractère associé.

Tableau 1-7 Types de données binaires

Type	Description	Commentaire pour une colonne
TINYBLOB(<i>n</i>)		Taille fixe (maximum de 255 octets).
BLOB(<i>n</i>)		Taille fixe (maximum de 65 535 octets).
MEDIUMBLOB(<i>n</i>)	Flot de <i>n</i> octets.	Taille fixe (maximum de 16 mégaoctets).
LONGBLOB(<i>n</i>)		Taille fixe (maximum de 4,29 gigaoctets).

Énumération

Deux types de collections sont proposés par MySQL.

- Le type ENUM définit une liste de valeurs permises (chaînes de caractères).
- Le type SET permettra de comparer une liste à une combinaison de valeurs permises à partir d'un ensemble de référence (chaînes de caractères).

Tableau 1-8 Types de données énumération

Type	Description
ENUM('valeur1', 'valeur2', ...)	Liste de 65 535 valeurs au maximum.
SET('valeur1', 'valeur2', ...)	Ensemble de référence (maximum de 64 valeurs).

Structure d'une table (DESCRIBE)

DESCRIBE (écriture autorisée DESC) est une commande qui vient de SQL*Plus d'Oracle qui a été reprise par MySQL. Elle permet d'extraire la structure brute d'une table ou d'une vue.

DESCRIBE [nomBase.] nomTableouVue [colonne];

Si la base n'est pas indiquée, il s'agit de celle en cours d'utilisation. Retrouvons la structure des tables Compagnie et Pilote précédemment créées. Le type de chaque colonne apparaît :

Tableau 1-9 Structure brute des tables

Résultat						Commentaires
<pre>mysql> DESCRIBE Pilote;</pre>						Les clés primaires sont NOT NULL (désignées par PRI dans la colonne Key).
<pre>+-----+-----+-----+-----+-----+ Field Type Null Key Default Extra +-----+-----+-----+-----+-----+ brevet char(6) NO PRI nom char(15) YES UNI NULL nbHVol double(7,2) YES NULL compa char(4) YES MUL NULL +-----+-----+-----+-----+-----+</pre>						
<pre>mysql> DESCRIBE Compagnie;</pre>						Les unicités sont désignées par UNI dans la colonne Key.
<pre>+-----+-----+-----+-----+-----+ Field Type Null Key Default Extra +-----+-----+-----+-----+-----+ comp char(4) NO PRI nrue int(3) YES NULL rue char(20) YES NULL ville char(15) YES Paris nomComp char(15) NO +-----+-----+-----+-----+-----+</pre>						

Restrictions



Les contraintes CHECK définies ne sont pas encore opérationnelles.

La fraction de seconde du type TIME n'est pas encore pris en charge 'D HH:MM:SS.fraction'.

Les noms des colonnes doivent être uniques pour une table donnée (il est en revanche possible d'utiliser le même nom de colonne dans plusieurs tables).

Les colonnes de type SET sont évaluées par des chaînes de caractères séparés par des « , » ('Airbus, Boeing'). En conséquence aucune valeur d'un SET ne doit contenir le symbole « , ».

Les noms des objets (base, tables, colonnes, contraintes, vues, etc.) ne doivent pas emprunter des mots-clés de MySQL : TABLE, SELECT, INSERT, IF... Si vous êtes « franco-français » ce ne vous gênera pas.

Index

Comme l'index de cet ouvrage vous aide à atteindre les pages concernées par un mot recherché, un index MySQL permet d'accélérer l'accès aux données d'une table. Le but princip

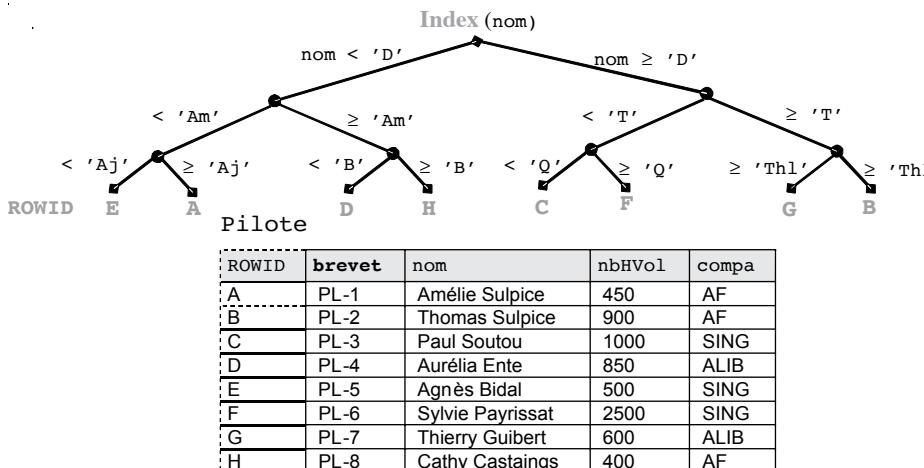
d'un index est d'éviter de parcourir une table séquentiellement du premier enregistrement jusqu'à celui visé (problème rencontré si c'est le Français nommé « Zidane » qu'on recherche dans une table non indexée de plus de soixante-six millions d'enregistrements...). Le principe d'un index est l'association de l'adresse de chaque enregistrement avec la valeur des colonnes indexées.

Sans index et pour n enregistrements, le nombre moyen d'accès nécessaire pour trouver un élément est égal à $n/2$. Avec un index, ce nombre tendra vers $\log(n)$ et augmentera donc de manière plus faible en fonction de la montée en charge des enregistrements. Si une table contient 1 000 enregistrements, alors l'usage d'un index accélérera l'accès d'un facteur 100 par rapport à un accès séquentiel.

Arbres balancés

La figure suivante illustre un index sous la forme d'un arbre. Cet index est basé sur la colonne nom de la table Pilote. Cette figure est caricaturale, car un index n'est pas un arbre binaire (plus de deux liens peuvent partir d'un nœud). Dans cet exemple, trois accès à l'index sont nécessaires pour adresser directement un pilote via son nom, au lieu d'en analyser huit au moins.

Figure 1-3 Index sur la colonne nom



Un index est associé à une table et peut être défini sur une ou plusieurs colonnes (on parle alors de « colonnes indexées »). Une table peut « héberger » plusieurs index. Ils sont mis à jour automatiquement après rafraîchissement de la table (ajouts et suppressions d'enregistrements ou modifications des colonnes indexées). Un index peut être déclaré unique si on sait que les valeurs dans les colonnes indexées seront toujours uniques.

La plupart des index de MySQL (**PRIMARY KEY**, **UNIQUE**, **INDEX**, et **FULLTEXT**) sont stockés dans des arbres équilibrés (*balanced trees* : *B-trees*). D'autres types d'index existent : ceux qui portent sur des colonnes **SPATIAL** (*reverse key* : *R-trees*), et ceux appliqués aux tables **MEMORY** (tables de hachage : *hash*).

La particularité des index *B-tree* est qu'ils conservent en permanence une arborescence symétrique (balancée). Toutes les feuilles sont à la même profondeur. Le temps de recherche est ainsi à peu près constant quel que soit l'enregistrement cherché. Le plus bas niveau de l'index (*leaf blocks*) contient les valeurs des colonnes indexées et le *rowid*. Toutes les feuilles de l'index sont chaînées entre elles. Pour les index non uniques (par exemple si on voulait définir un index sur la colonne **compa** de la table **Pilote**) le *rowid* est inclus dans la valeur de la colonne indexée. Ces index, premiers apparus, sont désormais très fiables et performants, ils ne se dégradent pas lors de la montée en charge de la table.

Création d'un index [CREATE INDEX]

Pour pouvoir créer un index dans sa base, la table à indexer doit appartenir à la base. Si l'utilisateur a le privilège **INDEX**, il peut créer et supprimer des index dans sa base. Un index est créé par l'instruction **CREATE INDEX** et supprimé par **DROP INDEX**.

La syntaxe de création d'un index est la suivante :

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX nomIndex  
[USING BTREE | HASH]  
ON nomTable (colonne1 [(taille1)] [ASC | DESC],...);
```

- **UNIQUE** permet de créer un index qui n'accepte pas les doublons.
- **FULLTEXT** permet de bénéficier de fonctions de recherche dans des textes (fot de caractères).
- **SPATIAL** permet de profiter de fonctions pour les données géographiques.
- **ASC** et **DESC** précisent l'ordre (croissant ou décroissant).

Créons deux index sur la table **Pilote**.

Tableau 1-10 Créations d'index

Instruction SQL	Commentaires
<pre>CREATE UNIQUE INDEX idx_Pilote_nom3 USING BTREE ON Pilote (nom(3) DESC);</pre>	Index <i>B-tree</i> , ordre décroissant sur les trois premiers caractères du nom des pilotes.
<pre>CREATE INDEX idx_Pilote_compa USING BTREE ON Pilote (compa);</pre>	Index <i>B-tree</i> , ordre croissant sur la colonne clé étrangère compa .



- Un index ralentit les rafraîchissements de la base (conséquence de la mise à jour de l’index ou des *bitmaps*). En revanche il accélère les accès.
- Il est conseillé de créer des index sur des colonnes (majoritairement des clés étrangères) utilisées dans les clauses de jointures (voir chapitre 4).
- Il est possible de créer des index pour toutes les colonnes d’une table (jusqu’à concurrence de 16).
- Les index sont pénalisants lorsqu’ils sont définis sur des colonnes très souvent modifiées si la table contient peu de lignes.

Destruction d’un schéma

Il vous sera utile d’écrire un script de destruction d’un schéma (j’entends « schéma » comme ensemble de tables, contraintes et index composant une base de données et non pas en tant qu’ensemble de tous les objets d’un utilisateur) pour pouvoir recréer une base propre. À ce niveau, si des données sont déjà présentes dans les tables, et que vous souhaitez les garder, il faudra utiliser une stratégie pour les réimporter dans les nouvelles tables. À ce niveau, l’ouvrage, vous n’en êtes pas là, et le script de destruction va vous permettre de corriger les erreurs de syntaxe du script de création des tables.

Nous avons vu qu’il fallait créer d’abord les tables « pères » puis les tables « fils » (les contraintes sont définies en même temps que les tables). L’ordre de destruction des tables, pour des raisons de cohérence, est inverse (il faut détruire les tables « fils » puis les tables « pères »). Dans notre exemple, il serait malvenu de supprimer la table `Compagnie` avant la table `Pilote`. En effet la clé étrangère `compa` n’aurait plus de sens.

Suppression d’une table (DROP TABLE)

Pour pouvoir supprimer une table dans une base, il faut posséder le privilège `DROP` sur la base. L’instruction `DROP TABLE` entraîne la suppression des données, de la structure, de la description dans le dictionnaire des données, des index, des déclencheurs associés (*triggers*) et la récupération de la place dans l’espace de stockage.

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    [nomBase.] nomTable1 [, [nomBase2.] nomTable2, ...]
    [RESTRICT | CASCADE]
```

- **TEMPORARY** : pour supprimer des tables temporaires. Les transactions en cours ne sont pas affectées. L'utilisation de **TEMPORARY** peut être un bon moyen de s'assurer qu'on n'a détruit pas accidentellement une table non temporaire...
- **IF EXISTS** : permet d'éviter qu'une erreur se produise si la table n'existe pas.
- **RESTRICT** et **CASCADE** ne sont pas encore opérationnels. Le premier permettra de vérifier qu'aucun autre élément n'utilise la table (vue, déclencheur, etc.). Le second répercute la destruction à tous les éléments référencés.

Les éléments qui utilisaient la table (vues, synonymes, fonctions et procédures) ne sont pas supprimés mais sont temporairement inopérants. Attention, une suppression ne peut pas être par la suite annulée.

Ordre des suppressions



Il suffit de relire à l'envers le script de création de vos tables pour en déduire l'ordre de suppression à écrire dans le script de destruction de votre schéma.

Le tableau suivant présente deux écritures possibles pour détruire des schémas. La première écriture n'est pas encore possible et il reste à voir comment, d'un point de vue des performances, MySQL programmera le **CASCADE**.

Tableau 1-11 Scripts équivalents de destruction

Avec CASCADE (pas encore opérationnel)	Les « fils » puis les « pères »
DROP TABLE Compagnie CASCADE; DROP TABLE Pilote;	DROP TABLE Pilote; DROP TABLE Compagnie;

L'objectif de ces exercices est de créer des tables, leur clé primaire et des contraintes de déclinaison (NOT NULL et CHECK).

Exercice

1.1 Présentation de la base de données

Une entreprise désire gérer son parc informatique à l'aide d'une base de données. Le bâtiment est composé de trois étages. Chaque étage possède son réseau (ou segment distinct) éthernet. Les réseaux traversent des salles équipées de postes de travail. Un poste de travail est une machine sur laquelle sont installés certains logiciels. Quatre catégories de postes de travail sont reconnues (stations Unix, terminaux X, PC Windows et PC NT). La base de données devra aussi décrire les installations de logiciels.

Les noms et types des colonnes sont les suivants :

Tableau 1-12 Caractéristiques des colonnes

Colonne	Commentaire	Type
indIP	trois premiers groupes IP (exemple : 130.120.80)	VARCHAR(15)
nomSegment	nom du segment	VARCHAR(20)
etage	étage du segment	TINYINT(1)
nSalle	numéro de la salle	VARCHAR(7)
nomSalle	nom de la salle	VARCHAR(20)
nbPoste	nombre de postes de travail dans la salle	TINYINT(2)
nPoste	code du poste de travail	VARCHAR(7)
nomPoste	nom du poste de travail	VARCHAR(20)
ad	dernier groupe de chiffres IP (exemple : 11)	VARCHAR(3)
typePoste	type du poste (UNIX, TX, PCWS, PCNT)	VARCHAR(9)
dateIns	date d'installation du logiciel sur le poste	dateTime
nLog	code du logiciel	VARCHAR(5)
nomLog	nom du logiciel	VARCHAR(20)
dateAch	date d'achat du logiciel	dateTime
version	version du logiciel	VARCHAR(7)
typeLog	type du logiciel (UNIX, TX, PCWS, PCNT)	VARCHAR(9)
prix	prix du logiciel	DECIMAL(6)
numIns	numéro séquentiel des installations	INTEGER(5)
dateIns	date d'installation du logiciel	TIMESTAMP
delai	intervalle entre achat et installation	SMALLINT
typeLP	types des logiciels et des postes	VARCHAR(9)
nomType	noms des types (Terminaux X, PC Windows...)	VARCHAR(20)

Exercice 1.2 Création des tables

Écrire puis exécuter le script SQL (que vous appellerez `creParc.sql`) de création des tables avec leur clé primaire (en gras dans le schéma suivant) et les contraintes suivantes :

- Les noms des segments, des salles et des postes sont non nuls.
- Le domaine de valeurs de la colonne `ad` s'étend de 0 à 255.
- La colonne `prix` est supérieure ou égale à 0.
- La colonne `dateIns` est égale à la date du jour par défaut.

Figure 1-4 Composition des tables

Segment

indIP	nomSegment	etage
--------------	------------	-------

Salle

nSalle	nomSalle	nbPoste	indIP
---------------	----------	---------	-------

Poste

nPoste	nomPoste	indIP	ad	typePoste	nSalle
---------------	----------	-------	----	-----------	--------

Logiciel

nLog	nomLog	dateAch	version	typeLog	prix
-------------	--------	---------	---------	---------	------

Installer

nPoste	nLog	numIns	dateIns	delai
---------------	-------------	---------------	----------------	-------

Types

typeLP	nomType
---------------	---------

Exercice 1.3 Structure des tables

Écrire puis exécuter le script SQL (que vous appellerez `descParc.sql`) qui affiche la description de toutes ces tables (en utilisant des commandes `DESCRIBE`). Comparer le résultat obtenu avec le schéma ci-dessus.

Exercice 1.4 Destruction des tables

Écrire puis exécuter le script SQL de destruction des tables (que vous appellerez `dropParc.sql`). Lancer ce script puis celui de la création des tables à nouveau.

Manipulation des données

Ce chapitre décrit l'aspect LMD (langage de manipulation des données) de MySQL. Verrons que SQL propose trois instructions pour manipuler des données :

- l'insertion d'enregistrements : `INSERT` ;
- la modification de données : `UPDATE` ;
- la suppression d'enregistrements : `DELETE` (et `TRUNCATE`).

Il existe d'autres possibilités que nous ne détaillerons pas dans ce chapitre, pour insérer des données en utilisant des outils d'importation ou de migration, tels MySQL Migration Kit, SQLPorter, Navicat, Intelligent Converters et MySQL Data Import de la société EM

Insertions d'enregistrements (`INSERT`)

Pour pouvoir insérer des enregistrements dans une table, il faut que vous ayez reçu le privilège `INSERT`. Il existe plusieurs possibilités d'insertion : l'insertion monoligne qui ajoute un enregistrement par instruction (que nous allons détailler maintenant) et l'insertion multiligne qui insère plusieurs enregistrements par une requête (que nous détaillerons au chapitre 4).

Syntaxe

La syntaxe simplifiée de l'instruction `INSERT` monoligne est la suivante :

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
       [INTO] [nomBase.] { nomTable | nomVue } [(nomColonne,...)]
       VALUES ({expression | DEFAULT},...),(...),...
       [ON DUPLICATE KEY UPDATE nomColonne = expression,...]
```

- `DELAYED` indique que l'insertion est différée (si la table est modifiée par ailleurs, le serveur attend qu'elle se libère pour y insérer périodiquement de nouveaux enregistrements si elle redevient active entre-temps).
- `LOW_PRIORITY` indique que l'insertion est différée à la libération complète de la table (option à ne pas utiliser sur des tables `MYISAM`).
- `HIGH_PRIORITY` annule l'option *low priority* du serveur.
- `IGNORE` indique que les éventuelles erreurs déclenchées suite à l'insertion seront ignorées en tant que *warnings*.

- ON DUPLICATE KEY UPDATE permet de mettre à jour l'enregistrement présent dans la table, qui a déclenché l'erreur de doublon (dans le cas d'un index UNIQUE ou d'une clé primaire). Dans ce cas le nouvel enregistrement n'est pas inséré, seul l'ancien est mis à jour.

À l'aide d'exemples, nous allons détailler les possibilités de cette instruction en considérant la majeure partie des types de données proposés par MySQL.

Renseigner toutes les colonnes

Ajoutons trois lignes dans la table `Compagnie` en alimentant toutes les colonnes de la table par des valeurs. La deuxième insertion utilise le mot-clé `DEFAULT` pour affecter explicitement la valeur par défaut à la colonne `ville`. La troisième insertion attribue explicitement la valeur `NULL` à la colonne `nrue`.

Tableau 2-1 Insertions

Web	Instruction SQL	Commentaires
	<pre>INSERT INTO Compagnie VALUES ('SING', 7, 'Camparols', 'Singapour', 'Singapore AL');</pre>	Toutes les valeurs sont renseignées dans l'ordre de la structure de la table.
	<pre>INSERT INTO Compagnie VALUES ('AC', 10, 'Gambetta', DEFAULT, 'Air France');</pre>	DEFAULT explicite.
	<pre>INSERT INTO Compagnie VALUES ('AN1', NULL, 'Hoche', 'Blagnac', 'Air Null');</pre>	NULL explicite.

Renseigner certaines colonnes

Insérons deux lignes dans la table `Compagnie` en ne précisant pas toutes les colonnes. La première insertion affecte implicitement la valeur par défaut à la colonne `ville`. La deuxième donne implicitement la valeur `NULL` à la colonne `nrue`.

Tableau 2-2 Insertions de certaines colonnes

Web	Instruction SQL	Commentaires
	<pre>INSERT INTO Compagnie(comp, nrue, rue, nomComp) VALUES ('AF', 8, 'Champs Elysées', 'Castanet Air');</pre>	DEFAULT implicite.
	<pre>INSERT INTO Compagnie(comp, rue, ville, nomComp) VALUES ('AN2', 'Foch', 'Blagnac', 'Air Nul2');</pre>	NULL sur nrue implicite.

La table `Compagnie` contient à présent les lignes suivantes :

Figure 2-1 Table après les insertions

Compagnie		Valeur NULL	Valeur par défaut	
comp	nrue	rue	ville	nomComp
SING	7	Camparols	Singapour	Singapore AL
AF	10	Gambetta	Paris	Air France
AN1		Hoche	Blagnac	Air Nul1
AC	8	Champs Elysées	Paris	Castanet Air
AN2		Foch	Blagnac	Air Nul2

Plusieurs enregistrements

Le script suivant ajoute trois nouvelles compagnies en une seule instruction INSERT.

```
INSERT INTO Compagnie VALUES
  ('LUFT',9,'Salas','Munich','Luftansa'),
  ('QUAN',1,'Kangouroo','Sydney','Quantas'),
  ('SNCM',3,'P. Paoli','Bastia','Corse Air');
```

Ne pas respecter des contraintes

Insérons des enregistrements dans la table Pilote, qui ne respectent pas des contraintes. Le tableau suivant décrit les messages renvoyés pour chaque erreur (le nom de la contrainte apparaît dans chaque message, les valeurs erronées sont notées en gras). La première erreur vient de la clé primaire, la seconde de l'unicité du nom. La troisième erreur signifie que la clé étrangère référence une clé primaire absente (ici une compagnie inexistante). Nous reviendrons sur ce dernier problème dans la section *Erreur ! Source du renvoi introuvable*. La dernière concerne la contrainte en ligne NOT

Tableau 2-3 Insertions

Web	Insertions vérifiant les contraintes	Insertions ne vérifiant pas les contraintes
	<pre>INSERT INTO Pilote VALUES ('PL-1', 'Louise Ente', 450, 'AF');</pre>	<pre>mysql> INSERT INTO Pilote VALUES('PL-1', 'Amélie Sulpice', 100, 'AF'); ERROR 1062 (23000): Duplicate entry 'PL-1' for key 1</pre>
	<pre>INSERT INTO Pilote VALUES ('PL-2', 'Jules Ente ', 900, 'AF');</pre>	<pre>mysql> INSERT INTO Pilote VALUES ('PL-4', 'Louise Ente', 450, 'AF'); ERROR 1062 (23000): Duplicate entry 'Louise Ente' for key 2</pre>
	<pre>INSERT INTO Pilote VALUES ('PL-3', 'Paul Soutou', 1000, 'SING');</pre>	<pre>mysql> INSERT INTO Pilote VALUES ('PL-5', 'Thomas Sulpice', 500, 'TOTO'); 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`)) mysql> INSERT INTO Pilote VALUES ('PL-6', NULL, 100, 'AF'); ERROR 1048 (23000): Column 'nom' cannot be null</pre>

Données binaires

Le type **BIT** permet de manipuler des suites variables de bits. Des fonctions sont disponibles pour programmer le « ET », le « OU » exclusif ou inclusif, etc. La table suivante contient deux colonnes de ce type. Notez l'utilisation du préfixe « b » pour initialiser un tel type.

```
CREATE TABLE Registres (nom CHAR(5), numero BIT(2), adresse BIT(16));
INSERT INTO Registres VALUES ('COM2', b'10', b'0000010011110111');
```

Énumérations

Le type **ENUM** est considéré comme une liste de chaînes de caractères. Toute valeur d'une colonne de ce type devra appartenir à cette liste établie lors de la création de la table. Supposons qu'on recense quatre types possibles de diplômes ('BTS', 'DUT', 'Licence' et 'INSA') pour chaque étudiant. On ne stocke qu'un seul diplôme par étudiant.

Figure 2-2 Table avec une colonne **ENUM**

UnCursus			ENUM
num	nom	{diplome }	
E1	F. Brouard	BTS	BTS, DUT, Licence INSA
E2	F. Degrelle	Licence	

Le script de la création de la table et des insertions est le suivant. Notez que les parenthèses sont optionnelles pour désigner la colonne ENUM.

Tableau 2-4 Insertions avec un **ENUM**

Création	Insertions (deux bonnes une illicite)
<pre>CREATE TABLE UnCursus (num CHAR(4), nom CHAR(15), diplome ENUM ('BTS','DUT','Licence','INSA'), CONSTRAINT pk_Cusus PRIMARY KEY(num));</pre>	<pre>mysql> INSERT INTO UnCursus VALUES ('E1', 'F. Brouard', ('BTS')); mysql> INSERT INTO UnCursus VALUES ('E2', 'F. Degrelle', 'Licence'); mysql> INSERT INTO UnCursus VALUES ('E3', 'Bug', ('MathSup')); ERROR 1265 (01000): Data truncated for column 'diplome' at row 1</pre>

Le type **SET** permet de comparer une liste à une combinaison de valeurs permises à partir d'un ensemble de référence (chaînes de caractères). Supposons qu'on désire stocker plusieurs diplômes par étudiant.

Figure 2-3 Table avec une colonne SET

Cursus			SET
num	nom	diplomes	SET
E1	F. Brouard	BTS, Licence	BTS, DUT, Licence INSA
E2	F. Degrelle	Licence, INSA , DUT	

Le script de la création de la table et des insertions est le suivant (même remarque pour les parenthèses).

Tableau 2-5 Insertions avec un SET

Création	Insertions (deux bonnes une illicite)
<pre>CREATE TABLE Cursus (num CHAR(4), nom CHAR(15), diplomes SET ('BTS','DUT','Licence','INSA'), CONSTRAINT pk_Cusus PRIMARY KEY(num));</pre>	<pre>mysql> INSERT INTO Cursus VALUES ('E1', 'F. Brouard',('BTS','Licence')); mysql> INSERT INTO Cursus VALUES ('E2', 'F. Degrelle','Licence,INSA,DUT');</pre>
	<pre>mysql> INSERT INTO Cursus VALUES ('E3', 'Bug', ('BTS,INSA,ENAC')); ERROR 1265 (01000): Data truncated column 'diplomes' at row 1</pre>

Dates et heures

Nous avons décrit au chapitre 1 les caractéristiques générales des types MySQL pour stocker des éléments de type date/heure. Étudions maintenant la manipulation de ces types. Verrons que MySQL peut les considérer soit en tant que chaînes de caractères soit comme nombres numériques.

Formats

Concernant les types DATETIME, DATE et TIMESTAMP les formats possibles sont les suivants :

- Chaînes de caractères 'YYYY-MM-DD HH:MM:SS' ou 'YY-MM-DD HH:MM:SS' (pour les colonnes DATE 'YYYY-MM-DD' ou 'YY-MM-DD'). Tout autre délimiteur est autorisé comme '2005.12.31 11%30%45' (pour les colonnes DATE, '65.12.31', '65/12/31', '65@12@31' sont équivalents et désignent tous le réveillon de l'année 1965).
- Considérés comme chaînes de caractères dans les formats suivants : 'YYYYMMDD HHMMSS' ou 'YYMMDDHHMMSS' (pour les DATE 'YYYYMMDD' ou 'YYMMDD') supposant que la chaîne ait un sens en tant que date. Ainsi '19650205063000' est interprété comme le 5 février 1965 à 6 heures et 30 minutes. Par contre '19650255' n'a pas de sens du fait du jour (il sera interprété comme '0000-00-00').

- Considérés comme numériques dans les formats suivants : YYYYMMDDHHMMSS ou YYMMDDHHMMSS (pour les DATE YYYYMMDD ou YYMMDD) en supposant que le nombre a un sens en tant que date. Ainsi 19650205063000 est interprété comme le 5 février 1965 à 6 heures et 30 minutes. Par contre 19650205069000 n'a pas de sens du fait des minutes (qui sera interprété comme 00000000000000).

Exemple avec DATE et DATETIME

Déclarons la table Pilote qui contient deux colonnes de type date/heure : DATE et DATETIME

```
CREATE TABLE Pilote
(brevet VARCHAR(6), nom VARCHAR(20), dateNaiss DATE,
 nbHVol DECIMAL(7,2), dateEmbauche DATETIME, compa VARCHAR(4),
 CONSTRAINT pk_Pilote PRIMARY KEY(brevet));
```

L'insertion du pilote initialise la date de naissance au 5 février 1965, ainsi que celle de l'embauche à la date du moment (heures, minutes, secondes) par la fonction SYSDATE.

```
INSERT INTO Pilote
VALUES ('PL-1', 'Christian Soutou', '1965-02-05', 900, SYSDATE(), 'AF')
```

Nous verrons au chapitre 4 comment extraire les années, mois, jours, heures, minutes et secondes. Nous verrons aussi qu'il est possible d'ajouter ou de soustraire des dates entre elles.

Exemple avec TIME et YEAR

Par analogie aux différents formats des dates, les heures (type TIME 'HH:MM:SS' ou 'HHH:MM:SS') peuvent aussi être manipulées sous la forme de chaînes ou de nombres.

- Chaîne 'D HH:MM:SS.fraction' avec le nombre de jours (0 à 34) et la fraction d'une seconde (pas encore opérationnelle), 'HH:MM:SS.fraction', 'HH:MM:SS', 'HH:MM:SS', 'D HH:MM:SS', 'D HH:MM', 'D HH', ou 'SS'.
- Chaîne sans les délimiteurs sous réserve que la chaîne ait un sens. Ainsi, '101112' est considéré comme '10:11:12', mais '109712' est incorrect du fait des minutes, il devient '00:00:00'.
- Nombre en raisonnant comme les chaînes. Ainsi 101112 est considéré comme '10:11:12', mais 109712 est incorrect du fait des minutes, il devient '00:00:00'. Les formats suivants sont aussi corrects : SS, MMSS, HHMMSS.

La table Pilote suivante contient la colonne pasVolDepuis pour stocker le délai depuis dernier vol et la colonne retraite qui indique l'année de retraite.

```
CREATE TABLE Pilote
(brevet VARCHAR(6), nom VARCHAR(20), pasVolDepuis TIME, retraite YEAR,
CONSTRAINT pk_Pilote PRIMARY KEY(brevet));
```

Les insertions suivantes utilisent différents formats. Le premier pilote n'a pas volé depuis 1 jour et 23 heures (47 heures) ; le second depuis 15 heures, 26 minutes 30 secondes ; le troisième depuis 4 jours et 23 heures (119 heures) ; le quatrième depuis 3 heures 27 minutes 50 secondes, et le dernier se croit plus précis que tous les autres en ajoutant une fraction qui sera pas prise en compte au niveau de la base.

```
INSERT INTO Pilote VALUES ('PL-1', 'Hait', '1 23:0:0', '2002');
INSERT INTO Pilote VALUES ('PL-2', 'Crampes', '152630', 2006);
INSERT INTO Pilote VALUES ('PL-3', 'Tuffery', '4 23:00', 05);
INSERT INTO Pilote VALUES ('PL-4', 'Mercier', 032750, '07');
INSERT INTO Pilote VALUES ('PL-5', 'Albaric', '1 23:0:0.457', '01')
```

L'état de la base est le suivant :

```
mysql> SELECT * FROM Pilote;
+-----+-----+-----+-----+
| brevet | nom   | pasVolDepuis | retraite |
+-----+-----+-----+-----+
| PL-1  | Hait  | 47:00:00    | 2002    |
| PL-2  | Crampes | 15:26:30  | 2006    |
| PL-3  | Tuffery | 119:00:00 | 2005    |
| PL-4  | Mercier | 03:27:50  | 2007    |
| PL-5  | Albaric | 47:00:00  | 2001    |
+-----+-----+-----+-----+
```



Si un dépassement se produit au niveau d'une colonne TIME, celle-ci est évaluée au maximum ou au minimum ('-850:00:00' et '999:00:00' sont respectivement convertis à '-838:59:59' et '838:59:59').

Exemple avec TIMESTAMP

Toute colonne du type TIMESTAMP est actualisée à chaque modification de l'enregistrement la première fois à l'INSERT, puis à chaque UPDATE (quelle que soit la colonne mise à jour). Déclarons la table Pilote qui contient une colonne de ce type.

```
CREATE TABLE Pilote
(brevet VARCHAR(6), nom VARCHAR(20), misaJour TIMESTAMP,
CONSTRAINT pk_Pilote PRIMARY KEY(brevet));
```

L'insertion du pilote suivant initialisera la colonne à la date système (comme SYSDATE)

```
INSERT INTO Pilote (brevet,nom) VALUES ('PL-1', 'Hait');
```

Par la suite, et à la différence d'un type DATE ou DATETIME, pour chaque modification d'un pilote, la colonne misaJour sera réactualisée avec la date de l'instant de la mise à jour.

Même si vous croyez mettre à NULL cette colonne avec une instruction UPDATE, elle contiendra toujours l'instant de votre vaine tentative !

Fonctions utiles

Les fonctions CURRENT_TIMESTAMP(), CURRENT_DATE() et CURRENT_TIME(), UTC TIME(), renseignent sur l'instant, la date, l'heure et l'heure GMT de la session en cours.

Il n'est pas nécessaire d'utiliser une table pour afficher une expression dans l'interface de commande. L'exemple suivant montre que la requête a été exécutée le 1^{er} novembre 2005 à 10 heures, 11 minutes et 27 secondes. Le client est sur le fuseau GMT+1h.

```
mysql> SELECT CURRENT_TIMESTAMP(), CURRENT_TIME(), CURRENT_DATE(), UTC_TIME();
+-----+-----+-----+-----+
| CURRENT_TIMESTAMP() | CURRENT_TIME() | CURRENT_DATE() | UTC_TIME() |
+-----+-----+-----+-----+
| 2005-11-01 10:11:27 | 10:11:27      | 2005-11-01     | 09:11:27    |
+-----+-----+-----+-----+
```

Séquences

Bien que « séquence » ne soit pas dans le vocabulaire de MySQL, car le mécanisme qu'il propose n'est pas aussi puissant que celui d'Oracle, MySQL offre la possibilité de générer automatiquement des valeurs numériques. Ces valeurs sont bien utiles pour composer des clés primaires de tables quand vous ne disposez pas de colonnes adéquates à cet effet. Ce mécanisme répond en grande partie à ce qu'on attendrait d'une séquence.

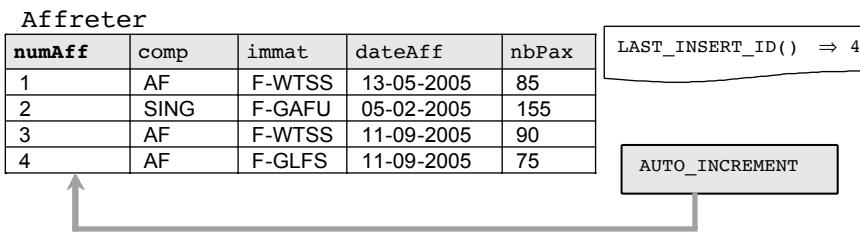


En attendant que MySQL offre peut-être dans une prochaine version un mécanisme plus riche que celui que nous allons étudier, je me permets d'appeler « séquence » une colonne indexée de type entier (INTEGER, SMALLINT, TINYINT, MEDIUMINT et BIGINT), définie à l'aide de la directive AUTO_INCREMENT. Cette colonne est clé primaire, ou unique et non nulle. Une séquence est en général affectée à une table, mais vous pouvez l'utiliser pour plusieurs tables ou variables.

Utilisation en tant que clé primaire

La figure suivante illustre la séquence appliquée à la colonne numAff pour initialiser les valeurs de la clé primaire de la table Affreter. La fonction LAST_INSERT_ID() retourne la dernière valeur de la séquence générée (ici le pas est de 1, la séquence débute par défaut à 1, nous verrons par la suite qu'il est possible d'utiliser une valeur différente).

Figure 2-4 Séquence appliquée à une clé primaire



Le tableau suivant décrit la création de cette table et différentes écritures pour les insertions.

Tableau 2-6 Séquence pour une clé primaire

Table	Insertions
CREATE TABLE Affreter (numAff SMALLINT AUTO_INCREMENT , comp CHAR(4), immat CHAR(6), dateAff DATE, nbPax SMALLINT(3), CONSTRAINT pk_Affreter PRIMARY KEY (numAff));	INSERT INTO Affreter (comp,immat,dateAff,nbPax) VALUES ('AF','F-WTSS','2005-05-13',85); INSERT INTO Affreter (comp,immat,dateAff,nbPax) VALUES ('SING','F-GAFU','2005-02-05',155); INSERT INTO Affreter VALUES (NULL,'AF','F-WTSS','2005-09-11',90); INSERT INTO Affreter VALUES (0,'AF','F-GLFS','2005-09-11',75);

Modification d'une séquence

La seule modification possible d'une séquence est celle qui consiste à changer la valeur de départ de la séquence (avec `ALTER TABLE`). Seules les valeurs à venir de la séquence modifiée seront changées (heureusement pour les données existantes des tables).

Supposons qu'on désire continuer à insérer des nouveaux affrètements à partir d'une valeur 100. Le prochain affrètement sera estimé à 100 et les insertions suivantes prendront en compte le nouveau point de départ tout en laissant intactes les données existantes des tables.

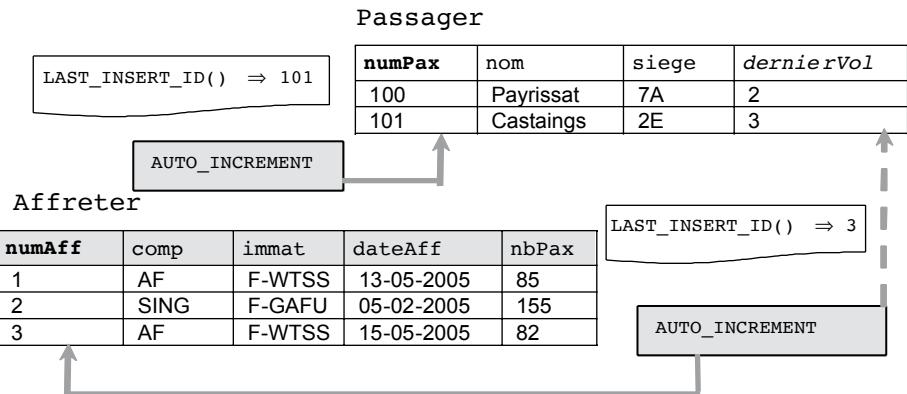
```
ALTER TABLE Affreter AUTO_INCREMENT = 100;  
INSERT INTO Affreter (comp,immat,dateAff,nbPax)  
VALUES ('SING', 'F-NEW', SYSDATE(), 77);  
mysql> SELECT * FROM Affreter ;
```

numAff	comp	immat	dateAff	nbPax
1	AF	F-WTSS	2005-05-13	85
2	SING	F-GAFU	2005-02-05	155
3	AF	F-WTSS	2005-09-11	90
4	AF	F-GLFS	2005-09-11	75
100	SING	F-NEW	2005-11-01	77

Utilisation en tant que clé étrangère

Créons deux séquences qui vont permettre de donner leur valeur aux clés primaires des deux tables illustrées à la figure suivante (les affrètements commencent à 1, les passagers à 100). Servons-nous aussi de la séquence de la table `Affreter` pour indiquer le dernier vol de chaque passager. La section *Intégrité référentielle* détaille les mécanismes relatifs aux clés étrangères.

Figure 2-5 Séquence appliquée à une clé étrangère



Le script SQL de définition et de manipulation des données est indiqué ci-après. La valeur de départ d'une séquence peut être définie à la fin de l'ordre `CREATE TABLE`. Notez également l'utilisation de la fonction `LAST_INSERT_ID` dans les insertions pour récupérer la valeur de la clé primaire.

Tableau 2-7 Séquence pour une clé étrangère

Tables	Insertions
<pre>CREATE TABLE Affreter (numAff SMALLINT AUTO_INCREMENT, comp CHAR(4), immat CHAR(6), dateAff DATE, nbPax SMALLINT(3), CONSTRAINT pk_Affreter PRIMARY KEY (numAff));</pre> <pre>CREATE TABLE Passager (numPax SMALLINT AUTO_INCREMENT, nom CHAR(15), siege CHAR(4), dernierVol SMALLINT, CONSTRAINT pk_Passager PRIMARY KEY(numPax), CONSTRAINT fk_Pax_vol_Affreter FOREIGN KEY (dernierVol) REFERENCES Affreter(numAff)) AUTO_INCREMENT = 100;</pre>	<pre>INSERT INTO Affreter (comp,immat,dateAff,nbPax) VALUES ('AF','F-WTSS','2005-05-13',85);</pre> <pre>INSERT INTO Affreter (comp,immat,dateAff,nbPax) VALUES ('SING','F-GAFU','2005-02-05',155);</pre> <pre>INSERT INTO Passager VALUES (NULL,'Payrissat','7A',LAST_INSERT_ID());</pre> <pre>INSERT INTO Affreter VALUES (NULL,'AF','F-WTSS','2005-05-15',82);</pre> <pre>INSERT INTO Passager VALUES (NULL,'Castaings','2E',LAST_INSERT_ID());</pre>

Modifications de colonnes

L'instruction UPDATE permet la mise à jour des colonnes d'une table. Pour pouvoir modifier des enregistrements d'une table, il faut que cette dernière soit dans votre base ou que vous ayez reçu le privilège UPDATE sur la table.

Syntaxe (UPDATE)

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

```
UPDATE [LOW_PRIORITY] [IGNORE] [nomBase.] nomTable
SET col_name1=expr1 [, col_name2=expr2 ...]
    SET      colonne1 = expression1 | (requête_SELECT) | DEFAULT
            [,colonne2 = expression2...]
    [WHERE (condition)]
    [ORDER BY listeColonnes]
    [LIMIT nbreLimite]
```

- **LOW_PRIORITY** indique que la modification est différée à la libération complète de la table (option à ne pas utiliser sur des tables MyISAM).
- **IGNORE** signifie que les éventuelles erreurs déclenchées suite aux modifications sont considérées en tant que *warnings*.
- La clause **SET** actualise une colonne en lui affectant une expression (valeur, valeur par défaut, calcul ou résultat d'une requête).

- La condition du WHERE filtre les lignes à mettre à jour dans la table. Si aucune condition n'est précisée, tous les enregistrements seront actualisés. Si la condition ne filtre aucune ligne, aucune mise à jour ne sera réalisée.
- ORDER BY indique l'ordre de modification des colonnes.
- LIMIT spécifie le nombre maximum d'enregistrements à changer (par ordre de clé primaire croissante).

Modification d'une colonne

Modifions la compagnie de code 'AN1' en affectant la valeur 50 à la colonne nrue.

```
UPDATE Compagnie SET nrue = 50 WHERE comp = 'AN1';
```

Modification de plusieurs colonnes

Modifions la compagnie de code 'AN2' en affectant simultanément la valeur 14 à la colonne nrue et la valeur par défaut ('Paris') à la colonne ville.

```
UPDATE Compagnie SET nrue = 14, ville = DEFAULT WHERE comp = 'AN2';
```

La table Compagnie contient à présent les données suivantes :

Figure 2-6 Table après les modifications

Compagnie		Modification 1		
comp	nrue	rue	ville	nomComp
SING	7	Camparols	Singapour	Singapore AL
AF	10	Gambetta	Paris	Air France
AN1	50	Hoche	Blagnac	Air Nul1
AC	8	Champs Elysées	Paris	Castanet Air
AN2	14	Foch	Paris	Air Nul2

Modifications 2

Modification de plusieurs enregistrements

Modifions les deux premières compagnies (par ordre de clé primaire, ici 'AC' et 'AF') en affectant la valeur 'Toulouse' à la colonne ville.

```
UPDATE Compagnie SET ville = 'Toulouse' LIMIT 2;
```

Ne pas respecter les contraintes

Il faut, comme pour les insertions, respecter les contraintes qui existent au niveau des colonnes. Dans le cas inverse, une erreur est renvoyée (le nom de la contrainte apparaît) et la mise à jour n'est pas effectuée.

Tableau 2-8 Table, données et contraintes

Web	Données	Table et contraintes																
	<p>Figure 2-7 Données</p> <p>Pilote</p> <table border="1"> <thead> <tr> <th>brevet</th><th>nom</th><th>nbHVol</th><th>compa</th></tr> </thead> <tbody> <tr> <td>PL-1</td><td>Louise Ente</td><td>450</td><td>AF</td></tr> <tr> <td>PL-2</td><td>Jules Ente</td><td>900</td><td>AF</td></tr> <tr> <td>PL-3</td><td>Paul Soutou</td><td>1000</td><td>SING</td></tr> </tbody> </table>	brevet	nom	nbHVol	compa	PL-1	Louise Ente	450	AF	PL-2	Jules Ente	900	AF	PL-3	Paul Soutou	1000	SING	<pre>CREATE TABLE Pilote (brevet CHAR(6), nom CHAR(15) NOT NULL, nbHVol DECIMAL(7,2), compa CHAR(4)) CONSTRAINT pk_Pilote PRIMARY KEY(brevet), CONSTRAINT ck_nbHVol CHECK (nbHVol BETWEEN 0 AND 2000) CONSTRAINT un_nom UNIQUE(nom), CONSTRAINT fk_Pil_compa_Comp FOREIGN KEY (compa) REFERENCES Compagnie(comp));</pre>
brevet	nom	nbHVol	compa															
PL-1	Louise Ente	450	AF															
PL-2	Jules Ente	900	AF															
PL-3	Paul Soutou	1000	SING															

À partir de la table Pilote, le tableau suivant décrit des modifications (certaines ne vérifient pas de contraintes). La mise à jour d'une clé étrangère est possible si elle n'est pas référencée par une clé primaire (voir la section *Intégrité référentielle*).

Tableau 2-9 Modifications

Web	Vérifiant les contraintes (sauf une !)	Ne vérifiant pas les contraintes
	<pre>--Modification d'une clé étrangère UPDATE Pilote SET compa = 'SING' WHERE brevet = 'PL-2'; -- Modification d'une clé primaire UPDATE Pilote SET brevet = 'PL3bis' WHERE brevet = 'PL-3'; --Passe outre la contrainte CHECK ! UPDATE Pilote SET nbHVol= 30000 WHERE brevet = 'PL-1';</pre>	<pre>mysql> UPDATE Pilote SET brevet='PL3bis' WHERE brevet='PL-1'; ERROR 1062 (23000): Duplicate entry 'PL-2' for key 1 mysql> UPDATE Pilote SET nom = NULL WHERE brevet = 'PL-1'; ERROR 1263 (22004): Column set to its default value; NULL supplied to a NULL column 'nom' at row 1 mysql> UPDATE Pilote SET nom='Paul Soutou' WHERE brevet = 'PL-1'; ERROR 1062 (23000): Duplicate entry 'Paul Soutou' for key 2 mysql> UPDATE Pilote SET compa='TOM' WHERE brevet = 'PL-1'; ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails (`bdsoutou/pilote` CONSTRAINT `fk_Pil_compa_Comp` FORE KEY (`compa`) REFERENCES `compagnie` (`comp`))</pre>



Pour l'heure, il n'est pas possible de modifier une table en utilisant une requête (dans la clause SET) portant sur cette même table.

Dates et intervalles

Le tableau suivant résume les principales opérations possibles entre des colonnes de type date-heure.

Tableau 2-10 Opérations entre colonnes date-heure

Opérande 1	Opérateur	Opérande 2	Résultat
DATE DATETIME	+ ou -	Interval	DATE DATETIME
DATE DATETIME	+ ou -	INTEGER	DATE DATETIME
TIME	+ ou -	TIME	TIME
TIME	+ ou -	INTEGER	TIME



Considérons la table suivante :

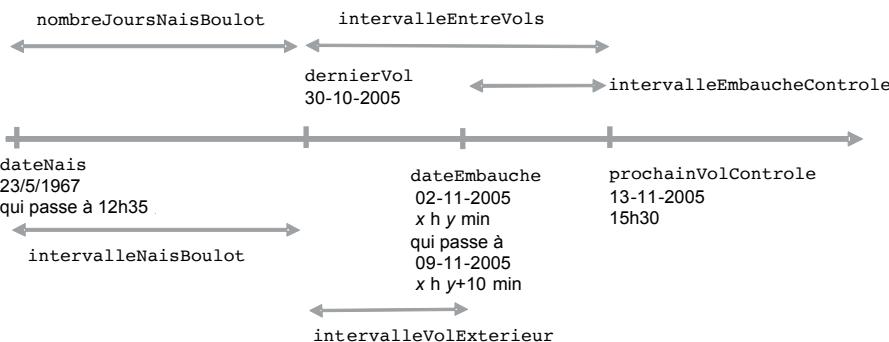
```
CREATE TABLE Pilote
(brevet VARCHAR(6), nom VARCHAR(20), dateNaiss DATETIME, dernierVol DATE
dateEmbauche DATETIME, prochainVolControle DATETIME,
nombreJoursNaissanceBoulot INTEGER(5),
intervalleNaissanceBoulot Decimal (20,6), intervalleVolExterieur
Decimal (10,6),
intervalleEntreVols Decimal (10,6), intervalleEmbaucheControle
TIME,
compa VARCHAR(4), CONSTRAINT pk_Pilote PRIMARY KEY(brevet));
```

À l'insertion du pilote 'Thierry Albaric' de la compagnie de code 'AF', initialisons sa date de naissance (25 mars 1967), la date de son dernier vol (30 octobre 2005), sa date d'embauche (1er celle du jour) et la date de son prochain contrôle en vol (13 novembre 2005, 15h30).

```
INSERT INTO Pilote VALUES
('PL-1', 'Thierry Albaric', '1967-03-25', '2005-10-30', SYSDATE(),
'2005-11-13 15:30:00', NULL, NULL, NULL, NULL, 'AF');
```

Les mises à jour par UPDATE sur cet enregistrement vont consister, sur la base de ces quatre dates, à calculer les intervalles illustrés à la figure suivante :

Figure 2-9 Intervalles à calculer



Modification d'une heure

On modifie une date en précisant une heure via la fonction en rajoutant le format 'HH:MM' quels que soient les délimiteurs (ici « : »).

```
UPDATE Pilote SET dateNaiss = '1967-03-25 12:35:00'  
WHERE brevet = 'PL-1';
```

Ajout d'un délai

On modifie la date d'embauche de 10 minutes après la semaine prochaine. L'ajout d'un délai s'opère par la fonction **DATE_ADD** couplée à la directive **DAY_MINUTE** qui permet de spécifier un jour, une heure et une minute. Ainsi la semaine à ajouter correspond au « 7 » paramètre, de même pour les 10 minutes.

```
UPDATE Pilote  
SET dateEmbauche = DATE_ADD(dateEmbauche, INTERVAL '7 0:10' DAY_MINUTE)  
WHERE brevet = 'PL-1';
```

Déférence entre deux dates

La différence entre deux dates peut se programmer à l'aide de la fonction **DATEDIFF** qui renvoie un entier correspondant au nombre de jours séparant les deux dates.

```
UPDATE Pilote  
SET nombreJoursNaisBoulot = DATEDIFF(dateEmbauche,dateNaiss)  
WHERE brevet = 'PL-1';
```

Cette même différence au format d'un intervalle plus précis (nombre de jours décimaux) requiert l'utilisation de la fonction **TIMESTAMPDIFF(interval, datetime1, datetime2)** que nous étudierons au chapitre 4 (il existe bien sûr d'autres possibilités de programmation). La fonction effectue la différence entre deux dates suivant un format d'intervalle donné (elle obtient des secondes qu'on divise par (24*3600) pour convertir en jours).

```

UPDATE Pilote SET
    intervalleNaisBoulot =
        TIMESTAMPDIFF(SECOND,dateNaiss,dateEmbauche)/(24*3600) ,
    intervalleEntreVols =
        TIMESTAMPDIFF(SECOND,dernierVol,prochainVolControle)/(24*3600)
    intervalleVolExterieur =
        TIMESTAMPDIFF(SECOND,dernierVol,dateEmbauche)/(24*3600)
WHERE brevet = 'PL-1';

```

Différence entre deux intervalles

La différence entre deux décimaux renvoie un décimal qu'on convertit en format TIME (s'il est inférieur à 839 h, soit 34.95 jours) par la fonction SEC_TO_TIME après l'avoir changé en secondes (multiplié par 24*3600).

```

UPDATE Pilote SET
    intervalleEmbaucheControle =
        SEC_TO_TIME((intervalleEntreVols - intervalleVolExterieur)*24*3600)
WHERE brevet = 'PL-1';

```

La ligne contient désormais les informations suivantes. Les données en gras correspondent aux mises à jour. On trouve qu'il a fallu 14 109,126875 jours pour que ce pilote soit embauché. 10,6511181 jours séparent le dernier vol du pilote du moment de son embauche. 14,645833 jours, séparent son dernier vol de son prochain contrôle en vol. La différence entre ces deux délais est de 95 heures, 52 minutes et 18 secondes.

Figure 2-10 Ligne modifiée par des calculs de dates

Pilote

brevet	nom	dateNaiss	dernierVol	dateEmbauche	prochainVolControle
PL-1	Thierry Albaric	1967-03-25 1967-03-25 12:35:00	2005-10-30	2005-11-02 15:27:42 2005-11-09 15:37:42	2005-11-13 15:30:00

nombreJoursNaisBoulot	intervalleNaisBoulot	intervalleVolExterieur
13186.1596	14109.126875	10.651181

intervalleEntreVols	intervalleEmbaucheControle	compa
14.645833	95:52:18	AF

Nous verrons au chapitre 4, comment convertir en jours, heures, minutes et secondes un décimal de grande taille.

Fonctions utiles



Les fonctions suivantes vous seront d'un grand secours pour manipuler des dates et des intervalles de temps.

- `DATE_FORMAT(date,format)` convertit une date (heure) suivant un certain format.
- `EXTRACT(type FROM date)` extrait une partie donnée d'une date (heure).
- `FROM_DAYS(n)` retourne une date à partir d'un entier (le 1/1/0001 correspond à 0).
`UNIX_TIMESTAMP(date)` retourne le nombre de secondes qui se sont écoulées depuis le 1^{er} janvier 1970 jusqu'à la date (heure) en paramètre.
- `GET_FORMAT(DATE|TIME|DATETIME, 'EUR' | 'USA' | 'JIS' | 'ISO' | 'INTERNAL')` retourne un format de date (heure).
- `SEC_TO_TIME(secondes)` convertit un nombre en un type TIME et son inverse `TIME_TO_SEC(time)`.
- `STR_TO_DATE(chaine,format)` convertit une chaîne en date (heure) suivant un certain format.
- `TIME(expression)` extrait d'une date (heure) un type TIME.
- `TIME_FORMAT(time,format)` convertit un intervalle suivant un certain format.

Les tableaux suivants présentent quelques exemples d'utilisation de ces fonctions :

Tableau 2-11 Quelques formats pour DATE_FORMAT et STR_TO_DATE

Expression	Résultat	Commentaire
<code>DATE_FORMAT(SYSDATE(), '%j')</code>	306	Ce n'est pas la vitesse de Data dans <i>Taxi2</i> , mais le numéro du jour de l'année (ici il s'agit du 2 novembre 2005).
<code>DATE_FORMAT(dateNaiss, '%W en %M %X')</code>	Saturday en March 1967	Affichage des libellés des jours des mois en anglais par défaut.
<code>STR_TO_DATE('11/09/2005 15:37:42', '%d/%m/%Y %H:%i:%s')</code>	2005-09-11 15:37:42	Conversion d'une chaîne typée française au format DATETIME.

Tableau 2-12 Utilisation de EXTRACT et UNIX_TIMESTAMP

Expression	Résultat	Commentaire
<code>EXTRACT(DAY FROM dateEmbauche)</code>	9	Extraction du jour contenu dans la colonne dateEmbauche.
<code>EXTRACT(MONTH FROM dateNaiss)</code>	3	Extraction du mois contenu dans la colonne dateNaiss.
<code>UNIX_TIMESTAMP(SYSDATE())/(24*3600)</code>	13089.8850	Le 2 novembre 2005 au soir, 13 089 jours et 8850 secondes depuis le 1 ^{er} janvier 1970.

L'instruction REPLACE consiste, comme son nom l'indique, à remplacer un enregistrement dans sa totalité (toutes ses colonnes). Il faut avoir les priviléges 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',
    'Air 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 qu'une 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 (`bdsoutou/pilote`, CONSTRAINT `fk_Pil_compa` FOREIGN KEY (`compa`) REFERENCES `compagnie` (`comp`))
```

Détruisons enfin les deux premières compagnies (triées par ordre croissant de clé, ici 'AN1' et 'AN2') 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 complètement 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.



Il n'est pas possible de « tronquer » une table qui est référencée par des clés étrangères actives. La solution consiste à désactiver les contraintes puis à tronquer la table.

Intégrité référentielle

L'intégrité référentielle forme le cœur de la cohérence d'une base de données relationnelle. Cette intégrité est fondée sur la relation entre clés étrangères et clés primaires (ou candidates) colonnes indexées uniques et non nulles) qui permettent de programmer des règles de gestion (exemple : l'affrètement d'un vol doit se faire par une compagnie et pour un avion tous deux existant dans la base de données). Ce faisant, la plupart des contrôles côté client (interfaces) sont ainsi déportés côté serveur.

Pour les règles de gestion plus complexes (exemple : l'affrètement d'un avion doit se faire par une compagnie qui a embauché au moins quinze pilotes dans les six derniers mois), il faut programmer un déclencheur (décris au chapitre 7). Il faut savoir que les déclencheurs sont plus pénalisants que des contraintes dans un mode transactionnel.



La contrainte référentielle concerne toujours deux tables – une table « père » aussi dite « maître » (*parent/referenced*) et une table « fils » (*child/dependent*) – possédant une ou plusieurs colonnes en commun. Pour la table « père », ces colonnes composent la clé primaire (ou candidate avec un index unique). Pour la table « fils », ces colonnes composent une clé étrangère.

Syntaxe

C'est seulement dans sa version 3.23.44 en 2002 (dix ans après Oracle), que MySQL a inclus dans son offre les contraintes référentielles pour les tables InnoDB. L'intégrité référentielle est programmée dans la table « fils » par la contrainte suivante. Il est conseillé de nommer cette contrainte, sinon MySQL s'en charge. Si la clé étrangère n'est pas déjà indexée, MySQL s'en charge aussi. Les deux tables ne doivent pas être temporaires.

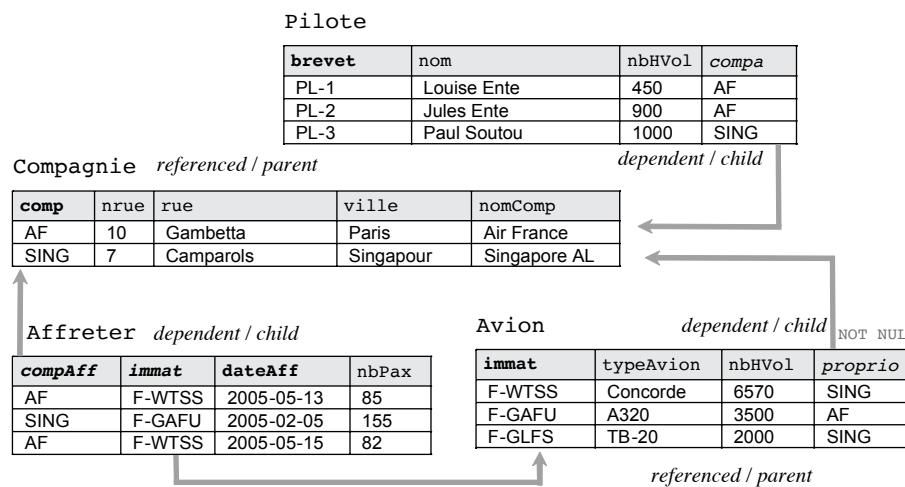
```
[CONSTRAINT nomContrainte] FOREIGN KEY [id] (listeColonneEnfant)
    REFERENCES nomTable (listeColonneParent)
    [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
    [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Cohérences assurées



L'exemple suivant illustre quatre contraintes référentielles. Une table peut être « père » pour une autre table, qui a une contrainte et « fils » pour une autre (c'est le cas de la table Avion).

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. 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
<pre>CREATE TABLE Compagnie (comp CHAR(4), nrue INTEGER(3), rue CHAR(20), ville CHAR(15), nomComp CHAR(15), CONSTRAINT pk_Compagnie PRIMARY KEY(comp));</pre>	<pre>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));</pre>

Contraintes côté « fils »

Indépendamment de l'écriture de la table « père », plusieurs écritures sont possibles au niveau de la table « fils » selon qu'on crée 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
<pre>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));</pre>	<pre>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 Companie(comp));</pre>

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) aucune contrainte NOT NULL n'est déclarée.

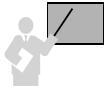
Décrivons à présent le script SQL qui convient à notre exemple (la syntaxe de création des deux premières tables a été discutée plus haut) et étudions ensuite les mécanismes programmés par ces contraintes. Décidons qu'un avion aura toujours un propriétaire (NOT NULL).

```

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 donné en anglais (il y est question de ne pouvoir ajouter un enregistrement « fils »).

Tableau 2-15 Insertions correctes et incorrectes

Web	Insertions correctes	Insertion incorrecte
	<pre>-- fils avec père INSERT INTO Pilote VALUES ('PL-3', 'Paul Soutou', 1000, 'SING'); -- fils sans père INSERT INTO Pilote VALUES ('PL-4', 'Un Connus', 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)</pre>	<pre>-- 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 (`bdssoutou/pilote`, CONSTRAINT `fk_Pil_compa_Comp` FOREIGN KEY (`comp`) REFERENCES `compagnie` (`comp`))</pre>

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) et enfin les affrètements.



Il suffit de relire le script de création de vos tables pour en déduire l'ordre d'insertion des enregistrements.

Cohérence du père vers le fils

En fonction des options choisies au niveau de la contrainte référentielle se trouvant dans table « fils » :

```
CONSTRAINT nomContrainte FOREIGN KEY ... REFERENCES ...
  [ ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION} ]
  [ ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION} ]
```

plusieurs scénarios sont possibles pour assurer la cohérence de la table « père » vers la table « fils »

- Prévenir la modification ou la suppression d'une clé primaire (ou candidate) de la table « père ». Cette alternative est celle par défaut. Soit vous n'ajoutez pas d'option à la clause REFERENCES – dans notre exemple, toutes les clés étrangères sont ainsi composées –, soit vous utilisez NO ACTION pour les directives ON DELETE et ON UPDATE. La suppression d'un avion n'est donc pas possible si ce dernier est référencé dans un affrètement.
- Propager la suppression des enregistrements « fils » associés à l'enregistrement « père » supprimé. Ce mécanisme est réalisé par la directive ON DELETE CASCADE. Dans notre exemple, nous pourrions ainsi décider de supprimer tous les affrètements dès qu'on retire un avion.
- Étendre la modification de la clé primaire de l'enregistrement « père » aux enregistrements « fils » associés. Ce mécanisme est réalisé par la directive ON UPDATE CASCADE. Dans notre exemple, nous pourrions ainsi décider de mettre à jour tous les affrètements dès qu'on modifie l'immatriculation d'un avion.
- Propager l'affectation de la valeur nulle aux clés étrangères des enregistrements « fils » associés à l'enregistrement « père » supprimé ou modifié. Ce mécanisme est réalisé par directive ON DELETE SET NULL (ou ON UPDATE SET NULL en cas de modification de la clé primaire du « père »). Dans ces deux cas, il ne faut pas poser de contrainte NOT NULL sur la clé étrangère. Dans notre exemple, nous pourrions ainsi décider de mettre NULL dans la colonne `compa` de la table `Pilote` pour chaque pilote d'une compagnie supprimée. Nous ne pourrions pas appliquer ce mécanisme à la table `Affreter` qui dispose de contraintes NOT NULL sur ses clés étrangères (car composant la clé primaire).



RESTRICT est une directive de la norme SQL qui n'est pas encore mise en œuvre sous MySQL. Elle concerne les SGBD compatibles avec les contraintes différenciées. Pour l'heure NO ACTION (qui en principe diffère les contraintes) et RESTRICT (qui ne diffère pas les contraintes) jouent le même rôle.

Les options DELETE CASCADE et DELETE SET NULL sont disponibles depuis version 3.23.50, celles relatives à ON UPDATE sont disponibles depuis la version 4.0.8.

Le tableau suivant décrit quelques alternatives de cohérence à notre base de données entre les tables Avion et Affreter puis Pilote et Compagnie.

Tableau 2-16 Alternatives de cohérence du « père » vers les « fils »

Web	Alternatives	Syntaxe / Message d'erreur
	Prévenir la modification de l'immatriculation d'un avion ou la suppression d'un avion.	--dans Affreter CONSTRAINT fk_Aff_na_Avion FOREIGN KEY(immat) REFERENCES Avion(immat) ON DELETE NO ACTION ON UPDATE NO ACTION mysql> DELETE FROM Avion; ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`bdsoutou/affreter`, CONSTRAINT `fk_Aff_na_Avio FOREIGN KEY (`immat`) REFERENCES `avion`(`immat ON DELETE NO ACTION ON UPDATE NO ACTION)
	Propager la suppression d'un avion.	--dans Affreter CONSTRAINT fk_Aff_na_Avion FOREIGN KEY(immat) REFERENCES Avion(immat) ON DELETE CASCADE
	Propager la modification de l'immatriculation d'un avion.	--dans Affreter CONSTRAINT fk_Aff_na_Avion FOREIGN KEY(immat) REFERENCES Avion(immat) ON UPDATE CASCADE
	Propager la modification du code de la compagnie dans les tables Pilote, Avion et Affreter. Affecter la valeur nulle dans la table Pilote suite à la suppression d'une compagnie, tout en préservant l'intégrité avec la table Avion.	--dans Affreter CONSTRAINT fk_Aff_comp_Compag FOREIGN KEY(compAff) REFERENCES Compagnie(comp ON UPDATE CASCADE --dans Avion CONSTRAINT fk_Avion_comp_Compag FOREIGN KEY(proprio) REFERENCES Compagnie(comp ON UPDATE CASCADE --dans Pilote CONSTRAINT fk_Pil_compa_Comp FOREIGN KEY (compa) REFERENCES Compagnie(comp ON DELETE SET NULL ON UPDATE CASCADE



Pour l'heure, aucun déclencheur ne peut être activé suite à la modification d'une colonne induite d'une action de répercussion (CASCADE ou SET NULL).

MySQL ne permet pas encore de propager une valeur par défaut (*set default*) comme la norme SQL le prévoit.

Le tableau suivant résume les conditions requises pour modifier l'état de la base de données en respectant l'intégrité référentielle :

Tableau 2-17 Instructions SQL sur les clés

Instructions	Table « père »	Table « fils »
INSERT	Correcte si la clé primaire (ou candidate) est unique.	Correcte si la clé étrangère est référencée dans la table « père » ou est nulle (partiellement ou en totalité).
UPDATE	Correcte si l'instruction ne laisse pas d'enregistrements dans la table « fils » ayant une clé étrangère non référencée.	Correcte si la nouvelle clé étrangère référence un enregistrement « père » existant.
DELETE	Correcte si aucun enregistrement de la table « fils » ne référence le ou les enregistrements détruits.	Correcte sans condition.
DELETE Cascade	Correcte sans condition.	Sans objet.
DELETE SET NULL	Correcte sans condition.	Sans objet.
UPDATE Cascade	Correcte sans condition.	Sans objet.
UPDATE SET NULL	Correcte s'il n'y a pas de NOT NULL dans la table « fils ».	Sans objet.

Insertions à partir d'un fichier

L'importation de données (au format fichier texte) dans une table peut être programmée l'aide de la directive `LOAD DATA INFILE`. Un tel mécanisme lit un fichier dans un répertoire du serveur et insère tout ou partie des informations dans une table. La syntaxe simplifiée de cette directive est la suivante :

```
LOAD DATA INFILE 'nomEtCheminFichier.txt'  
[REPLACE | IGNORE] INTO TABLE nomTable  
[FIELDS [TERMINATED BY 'string' ]  
[[OPTIONALLY] ENCLOSED BY 'char' ]  
[ESCAPED BY 'char' ] ]  
[LINES [STARTING BY 'string' ] [TERMINATED BY 'string' ] ]  
[IGNORE number LINES];
```

- **REPLACE** : option à utiliser pour remplacer systématiquement les anciens enregistrements par les nouveaux (valeur de clé primaire ou d'index unique).
- **IGNORE** : fait en sorte de ne pas insérer des enregistrements de clé primaire ou d'index unique déjà présents dans la table.

- **FIELDS** décrit comment sont formatées dans le fichier les valeur à insérer dans la colonne. En l'absence de cette clause, TERMINATED BY vaut '\t', ENCLOSED BY vaut ESCAPED BY vaut '\\'.
 - FIELDS TERMINATED BY décrit le caractère qui sépare deux valeurs de colonne.
 - 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 déclarations ou des commentaires).

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 comme cet effet (brevet VARCHAR(6), nom VARCHAR(16), nbHVol DECIMAL(7,2), compta CHAR(4) et PRIMARY KEY(brevet)). Notez l'utilisation du double « \\ » pour gérer une arborescence Windows. Le caractère NULL est importé par le caractère « \\N ».

Figure 2-12 Importation de données

```
import-Pilote"PL-1";"Gratien Viel";"450.00";"AF"$  
import-Pilote"PL-2";"Didier Donsez";"0.00";"AF"$  
import-Pilote"PL-3";"Richard Grin";"1000.00";"SING"$  
import-Pilote"PL-4";"Placide Fresnais";"2450.00";"CAST"$  
import-Pilote"PL-5";"Daniel Vielle";"\N";"AF"$
```

```
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 compta, nom;  
+-----+-----+-----+-----+  
| brevet | nom          | nbHVol | compta |  
+-----+-----+-----+-----+  
| 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   |  
+-----+-----+-----+-----+
```

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
	p1	Poste 1	130.120.80	01	TX
	p2	Poste 2	130.120.80	02	UNIX
	p3	Poste 3	130.120.80	03	TX
	p4	Poste 4	130.120.80	04	PCWS
	p5	Poste 5	130.120.80	05	PCWS
	p6	Poste 6	130.120.80	06	UNIX
	p7	Poste 7	130.120.80	07	TX
	p8	Poste 8	130.120.81	01	UNIX
	p9	Poste 9	130.120.81	02	TX
	p10	Poste 10	130.120.81	03	UNIX
	p11	Poste 11	130.120.82	01	PCNT
	p12	Poste 12	130.120.82	02	PCWS

Tableau 2-18 Données des tables (suite)

Table	Données				
Logiciel	NLOG	NOMLOG	DATEACH	VERSION	TYPELOG
log1	Oracle	6	1995-05-13	6.2	UNIX
log2	Oracle	8	1999-09-15	8i	UNIX
log3	SQL Server		1998-04-12	7	PCNT
log4	Front Page		1997-06-03	5	PCWS
log5	WinDev		1997-05-12	5	PCWS
log6	SQL*Net			2.0	UNIX
log7	I. I. S.		2002-04-12	2	PCNT
log8	DreamWeaver		2003-09-21	2.0	BeOS
Types	TYPELP	NOMTYPE			
TX	Terminal	X-Window			
UNIX	Système	Unix			
PCNT	PC	Windows NT			
PCWS	PC	Windows			
NC	Network	Computer			

Exercice 2.2 Gestion d'une séquence

Dans ce même script, gérer la séquence associée à la colonne numIns commençant à la valeur manière à insérer les enregistrements suivants :

Tableau 2-19 Données de la table Installer

Table	Données				
Installer	NPOSTE	NLOG	NUMINS	DATEINS	DELAI
p2		log1	1	2003-05-15	
p2		log2	2	2003-09-17	
p4		log5	3		
p6		log6	4	2003-05-20	
p6		log1	5	2003-05-20	
p8		log2	6	2003-05-19	
p8		log6	7	2003-05-20	
p11		log3	8	2003-04-20	
p12		log4	9	2003-04-20	
p11		log7	10	2003-04-20	
p7		log7	11	2002-04-01	

Exercice 2.3 Modification de données

Écrire le script `modification.sql` qui permet de modifier (avec `UPDATE`) la colonne `etage` (pour l'instant nulle) de la table `Segment`, afin d'affecter un numéro d'étage correct (0 pour le segment 130.120.80, 1 pour le segment 130.120.81, 2 pour le segment 130.120.82).

Diminuer de 10 % le prix des logiciels de type 'PCNT'.

Vérifier :

```
SELECT * FROM Segment;  
SELECT nLog, typeLog, prix FROM Logiciel;
```

Évolution d'un schéma

L'évolution d'un schéma est un aspect très important à prendre en compte, car il répond aux besoins de maintenance des applicatifs qui utilisent la base de données. Nous verrons qu'il est possible de modifier une base de données d'un point de vue structurel (colonnes et index) mais aussi comportemental (contraintes).

L'instruction principalement utilisée est `ALTER TABLE` (commande du LDD) qui permet d'ajouter, de renommer, de modifier et de supprimer des colonnes d'une table. Cette commande permet aussi d'ajouter et de supprimer des contraintes. Avant de détailler ces mécanismes, étudions la commande qui permet de renommer une table.

Renommer une table (RENAME)

L'instruction `RENAME` renomme une ou plusieurs tables ou vues. Il faut posséder le privilège `ALTER` et `DROP` sur la table d'origine, et `CREATE` sur la base.

```
RENAME [ nomBase.] ancienNomTable TO [ nomBase.] nouveauNomTable
      [, [ nomBase.] ancienNom2 TO [ nomBase.] nouveauNom2] ;
```

Les contraintes d'intégrité, index et prérogatives associés à l'ancienne table sont automatiquement transférés sur la nouvelle. En revanche, les vues et procédures cataloguées sont indépendantes et doivent être recréées.

Il est aussi possible d'utiliser l'option `RENAME TO` de l'instruction `ALTER TABLE` pour renommer une table existante. Le tableau suivant décrit comment renommer la table `Pilote` sans perturber l'intégrité référentielle :

Tableau 3-1 Renommer une table

Commande RENAME	Commande ALTER TABLE
<code>RENAME Pilote TO Naviguant;</code>	<code>ALTER TABLE Pilote RENAME TO Naviguant;</code>

Modifications structurelles (ALTER TABLE)

Considérons la table suivante que nous allons faire évoluer :

Figure 3-1 Table à modifier

```
CREATE TABLE Pilote  
    (brevet VARCHAR(4), nom VARCHAR(20));  
  
INSERT INTO Pilote  
    VALUES ('PL-1', 'Agnès Labat');
```

Pilote	
brevet	nom
PL-1	Agnès Labat

Ajout de colonnes

La directive ADD de l'instruction ALTER TABLE permet d'ajouter une nouvelle colonne à une table. Cette colonne est initialisée à NULL pour tous les enregistrements (à moins de spécifier une contrainte DEFAULT, auquel cas tous les enregistrements de la table sont mis à jour avec une valeur non nulle).



Il est possible d'ajouter une colonne en ligne NOT NULL seulement si la table est vide ou si une contrainte DEFAULT est définie sur la nouvelle colonne (dans le cas inverse, il faudra utiliser MODIFY à la place de ADD).

Le script suivant ajoute trois colonnes à la table Pilote. La première instruction insère une colonne nbHVol en l'initialisant à NULL pour tous les pilotes (ici il n'en existe qu'une seule). La deuxième commande ajoute deux colonnes initialisées à une valeur non nulle. La colonne ville ne sera jamais nulle.



```
ALTER TABLE Pilote ADD (nbHVol DECIMAL(7,2));  
ALTER TABLE Pilote  
    ADD (compa VARCHAR(4) DEFAULT 'AF',  
         ville VARCHAR(30) DEFAULT 'Paris' NOT NULL);
```

La table est désormais la suivante :

Figure 3-2 Table après l'ajout de colonnes

Pilote				
brevet	nom	nbHVol	compa	ville
PL-1	Agnès Labat		AF	Paris

Renommer des colonnes

Il faut utiliser l'option **CHANGE** de l'instruction **ALTER TABLE** pour renommer une colonne existante. Le nouveau nom de colonne ne doit pas être déjà utilisé dans la table. Le type de colonne (et une éventuelle contrainte) doit être reprécisé. La position de la colonne peut aussi être modifiée en même temps. La syntaxe générale de cette option est la suivante :

```
ALTER TABLE [nomBase].nomTable CHANGE [COLUMN] ancienNom  
nouveauNom typeMySQL [NOT NULL | NULL] [DEFAULT valeur]  
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]  
[COMMENT 'chaine'] [REFERENCES ...]  
[FIRST|AFTER nomColonne];
```

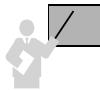
L'instruction suivante permet de renommer la colonne **ville** en **adresse** en la positionnant avant la colonne **compa** :

```
ALTER TABLE Pilote CHANGE ville adresse VARCHAR(30) AFTER nbHVol
```

Modifier le type des colonnes

L'option **MODIFY** de l'instruction **ALTER TABLE** modifie le type d'une colonne existante sans pour autant la renommer. La syntaxe générale de cette instruction est la suivante : les options sont les mêmes que pour **CHANGE** :

```
ALTER TABLE [nomBase].nomTable MODIFY [COLUMN] nomColonneAModifier  
typeMySQL [NOT NULL | NULL] [DEFAULT valeur]  
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]  
[COMMENT 'chaine'] [REFERENCES ...]  
[FIRST|AFTER nomColonne];
```



Il est possible d'augmenter la taille d'une colonne numérique (largeur ou précision) – ou d'agrandir une chaîne de caractères (CHAR et VARCHAR) – ou de la diminuer si toutes les données présentes dans la colonne peuvent s'adapter à la nouvelle taille.

Attention à ne pas réduire les colonnes indexées à une taille inférieure à celle déclarée lors de la création de l'index.

Les contraintes en ligne peuvent être aussi modifiées par cette instruction. Une fois la colonne changée, les nouvelles contraintes s'appliqueront aux mises à jour ultérieures de la table, les données présentes devront toutes vérifier cette nouvelle contrainte.

Le tableau suivant présente différentes modifications de colonnes :

Tableau 3-2 Modifications de colonnes

Instructions SQL	Commentaires
<pre>ALTER TABLE Pilote MODIFY compa VARCHAR(6) DEFAULT 'SING'; INSERT INTO Pilote (brevet, nom) VALUES ('PL-2', 'Laurent Boutrand');</pre>	Augmente la taille de la colonne compa et change la contrainte de valeur par défaut puis insère un nouveau pilote.
<pre>ALTER TABLE Pilote MODIFY compa CHAR(4) NOT NULL;</pre>	Diminue la colonne et modifie également son type de VARCHAR en CHAR tout en le déclarant NOT NULL (possible car les données contenues dans la colonne ne dépassent pas quatre caractères).
<pre>ALTER TABLE Pilote MODIFY compa CHAR(4);</pre>	Rend possible l'insertion de valeur nulle dans la colonne compa.

La table est désormais la suivante :

Figure 3-3 Après modification des colonnes

Pilote				
brevet	nom	nbHVol	adresse	compa
PL-1	Agnès Labat		Paris	AF
PL-2	Laurent Boutrand		Paris	SING

CHAR (4)
NULL possible
Défaut : 'SING'

Valeurs par défaut

L’option ALTER COLUMN de l’instruction ALTER TABLE modifie la valeur par défaut d’une colonne existante. La syntaxe générale de cette instruction est la suivante :

```
ALTER TABLE [ nomBase ]. nomTable ALTER [COLUMN] nomColonneAModifier
    {SET DEFAULT 'chaine' | DROP DEFAULT};
```

Le script suivant définit une valeur par défaut pour la colonne adresse puis supprime celle relative à la colonne compa :

```
ALTER TABLE Pilote ALTER COLUMN adresse SET DEFAULT 'Blagnac';
ALTER TABLE Pilote ALTER COLUMN compa DROP DEFAULT;
```

Supprimer des colonnes

L’option DROP de l’instruction ALTER TABLE permet de supprimer une colonne (aussi un index ou une clé que nous étudierons plus loin). La possibilité de supprimer une colonne évite aux administrateurs d’exporter les données, de recréer une nouvelle table, d’importer les

données et de recréer les éventuels index et contraintes. Lorsqu'une colonne est supprimée, les index qui l'utilisent sont mis à jour voire éliminés si toutes les colonnes qui le composent sont effacées. La syntaxe des ces options est la suivante :

```
ALTER TABLE [nomBase].nomTable DROP
{ [COLUMN] nomColonne | PRIMARY KEY
| INDEX nomIndex | FOREIGN KEY nomContrainte }
```



Il n'est pas possible de supprimer avec cette instruction :

- toutes les colonnes d'une table ;
- les colonnes qui sont clés primaires (ou candidates par UNIQUE) référencées par des clés étrangères.

La suppression de la colonne `adresse` de la table `Pilote` est programmée par l'instruction suivante :

```
ALTER TABLE Pilote DROP COLUMN adresse;
```

Modifications comportementales

Nous étudions dans cette section les mécanismes d'ajout, de suppression, d'activation et désactivation de contraintes.



Faisons évoluer le schéma suivant. Les seules contraintes existantes sont les clés primaires nommées `pk_Compagnie`, pour la table `Compagnie`, et `pk_Avion` pour la table `Avion`.

Figure 3-4 Schéma à faire évoluer

Compagnie

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

Affreter

compAff	immat	dateAff	nbPax
AF	F-WTSS	2003-05-13	85
SING	F-GAFU	2003-02-05	155
AF	F-WTSS	2003-05-15	82

Avion

immat	typeAvion	nbHVol	proprio
F-WTSS	Concorde	6570	SING
F-GAFU	A320	3500	AF
F-GLFS	TB-20	2000	SING

Ajout de contraintes

Jusqu'à présent, nous avons créé les contraintes en même temps que les tables. Il est possible de créer des tables sans contraintes (dans ce cas l'ordre de génération n'est pas important).

peut même les élaborer par ordre alphabétique), puis d'ajouter les contraintes. Les outils de conception (*Win'Design*, *Designer* ou *PowerAMC*) adoptent cette démarche lors de la génération automatique de scripts SQL.

La directive ADD CONSTRAINT de l'instruction ALTER TABLE permet d'ajouter une contrainte à une table. Il est aussi possible d'ajouter un index. La syntaxe générale est la suivante :

```
ALTER TABLE [nomBase].nomTable ADD
{ INDEX [nomIndex] [typeIndex] (nomColonne1,...)
| CONSTRAINT nomContrainte typeContrainte }
```

Trois types de contraintes sont possibles :

- UNIQUE (colonne1 [,colonne2]...)
- PRIMARY KEY (colonne1 [,colonne2]...)
- FOREIGN KEY (colonne1 [,colonne2]...)

```
REFERENCES nomTablePère (col1 [,col2]...)
[ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
[ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION}]
```

Unicité

Ajoutons la contrainte d'unicité portant sur la colonne du nom de la compagnie. Un index est automatiquement généré sur cette colonne à présent :

```
ALTER TABLE Compagnie ADD CONSTRAINT un_nomC UNIQUE (nomComp);
```

Clé étrangère

Ajoutons la clé étrangère (indexée) à la table Avion au niveau de la colonne *proprio* en la assignant une contrainte NOT NULL :

```
ALTER TABLE Avion ADD INDEX (proprio);
ALTER TABLE Avion ADD CONSTRAINT fk_Avion_comp_Compag
FOREIGN KEY(proprio) REFERENCES Compagnie(comp);
ALTER TABLE Avion MODIFY proprio CHAR(4) NOT NULL;
```

Clé primaire

Ajoutons à la table Affreter, en une seule instruction, sa clé primaire et deux clés étrangères (une vers la table Avion et l'autre vers Compagnie) :

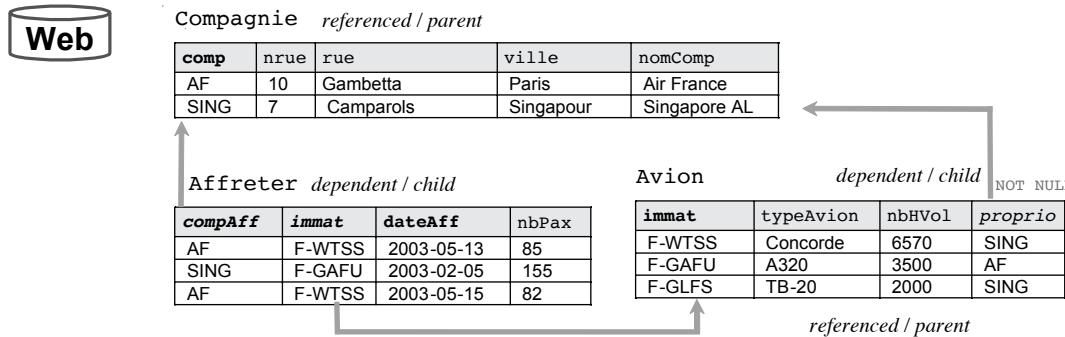
```
ALTER TABLE Affrete ADD (
CONSTRAINT pk_Affreter PRIMARY KEY (compAff, immat, dateAff)
CONSTRAINT fk_Aff_na_Avion FOREIGN KEY(immat) REFERENCES
Avion(immat),
CONSTRAINT fk_Aff_comp_Compag FOREIGN KEY(compAff)
REFERENCES Compagnie(comp));
```



Pour que l'ajout ou la modification d'une contrainte soient possibles, il faut que les données présentes dans la table concernée ou référencée respectent la nouvelle contrainte.

Les tables contiennent à présent les contraintes suivantes :

Figure 3-5 Après ajout de contraintes



Suppression de contraintes

Il n'existe pas encore l'option `DROP CONSTRAINT` à l'instruction `ALTER TABLE` pour supprimer une contrainte de nom donné (peut-être parce que les contraintes `CHECK` ne sont pas encore considérées, et car la prise en compte des contraintes a été étalée au fil des versions successives de MySQL). Il faut donc utiliser une directive différente de l'instruction `ALTER TABLE` pour supprimer chaque type de contrainte.

Contrainte NOT NULL

Il faut utiliser la directive `MODIFY` de l'instruction `ALTER TABLE` pour supprimer une contrainte `NOT NULL` existante sur une colonne. Dans notre exemple, détruisons la contrainte `NOT NULL` de la clé étrangère `proprio` dans la table `Avion`.

```
ALTER TABLE Avion MODIFY proprio CHAR(4) NULL;
```

Contrainte UNIQUE

Il faut utiliser la directive `DROP INDEX` de l'instruction `ALTER TABLE` pour supprimer une contrainte d'unicité. Dans notre exemple, effaçons la contrainte portant sur le nom `compagnie`. L'index est également détruit.

```
ALTER TABLE Compagnie DROP INDEX un_nomC;
```

Clé étrangère

L’option **DROP FOREIGN KEY** de l’instruction **ALTER TABLE** permet de supprimer une clé étrangère d’une table. La syntaxe générale est la suivante :

```
| ALTER TABLE [nomBase].nomTable DROP FOREIGN KEY nomContrainte;
```

Le nom de la contrainte est celui qui a été déclaré lors de la création de la table, soit lors de sa modification (dans **CREATE TABLE** ou **ALTER TABLE**).



Si la contrainte a été définie sans être nommée, son nom est généré par le moteur InnoDB et l’instruction **SHOW CREATE TABLE [nomBase].nomTable** permet de le découvrir.

Détruisons la clé étrangère de la colonne **proprio**.

```
| ALTER TABLE Avion DROP FOREIGN KEY fk_Avion_comp_Compag;
```

Clé primaire

L’option **DROP PRIMARY KEY** de l’instruction **ALTER TABLE** permet de supprimer une clé primaire. Dans des versions précédentes de MySQL, si aucune clé primaire n’existait, la première contrainte **UNIQUE** disparaissait à la place. Ce n’est plus le cas depuis la version 5.7.



Si la colonne clé primaire à supprimer contient des clés étrangères, il faut d’abord retirer les contraintes de clé étrangère. Si la clé primaire à supprimer est référencée par des clés étrangères d’autres tables, il faut d’abord ôter les contraintes de clé étrangère de ces autres tables.

Ainsi, pour supprimer la clé primaire de la table **Affreter**, il faut d’abord enlever les deux contraintes de clé étrangère concernant des colonnes composant la clé primaire.

```
| ALTER TABLE Affreter DROP FOREIGN KEY fk_Aff_na_Avion;  
| ALTER TABLE Affreter DROP FOREIGN KEY fk_Aff_comp_Compag;  
ALTER TABLE Affreter DROP PRIMARY KEY;
```

La figure suivante illustre les contraintes qui restent actives : les clés primaires des tables **Compagnie** et **Avion**, et une contrainte de non nullité.

Aucun ordre particulier n’est nécessaire pour supprimer ces trois contraintes, car il n’y a plus de contrainte référentielle active.

Figure 3-6 Après suppression de contraintes

Compagnie

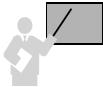
comp	nrule	rue	ville	nomComp
AF	10	Gambetta	Paris	Air France
SING	7	Camparols	Singapour	Singapore AL

Affreter

compAff	immat	dateAff	nbPax
AF	F-WTSS	2003-05-13	85
SING	F-GAFU	2003-02-05	155
AF	F-WTSS	2003-05-15	82

Avion

immat	typeAvion	nbHVol	proprio
F-WTSS	Concorde	6570	SING
F-GAFU	A320	3500	AF
F-GLFS	TB-20	2000	SING



Concernant tout schéma, il faut éliminer d'abord les contraintes de clé étrangère des « fils » puis « père » puis les contraintes de clé primaire. Il suffit, pour éviter toute incohérence de détruire les contraintes dans l'ordre inverse d'apparition dans le script de création.

Désactivation des contraintes

La désactivation des contraintes référentielles peut être intéressante pour accélérer des procédures de chargement d'importation et d'exportation massives de données externes. Ce mécanisme améliore aussi les performances de programmes *batchs* qui ne modifient pas les données concernées par l'intégrité référentielle, ou pour lesquelles on vérifie la cohérence de la base à la fin. En effet les index ne sont pas mis à jour pour chaque insertion ou modification et aucun ordre n'est requis pour insérer ou supprimer des enregistrements.

Syntaxe

L'instruction `SET FOREIGN_KEY_CHECKS=0` permet de désactiver temporairement (jusqu'à la réactivation) toutes les contraintes référentielles d'une base.



N'essayez pas de désactiver l'intégrité référentielle avec l'option `DISABLE KEYS` de l'instruction `ALTER TABLE`. Cette option concerne les tables MyISAM et spécifie seulement de ne pas mettre à jour les index non uniques.

Exemple

En considérant l'exemple suivant, désactivons les contraintes d'intégrité référentielle et tentons d'insérer des enregistrements ne respectant aucune contrainte (d'intégrité référentielle et autres).

Figure 3-7 Avant la désactivation de contraintes

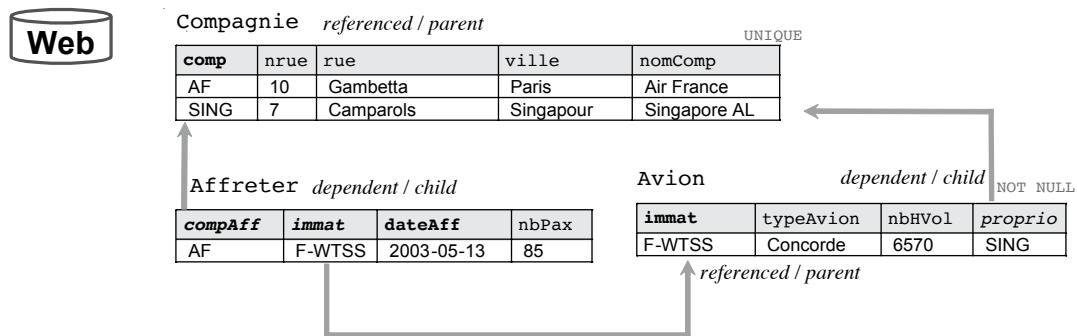


Tableau 3-3 Insertions après la désactivation de l'intégrité référentielle

Web	Instructions valides	Instructions non valides
	mysql> SET FOREIGN_KEY_CHECKS=0;	
	mysql> INSERT INTO Avion VALUES ('F-GLFS', 'TB-22', 500, ' Toto '); Query OK, 1 row affected (0.04 sec)	mysql> INSERT INTO Compagnie VALUES ('GTR', 1, 'Brassens', 'Blagnac', ' France '); ERROR 1062 (23000): Duplicate entry 'Air France' for key 2
	mysql> INSERT INTO Avion VALUES (' Bidon1 ', 'TB-21', 1000, 'AF'); Query OK, 1 row affected (0.10 sec)	mysql> INSERT INTO Avion VALUES ('Bidon1', 'TB-20', 2000, NULL); ERROR 1048 (23000): Column 'proprio' cannot be null
	mysql> INSERT INTO Affreter VALUES ('AF', ' Toto ', '2005-05-13', 0); Query OK, 1 row affected (0.10 sec)	mysql> INSERT INTO Avion VALUES ('F-GLFS', 'TB-21', 1000, 'AF'); ERROR 1062 (23000): Duplicate entry 'F-GLFS' for key 1
	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)	

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.

Figure 3-8 Après la désactivation des contraintes référentielles

Compagnie

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

Avion

immat	typeAvion	nbHVol	proprio
F-WTSS	Concorde	6570	SING
F-GLFS	TB-22	500	Toto

Affreter

compAff	immat	dateAff	nbPax
AF	F-WTSS	2003-05-13	82
AF	Toto	2005-05-13	0
GTI	F-WTSS	2005-11-07	10
GTI	Toto	2005-11-07	40

Réactivation des contraintes

L'instruction `SET FOREIGN_KEY_CHECKS=1` permet de réactiver toutes les contraintes référentielles d'une base.



N'essayez pas de désactiver l'intégrité référentielle avec l'option `ENABLE KEYS` de l'instruction `ALTER TABLE`. Cette option concerne les tables MyISAM et spécifie de mettre à jour les clés non uniques.

Syntaxe

La réactivation totale de l'intégrité référentielle de la base se programme ainsi :

```
mysql> SET FOREIGN_KEY_CHECKS=1;
```



L'intégrité est assurée de nouveau mais ne concerne que les mises à jour à venir (ajout d'enregistrements, modifications de colonnes et suppressions d'enregistrements). Les données actuelles présentes dans les tables qui ne vérifient pas l'intégrité sont toujours en b

Récupération de données erronées

Il n'existe pas pour l'heure de méthode de récupération automatique comme Oracle propose, par exemple, avec la directive `EXCEPTIONS INTO` de l'instruction `ALTER TABLE`. En conséquence, il faut programmer des requêtes d'extraction (étudiées au chapitre 4) permettant des recherches d'enregistrements sous critères.

Le tableau suivant décrit les deux requêtes à programmer afin d'extraire les enregistrements posant problème. Ici nous extrayons les avions qui réfèrent une compagnie inexiste et les affrètements qui réfèrent une compagnie inexistante ou un avion inexistant. Une

extraits, il faudra statuer pour chacun d'eux entre une modification de telle ou telle colonne ou bien une suppression. Il apparaît que quatre enregistrements ne respectent pas des contraintes

Tableau 3-4 Enregistrements posant problème

Table Avion		Table Affreter		
mysql> SELECT immat, proprio FROM Avion		mysql> SELECT compAff, immat, dateAff		
WHERE proprio NOT IN (SELECT comp FROM Compagnie);		FROM Affreter WHERE compAff NOT IN (SELECT comp FROM Compagnie)		

immat	proprio
F-GLFS	Toto

compAff	immat	dateAff
GTI	F-WTSS	2005-11-07
AF	Toto	2005-05-13
GTI	Toto	2005-11-07



Désactivez de nouveau l'intégrité référentielle avant de modifier les enregistrements ne vérifiant pas les contraintes, car, sinon, certaines modifications cohérentes ne pourraient avoir lieu à cause de la vérification référentielle sur des valeurs erronées (dans notre exemple si on veut modifier la compagnie 'GTI' du dernier affrètement, l'erreur portera sur la colonne immat).

Dans notre exemple, choisissons :

- D'affecter la compagnie 'AF' aux avions appartenant à des compagnies non référencées dans la table Avion. Notez qu'il faut compléter par deux espaces le code compagnie (CHAR, inutile s'il avait été VARCHAR), et qu'on utilise la même requête que précédemment.

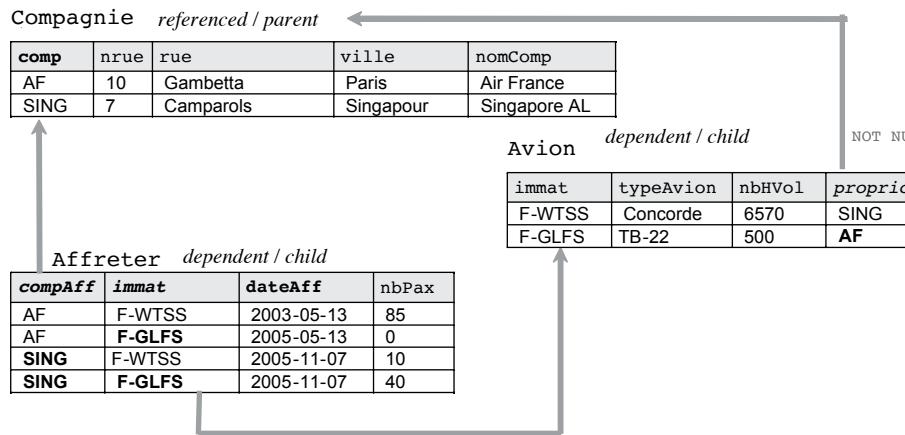
```
SET FOREIGN_KEY_CHECKS=0;
UPDATE Avion SET proprio = 'AF '
    WHERE proprio NOT IN (SELECT comp FROM Compagnie) ;
```

- De modifier la compagnie 'GTI' par la compagnie 'SING' dans la table Affreter et toutes les immatriculations d'avion inexistant en 'F-GLFS'.

```
UPDATE Affreter SET compAff ='SING' WHERE compAff = 'GTI ' ;
UPDATE Affreter SET immat='F-GLFS'
    WHERE immat NOT IN (SELECT immat FROM Avion) ;
```

Maintenant, il conviendra de réactiver l'intégrité référentielle. L'état de la base avec les contraintes réactivées est le suivant (les mises à jour sont en gras) :

Figure 3-9 Tables après modifications et réactivation des contraintes



Contraintes différencées

Les contraintes que nous avons étudiées jusqu'à maintenant sont des contraintes immédiates (*immediate*) qui sont contrôlées après chaque instruction. Une contrainte est dite « différée » (*deferred*) si elle déclenche sa vérification seulement à la fin de la transaction (après une instruction `commit` rencontrée). Pour l'heure, MySQL avec InnoDB ne propose pas ce type de programmation.

Les objectifs de ces exercices sont :

- d'ajouter et de modifier des colonnes ;
- d'ajouter des contraintes ;
- de traiter les erreurs.

Exercice

3.1 Ajout de colonnes

Écrire le script `évolution.sql` qui contient les instructions nécessaires pour ajouter les colonnes suivantes (avec `ALTER TABLE`). Le contenu de ces colonnes sera modifié ultérieurement.

Tableau 3-5 Données de la table `Installer`

Table	Nom, type et signification des nouvelles colonnes
Segment	<code>nbSalle TINYINT(2)</code> : nombre de salles par défaut = 0. <code>nbPoste TINYINT(2)</code> : nombre de postes par défaut = 0.
Logiciel	<code>nbInstall TINYINT(2)</code> : nombre d'installations par défaut = 0.
Poste	<code>nbLog TINYINT(2)</code> : nombre de logiciels installés par défaut = 0.

Vérifier la structure et le contenu de chaque table avec `DESCRIBE` et `SELECT`.

Exercice

3.2 Modification de colonnes

Dans ce même script, rajouter les instructions nécessaires pour :

- augmenter la taille dans la table `Salle` de la colonne `nomSalle` (passer à `VARCHAR(30)`) ;
- diminuer la taille dans la table `Segment` de la colonne `nomSegment` à `VARCHAR(15)` ;
- tenter de diminuer la taille dans la table `Segment` de la colonne `nomSegment` à `VARCHAR(14)`. Pourquoi la commande n'est-elle pas possible ?

Vérifier la structure et le contenu de chaque table avec `DESCRIBE` et `SELECT`.

Exercice

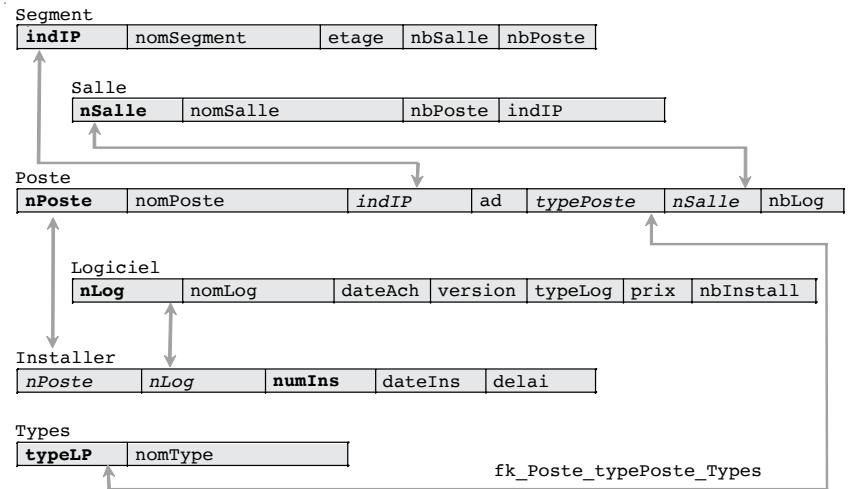
3.3 Ajout de contraintes

Ajouter la contrainte afin de s'assurer qu'on ne puisse installer plusieurs fois le même logiciel sur un poste de travail donné.

Ajouter les contraintes de clés étrangères pour assurer l'intégrité référentielle (avec `ALTER TABLE ... ADD CONSTRAINT...`) entre les tables suivantes. Adopter les conventions recommandées dans chapitre 1 (comme indiqué pour la contrainte entre `Poste` et `Types`).

Si l'ajout d'une contrainte référentielle renvoie une erreur, vérifier les enregistrements des tables « pères » et « fils » (notamment au niveau de la casse des chaînes de caractères, 'Tx' est différent de 'TX' par exemple).

Figure 3-10 Contraintes référentielles à créer



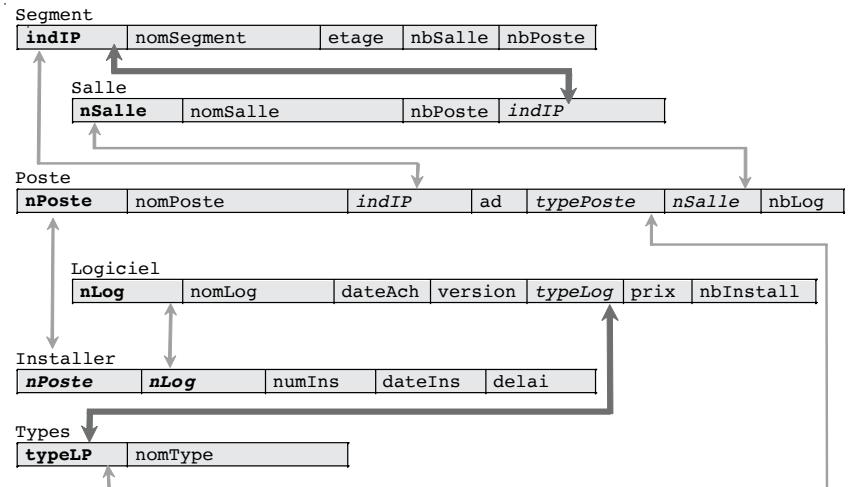
Modifier le script SQL de destruction des tables (dropParc.sql) en fonction des nouvelles contraintes. Lancer ce script puis tous ceux écrits jusqu'ici.

Exercice

3.4 Traitements des erreurs

Tentez d'ajouter les contraintes de clés étrangères entre les tables Salle et Segment et Logiciel et Types (en gras dans le schéma suivant).

Figure 3-11 Contraintes référentielles à créer



La mise en place de ces contraintes doit renvoyer une erreur car :

- Il existe des salles ('s22' et 's23') ayant un numéro de segment inexistant dans la table Segment.
- Il existe un logiciel ('log8') dont le type n'est pas référencé dans la table Types.

Extraire les enregistrements qui posent problème (numéro des salles pour le premier cas, numéro du logiciel pour le second). Supprimer les enregistrements de la table Salle qui posent problème. Ajouter le type de logiciel ('BeOS', 'Système Be') dans la table Types.

Exécuter à nouveau l'ajout des deux contraintes de clé étrangère. Vérifier que les instructions renvoient plus d'erreur et que les deux requêtes d'extraction ne renvoient aucune donnée.

Interrogation des données

Ce chapitre traite de l'aspect le plus connu du langage SQL qui concerne l'extraction de données par requêtes (nom donné aux instructions SELECT). Une requête permet de rechercher des données dans une ou plusieurs tables ou vues, à partir de critères simples ou complexes. Les instructions SELECT peuvent être exécutées dans l'interface de commande (voir les exemples de ce chapitre) ou au sein d'un programme SQL (procédure cataloguée), PHP, Java, C, etc.

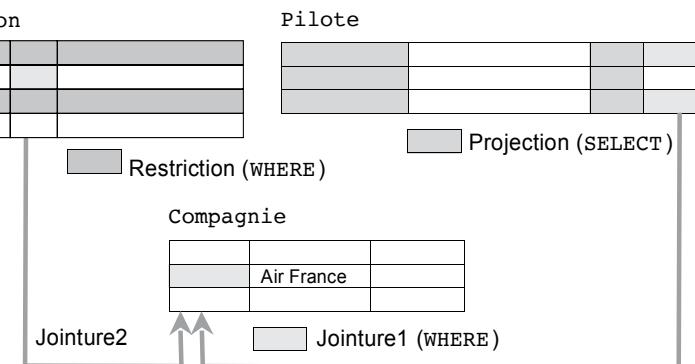
Généralités



L'instruction SELECT est une commande déclarative (elle décrit ce que l'on cherche sans spécifier le moyen d'opérer). À l'inverse, une instruction procédurale (comme un programme) développerait le moyen pour réaliser l'extraction de données (comme le chemin à emprunter entre des tables ou une itération pour parcourir un ensemble d'enregistrements).

La figure suivante schématise les principales fonctionnalités de l'instruction SELECT. Cette figure est composée d'une directive FROM qui précise la (les) table(s) interrogée(s), et d'une directive WHERE qui contient les critères.

Figure 4-1 Possibilités de l'instruction SELECT



- La restriction qui est programmée dans le WHERE de la requête permet de restreindre recherche à une ou plusieurs lignes. Dans notre exemple une restriction répond à 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 des 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 (aucune jointure ou opérateur ensembliste ne sont employés dans l'interrogation).

Tableau 4-1 Extraction d'expressions

Besoin	Requête et résultat
Aucun, utilisation probablement la plus superflue.	mysql> SELECT 'Il reste encore beaucoup de pages?'; +-----+ Il reste encore beaucoup de pages? +-----+ Il reste encore beaucoup de pages? +-----+
J'ai oublié ma montre !	mysql> SELECT SYSDATE() "Maintenant :"; +-----+ Maintenant : +-----+ 2005-11-07 09:15:46 +-----+
Pour les matheux qui voudraient retrouver le résultat de 2^{14} , le carré du cosinus de 3π sur 2 et e^1 .	mysql> SELECT POWER(2,14), POWER(COS(135*3.14159265359/180),2) "Environ" ,EXP(1); +-----+-----+-----+ POWER(2,14) Environ EXP(1) +-----+-----+-----+ 16384 0.50000000000015 2.718281828459 +-----+-----+-----+

Projection (éléments du SELECT)

Étudions la partie de l'instruction SELECT qui permet de programmer l'opérateur de projection (surligné dans la syntaxe suivante) :

```
SELECT [ { DISTINCT | DISTINCTROW } | ALL ] listeColonnes
  FROM nomTable [aliasTable]
  [ clauseOrdonnancement ]
  [ LIMIT [rangDépart,] nbLignes ];
```

- DISTINCT et DISTINCTROW jouent le même rôle, à savoir ne pas inclure les dupliques.
- ALL prend en compte les duplicitas (option par défaut).
- *listeColonnes* : { * | *expression1* [[AS] *alias1*] [, *expression2* [[AS] *alias2*] ...]}

 - * : extrait toutes les colonnes de la table.
 - *expression* : nom de colonne, fonction SQL, constante ou calcul.
 - *alias* : renomme l'expression (nom valable pendant la durée de la requête).

- FROM désigne la table (qui porte un alias ou non) à interroger.

- *clauseOrdonnancement* : tri sur une ou plusieurs colonnes ou expressions.
 - **LIMIT** pour limiter le nombre de lignes après résultat.
- Interrogeons la table suivante en utilisant chaque option:

Figure 4-2 Table Pilote

Pilote			
brevet	nom	nbHVol	compa
PL-1	Gratien Viel	450	AF
PL-2	Didier Donsez	0	AF
PL-3	Richard Grin	1000	SING
PL-4	Placide Fresnais	2450	CAST
PL-5	Daniel Vielle		AF

VARCHAR(6) VARCHAR(16) DECIMAL(7,2) CHAR(4)

Extraction de toutes les colonnes

L'extraction de toutes les colonnes d'une table requiert l'utilisation du symbole « * ».

Tableau 4-2 Utilisation de « * »

Requête SQL	Résultat
<pre>mysql> SELECT * FROM Pilote;</pre>	<pre>+-----+-----+-----+-----+ brevet nom nbHVol compa +-----+-----+-----+-----+ PL-1 Gratien Viel 450.00 AF PL-2 Didier Donsez 0.00 AF PL-3 Richard Grin 1000.00 SING PL-4 Placide Fresnais 2450.00 CAST PL-5 Daniel Vielle NULL AF +-----+-----+-----+-----+</pre>

Extraction de certaines colonnes

La liste des colonnes à extraire se trouve dans la clause **SELECT**.

Tableau 4-3 Liste de colonnes

Requête SQL	Résultat
<pre>SELECT compa, brevet FROM Pilote;</pre>	<pre>+-----+-----+ compa brevet +-----+-----+ AF PL-1 AF PL-2 SING PL-3 CAST PL-4 AF PL-5 +-----+-----+</pre>

Alias

Les alias permettent de renommer des colonnes à l'affichage ou des tables dans la requête. 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;
```



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 une 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.

Tableau 4-5 Gestion des duplicates

Web	Avec duplicita	Sans duplicita
	<pre>SELECT compa FROM Pilote;</pre> <pre>+-----+ compa +-----+ AF AF SING CAST AF +-----+</pre>	<pre>SELECT DISTINCT(compa) FROM Pilote;</pre> <pre>+-----+ compa +-----+ AF SING CAST +-----+</pre>

Expressions et valeurs nulles



Il est possible d'évaluer et d'afficher simultanément des expressions dans la clause SELECT (types numériques, DATE et DATETIME).

Les opérateurs arithmétiques sont évalués par ordre de priorité (*, /, + et -).

Le résultat d'une expression comportant une valeur NULL est évalué à NULL.

Nous avons déjà étudié les opérations sur les dates et intervalles (chapitre 2). Dans l'exemple suivant, l'expression $10 * nbHVol + 5 / 2$ est calculée en multipliant par 10 le nombre d'heures de vol, puis en ajoutant le résultat de 5 divisé par 2. Dans le second exemple, on convertit moment actuel (année, mois, jour, heures, minutes et secondes) en un entier.

Tableau 4-6 Expressions numériques

Web	Requête	Résultat
	<pre>SELECT brevet, nbHVol, nbHVol*nbHVol AS auCarre, 10*nbHVol+5/2 FROM Pilote;</pre>	<pre>+-----+-----+-----+-----+ brevet nbHVol auCarre 10*nbHVol+5/2 +-----+-----+-----+-----+ PL-1 450.00 202500.0000 4502.5000 PL-2 0.00 0.0000 2.5000 PL-3 1000.00 1000000.0000 10002.5000 PL-4 2450.00 6002500.0000 24502.5000 PL-5 NULL NULL NULL +-----+-----+-----+-----+</pre>
	<pre>SELECT SYSDATE() + 0 ;</pre>	<pre>+-----+ SYSDATE() + 0 +-----+ 20051107145522 +-----+</pre>

Ordonnancement

Pour trier le résultat d'une requête, il faut spécifier la clause d'ordonnancement par **ORDER BY** de la manière suivante :

```
ORDER BY
{ expression1 | position1 | alias1 } [ASC | DESC]
[,{expr2 | position2 | alias2} [ASC | DESC]
```

- **expression** : nom de colonne, fonction SQL, constante, calcul.
- **position** : entier qui désigne l'expression (au lieu de la nommer) dans son ordre d'ordonnancement dans la clause **SELECT**.
- ASC ou DESC : tri ascendant ou descendant (par défaut ASC).

Dans l'exemple suivant, on remarque que la valeur NULL est considérée comme plus grande que 0.

Tableau 4-7 Ordonnancement

Web	Options par défaut	Ordre décroissant (et NULL)
	mysql> SELECT brevet, nom FROM Pilote ORDER BY nom;	mysql> SELECT brevet, nbHVol FROM Pilote ORDER BY nbHvol DESC
	+-----+-----+ brevet nom +-----+-----+ PL-5 Daniel Vielle PL-2 Didier Donsez PL-1 Gratien Viel PL-4 Placide Fresnais PL-3 Richard Grin +-----+-----+	+-----+-----+ brevet nbHVol +-----+-----+ PL-4 2450.00 PL-3 1000.00 PL-1 450.00 PL-2 0.00 PL-5 NULL +-----+-----+

Concaténation

L'opérateur de concaténation se programme à l'aide de la fonction **CONCAT** qui admet plusieurs chaînes de caractères en paramètre. Cette fonction permet de concaténer différentes expressions (colonnes, calculs, résultats de fonctions SQL ou constantes), sous réserve d'éventuelles conversions (*casting*). La colonne résultante est considérée comme une chaîne de caractères.

L'exemple suivant présente un alias dans l'en-tête de colonne ("Embauche") qui modifie les résultats. La concaténation concerne deux colonnes et la constante "vole pour".

Tableau 4-8 Concaténation

Web	Requête	Résultat
		+-----+
		brevet Embauche
	SELECT brevet,	+-----+
	CONCAT (nom, ' vole pour ',compa)	PL-1 Gratien Viel vole pour AF
	AS "Embauche" FROM Pilote;	PL-2 Didier Donsez vole pour AF
		PL-3 Richard Grin vole pour SING
		PL-4 Placide Fresnais vole pour CAST
		PL-5 Daniel Vielle vole pour AF
		+-----+

Insertion multiligne

Nous pouvons maintenant décrire l'insertion multiligne évoquée au chapitre précédent. Dans l'exemple suivant, il s'agit d'insérer tous les pilotes de la table Pilote (en considérant nom, le nombre d'heures de vol et la compagnie) dans la table NomsetHVoldesPilotes

Tableau 4-9 Insertion multiligne

Web	Création et insertion	Requête et résultat
	<pre>CREATE TABLE NomsetHVoldesPilotes (nom VARCHAR(16), nbHVol DECIMAL(7,2), compa CHAR(4)); INSERT INTO NomsetHVoldesPilotes SELECT nom,nbHVol,compa FROM Pilote;</pre>	<pre>mysql> SELECT * FROM NomsetHVoldesPilotes +-----+-----+-----+ nom nbHVol compa +-----+-----+-----+ Gratien Viel 450.00 AF Didier Donsez 0.00 AF Richard Grin 1000.00 SING Placide Fresnais 2450.00 CAST Daniel Vielle NULL AF +-----+-----+-----+</pre>

Notez que les instructions (CREATE TABLE et INSERT...) peuvent être programmées en une instruction (option AS SELECT de la commande CREATE TABLE) :

```
CREATE TABLE NomsetHVoldesPilotes AS SELECT nom, nbHVol, compa
  FROM Pilote;
```

Limitation du nombre de lignes

Pour limiter le nombre de lignes à extraire à partir du résultat d'une requête, il faut spécifier la clause LIMIT de la manière suivante :

```
LIMIT [rangDépart,] nbLignes
```

Le premier entier précise le rang de la première ligne sélectionnée (en fonction du tri du résultat). Le second entier indique le nombre maximum de lignes à extraire. La première ligne considérée comme présente au rang 0. Ainsi « `LIMIT n` » équivaut à « `LIMIT 0, n` ». L'exemple suivant illustre deux utilisations de la clause `LIMIT` :

Tableau 4-10 Limitation des résultats

Web	Requête	Résultat
	Deuxième et troisième (ordre de clé) pilote.	+-----+-----+-----+-----+ brevet nom nbHVol compa +-----+-----+-----+-----+ PL-2 Didier Donsez 0.00 AF PL-3 Richard Grin 1000.00 SING +-----+-----+-----+-----+
	<code>SELECT * FROM Pilote LIMIT 1,2;</code>	
	Les deux pilotes les plus expérimentés (par ordre du nombre d'heures de vol).	+-----+-----+-----+-----+ brevet nom nbHVol compa +-----+-----+-----+-----+ PL-4 Placide Fresnais 2450.00 CAST PL-3 Richard Grin 1000.00 SING +-----+-----+-----+-----+
	<code>SELECT * FROM Pilote ORDER BY nbHVol DESC LIMIT 2;</code>	

Restriction (WHERE)

Les éléments de la clause `WHERE` d'une requête permettent de programmer l'opérateur de restriction. Cette clause limite la recherche aux enregistrements qui respectent une condition simple ou complexe. Cette section s'intéresse à la partie surlignée de l'instruction `SELECT` suivante :

```
| SELECT [ { DISTINCT | DISTINCTROW } | ALL ] listeColonnes  
|   FROM nomTable [aliasTable]  
|   [ WHERE condition ] ;
```

- *condition* composée de colonnes, d'expressions, de constantes liées deux à deux par des opérateurs :
 - de comparaison (`>`, `=`, `<`, `>=`, `<=`, `<>`) ;
 - logiques (NOT, AND ou OR) ;
 - intégrés (BETWEEN, IN, LIKE, IS NULL).

Interrogeons la table suivante en utilisant chaque catégorie d'opérateur :

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 Vieille	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 utilisées 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

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 "de Air-France"
```

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

brevet	de Air-France
PL-1	Gratien Viel
PL-2	Didier Donsez

```
SELECT brevet, nom, prime FROM Pilote
```

```
WHERE prime <> 500 ;
```

brevet	nom	prime
PL-3	Richard Grin	90
PL-5	Daniel Vieille	600
PL-6	Francoise Tort	0

Opérateurs logiques



- L'ordre de priorité des opérateurs logiques est NOT, AND et OR.
- Les opérateurs de comparaison ($>$, $=$, $<$, \geq , \leq , \neq) sont prioritaires par rapport à NOT.
- Les parenthèses permettent de modifier ces règles de priorité.

La première requête de l'exemple suivant contient une condition composée de trois prédictifs qui sont évalués par ordre de priorité (d'abord AND puis OR). La conséquence est l'affichage des pilotes de la compagnie 'SING' avec les pilotes de 'AF' ayant moins de 500 heures de vol.

La deuxième requête force la priorité avec les parenthèses (AND et OR sur le même prédictif d'égalité). La conséquence est l'affichage des pilotes ayant moins de 500 heures de vol des deux compagnies 'SING' et 'AF'.

Tableau 4-12 Opérateurs logiques

Requête	Résultat sous SQL*Plus
<p>Web</p> <pre>SELECT brevet, nom, compa FROM Pilote WHERE (compa = 'SING' OR compa = 'AF' AND nbHVol < 500);</pre>	<pre>+-----+-----+-----+ brevet nom compa +-----+-----+-----+ PL-1 Gratien Viel AF PL-2 Didier Donsez AF PL-3 Richard Grin SING PL-5 Daniel Vielle SING +-----+-----+-----+</pre>
<pre>SELECT brevet, nom, compa FROM Pilote WHERE ((compa = 'SING' OR compa = 'AF') AND nbHVol < 500);</pre>	<pre>+-----+-----+-----+ brevet nom compa +-----+-----+-----+ PL-1 Gratien Viel AF PL-2 Didier Donsez AF PL-5 Daniel Vielle SING +-----+-----+-----+</pre>

Opérateurs intégrés

Les opérateurs intégrés sont BETWEEN, IN, LIKE et IS NULL.

Tableau 4-13 Opérateurs intégrés

Opérateur	Exemple
Web BETWEEN <i>limiteInf</i> AND <i>limiteSup</i> teste l'appartenance à un intervalle de valeurs.	SELECT brevet, nom, nbHVol FROM Pilote WHERE nbHVol BETWEEN 399 AND 1000; +-----+-----+-----+ brevet nom nbHVol +-----+-----+-----+ PL-1 Gratien Viel 450.00 PL-3 Richard Grin 1000.00 PL-5 Daniel Vielle 400.00 +-----+-----+-----+
IN (<i>listeValeurs</i>) compare une expression avec une liste de valeurs.	SELECT brevet, nom, compa FROM Pilote WHERE compa IN ('CAST', 'SING'); +-----+-----+-----+ brevet nom compa +-----+-----+-----+ PL-3 Richard Grin SING PL-4 Placide Fresnais CAST PL-5 Daniel Vielle SING PL-6 Francoise Tort CAST +-----+-----+-----+
LIKE (<i>expression</i>) compare de manière générique des chaînes de caractères à une expression. Le symbole % remplace un ou plusieurs caractères. Le symbole _ remplace un caractère. Ces symboles peuvent se combiner. Utilisez de préférence des colonnes VARCHAR ou complétez si nécessaire par des blancs jusqu'à la taille maximale pour des CHAR.	SELECT brevet, nom, compa FROM Pilote WHERE compa LIKE ('%A%'); +-----+-----+-----+ brevet nom compa +-----+-----+-----+ PL-1 Gratien Viel AF PL-2 Didier Donsez AF PL-4 Placide Fresnais CAST PL-6 Francoise Tort CAST +-----+-----+-----+ SELECT brevet, nom, compa FROM Pilote WHERE compa LIKE ('A_'); +-----+-----+-----+ brevet nom compa +-----+-----+-----+ PL-1 Gratien Viel AF PL-2 Didier Donsez AF +-----+-----+-----+
IS NULL compare une expression (colonne, calcul, constante) à la valeur NULL. La négation s'écrit soit « <i>expression</i> IS NOT NULL » soit « NOT (<i>expression</i> IS NULL) ».	SELECT nom, prime, nbHVol FROM Pilote WHERE prime IS NULL OR nbHVol IS NULL +-----+-----+-----+ nom prime nbHVol +-----+-----+-----+ Didier Donsez NULL 0.00 Francoise Tort 0 NULL +-----+-----+-----+



Il n'est pas permis d'utiliser un alias de colonne dans la clause WHERE. Cette recommandation de la norme s'explique par le fait que certaines expressions pourraient ne pas être déterminées pendant que la condition WHERE est évaluée.

Ainsi, la requête suivante renvoie une erreur alors qu'elle ne contient pourtant pas d'expression litigieuse. Il faudra ici remplacer « c1 » par « aliasDesPilotes.compa ».

```
mysql> SELECT aliasDesPilotes.compa AS c1, aliasDesPilotes.nom  
        FROM Pilote aliasDesPilotes WHERE c1 = 'AF';  
ERROR 1054 (42S22): Unknown column 'c1' in 'where clause'
```

Fonctions

MySQL propose un grand nombre de fonctions qui s'appliquent dans les clauses SELECT et WHERE d'une requête. La syntaxe générale d'une fonction est la suivante :



```
nomFonction(colonne1 | expression1 [,colonne2 | expression2 ...])
```

- Une fonction monoligne agit sur une ligne à la fois et ramène un résultat par ligne. On distingue quatre familles de fonctions monolignes : caractères, numériques, dates et conversions de types de données. Ces fonctions peuvent se combiner entre elles (exemples : MAX(COS(ABS(n))) désigne le maximum des cosinus de la valeur absolue de la colonne n).
- Une fonction multiligne (fonction d'agrégat) agit sur un ensemble de lignes pour ramener un résultat (voir la section *Regroupements*).

Caractères

Interrogeons la table suivante en utilisant des fonctions pour les caractères :

Figure 4-4 Table Pilote

Pilote

brevet	prenom	nom	surnom	compa
PL-1	Gratien	viel	dba	AF
PL-2	Didier	donsez	smith	AF
PL-3	richard	Grin	Faucon	SING
PL-4	placide	Fresnais	cool	CAST
PL-5	Daniel	vienne	jones	SING
PL-6	Francoise	tort	NormaleSup	CAST

La plupart des fonctions pour les caractères acceptent une chaîne de caractères en paramètre de nature CHAR ou VARCHAR. Le tableau suivant décrit les principales fonctions :

Tableau 4-14 Fonctions pour les caractères

Fonction	Objectif	Exemple			
<code>ASCII(c)</code>	Retourne le caractère ASCII équivalent.	<code>ASCII('A')</code> donne 65			
<code>CHAR(n)</code>	Retourne le caractère équivalent dans le jeu de caractères en cours.	<code>CHR(161) CHR(162)</code> donne 16			
<code>CONCAT(c1, c2)</code>	Concatène deux chaînes.	<pre>SELECT CONCAT(CONCAT(nom, ' vole pour '), compa) "Personnel" FROM Pilote;</pre> <table border="1"> <tr><td> Personnel</td></tr> <tr><td> ...</td></tr> <tr><td> viel travaille pour AF</td></tr> </table>	Personnel	...	viel travaille pour AF
Personnel					
...					
viel travaille pour AF					
<code>FIELD(c, c1, c2...)</code>	Retourne l'index qui correspond à la première égalité entre <i>c</i> et <i>c1</i> , <i>c</i> et <i>c2</i> , etc. 0 si aucune égalité n'est trouvée.	<pre>SELECT FIELD('Air', 'air', 'Airbus', 'Air') "Attention à la casse!";</pre> <table border="1"> <tr><td> Attention à la casse!</td></tr> <tr><td> 1</td></tr> </table>	Attention à la casse!	1	
Attention à la casse!					
1					
<code>INSERT(c1, pos, t, c2)</code>	Modifie la chaîne <i>c1</i> en insérant <i>t</i> caractères de la sous-chaîne <i>c2</i> à partir de la position <i>pos</i> .	<pre>SELECT INSERT('Compxxxie : Airbus ', 5, 3, 'agn') "Qui?";</pre> <table border="1"> <tr><td> Qui?</td></tr> <tr><td> Compagnie : Airbus</td></tr> </table>	Qui?	Compagnie : Airbus	
Qui?					
Compagnie : Airbus					
<code>INSTR(c1, c2)</code>	Premier indice d'une sous-chaîne <i>c1</i> dans une chaîne <i>c2</i> . Exemple : indice de 'Air' dans la chaîne.	<pre>SELECT INSTR('Infos-Air : AirBus pour Air-France', 'Air') "Indice";</pre> <table border="1"> <tr><td> Indice</td></tr> <tr><td> 7</td></tr> </table>	Indice	7	
Indice					
7					

Tableau 4-14 Fonctions pour les caractères (suite)

Fonction	Objectif	Exemple
LOWER(<i>c</i>)	Tout en minuscules.	<pre>SELECT CONCAT(LOWER(prenom) , ' ', LOWER(nom)) "Etat civil" FROM Pilote WHERE compa = 'SING'; +-----+ Etat civil +-----+ richard grin daniel vielle </pre>
LOCATE(<i>c1,c2,pos</i>)	Premier indice d'une sous-chaîne <i>c1</i> dans une chaîne <i>c2</i> à partir de la position <i>pos</i> . Exemple : indice de 'Air' dans la chaîne à partir du 9 ^e caractère.	<pre>SELECT LOCATE('Air','Infos-Air : Airbus pour Air-France',9) "Indice après 9"; +-----+ Indice après 9 +-----+ 13 +-----+</pre>
LENGTH(<i>c</i>)	Longueur de la chaîne.	<pre>SELECT LENGTH('Infos-Air : Airbus pour Air-France') "Taille"; +-----+ Taille +-----+ 35 +-----+</pre>
LEFT(<i>c,n</i>)	Extrait les <i>n</i> premiers caractères à <i>c</i> en partant de la gauche.	<pre>SELECT LEFT('A380 à BlagnacB747B747','Bye les Jumbo'; +-----+ Bye les Jumbo +-----+ A380 à Blagnac +-----+</pre>
LPAD(<i>c1,n,c2</i>)	Insertion à gauche de <i>c2</i> dans <i>c1</i> sur <i>n</i> caractères.	<pre>SELECT LPAD('Rien',20,'---') "sur 20" +-----+ sur 20 +-----+ ---Rien +-----+</pre>
REPLACE(<i>c1,c2,c3</i>)	Recherche les <i>c2</i> présentes dans <i>c1</i> et les remplace par <i>c3</i> .	<pre>SELECT REPLACE('Matra et Aerospatiale', 'Matra', 'EADS') "Changement"; +-----+ Changement +-----+ EADS et Aerospatiale +-----+</pre>

Tableau 4-14 Fonctions pour les caractères (suite)

Fonction	Objectif	Exemple
REVERSE(<i>c</i>)	Retourne la chaîne renversée.	<pre>SELECT REVERSE('cangalB à 083A') "Miroir"; +-----+ Miroir +-----+ A380 à Blagnac +-----+</pre>
RIGHT(<i>c,n</i>)	Extrait les <i>n</i> derniers caractères à <i>c</i> en partant de la droite.	<pre>SELECT RIGHT('B747B747A380 à Blagnac',14) "Bye les Jumbo"; +-----+ Bye les Jumbo +-----+ A380 à Blagnac +-----+</pre>
RPAD(<i>c1,n,c2</i>)	Insertion à droite de <i>c2</i> dans <i>c1</i> sur <i>n</i> caractères.	<pre>SELECT RPAD('Rien',19,'---') "sur 19"; +-----+ sur 19 +-----+ Rien---.---.---.--- +-----+</pre>
SOUNDEX(<i>c</i>)	Extrait la phonétique d'une expression (<i>in english only</i>).	<pre>SELECT nom, surnom, compa FROM Pilote WHERE SOUNDEX(surnom) IN (SOUNDEX('SMYTHE'), SOUNDEX('John')); +-----+-----+-----+ nom surnom compa +-----+-----+-----+ donsez smith AF vielle jone SING +-----+-----+-----+</pre>
SUBSTR(<i>c,n,[t]</i>)	Extraction de la sous-chaîne <i>c</i> commençant à la position <i>n</i> sur <i>t</i> caractères.	<pre>SELECT SUBSTR('Air France à Blagnac Con!',12,9) "Où ça?"; +-----+ Où ça? +-----+ à Blagnac +-----+</pre>

Tableau 4-14 Fonctions pour les caractères (suite)

Fonction	Objectif	Exemple
TRIM(<i>c1</i> FROM <i>c2</i>)	Enlève les caractères <i>c1</i> à la chaîne <i>c2</i> (options LEADING et TRAILING pour préciser le sens du découpage). Existent aussi LTRIM et RTRIM qui enlèvent des espaces respectivement au début ou à la fin d'une chaîne.	SELECT TRIM('B' FROM 'BA380 à BlagnacBBBB') "Bye les Jumbo"; +-----+ Bye les Jumbo +-----+ A380 à Blagnac +-----+
UPPER	Tout en majuscules.	SELECT CONCAT(UPPER(prenom) , ' ', UPPER(nom)) "Pilotes de CAST" FROM Pilote WHERE compa = 'CAST'; +-----+ Pilotes de CAST +-----+ PLACIDE FRESNAIS FRANCOISE TORT +-----+

Numériques

En plus des opérateurs arithmétiques disponibles dans les langages de programmation (+, -, et DIV pour la division entière), MySQL supporte un grand nombre de fonctions numériques.

Tableau 4-15 Fonctions numériques

Fonction	Objectif	Exemple
ABS(<i>n</i>)	Valeur absolue de <i>n</i> .	
ACOS(<i>n</i>)	Arc cosinus (<i>n</i> de -1 à 1), retour exprimé en radians (de 0 à pi).	
ATAN(<i>n</i>)	Arc tangente ($\forall n$), retour exprimé en radians (de -pi/2 à pi/2).	
CEIL(<i>n</i>)	Plus petit entier \geq à <i>n</i> .	CEIL(15.7) retourne 16 .
COS(<i>n</i>)	Cosinus de <i>n</i> exprimé en radians de 0 à 2 pi .	COS(60*PI()/180) retourne 0.5
COT(<i>n</i>)	Cotangente de <i>n</i> exprimée en radians.	COT(30*PI()/180) retourne 1.7320508075689.
DEGREES(<i>n</i>)	Conversion de radians en degrés.	DEGREES(PI()/2) retourne 90.

Tableau 4-15 Fonctions numériques (suite)

Fonction	Objectif	Exemple
<code>EXP(n)</code>	e (2.71828183) à la puissance n .	
<code>FLOOR(n)</code>	Plus grand entier \leq à n .	<code>FLOOR(15.7)</code> retourne 15.
<code>LN(n)</code>	Logarithme népérien de n .	
<code>LOG(n) (m,n)</code>	Logarithme de n dans une base m .	
<code>MOD(m,n)</code>	Reste de la division entière de m par n .	
<code>POW(m,n)</code>	m puissance n .	
<code>RADIANS(n)</code>	Conversion de degrés en radians.	<code>RADIANS(90)</code> retourne 1.5707963267949.
<code>RAND()</code>	Flottant aléatoire (à 14 décimales) entre 0 et 1.	<code>ROUND(17.567,2)</code> retourne 17,57.
<code>ROUND(m,n)</code>	Arrondi à une ou plusieurs décimales.	<code>ROUND(17.567,2)</code> retourne 17,57.
<code>SIGN(n)</code>	Retourne le signe d'un nombre (-1, 0 ou 1).	
<code>SIN(n)</code>	Sinus de n exprimé en radians de 0 à 2 pi.	<code>SIN(30*PI() /180)</code> retourne 0.5.
<code>SINH(n)</code>	Sinus hyperbolique de n .	
<code>SQRT(n)</code>	Racine carrée de n .	
<code>TAN(n)</code>	Tangente de n exprimée en radians de 0 à 2 pi.	
<code>TRUNCATE(n,m)</code>	Coupure de n à m décimales.	<code>TRUNC(15.79,1)</code> retourne 15.7.

Fonction pour les bits

Les opérateurs suivants sont disponibles jusqu'à 64 bits (`BIGINT`). On peut en utiliser certains en passant des paramètres en base 10, en binaire ou de type chaîne de caractères.

Tableau 4-16 Fonctions pour les bits

Fonction	Objectif	Exemple
<code>OR : </code>	OU bits à bits.	<code>b'0100' b'1100'</code> retourne 12.
<code>AND : &</code>	ET bits à bits.	<code>b'0100' & b'1100'</code> retourne 4.
<code>XOR : ^</code>	OU exclusif bits à bits.	<code>b'0100' ^ b'1100'</code> retourne 8.
<code>SHL : <<</code>	Décalage à gauche de n positions.	<code>3 << 2</code> retourne 12.
<code>SHR : >></code>	Décalage à droite de n positions.	<code>b'0100' >> 2</code> retourne 1.
<code>Complément à 1 : ~</code>	Inversion de chaque bit.	<code>3+(-3+1)</code> retourne 1 (ici on programme le complément à 2).
<code>BIN(n)</code>	Chaîne qui représente la valeur binaire de n .	<code>BIN(12)</code> retourne '1100'.

Tableau 4-16 Fonctions pour les bits (suite)

Fonction	Objectif	Exemple
BIT_LENGTH(<i>c</i>)	Taille de la chaîne en bits.	BIT_LENGTH('GTR') retourne (3 octets).
HEX(<i>ns</i>)	Chaîne en hexadécimal représentant <i>ns</i> (nombre ou chaîne).	HEX(254) retourne 'FE'.
OCT(<i>n</i>)	Chaîne en octal représentant <i>n</i> .	OCT(12) retourne 14.
OCTET_LENGTH(<i>c</i>)	Synonyme de LENGTH().	
UNHEX(<i>c</i>)	Fonction inverse de HEX.	UNHEX('53514C') retourne 'S

Dates

Le tableau suivant décrit les principales fonctions pour les dates :

Tableau 4-17 Fonctions pour les dates

Fonction	Objectif	Retour
ADDDATE(<i>date</i> , <i>n</i>)	Ajoute <i>n</i> jours à une date (heure).	DATE ou DATETIME
ADDTIME(<i>date1</i> , <i>date2</i>)	Ajoute les deux dates avec <i>date1</i> TIME ou DATETIME, et <i>date2</i> TIME.	TIME ou DATETIME
CURDATE(), CURRENT_DATE ou CURRENT_DATE()	Date courante ('YYYY-MM-DD' ou YYYYMMDD).	INT ou DATE
CURTIME(), CURRENT_TIME ou CURRENT_TIME()	Heure courante ('HH:MM:SS' or HHMMSS).	INT ou DATE
CURRENT_TIMESTAMP, CURRENT_TIMESTAMP() ou NOW()	Date et heure courantes ('YYYY-MM-DD HH:MM:SS' ou YYYYMMDDHHMMSS).	INT ou DATETIME
DATE(<i>datet</i>)	Extrait une date à partir d'une expression de type DATETIME.	DATE
DATEDIFF(<i>date1</i> , <i>date2</i>)	Nombre entier de jours entre les 2 dates.	INT
DATE_ADD(<i>date</i> , INTERVAL <i>expr</i> <i>type</i>)	Ajoute un intervalle à une date (heure). <i>expr</i> désigne un intervalle. <i>type</i> indique comment interpréter le format de l'expression (voir tableau suivant).	DATE ou DATETIME
DATE_FORMAT(<i>date</i> , <i>format</i>)	Présente la date selon un format (voir tableau suivant).	VARCHAR
DATE_SUB(<i>date</i> , INTERVAL <i>expr</i> <i>type</i>)	Soustrait un intervalle à une date (heure). Mêmes paramètres que DATE_ADD.	DATE ou DATETIME
DAYNAME(<i>date</i>)	Nom du jour en anglais.	VARCHAR
DAY(<i>date</i>) ou DAYOFMONTH(<i>date</i>)	Numéro du jour dans le mois (0 à 31).	INT
DAYOFYEAR(<i>date</i>)	Numéro du jour dans l'année (0 à 366).	INT

Tableau 4-17 Fonctions pour les dates (suite)

Fonction	Objectif	Retour
<code>EXTRACT(type FROM date)</code>	Extrait une partie d'une date selon un type d'intervalle (comme pour DATE_ADD).	INT
<code>FROM_DAYS(n)</code>	Retourne une date à partir d'un nombre de jours (le calendrier année 0 débute à n=365).	DATE
<code>FROM_UNIXTIME(unix[,format])</code>	Retourne une date (heure) à partir d'une estampille Unix (nombre de jours depuis le 1/1/1970). Utilisation possible d'un format.	INT ou DATETIME
<code>HOUR(time)</code>	Extrait l'heure d'un temps.	INT
<code>LAST_DAY(date)</code>	Dernier jour du mois d'une date (heure).	DATE
<code>LOCALTIME, LOCALTIME(), LOCALTIMESTAMP, LOCALTIMESTAMP()</code>	Synonymes de NOW().	
<code>MAKEDATE(annee,njour)</code>	Construit une date à partir d'une année et d'un nombre de jours (>0, si njour>365, l'année s'incrémentera automatiquement).	DATE
<code>MAKETIME(heure,minute,seconde)</code>	Construit une heure.	TIME
<code>MICROSECOND(date)</code>	Extrait les microsecondes d'une date-heure.	INT
<code>MINUTE(time)</code>	Extrait les minutes d'un temps.	INT
<code>MONTH(date), MONTHNAME(date)</code>	Retourne respectivement le numéro et le nom du mois d'une date-heure.	INT, VARCHAR
<code>NOW()</code>	Date et heure courantes au format 'YYYY-MM-DD HH:MM:SS' ou YYYYMMDDHHMMSS.	DATETIME ou INT
<code>PERIOD_DIFF(int1,int2)</code>	Nombre de mois séparant les deux dates au format YYMM or YYYYMM.	INT
<code>SECOND(time)</code>	Extrait les secondes d'un temps.	INT
<code>SEC_TO_TIME(secondes)</code>	Construit une heure au format 'HH:MM:SS' ou HHMMSS.	TIME ou INT
<code>STR_TO_DATE(c,format)</code>	Construit une date (heure) selon un certain format. C'est l'inverse de DATE_FORMAT().	DATE ou DATETIME ou TIME
<code>SUBDATE(date,n)</code>	Retranche n jours à une date (heure).	DATE ou DATETIME
<code>SUBTIME(date1,date2)</code>	Retranche date2 (TIME) à date1 (TIME ou DATETIME).	TIME ou DATETIME
<code>SYSDATE()</code>	Date et heure courantes au format 'YYYY-MM-DD HH:MM:SS' ou YYYYMMDDHHMMSS (différence avec NOW voir chapitre 1).	DATETIME ou INT
<code>TIME(datetime)</code>	Extrait le temps d'une date-heure.	TIME

Tableau 4-17 Fonctions pour les dates (suite)

Fonction	Objectif	Retour
<code>TIMEDIFF(tdate1, tdate2)</code>	Temps entre 2 temps ou 2 dates ou 2 dates-heure.	TIME
<code>TIMESTAMP(date)</code>	Construit une estampille à partir d'une date (heure).	TIMESTAM
<code>TIMESTAMPADD(intervalle, int, date)</code>	Ajoute à la date (heure) un intervalle (<i>int</i>) du type FRAC_SECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, ou YEAR.	TIMESTAM
<code>TIMESTAMPDIFF(intervalle, int, date)</code>	Retranche à la date (heure) un intervalle du type (idem précédent).	TIMESTAM
<code>TIME_TO_SEC(time)</code>	Retourne le nombre de secondes équivalent au temps.	INT
<code>TO_DAYS(date)</code>	Retourne un nombre de jours à partir d'une date ('YYYY-MM-DD' ou YYYYMMDD). Inverse de <code>FROM_DAYS()</code> .	INT
<code>UNIX_TIMESTAMP(date)</code>	Retourne le nombre de secondes depuis le 1/1/1970 jusqu'à la date (heure) passée en paramètre (ou entier au format YYMMDD YYYYMMDD). Inverse de <code>FROM_UNIXTIME()</code> .	INT
<code>UTC_DATE(), UTC_TIME(), UTC_TIMESTAMP()</code>	Retournent respectivement la date, l'heure et l'estampille au méridien de Greenwich.	DATE, TIME, DATETIME
<code>WEEKDAY(date)</code>	Numéro du jour (0 : <i>lundi</i> , 1 : <i>mardi</i> , ... 6 : <i>dimanche</i>) d'une date (heure).	INT
<code>WEEKOFYEAR(date)</code>	Numéro de la semaine en cours (1 à 53).	INT

Tableau 4-18 Paramètres d'intervalle pour les fonctions DATE_ADD et DATE_SUB

Paramètre type	Paramètre expr
MICROSECOND	<i>n</i>
SECOND	<i>n</i>
MINUTE	<i>n</i>
HOUR	<i>nn</i>
DAY	<i>nn</i>
WEEK	<i>n</i>
MONTH	<i>nn</i>
YEAR	<i>nnnn</i>
SECOND_MICROSECOND	'ss.microsec'
MINUTE_MICROSECOND	'mi.microsec'
MINUTE_SECOND	'mi:ssS'
HOUR_MICROSECOND	'hh.microsec'
HOUR_SECOND	'hh:mi:ss'
HOUR_MINUTE	'hh:mi'
DAY_MICROSECOND	'dd.microsec'
DAY_SECOND	'dd hh:mi:ss'
DAY_MINUTE	'dd hh:mi'
DAY_HOUR	'dd hh'
YEAR_MONTH	'yyyy-mm'

Tableau 4-19 Principaux formats pour les fonctions DATE_FORMAT et STR_TO_DATE

Format	Description
%a	Nom du jour en anglais abrévié (Sun..Sat)
%b	Nom du mois en anglais abrévié (Jan..Dec)
%c	Mois (0..12)
%e	Jour du mois (0..31)
%f	Microsecondes (000000..999999)
%H	Heures (00..23)
%i	Minutes (00..59)
%j	Jour de l'année (001..366)
%M	Nom du mois en anglais (January..December)
%s	Secondes (00..59)
%T	Time sur 24 heures (hh:mm:ss)
%u	Numéro de semaine (00..53)
%W	Nom du jour en anglais (Sunday..Saturday)
%w	Jour de la semaine (0=Sunday..6=Saturday)
%Y	Année sur 4 positions

Quelques exemples d'utilisation (date du jour : mercredi 9 novembre 2005) sont donnés le tableau suivant :

Tableau 4-20 Exemples de fonctions pour les dates

Web	Besoin et fonction	Résultat
	Date dans 31 jours. SELECT ADDDATE ('2005-11-9', 31);	+-----+ ADDDATE('2005-11-9', 31) +-----+ 2005-12-10 +-----+
	1 jour et 1 microseconde après le 9/11/2005, 11 heures, 1 microseconde. SELECT ADDTIME ('2005-11-09 22:59:59.999999','1 0:0:0.000001') "exemple ADDTIME";	+-----+ exemple ADDTIME +-----+ 2005-11-10 23:00:00.000000 +-----+
	Rendez-vous dans 4 mois. SELECT DATE_ADD (CURRENT_TIMESTAMP, INTERVAL '4' MONTH) "RDV";	+-----+ RDV +-----+ 2006-03-09 17:07:33 +-----+
	Rendez-vous dans 7 jours, 1 heure et 30 minutes. SELECT DATE_ADD (CURRENT_TIMESTAMP, INTERVAL '7 01:30:00' DAY_SECOND) "RDV 1sem 1h30";	+-----+ RDV 1sem 1h30 +-----+ 2005-11-16 18:53:03 +-----+
	Aujourd'hui en anglais. SELECT DATE_FORMAT (SYSDATE(), '%w %d %M %Y') "%Y') "Today in English";	+-----+ Today in English +-----+ Wednesday 09 November 2005 +-----+
	Extraction au format numérique du jour, heures et minutes. SELECT EXTRACT (DAY_MINUTE FROM '2005-11-09 01:02:03') "DAY_MINUTE";	+-----+ DAY_MINUTE +-----+ 90102 +-----+

Conversions

MySQL autorise des conversions de types implicites ou explicites.

Implicites



Il est possible d'affecter, dans une expression ou dans une instruction SQL (INSERT, UPDATE...), une donnée de type numérique (ou date-heure) à une donnée de type VARCHAR (ou CHAR). Il en va de même pour l'affectation d'une colonne VARCHAR par une donnée de type date-heure (ou numérique). On parle ainsi de conversions implicites.

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
<code>BINARY(expr)</code>	L'expression en bits.	Pour le premier pilote de notre dernier exemple, le test <code>BINARY(brevet) = BINARY('pl-1')</code> renverra faux.
<code>CAST(expression AS typeMySQL)</code>	L'expression dans le type en paramètre (BINARY, CHAR, DATE, DATETIME, DECIMAL, SIGNED, TIME, UNSIGNED).	<code>CAST(2 AS CHAR)</code> retourne '2'.
<code>CONVERT(c, jeu-car)</code>	La chaîne <i>c</i> dans le jeu de caractères passé en paramètre.	<code>CONVERT('Ã È Í Ø' USING cp850)</code> jeu de caractère DOS, retourne "? È 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).

Énumérations

Nous avons vu au chapitre 2 comment manipuler les deux types d'énumérations que MySQL propose (ENUM et SET). Étudions à présent quelques fonctions relatives à ces types.

Type ENUM

Chaque valeur de l'énumération est associée à un indice commençant à 1. Ainsi il est possible de retrouver la position d'une valeur au sein de son énumération comme l'illustre l'exemple suivant, décrit au chapitre 2.

Tableau 4-22 Exemples d'extraction d'indices d'une énumération ENUM

Table et données			Extraction									
Figure 4-5 Table et données colonne ENUM			<pre>mysql> SELECT nom, diplome, diplome+ FROM UnCursus ;</pre>									
UnCursus			<table border="1"><thead><tr><th>nom</th><th>diplome</th><th>diplome+</th></tr></thead><tbody><tr><td>F. Brouard</td><td>BTS</td><td>1</td></tr><tr><td>F. Degrelle</td><td>Licence</td><td>3</td></tr></tbody></table>	nom	diplome	diplome+	F. Brouard	BTS	1	F. Degrelle	Licence	3
nom	diplome	diplome+										
F. Brouard	BTS	1										
F. Degrelle	Licence	3										
ENUM			<table border="1"><thead><tr><th>ENUM</th></tr></thead><tbody><tr><td>BTS, DUT, Licence INSA</td></tr></tbody></table>	ENUM	BTS, DUT, Licence INSA							
ENUM												
BTS, DUT, Licence INSA												

L'indice d'une valeur vide (colonne valuée à "") ou " " dans l'INSERT) est 0, celui d'une valeur NULL est NULL.

Type SET

Il est possible d'extraire des enregistrements en comparant des ensembles entre eux pour tester l'appartenance d'éléments au sein d'une énumération SET. L'exemple suivant (décrit au chapitre 2) illustre deux possibilités d'extraction.

Web	Table et données	Extraction												
	<p>Figure 4-6 Table et données colonne SET</p> <p>Cursus</p> <table border="1"> <thead> <tr> <th>num</th><th>nom</th><th>{diplomes}</th></tr> </thead> <tbody> <tr> <td>E1</td><td>F. Brouard</td><td>BTS, Licence</td></tr> <tr> <td>E2</td><td>F. Degrelle</td><td>Licence, INSA , DUT</td></tr> <tr> <td>E0</td><td>F. Peyrard</td><td>INSA, DUT</td></tr> </tbody> </table> <p>SET BTS, DUT, Licence INSA</p>	num	nom	{diplomes}	E1	F. Brouard	BTS, Licence	E2	F. Degrelle	Licence, INSA , DUT	E0	F. Peyrard	INSA, DUT	<pre>SELECT nom, diplomes FROM Cursus WHERE FIND_IN_SET('Licence',diplomes)>0</pre> <pre>+-----+-----+ nom diplomes +-----+-----+ F. Brouard BTS,Licence F. Degrelle DUT,Licence,INSA +-----+-----+</pre>
num	nom	{diplomes}												
E1	F. Brouard	BTS, Licence												
E2	F. Degrelle	Licence, INSA , DUT												
E0	F. Peyrard	INSA, DUT												
		<pre>SELECT nom, diplomes FROM Cursus WHERE diplomes = 'BTS,Licence';</pre> <pre>+-----+-----+ nom diplomes +-----+-----+ F. Brouard BTS,Licence +-----+-----+</pre>												



Il est possible d'écrire des extractions basées sur l'opérateur LIKE (exemple : « SELECT FROM Cursus WHERE diplomes LIKE ('%Licence%') »). Cela n'est cependant pas recommandé, car le mot 'Licence' peut être présent dans l'ensemble non pas en tant qu'élément, mais en tant que sous-chaine d'un élément.

Autres fonctions

D'autres fonctions n'appartenant pas à la classification précédente sont présentées dans tableau suivant :

Tableau 4-24 Autres fonctions

Fonction	Objectif	Exemple
DEFAULT(<i>colonne</i>)	Valeur par défaut d'une colonne (NULL si aucune).	
FORMAT(<i>nombre,nb</i>)	Formate un nombre arrondi à <i>nb</i> décimales de la manière suivante :#,###,##.##'.	FORMAT(1234567.8901, 1) retourne '1,234,567.9'.
GREATEST(<i>expression[, expression]...</i>)	Retourne la plus grande des expressions.	GREATEST('Raffarin','Chirac','X-Men') retourne 'X-Men'.

Tableau 4-24 Autres fonctions (suite)

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

Regroupements

Cette section traite à la fois des regroupements de lignes (agrégats) et des fonctions de groupement (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]...)` pour pouvoir de regrouper des lignes selon la valeur des expressions (colonnes, fonction, constante, cas, etc.).
- *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

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

Fonctions de groupe

Nous étudions dans cette section les fonctions usuelles. D'autres sont proposées pour manipuler des cubes (*datawarehouse*).

Le tableau suivant présente les principales fonctions. L'option DISTINCT évite les duplicates (pris en compte sinon par défaut). À l'exception de COUNT, toutes les fonctions ignorent les valeurs NULL (il faudra utiliser IFNULL pour contrer cet effet).

Tableau 4-25 Fonctions de groupe

Fonction	Objectif
AVG([DISTINCT] expr)	Moyenne de <i>expr</i> (nombre).
COUNT({* [DISTINCT] expr})	Nombre de lignes (* toutes les lignes, <i>expr</i> pour les colonnes non nulles).
GROUP_CONCAT(expr)	Composition d'un ensemble de valeurs.
MAX([DISTINCT] expr)	Maximum de <i>expr</i> (nombre, date, chaîne).
MIN([DISTINCT] expr)	Minimum de <i>expr</i> (nombre, date, chaîne).
STDDEV(expr)	Écart type de <i>expr</i> (nombre).
SUM([DISTINCT] expr)	Somme de <i>expr</i> (nombre).
VARIANCE(expr)	Variance de <i>expr</i> (nombre).

Utilisées sans GROUP BY, ces fonctions s'appliquent à la totalité ou à une seule partie d'une table comme le montrent les exemples suivants :

Tableau 4-26 Exemples de fonctions de groupe

Web	Fonction	Exemples
	AVG	Moyenne des heures de vol et des primes des pilotes de la compagnie 'AF'. <pre>SELECT AVG(nbHVol), AVG(prime) FROM Pilote WHERE compa = 'AF'; +-----+-----+ AVG(nbHVol) AVG(prime) +-----+-----+ 283.333333 550.0000 +-----+-----+</pre>
	COUNT	Nombre de pilotes, d'heures de vol et de primes (toutes et distinctes) recensées dans la table. <pre>SELECT COUNT(*), COUNT(nbHVol), COUNT(prime), COCOUNT(DISTINCT prime) FROM Pilote; +-----+-----+-----+ COUNT(*) COUNT(nbHVol) COUNT(prime) COUNT(DISTINCT prime) +-----+-----+-----+ 6 5 4 3 +-----+-----+-----+</pre>

Tableau 4-26 Exemples de fonctions de groupe (suite)

Fonction	Exemples
GROUP _ CONCAT	Nom des pilotes de la compagnie 'AF'. <pre>SELECT compa, GROUP_CONCAT(nom) FROM Pilote GROUP BY compa;</pre> <pre>+-----+-----+ compa GROUP_CONCAT(nom) +-----+-----+ AF Gratien Viel,Didier Donsez,Daniel Vielle CAST Francoise Tort SING Richard Grin,Placide Fresnais +-----+-----+</pre>
MAX – MIN	Nombre d'heures de vol le plus élevé, date d'embauche la plus récente. Nombre d'heures de vol le moins élevé, date d'embauche la plus ancienne. <pre>SELECT MAX(nbHVol), MAX(embauche) "Date+", MIN(prime), MIN(embauche) "Date-" FROM Pilote;</pre> <pre>+-----+-----+-----+ MAX(nbHVol) Date+ MIN(prime) Date- +-----+-----+-----+ 2450.00 2001-09-21 0 1965-02-05 +-----+-----+-----+</pre>
STDEV – SUM – VARIANCE	Écart type des primes, somme des heures de vol, variance des primes des pilotes de la compagnie 'AF'. <pre>SELECT STDDEV(prime), SUM(nbHVol), VARIANCE(prime) FROM Pilote WHERE compa = 'AF';</pre> <pre>+-----+-----+-----+ STDDEV(prime) SUM(nbHVol) VARIANCE(prime) +-----+-----+-----+ 50.0000 850.00 2500.0000 +-----+-----+-----+</pre>

Étudions à présent ces fonctions dans le cadre de regroupements de lignes.

Étude du GROUP BY et HAVING

Le regroupement de lignes dans une requête se programme au niveau surligné de l'instruction SQL suivante :

```
SELECT col1[, col2...], fonction1Groupe(...)[,fonction2Groupe(...)...]
FROM nomTable [ WHERE condition ]
GROUP BY {col1 | expr1 | position1} [, {col2... } ]
[ HAVING condition ]
[ORDER BY {col1 | expr1 | position1} [ASC | DESC] [, {col1 ... } ]]
```

- La clause WHERE de la requête permet d'exclure des lignes pour chaque regroupement, ou rejeter des groupements entiers. Elle s'applique donc à la totalité de la table.
- La clause GROUP BY liste les colonnes du regroupement.
- La clause HAVING permet de poser des conditions sur chaque regroupement.
- La clause ORDER BY permet de trier le résultat (déjà étudiée).



Les colonnes présentes dans le SELECT doivent apparaître dans le GROUP BY. Seules des fonctions ou expressions peuvent exister en plus dans le SELECT.

Les alias de colonnes ne peuvent pas être utilisés dans la clause GROUP BY.

Dans l'exemple suivant, en groupant sur la colonne compa, trois ensembles de lignes (groupements) sont composés. Il est alors possible d'appliquer des fonctions de groupe à chacun de ces ensembles (dont le nombre n'est pas précisé dans la requête ni limité par le système qui parcourt toute la table).

Figure 4-8 Groupement sur la colonne compa

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-5	Daniel Vieille	400	600	1965-01-16	A340	AF
PL-6	Francoise Tort		0	2000-12-24	A340	CAST
PL-3	Richard Grin	1000		2001-09-11	A320	SING
PL-4	Placide Fresnais	2450	500	2001-09-21	A330	SING

Il est aussi possible de grouper sur plusieurs colonnes (par exemple ici sur les colonnes compa et typeAvion pour classifier les pilotes selon ces deux critères).

Utilisées avec GROUP BY, les fonctions s'appliquent désormais à chaque regroupement comme le montrent les exemples suivants :

Tableau 4-27 Exemples de fonctions de groupe avec GROUP BY



Fonction	Exemples												
AVG	<p>Moyenne des heures de vol et des primes pour chaque compagnie.</p> <pre>SELECT compa, AVG(nbHVol), AVG(prime) FROM Pilote GROUP BY (compa) ;</pre> <table border="1"> <thead> <tr> <th>compa</th> <th>AVG(nbHVol)</th> <th>AVG(prime)</th> </tr> </thead> <tbody> <tr> <td>AF</td> <td>283.333333</td> <td>550.0000</td> </tr> <tr> <td>CAST</td> <td>NULL</td> <td>0.0000</td> </tr> <tr> <td>SING</td> <td>1725.000000</td> <td>500.0000</td> </tr> </tbody> </table>	compa	AVG(nbHVol)	AVG(prime)	AF	283.333333	550.0000	CAST	NULL	0.0000	SING	1725.000000	500.0000
compa	AVG(nbHVol)	AVG(prime)											
AF	283.333333	550.0000											
CAST	NULL	0.0000											
SING	1725.000000	500.0000											

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

Fonction	Exemples																		
COUNT	<p>Nombre de pilotes (et ceux qui ont de l'expérience du vol) par compagnie.</p> <pre>SELECT compa, COUNT(*), COUNT(nbHVol) FROM Pilote GROUP BY (compa);</pre> <table border="1"> <thead> <tr> <th>compa</th> <th>COUNT(*)</th> <th>COUNT(nbHVol)</th> </tr> </thead> <tbody> <tr> <td>AF</td> <td>3</td> <td>3</td> </tr> <tr> <td>CAST</td> <td>1</td> <td>0</td> </tr> <tr> <td>SING</td> <td>2</td> <td>2</td> </tr> </tbody> </table>	compa	COUNT(*)	COUNT(nbHVol)	AF	3	3	CAST	1	0	SING	2	2						
compa	COUNT(*)	COUNT(nbHVol)																	
AF	3	3																	
CAST	1	0																	
SING	2	2																	
MAX	<p>Nombre d'heures de vol le plus élevé, date d'embauche la plus récente pour chaque compagnie.</p> <pre>SELECT compa, MAX(nbHVol), MAX(embauche) "Date" FROM Pilote GROUP BY (compa);</pre> <table border="1"> <thead> <tr> <th>compa</th> <th>MAX(nbHVol)</th> <th>Date</th> </tr> </thead> <tbody> <tr> <td>AF</td> <td>450.00</td> <td>1995-05-13</td> </tr> <tr> <td>CAST</td> <td>NULL</td> <td>2000-12-24</td> </tr> <tr> <td>SING</td> <td>2450.00</td> <td>2001-09-21</td> </tr> </tbody> </table>	compa	MAX(nbHVol)	Date	AF	450.00	1995-05-13	CAST	NULL	2000-12-24	SING	2450.00	2001-09-21						
compa	MAX(nbHVol)	Date																	
AF	450.00	1995-05-13																	
CAST	NULL	2000-12-24																	
SING	2450.00	2001-09-21																	
STDEV — SUM (avec WHERE)	<p>Écarts types des primes et sommes des heures de vol des pilotes volant sur 'A320' de chaque compagnie.</p> <pre>SELECT compa, STDEV(prime), SUM(nbHVol) FROM Pilote WHERE typeAvion = 'A320' GROUP BY (compa);</pre> <table border="1"> <thead> <tr> <th>compa</th> <th>STDEV(prime)</th> <th>SUM(nbHVol)</th> </tr> </thead> <tbody> <tr> <td>AF</td> <td>0.0000</td> <td>450.00</td> </tr> <tr> <td>SING</td> <td>NULL</td> <td>1000.00</td> </tr> </tbody> </table>	compa	STDEV(prime)	SUM(nbHVol)	AF	0.0000	450.00	SING	NULL	1000.00									
compa	STDEV(prime)	SUM(nbHVol)																	
AF	0.0000	450.00																	
SING	NULL	1000.00																	
Plusieurs colonnes dans le GROUP BY	<p>Nombre de pilotes qualifiés par type d'appareil et par compagnie.</p> <pre>SELECT compa,typeAvion, COUNT(brevet) FROM Pilote GROUP BY compa,typeAvion ;</pre> <table border="1"> <thead> <tr> <th>compa</th> <th>typeAvion</th> <th>COUNT(brevet)</th> </tr> </thead> <tbody> <tr> <td>AF</td> <td>A320</td> <td>2</td> </tr> <tr> <td>AF</td> <td>A340</td> <td>1</td> </tr> <tr> <td>CAST</td> <td>A340</td> <td>1</td> </tr> <tr> <td>SING</td> <td>A320</td> <td>1</td> </tr> <tr> <td>SING</td> <td>A330</td> <td>1</td> </tr> </tbody> </table>	compa	typeAvion	COUNT(brevet)	AF	A320	2	AF	A340	1	CAST	A340	1	SING	A320	1	SING	A330	1
compa	typeAvion	COUNT(brevet)																	
AF	A320	2																	
AF	A340	1																	
CAST	A340	1																	
SING	A320	1																	
SING	A330	1																	

Fonction	Exemples						
GROUP BY et HAVING	<p>Compagnies (et nombre de leurs pilotes) ayant plus d'un pilote.</p> <pre>SELECT compa, COUNT(brevet) FROM Pilote GROUP BY(compa) HAVING COUNT(brevet)>=2 ;</pre> <table border="1"> <thead> <tr> <th>compa</th> <th>COUNT(brevet)</th> </tr> </thead> <tbody> <tr> <td>AF</td> <td>3</td> </tr> <tr> <td>SING</td> <td>2</td> </tr> </tbody> </table>	compa	COUNT(brevet)	AF	3	SING	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 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 fonctionne. Le nom des colonnes n'a pas d'importance. Il est possible de comparer plusieurs colonnes de deux tables.

Exemple

Étudions à présent chaque opérateur à partir de l'exemple composé des deux tables suivantes :

- `AvionsdeAF(immat CHAR(6), typeAvion CHAR(10), nbHVol DECIMAL(10,2))`
- `AvionsdeSING(immatriculation CHAR(6), typeAv CHAR(10), prixAchat DECIMAL(14,2))`

Il est vraisemblable que seules les deux premières colonnes doivent être comparées. Bien que permises par MySQL, la programmation de l'union, l'intersection ou la différence entre les prix des avions et les heures de vol (deux colonnes numériques) ne seraient pas valides du point de vue sémantique.

Figure 4-9 Tables pour les opérateurs ensemblistes



AviondeAF			AviondeSING		
immat	typeAvion	nbHVol	immatriculation	typeAv	PrixAchat
F-WTSS	Concorde	6570	S-ANSI	A320	104 500
F-GLFS	A320	3500	S-AVEZ	A320	156 000
F-GTMP	A340		S-SMILE	A330	198 000
			F-GTMP	A340	204 500

Intersection



L'intersection entre deux ensembles homogènes se programme à l'aide d'une requête du type : `SELECT DISTINCT ensemble1 FROM Table1 WHERE ensemble1 IN (SELECT ensemble2 FROM Table2)`. Comme l'opérateur d'intersection, cette requête est commutative (*Table1* joue le rôle de *Table2*, et *ensemble1* celui d'*ensemble2*).

Notez qu'à l'affichage le nom des colonnes est donné par la première requête. La deuxième requête fait apparaître deux colonnes dans le SELECT.

Tableau 4-28 Exemples d'intersections



Besoin	Requête
Quels sont les types d'avions que les deux compagnies exploitent en commun ?	<pre>SELECT DISTINCT typeAvion FROM AvionsdeAF WHERE typeAvion IN (SELECT typeAv FROM AvionsdeSING);</pre> +-----+ typeAvion +-----+ A320 A340 +-----+

Tableau 4-28 Exemples d'intersections (suite)

Besoin	Requête
Quels sont les avions qui sont exploités par les deux compagnies en commun ?	<pre>SELECT DISTINCT immatriculation ,typeAv FROM AvionsdeSING WHERE immatriculation IN (SELECT immat FROM AvionsdeAF) AND typeAv IN (SELECT typeAvion FROM AvionsdeAF); +-----+-----+ immatriculation typeAv +-----+-----+ F-GTMP A340 +-----+-----+</pre>

Si vous voulez continuer ce raisonnement en vous basant sur trois compagnies, il suffit d'ajouter une clause WHERE dans les requêtes imbriquées qui interrogera la troisième compagnie. Ce principe se généralise, et, pour n compagnies, il faudra n imbrications de requêtes.

Opérateurs UNION et UNION ALL



Les opérateurs UNION et UNION ALL sont commutatifs. L'opérateur UNION permet d'éviter les duplicates (comme DISTINCT dans un SELECT). UNION ALL ne les élimine pas.

Tableau 4-29 Exemples avec les opérateurs UNION

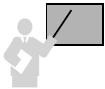
Besoin	Requête
Web Quels sont tous les types d'avions que les deux compagnies exploitent ?	<pre>SELECT typeAvion FROM AvionsdeAF UNION SELECT typeAv FROM AvionsdeSING; +-----+ typeAvion +-----+ A320 A340 Concorde A330 +-----+</pre>

Tableau 4-29 Exemples avec les opérateurs UNION (suite)

Web	Besoin	Requête
	Même requête avec les duplicates. On extrait les types de la compagnie 'AF', suivis des types de la compagnie 'SING'.	<pre>SELECT typeAvion FROM AvionsdeAF UNION ALL SELECT typeAv FROM AvionsdeSING; +-----+ typeAvion +-----+ A320 A340 Concorde A340 A320 A320 A330 +-----+</pre>

Ce principe se généralise à l'union de n ensembles par n requêtes reliées avec $n-1$ clause UNION ou UNION ALL.

Différence



La différence entre deux ensembles homogènes se programme à l'aide d'une requête du type SELECT DISTINCT *ensemble1* FROM *Table1* WHERE *ensemble1* NOT IN (SELECT *ensemble2* FROM *Table2*). Comme l'opérateur ensembliste, cette requête n'est pas commutative, car elle programme *ensemble1-ensemble2*.

Tableau 4-30 Exemples de différences

Web	Besoin	Requête
	Quels sont les types d'avions exploités par la compagnie 'AF', mais pas par 'SING' ?	<pre>SELECT DISTINCT typeAvion FROM AvionsdeAF WHERE typeAvion NOT IN (SELECT typeAv FROM AvionsdeSING); +-----+ typeAvion +-----+ Concorde +-----+</pre>
	Quels sont les types d'avions exploités par la compagnie 'SING,' mais pas par 'AF' ?	<pre>SELECT DISTINCT typeAv FROM AvionsdeSING WHERE typeAv NOT IN (SELECT typeAvion FROM AvionsdeAF); +-----+ typeAv +-----+ A330 +-----+</pre>

Ce principe se généralise à la différence entre n ensembles par imbrication de n requêtes (dans le bon ordre).



La directive NOT IN doit être utilisée avec prudence car elle retourne *faux* si un membre ramené par la sous-interrogation est NULL.

Ordonner les résultats



Le résultat d'une requête contenant des opérateurs ensemblistes est trié, par défaut, par ordre croissant, sauf avec l'opérateur UNION ALL.



La clause ORDER BY n'est utilisable qu'une fois en fin d'une requête incluant des opérateurs ensemblistes. Cette clause accepte le nom des colonnes de la première requête ou la position de ces colonnes.

Le tableau suivant présente trois écritures différentes de la même requête ensembliste contenant une clause ORDER BY. Le besoin est de connaître tous les types d'avions que les deux compagnies exploitent (classement par ordre décroissant).

Notez que la troisième requête produit le même résultat en faisant intervenir un SELECT (aliasé) dans le FROM. Ce mécanisme permet de construire dynamiquement la table à interroger.

Tableau 4-31 Exemples avec la clause ORDER BY

Technique	Requête
Nom de la colonne.	<pre>SELECT typeAvion FROM AvionsdeAF UNION SELECT typeAv FROM AvionsdeSING ORDER BY typeAvion DESC ;</pre>
Position de la colonne.	<pre>... ORDER BY 1 DESC ;</pre>
SELECT dans le FROM.	<pre>SELECT T.typeAvion FROM (SELECT typeAvion FROM AvionsdeAF UNION SELECT typeAv FROM AvionsdeSING) T ORDER BY T.typeAvion DESC;</pre> <pre>+-----+ typeAvion +-----+ Concorde A340 A330 A320 +-----+</pre>



Il faut affecter des alias aux expressions de la première requête pour pouvoir les utiliser le ORDER BY final.

Pour illustrer cette restriction, supposons que nous désirions faire la liste des avions avec le prix d'achat augmenté de 20 %, liste triée en fonction de cette dernière hausse. Le problème est que la table AvionsdeAF ne possède pas une telle colonne. Ajoutons donc au SELECT cette table, dans le tableau suivant, la valeur 0 pour rendre possible l'opérateur UNION.

Tableau 4-32 Alias pour ORDER BY

Requête	Résultat
<pre>SELECT immatriculation, 1.2*prixAchat px FROM Avions- deSING UNION SELECT immat, 0 FROM AvionsdeAF ORDER BY px DESC;</pre>	+-----+-----+ immatriculation px +-----+-----+ F-GTMP 245400.000 S-MILE 237600.000 S-AVEZ 187200.000 S-ANSI 125400.000 F-GLFS 0.000 F-GTMP 0.000 F-WTSS 0.000 +-----+-----+

Produit cartésien

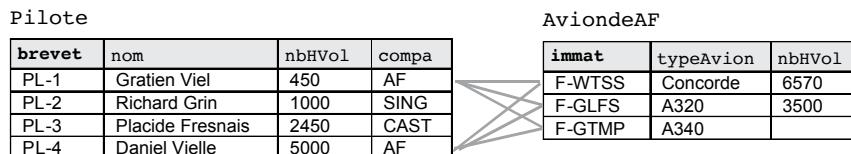
En mathématiques, le produit cartésien de deux ensembles E et F est l'ensemble des couples (x, y) où $x \in E$ et $y \in F$. En transposant au modèle relationnel, le produit cartésien de deux tables $T1$ et $T2$ est l'ensemble des enregistrements (x, y) où $x \in T1$ et $y \in T2$.



Le produit cartésien total entre deux tables $T1$ et $T2$ se programme sous SQL en positionnant les deux tables dans la clause FROM, sans ajouter de conditions dans la clause WHERE. Si les conditions sont de la forme $c1$ opérateur $c2$ avec $c1 \in T1$ et $c2 \in T2$, on parle de jointure. Si les conditions sont de la forme $c1$ opérateur *valeur1* ou $c2$ opérateur *valeur2*, on parlera de produit cartésien restreint.

Le produit cartésien restreint, illustré par l'exemple suivant, exprime les combinaisons d'informations sur les pilotes et les avions qu'il est possible de réaliser en considérant les pilotes de la compagnie 'AF' et les avions de la table AvionsdeAF.

Figure 4-10 Produit cartésien d'enregistrements de tables



Le nombre d'enregistrements résultant d'un produit cartésien est égal au produit du nombre d'enregistrements des deux tables mises en relation.

Dans le cadre de notre exemple, le nombre d'enregistrements du produit cartésien sera de 2 pilotes x 3 avions = 6 enregistrements. Le tableau suivant décrit la requête SQL permettant de construire le produit cartésien restreint de notre exemple. Les alias distinguent les colonnes s'il advenait qu'il en existe de même nom entre les deux tables.

Tableau 4-33 Produit cartésien

Web	Besoin	Requête														
	<p>Quels sont les couples possibles (<i>avion, pilote</i>) en considérant les avions et les pilotes de la compagnie 'AF' ?</p> <p><i>6 lignes extraites</i></p>	<pre>SELECT p.brevet, avAF.immat FROM Pilote p, AvionsdeAF avAF WHERE p.compa = 'AF';</pre> <table border="1"> <thead> <tr> <th>brevet</th> <th>immat</th> </tr> </thead> <tbody> <tr> <td>PL-1</td> <td>F-GLFS</td> </tr> <tr> <td>PL-1</td> <td>F-GTMP</td> </tr> <tr> <td>PL-1</td> <td>F-WTSS</td> </tr> <tr> <td>PL-4</td> <td>F-GLFS</td> </tr> <tr> <td>PL-4</td> <td>F-GTMP</td> </tr> <tr> <td>PL-4</td> <td>F-WTSS</td> </tr> </tbody> </table>	brevet	immat	PL-1	F-GLFS	PL-1	F-GTMP	PL-1	F-WTSS	PL-4	F-GLFS	PL-4	F-GTMP	PL-4	F-WTSS
brevet	immat															
PL-1	F-GLFS															
PL-1	F-GTMP															
PL-1	F-WTSS															
PL-4	F-GLFS															
PL-4	F-GTMP															
PL-4	F-WTSS															

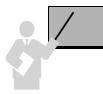
Bilan

Seules les colonnes de même type et représentant la même sémantique peuvent être comparées à l'aide de termes ensemblistes. Il est possible d'ajouter des expressions (constantes ou calculs) à une requête pour rendre homogènes les deux requêtes, et permettre ainsi l'utilisation d'un opérateur ensembliste.



Parce qu'il faut utiliser NOT IN avec prudence, différentes alternatives aux opérateurs ensemblistes existent sur la base de jointures de type SQL2 (qui sont détaillées dans la prochaine section), elles sont détaillées par Pierre Caboche et mises en ligne sur le site Web associé à l'ouvrage.

Les jointures permettent d'extraire des données issues de plusieurs tables. Le processus de normalisation du modèle relationnel est basé sur la décomposition et a pour conséquence d'augmenter le nombre de tables d'un schéma. Ainsi, la majorité des requêtes utilisent des jointures nécessaires pour pouvoir extraire des données de tables distinctes.



Une jointure met en relation deux tables sur la base d'une clause de jointure (comparaison de colonnes). Généralement, cette comparaison fait intervenir une clé étrangère d'une table et une clé primaire d'une autre table (car le modèle relationnel est fondamentalement basé sur les valeurs).

En considérant les tables suivantes, les seules jointures logiques doivent se faire sur l'égalité soit des colonnes **comp** et **compa** soit des colonnes **brevet** et **chefPil**. Ces jointures permettront d'afficher des données d'une table (ou des deux tables) tout en posant des conditions sur une table (ou les deux). Par exemple, l'affichage du nom des compagnies (colonnes de la table **Compagnie**) qui ont embauché un pilote ayant moins de 500 heures de vol (conditions posées sur la table **Pilote**).

Figure 4-11 Deux tables à mettre en jointure

Compagnie

comp	nrule	rue	ville	nomComp
AF	124	Port Royal	Paris	Air France
SING	7	Camparols	Singapour	Singapore AL
CAST	1	G. Brassens	Blagnac	Castanet AL

Pilote

brevet	nom	nbHVol	compa	chefPil
PL-1	Pierre Lamothe	450	AF	PL-4
PL-2	Didier Linxe	900	AF	PL-4
PL-3	Christian Soutou	1000	SING	
PL-4	Henri Alquié	3400	AF	

Classification

Une jointure peut s'écrire, dans une requête SQL, de différentes manières :

- « relationnelle » (aussi appelée « SQL89 » pour rappeler la version de la norme SQL) ;
- « SQL2 » (aussi appelée « SQL92 ») ;
- « procédurale » (qui qualifie la structure de la requête) ;
- « mixte » (combinaison des trois approches précédentes).

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 des 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

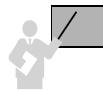
Bien que, dans le vocabulaire courant, on ne parle que de « jointures » en fonction de la nature de l'opérateur utilisé dans la requête, de la clause de jointure et des tables concernées, on distingue :

- 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 applicables 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 T_1 , T_2 et T_3 en jointure, il faut utiliser deux clauses de jointures (une entre T_1 et T_2 et l'autre entre T_2 et T_3). Pour n tables, il faut $n-1$ clauses de jointures. Si vous oubliez une clause de jointure, un produit cartésien restreint est générée.

Étudions à présent chaque type de jointure avec les syntaxes « relationnelle » et « SQL2003 ».

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 :

- l'identité des pilotes de la compagnie de nom 'Air France' ayant plus de 500 heures de vol (requête R1) ;
- les coordonnées des compagnies qui embauchent des pilotes de moins de 500 heures de vol (requête R2).

La jointure qui résoudra la première requête est illustrée dans la figure par les données grisées tandis que la deuxième jointure est représentée par les données en gras.

Figure 4-12 Équijointures



Écriture « relationnelle »



- MySQL recommande d'utiliser des alias de tables pour améliorer les performances.
- Les alias sont obligatoires quand des colonnes de différentes tables portent le même nom ou dans le cas d'autojointures.

Écriture « SQL2 »



- La clause `JOIN ... ON condition` permet de programmer une équijoindre.
- L'utilisation de la directive `INNER` devant `JOIN...` est optionnelle et est appliquée par défaut.

Le tableau suivant détaille ces requêtes avec les deux syntaxes. Les clauses de jointures sont grises.

Tableau 4-35 Exemples d'équi jointures

Web	Requête	Jointure relationnelle	Jointure SQL2
	R1	<pre>SELECT brevet, nom FROM Pilote, Compagnie WHERE comp = compa AND nomComp = 'Air France' AND nbHVol > 500;</pre>	<pre>SELECT brevet, nom FROM Compagnie JOIN Pilote ON comp = compa WHERE nomComp = 'Air France' AND nbHVol > 500;</pre>
	R2	<pre>SELECT cpg.nomComp, cpg.nrue, cpg.rue, cpg.ville FROM Pilote pil, Compagnie cpg WHERE cpg.comp = pil.compa AND pil.nbHVol < 500;</pre>	<pre>SELECT nomComp, nrue, rue, ville FROM Compagnie INNER JOIN Pilote ON comp = compa WHERE nbHVol < 500;</pre>

Autojointure

Cas particulier de l'équi jointure, l'autojointure relie une table à elle même.

Extrayons par exemple :

- l'identité des pilotes placés sous la responsabilité des pilotes de nom 'A' (requête R3) ;
- la somme des heures de vol des pilotes placés sous la responsabilité des chefs pilotes compagnie de nom 'Air France' (requête R4).

Ces requêtes doivent être programmées à l'aide d'une autojointure, car elles imposent de parcourir deux fois la table Pilote (examen de chaque pilote en le comparant à un autre). Les autojointures sont réalisées entre les colonnes `brevet` et `chefPil`.

La jointure de la première requête est illustrée dans la figure par les données soulignées en noir, tandis que la deuxième jointure est mise en valeur par les données soulignées en gras.

Compagnie

comp	nrule	rue	ville	nomComp
AF	124	Port Royal	Paris	Air France
SING	7	Camparols	Singapour	Singapore AL
CAST	1	G. Brassens	Blagnac	Castanet AL

Pilote

brevet	nom	nbHVol	compa	chefPil
PL-1	Pierre Lamothe	450	AF	PL-4
PL-2	Didier Linxe	900	AF	PL-4
PL-3	Christian Soutou	1000	SING	
PL-4	Henri Alquié	3400	AF	

Le tableau suivant détaille ces requêtes, les clauses d'autojointures sont surlignées. Dans les deux syntaxes, il est impératif d'utiliser un alias de table. Concernant l'écriture « SQL2 », on remarque que les clauses JOIN s'imbriquent (pour joindre plus de deux tables).

Tableau 4-36 Exemples d'autojointures

Web	Requête	Jointure relationnelle	Jointure SQL2
R3	<pre>SELECT p1.brevet, p1.nom FROM Pilote p1, Pilote p2 WHERE p1.chefPil = p2.brevet AND p2.nom LIKE '%Alquié%';</pre>	<pre>+-----+-----+ brevet nom +-----+-----+ PL-1 Pierre Lamothe PL-2 Didier Linxe +-----+-----+</pre>	<pre>SELECT p1.brevet, p1.nom FROM Pilote p1 JOIN Pilote p2 ON p1.chefPil = p2.brevet WHERE p2.nom LIKE '%Alquié%';</pre>
R4	<pre>SELECT SUM(p1.nbHVol) FROM Pilote p1, Pilote p2, Compagnie cpg WHERE p1.chefPil = p2.brevet AND cpg.comp = p2.compa AND cpg.nomComp = 'Air France';</pre>	<pre>+-----+ SUM(p1.nbHVol) +-----+ 1350.00 +-----+</pre>	<pre>SELECT SUM(p1.nbHVol) FROM Pilote p1 JOIN Pilote p2 ON p1.chefPil = p2.brevet JOIN Compagnie ON comp = p2.compa WHERE nomComp = 'Air France';</pre>



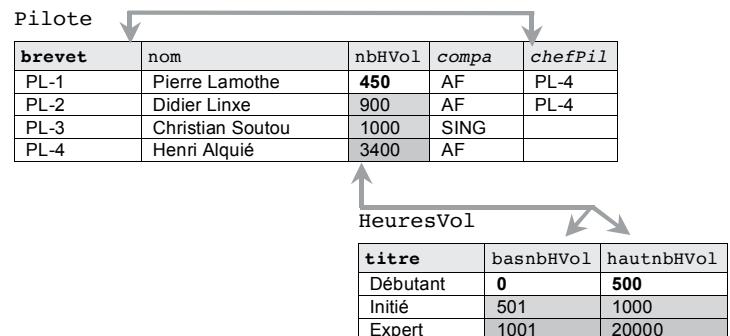
Les requêtes d'inéquijointures font intervenir tout type d'opérateur ($<$, $>$, $<$, $>$, $=$, \neq , BETWEEN, LIKE, IN). À l'inverse des équijointures, la clause d'une inéquijointure n'est pas basée sur l'égalité de clés primaires (ou candidates) et de clés étrangères.

En considérant les tables suivantes, extrayons par exemple :

- les pilotes ayant plus d'expérience que le pilote de numéro de brevet 'PL-2' (requête R5).
- le titre de qualification des pilotes en raisonnant sur la comparaison des heures de vol d'un ensemble de référence, ici la table HeuresVol (requête R6). Dans notre exemple, il s'agit par exemple de retrouver le fait que le premier pilote est débutant.

La jointure qui résoudra la deuxième requête est illustrée par les niveaux de gris.

Figure 4-14 Inéquijointures



Le tableau suivant détaille ces requêtes, les clauses d'inéquijointures sont surlignées :

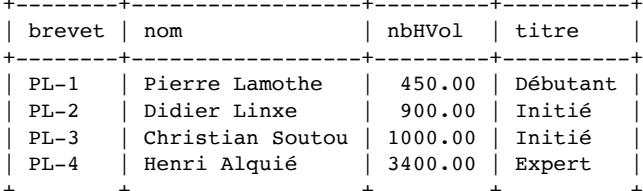
Tableau 4-37 Exemples d'inéquijointures

Requête	Jointure relationnelle	Jointure SQL2
R5	<pre>SELECT p1.brevet, p1.nom, p1.nbHVol, p2.nbHVol "Référence" FROM Pilote p1, Pilote p2 WHERE p1.nbHVol > p2.nbHVol AND p2.brevet = 'PL-2';</pre>	<pre>SELECT p1.brevet, p1.nom, p1.nbHVol, p2.nbHVol "Référence" FROM Pilote p1 JOIN Pilote p2 ON p1.nbHVol>p2.nbHVol WHERE p2.brevet = 'PL-2';</pre>

brevet	nom	nbHVol	Référence
PL-3	Christian Soutou	1000.00	900.00
PL-4	Henri Alquié	3400.00	900.00

Tableau 4-37 Exemples d'inéquijoointures (suite)

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



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 jointures 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).

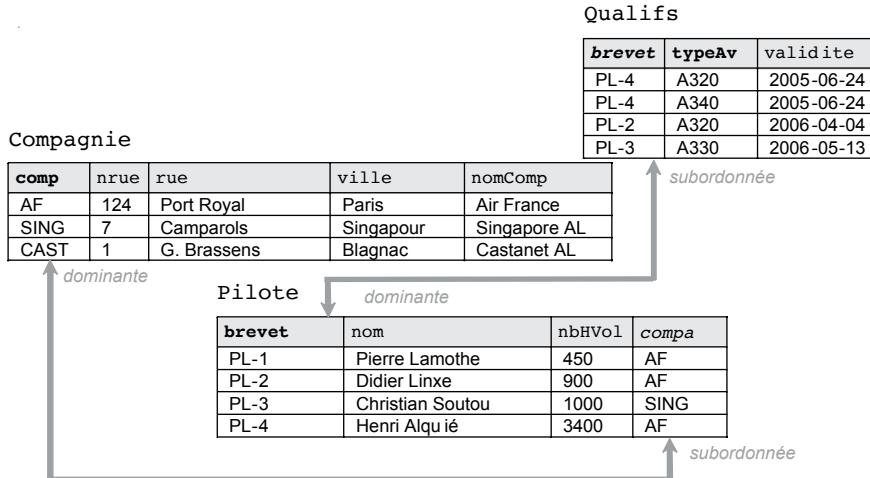
Jointures unilatérales

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 :

Figure 4-15 Jointures externes unilatérales



Écriture « SQL2 »



Le sens de la directive de jointure externe LEFT ou RIGHT de la clause OUTER JOIN dépend de la table dominante.

Le tableau suivant détaille les requêtes de notre exemple, les clauses de jointures externes unilatérales sont grisées. Les tables dominantes sont notées en gras (Compagnie pour la première requête et Pilote pour la deuxième).

Tableau 4-38 Écritures équivalentes de jointures externes unilatérales

Requête	Jointures relationnelles	Jointures SQL2																		
R7	Sans objet.	<pre>SELECT nomComp, brevet, nom FROM Compagnie LEFT OUTER JOIN Pilote ON comp = compa; --équivalent à SELECT nomComp, brevet, nom FROM Pilote RIGHT OUTER JOIN Compagnie ON comp = compa;</pre> <table border="1"> <thead> <tr> <th>nomComp</th> <th>brevet</th> <th>nom</th> </tr> </thead> <tbody> <tr> <td>Air France</td> <td>PL-1</td> <td>Pierre Lamothe</td> </tr> <tr> <td>Air France</td> <td>PL-2</td> <td>Didier Linxe</td> </tr> <tr> <td>Air France</td> <td>PL-4</td> <td>Henri Alquié</td> </tr> <tr> <td>Castanet AL</td> <td>NULL</td> <td>NULL</td> </tr> <tr> <td>Singapore AL</td> <td>PL-3</td> <td>Christian Soutou</td> </tr> </tbody> </table>	nomComp	brevet	nom	Air France	PL-1	Pierre Lamothe	Air France	PL-2	Didier Linxe	Air France	PL-4	Henri Alquié	Castanet AL	NULL	NULL	Singapore AL	PL-3	Christian Soutou
nomComp	brevet	nom																		
Air France	PL-1	Pierre Lamothe																		
Air France	PL-2	Didier Linxe																		
Air France	PL-4	Henri Alquié																		
Castanet AL	NULL	NULL																		
Singapore AL	PL-3	Christian Soutou																		

Tableau 4-38 Écritures équivalentes de jointures externes unilatérales

Requête	Jointures relationnelles	Jointures SQL2																		
R8	Sans objet.	<pre>SELECT qua.typeAv, pil.brevet, pil.nom FROM Qualifs qua RIGHT OUTER JOIN Pilote pil ON pil.brevet = qua.brevet; --équivalent à SELECT qua.typeAv, pil.brevet, pil.nom FROM Pilote pil LEFT OUTER JOIN Qualifs qua ON pil.brevet = qua.brevet;</pre> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <table border="1"> <thead> <tr> <th>typeAv</th> <th>brevet</th> <th>nom</th> </tr> </thead> <tbody> <tr> <td>NULL</td> <td>PL-1</td> <td>Pierre Lamothe</td> </tr> <tr> <td>A320</td> <td>PL-2</td> <td>Didier Linxe</td> </tr> <tr> <td>A330</td> <td>PL-3</td> <td>Christian Soutou</td> </tr> <tr> <td>A320</td> <td>PL-4</td> <td>Henri Alquié</td> </tr> <tr> <td>A340</td> <td>PL-4</td> <td>Henri Alquié</td> </tr> </tbody> </table> </div>	typeAv	brevet	nom	NULL	PL-1	Pierre Lamothe	A320	PL-2	Didier Linxe	A330	PL-3	Christian Soutou	A320	PL-4	Henri Alquié	A340	PL-4	Henri Alquié
typeAv	brevet	nom																		
NULL	PL-1	Pierre Lamothe																		
A320	PL-2	Didier Linxe																		
A330	PL-3	Christian Soutou																		
A320	PL-4	Henri Alquié																		
A340	PL-4	Henri Alquié																		

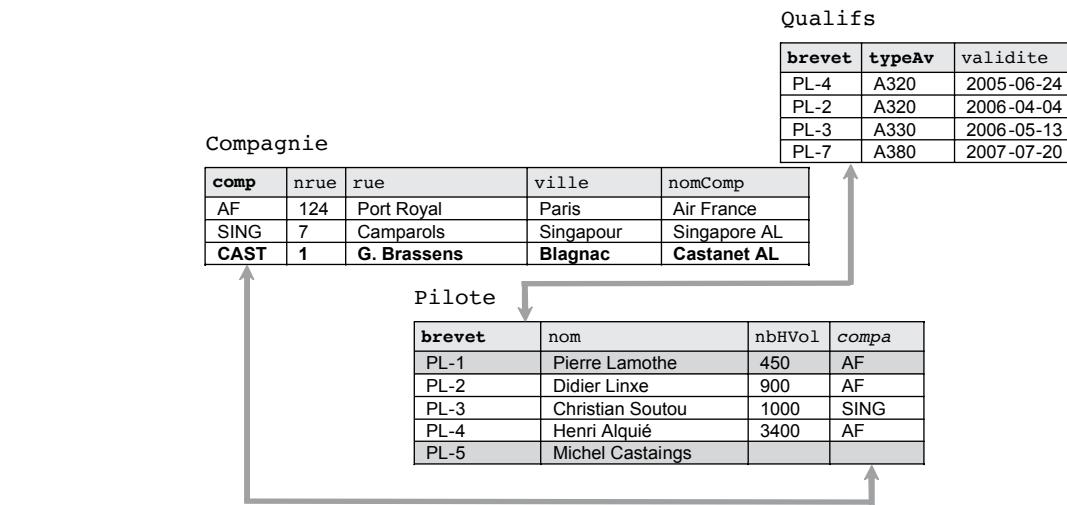
Jointures bilatérales

Les deux tables jouent un rôle symétrique, il n'y a pas de table dominante. Ce type de jointure permet d'extraire des enregistrements qui ne répondent pas aux critères de jointure des deux côtés de la clause de jointure.

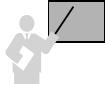
En considérant les tables suivantes, une jointure externe bilatérale permet d'extraire, par exemple :

- la liste des compagnies et leurs pilotes, incluant les compagnies n'ayant pas de pilote et les pilotes rattachés à aucune compagnie (requête R9) ;
- la liste des pilotes et leurs qualifications, incluant les pilotes n'ayant pas encore d'expérience et les qualifications associées à des pilotes inconnus (requête R10).

Figure 4-16 Jointures externes bilatérales



Écriture « SQL2 »



La directive FULL OUTER JOIN permet d'ignorer l'ordre (et donc le sens de la jointure) des tables dans la requête.

Le seul problème, c'est que :



MySQL ne prend pas encore en charge, en version 5.0, la directive FULL OUTER JOIN.

Le tableau suivant détaille les requêtes de notre exemple, les clauses de jointures externes bilatérales sont surlignées. Les enregistrements qui ne respectent pas la condition de jointure sont surlignés.

Tableau 4-39 Jointures externes bilatérales

Web	Requête	Jointure relationnelle	Jointures (théoriques) SQL2
	R9	Sans objet.	<pre>SELECT nomComp, brevet, nom FROM Pilote FULL OUTER JOIN Compagnie ON comp = compa; --équivalent à SELECT nomComp, brevet, nom FROM Compagnie FULL OUTER JOIN Pilote ON comp = compa;</pre>
			NOMCOMP BREVET NOM
			----- -----
			Air France PL-4 Henri Alquié Air France PL-1 Pierre Lamothe Air France PL-2 Didier Linxe Singapore AL PL-3 Christian Soutou Castanet AL PL-5 Michel Castaings
	R10	Sans objet.	<pre>SELECT qua.typeAv, pil.brevet, pil.nom FROM Pilote pil FULL OUTER JOIN Qualifs qua ON pil.brevet = qua.brevet; --équivalent à SELECT qua.typeAv, pil.brevet, pil.nom FROM Qualifs qua FULL OUTER JOIN Pilote pil ON pil.brevet = qua.brevet;</pre>
			TYPE BREVET NOM
			----- -----
			A320 PL-4 Henri Alquié A320 PL-2 Didier Linxe A330 PL-3 Christian Soutou A380 PL-1 Pierre Lamothe PL-5 Michel Castaings

Jointures procédurales



Les jointures procédurales sont écrites par des requêtes qui contiennent des sous-interrogations (SELECT imbriqué). Chaque clause FROM ne contient qu'une seule table.

```
SELECT colonnesTable1 FROM [nomBase.]nomTable1
  WHERE colonne(s) | expression(s) { IN | = | opérateur }
        (SELECT colonne(s) de la Table2 FROM [nomBase.]nomTable2
          WHERE colonne(s) | expression(s) { IN | = | opérateur }
            (SELECT ...)
            [AND (conditionsTable2)]
        )
  [AND (conditionsTable1)];
```

Cette forme d'écriture n'est pas la plus utilisée, mais elle permet de mieux visualiser certaines jointures. Elle est plus complexe à écrire, car l'ordre d'apparition des tables dans les clauses FROM a son importance.



Seules les colonnes de la table qui se trouve au niveau du premier SELECT peuvent être extraites.

La sous-interrogation doit être placée entre parenthèses. Elle ne doit pas comporter de clause ORDER BY, mais peut inclure GROUP BY et HAVING.

Le résultat d'une sous-interrogation est utilisé par la requête de niveau supérieur. Une interrogation est exécutée avant la requête de niveau supérieur.

Une sous-interrogation peut ramener une ou plusieurs lignes. Les opérateurs =, >, <, >= et <= permettent d'en extraire une, les opérateurs IN, ANY et ALL permettent d'en ramener plusieurs.

Sous-interrogations monolignes

Le tableau suivant détaille quelques sous-interrogations monolignes. Nous nous basons sur certaines requêtes déjà étudiées (forme relationnelle et SQL2).

Tableau 4-40 Sous-interrogations monolignes

Opérateur	Besoin	Requête
Web	= pour les équijoointures ou auto-jointures (= teste une ligne)	<i>R1</i> (Pilotes de la compagnie de nom 'Air France' ayant plus de 500 heures de vol.) SELECT brevet, nom FROM Pilote WHERE compa = (SELECT comp FROM Compagnie WHERE nomComp = 'Air Franc AND nbHVol>500;
		<i>R3</i> (Pilotes sous la responsabilité du pilote de nom 'Alquié') SELECT brevet, nom FROM Pilote WHERE chefPil = (SELECT brevet FROM Pilote WHERE nom LIKE '%Alquié%');
> pour les inéquijoointures	<i>R5</i> (Pilotes ayant plus d'expérience que le pilote de brevet 'PL-2'.)	SELECT brevet, nom, nbHVol FROM Pilote WHERE nbHVol > (SELECT nbHVol FROM Pilote WHERE brevet = 'PL-2');

Sous-interrogations multilignes (IN, ALL et ANY)

Les opérateurs multilignes sont les suivants :



- IN compare un élément à une donnée quelconque d'une liste ramenée par la sous-interrogation. Cet opérateur est utilisé pour les équijointures et les autojointures (et les intersections). L'opérateur NOT IN sera employé pour les jointures externes (et les différences).
- ANY compare l'élément à chaque donnée ramenée par la sous-interrogation. L'opérateur « =ANY » équivaut à IN. L'opérateur « <ANY » signifie « inférieur à au moins une des valeurs », donc « inférieur au maximum ». L'opérateur « >ANY » signifie « supérieur à au moins une des valeurs », donc « supérieur au minimum ».
- ALL compare l'élément à tous ceux ramenés par la sous-interrogation. L'opérateur « <ALL » signifie « inférieur au minimum » et « >ALL » signifie « supérieur au maximum ».

Le tableau suivant détaille quelques sous-interrogations multilignes. Le dernier exemple programme une partie d'une jointure externe.

Tableau 4-41 Sous-interrogations multilignes

Opérateur	Besoin	Requête
IN	R2. Coordonnées des compagnies qui embauchent des pilotes de moins de 500 heures de vol.	<pre>SELECT nomComp, nrule, rue, ville FROM Compagnie WHERE comp IN (SELECT compa FROM Pilote WHERE nbHVol < 500);</pre>
= et IN	R4. Somme des heures de vol des pilotes placés sous la responsabilité des chefs pilotes de la compagnie de nom 'Air France'.	<pre>SELECT SUM(nbHVol) FROM Pilote WHERE chefPil IN (SELECT brevet FROM Pilote WHERE compa = (SELECT comp FROM Compagnie WHERE nomComp = 'Air France')));</pre>
NOT IN	Compagnies n'ayant pas de pilote.	<pre>SELECT nomComp, nrule, rue, ville FROM Compagnie WHERE comp NOT IN (SELECT compa FROM Pilote WHERE compa IS NOT NULL);</pre>



La directive NOT IN doit être utilisée avec prudence car elle retourne *faux* si un membre ramené par la sous-interrogation est NULL.

Afin d'illustrer les opérateurs ANY et ALL, considérons la table suivante. Nous avons mis en gras les nombres d'heures minimal et maximal des A320, en grisé les nombres d'heures minimal et maximal des avions de la compagnie 'AF'.

Figure 4-17 Table Avion

Avions

immat	typeAv	nbHVol	compa
A1	A320	1000	AF
A2	A330	1500	AF
A3	A320	550	SING
A4	A340	1800	SING
A5	A340	200	AF
A6	A330	100	AF

Le tableau suivant détaille quelques jointures procédurales utilisant les opérateurs ALL et ANY :

Tableau 4-42 Opérateurs ALL et ANY

Opérateur	Besoin	Requête et résultat
ANY	<i>R11.</i> Avions dont le nombre d'heures de vol est inférieur à celui de n'importe quel A320.	<pre>SELECT immat, typeAv, nbHVol FROM Avion WHERE nbHVol < ANY (SELECT nbHVol FROM Avion WHERE typeAv='A320'); +-----+-----+-----+ immat typeAv nbHVol +-----+-----+-----+ A3 A320 550.00 A5 A340 200.00 A6 A330 100.00 +-----+-----+-----+</pre>
	<i>R12.</i> Compagnies et leurs avions dont le nombre d'heures de vol est supérieur à celui de n'importe quel avion de la compagnie de code 'SING'.	<pre>SELECT immat, typeAv, nbHVol, compa FROM Avion WHERE nbHVol > ANY (SELECT nbHVol FROM Avion WHERE compa = 'SING'); +-----+-----+-----+-----+ immat typeAv nbHVol compa +-----+-----+-----+-----+ A1 A320 1000.00 AF A2 A330 1500.00 AF A4 A340 1800.00 SING +-----+-----+-----+-----+</pre>

Tableau 4-42 Opérateurs ALL et ANY (suite)

Opérateur	Besoin	Requête et résultat									
ALL	R13. Avions dont le nombre d'heures de vol est inférieur à tous les A320.	<pre>SELECT immat, typeAv, nbHVol FROM Avion WHERE nbHVol < ALL (SELECT nbHVol FROM Avion WHERE typeAv='A320') ;</pre> <table border="1"> <thead> <tr> <th>immat</th> <th>typeAv</th> <th>nbHVol</th> </tr> </thead> <tbody> <tr> <td>A5</td> <td>A340</td> <td>200.00</td> </tr> <tr> <td>A6</td> <td>A330</td> <td>100.00</td> </tr> </tbody> </table>	immat	typeAv	nbHVol	A5	A340	200.00	A6	A330	100.00
immat	typeAv	nbHVol									
A5	A340	200.00									
A6	A330	100.00									
	R14. Compagnies et leurs avions dont le nombre d'heures de vol est supérieur à tous les avions de la compagnie de code 'AF'.	<pre>SELECT immat, typeAv, nbHVol, compa FROM Avion WHERE nbHVol > ALL (SELECT nbHVol FROM Avion WHERE compa = 'AF') ;</pre> <table border="1"> <thead> <tr> <th>immat</th> <th>typeAv</th> <th>nbHVol</th> <th>compa</th> </tr> </thead> <tbody> <tr> <td>A4</td> <td>A340</td> <td>1800.00</td> <td>SING</td> </tr> </tbody> </table>	immat	typeAv	nbHVol	compa	A4	A340	1800.00	SING	
immat	typeAv	nbHVol	compa								
A4	A340	1800.00	SING								

Jointures mixtes

Une jointure mixte combine des clauses de jointures relationnelles, procédurales (avec des sous-interrogations) ou des clauses de jointures SQL2.

Jointure relationnelle procédurale

La jointure mixte suivante combine une clause de jointure relationnelle (en gras) avec une jointure procédurale (en souligné) pour programmer la requête R4 :

```
SELECT SUM(p1.nbHVol)
      FROM Pilote p1, Pilote p2
     WHERE p1.chefPil = p2.brevet
       AND p2.compa = (SELECT comp FROM Compagnie WHERE nomComp =
          'Air France') ;
```

Ce type d'écriture peut être intéressant s'il n'est pas nécessaire d'afficher des colonnes de tables présentes dans les sous-interrogations, ou si l'on désire appliquer des fonctions à des regroupements.

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 pilote 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	nbtVol	compa
PL-1	Pierre Lamothe	450	AF
PL-2	Didier Linxe	900	AF
PL-3	Christian Soutou	1000	SING
PL-4	Henri Alquié	3400	AF
PL-5	Michel Castaings		

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	b	
compa	nbpil	total
AF	3	5
SING	1	
	1	

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 enregistrement.

ment extrait par la requête de niveau supérieur. Cette technique peut être aussi utilisée dans les ordres UPDATE et DELETE. La forme générale d'une sous-interrogation synchronisée est suivante. Les alias des tables sont utiles pour pouvoir manipuler des colonnes de tables de différents niveaux.

```
SELECT alias1.c
  FROM nomTable1 alias1
 WHERE colonne(s) opérateur (SELECT alias2.z...
                                FROM nomTable2 alias2
                                WHERE alias1.x opérateur alias2.y)
 [AND (conditionsTable1)];
```

Une sous-interrogation synchronisée peut ramener une ou plusieurs lignes. Différents opérateurs peuvent être employés ($=, >, <, >=, <=, \text{ EXISTS}$).

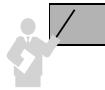
Opérateur mathématique

Le tableau suivant détaille un exemple d'opérateur mathématique appliqué à une sous-interrogation synchronisée :

Tableau 4-44 Sous-interrogation synchronisée

Besoin	Requête et résultat																
R15. Avions dont le nombre d'heures de vol est supérieur au nombre d'heures de vol moyen des avions de leur compagnie (ici 700 heures pour 'AF' et 1115 heures pour 'SING').	<pre>SELECT avi1.* FROM Avion avi1 WHERE avi1.nbHVol > (SELECT AVG(avi2.nbHVol) FROM Avion avi2 WHERE avi2.compa = avi1.compa);</pre> <table border="1"><thead><tr><th>immat</th><th>typeAv</th><th>nbHVol</th><th>compa</th></tr></thead><tbody><tr><td>A1</td><td>A320</td><td>1000.00</td><td>AF</td></tr><tr><td>A2</td><td>A330</td><td>1500.00</td><td>AF</td></tr><tr><td>A4</td><td>A340</td><td>1800.00</td><td>SING</td></tr></tbody></table>	immat	typeAv	nbHVol	compa	A1	A320	1000.00	AF	A2	A330	1500.00	AF	A4	A340	1800.00	SING
immat	typeAv	nbHVol	compa														
A1	A320	1000.00	AF														
A2	A330	1500.00	AF														
A4	A340	1800.00	SING														

Opérateur EXISTS



L'opérateur EXISTS permet d'interrompre la sous-interrogation dès le premier enregistrement trouvé. La valeur FALSE est retournée si aucun enregistrement n'est extrait par la sous-interrogation.

Utilisons la table suivante pour décrire l'utilisation de l'opérateur EXISTS :

Figure 4-19 Utilisation de EXISTS

brevet	nom	nbHVol	compa	chefPil
PL-1	Pierre Lamothe	450	AF	
PL-2	Didier Linxe	900	AF	PL-4
PL-3	Christian Soutou	1000	SING	PL-4
PL-4	Henri Alquié	3400	AF	
PL-5	Michel Castaings			

La sous-interrogation synchronisée est surlignée dans le script suivant :

Tableau 4-45 Opérateur EXISTS

Web	Besoin	Requête et résultat
	<p>R15. Pilotes ayant au moins un pilote sous leur responsabilité.</p>	<pre>SELECT pill.brevet, pill.nom, pill.compa FROM Pilote pill WHERE EXISTS (SELECT pil2.* FROM Pilote pil2 WHERE pil2.chefPil = pill.brevet)</pre> <pre>+-----+-----+-----+ brevet nom compa +-----+-----+-----+ PL-4 Henri Alquié AF +-----+-----+-----+</pre>

Opérateur NOT EXISTS



L'opérateur NOT EXISTS retourne la valeur TRUE si aucun enregistrement n'est extrait par la sous-interrogation. Cet opérateur peut être utilisé pour écrire des jointures externes.

Tableau 4-46 Opérateur NOT EXISTS

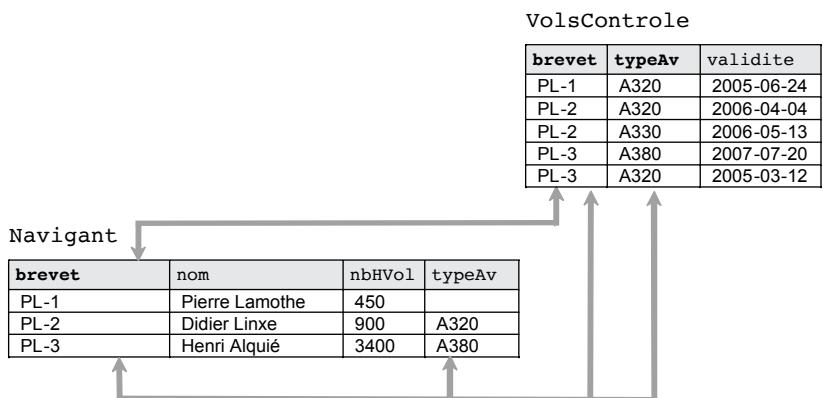
Besoin	Requête et résultat
Liste des compagnies n'ayant pas de pilote.	<pre>SELECT cpg.* FROM Compagnie cpg WHERE NOT EXISTS (SELECT compa FROM Pilote WHERE compa = cpg.compa)</pre> <pre>+-----+-----+-----+-----+ comp nrue rue ville nomComp +-----+-----+-----+-----+ CAST 1 G. Brassens Blagnac Castanet +-----+-----+-----+-----+</pre>

Autres directives SQL2

Étudions enfin les autres options des jointures SQL2 (NATURAL JOIN, USING et CROSS JOIN).

Considérons le schéma suivant (des colonnes portent le même nom). La colonne typeAv dans la table Navigant désigne le type d'appareil sur lequel le pilote est instructeur.

Figure 4-20 Deux tables à mettre en jointure naturelle



Opérateur NATURAL JOIN



La jointure naturelle est programmée par la clause NATURAL JOIN. La clause de jointure est automatiquement construite sur la base de toutes les colonnes portant le même nom entre les deux tables.

Les concepteurs devraient ainsi penser à nommer d'une manière semblable clés primaires et clés étrangères ! Ce principe n'est pas souvent appliqué aux bases volumineuses.

Le tableau suivant détaille deux écritures possibles d'une jointure naturelle. La clause de jointure est basée sur les colonnes (brevet, typeAv). Une clause WHERE aurait pu aussi être programmée.

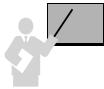
Tableau 4-47 Jointures naturelles

Besoin	Jointures SQL2 et résultat
Navigants qualifiés sur un type d'appareil et instructeurs sur ce même type.	<pre>SELECT brevet, nom, typeAv, validite FROM Naviguant NATURAL JOIN VolsControle; --équivalent à SELECT brevet, nom, typeAv, validite FROM VolsControle NATURAL JOIN Naviguant;</pre>

Tableau 4-47 Jointures naturelles (suite)

Besoin	Jointures SQL2 et résultat		
brevet	nom	typeAv	validite
PL-2	Didier Linxe	A320	2006-04-04
PL-3	Henri Alquié	A380	2007-07-20

Opérateur USING



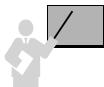
La directive **USING(*col1, col2...*)** de la clause JOIN programme une jointure naturelle entre à un ensemble de colonnes. Il ne faut pas utiliser d'alias de tables dans la liste colonnes.

Dans notre exemple, on peut restreindre la jointure naturelle aux colonnes *brevet* et *typeAv*. Si on les positionnait (*brevet*, *typeAv*) dans la directive USING, cela revient à construire un **NATURAL JOIN**. Le tableau suivant détaille deux écritures d'une jointure naturelle restreinte :

Tableau 4-48 Jointures naturelles restreintes

Web	Besoin	Jointures SQL2 et résultat																		
	Nom des navigants avec leurs qualifications et dates de validité.	<pre>SELECT nom, v.typeAv, v.validite FROM Naviguant JOIN VolsControle v USING(brevet); SELECT nom, v.typeAv, v.validite FROM VolsControle JOIN Naviguant USING(brevet);</pre>																		
<table border="1"> <tbody> <tr> <td>nom</td> <td>typeAv</td> <td>validite</td> </tr> <tr> <td>Pierre Lamothe</td> <td>A320</td> <td>2005-06-24</td> </tr> <tr> <td>Didier Linxe</td> <td>A320</td> <td>2006-04-04</td> </tr> <tr> <td>Didier Linxe</td> <td>A330</td> <td>2006-05-13</td> </tr> <tr> <td>Henri Alquié</td> <td>A380</td> <td>2007-07-20</td> </tr> <tr> <td>Henri Alquié</td> <td>A320</td> <td>2005-03-12</td> </tr> </tbody> </table>			nom	typeAv	validite	Pierre Lamothe	A320	2005-06-24	Didier Linxe	A320	2006-04-04	Didier Linxe	A330	2006-05-13	Henri Alquié	A380	2007-07-20	Henri Alquié	A320	2005-03-12
nom	typeAv	validite																		
Pierre Lamothe	A320	2005-06-24																		
Didier Linxe	A320	2006-04-04																		
Didier Linxe	A330	2006-05-13																		
Henri Alquié	A380	2007-07-20																		
Henri Alquié	A320	2005-03-12																		

Opérateur CROSS JOIN



La directive **CROSS JOIN** programme un produit cartésien qu'on peut restreindre dans la clause WHERE.

Le tableau suivant présente deux écritures d'un produit cartésien (seul l'ordre d'affichage des colonnes change) :

Tableau 4-49 Produit cartésien

Web	Besoin	Jointures SQL2 et résultat					
	Combinaison de toutes les lignes des deux tables.	SELECT * FROM Naviguant CROSS JOIN VolsControle -- équivalent à SELECT * FROM VolsControle CROSS JOIN Naviguant;					
		<pre>+-----+-----+-----+-----+-----+-----+ brevet nom nbHVol typeAv brevet typeAv validite +-----+-----+-----+-----+-----+-----+ PL-1 Pierre Lamothe 450.00 NULL PL-1 A320 2005-06-24 PL-2 Didier Linxe 900.00 A320 PL-1 A320 2005-06-24 PL-3 Henri Alquié 3400.00 A380 PL-1 A320 2005-06-24 PL-1 Pierre Lamothe 450.00 NULL PL-2 A320 2006-04-04 ... 15 rows in set</pre>					

Division

La division est un opérateur algébrique et non ensembliste. Cet opérateur est semblable sur principe à l'opération qu'on apprend au CM2 (oubliée plus tard en terminale à cause des calculatrices). La division est un opérateur binaire comme la jointure, car il s'agit de diviser une table (ou partie de) par une autre table (ou partie de). Il est possible d'opérer une division à partir d'une seule table, en ce cas on divise deux parties de cette table (analogie aux autojointures).



L'opérateur de division n'est pas fourni par MySQL (ni par aucun de ses concurrents d'ailleurs). Il n'existe donc malheureusement pas d'instruction DIVIDE.



Est-ce la complexité ou le manque d'intérêt qui freinent les éditeurs de logiciels à programmer ce concept ? La question reste en suspens, alors si vous avez un avis à ce sujet, faites-moi signe !

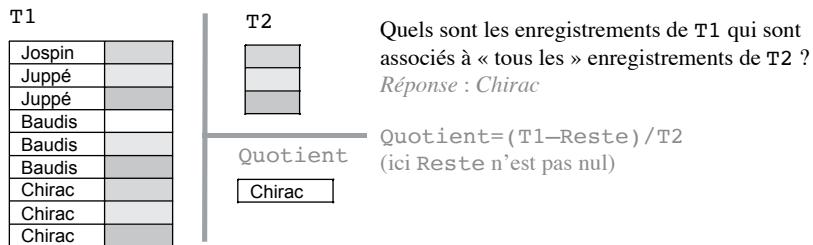
Cet opérateur permet de traduire le terme « pour tous les » des requêtes qu'on désire programmer en SQL.

On peut aussi dire que lorsque vous voulez comparer un ensemble avec un groupe de référence, il faut programmer une division.

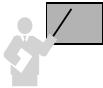
La division peut se programmer sous MySQL à l'aide d'une différence (NOT IN) et la fonction NOT EXISTS.

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 que l'auteur énoncé ci-avant, à savoir comparer un ensemble (la table $T1$) avec un ensemble de différences (la table $T2$).

Figure 4-21 Division



Définition



La division de la table $T1[a_1, \dots, a_n, b_1, \dots, b_n]$ par la table $T2[b_1, \dots, b_n]$ (la structure de $T2$ est incluse dans la structure de $T1$) donne la table $T3[a_1, \dots, a_n]$ qui contient les enregistrements vérifiant $t_i \in T3$ (de structure $[a_1, \dots, a_n]$), $t_j \in T2$ (t_j de structure $[b_1, \dots, b_n]$) et $t_i, t_j \in T1$ (de structure $[a_1, \dots, a_n, b_1, \dots, b_n]$).

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, car 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 concevables dans les deux tables sont grisesées.

Figure 4-22 Divisions à programmer

The diagram illustrates the division process. A central 'Web' database icon is connected to three tables: 'Affretements', 'Compagnie', and 'Résultat'. The 'Affretements' table contains data for aircraft (immat, typeAv, compa, dateAff). The 'Compagnie' table lists airline companies (comp, nomComp, pays). The 'Résultat' table shows the result of the division, containing rows where an aircraft from 'Affretements' has no corresponding entry in 'Compagnie'.

Affretements			
immat	typeAv	compa	dateAff
A1	A320	SING	1965-05-13
A2	A340	AF	1968-06-22
A3	Mercure	AF	1965-02-05
A4	A330	ALIB	1965-01-16
A3	Mercure	ALIB	1942-03-05
A3	Mercure	SING	1987-03-01

Compagnie		
comp	nomComp	pays
AF	Air France	F
ALIB	Air Lib	F
SING	Singapore AL	SG

Résultat	
immat	typeAv
A3	Mercure

L’opérateur de différence (programmé avec NOT IN) combiné à la fonction EXISTS permet de programmer ces deux comparaisons (un ensemble inclus dans un autre et une égalité d’ensembles). Il existe d’autres solutions à base de regroupements et de sous-interrogations (synchronisées ou pas) que nous n’étudierons pas, parce qu’elles me semblent plus compliquées. Écrivons à présent ces deux divisions à l’aide de requêtes SQL.

Division inexacte

Pour programmer le fait qu’un ensemble est seulement inclus dans un autre (ici $A \subset B$), il faut qu’il n’existe pas d’élément dans l’ensemble $\{A-B\}$. La différence se programme à l’aide de l’opérateur NOT IN, l’inexistence d’élément se programme à l’aide de la fonction NOT EXISTS comme le montre la requête suivante :

```

SELECT DISTINCT immat, typeAv FROM Affretements aliasAff
WHERE NOT EXISTS
  (SELECT DISTINCT comp FROM Compagnie WHERE pays = 'F'
    AND comp NOT IN
      (SELECT compa FROM Affretements WHERE immat = aliasAff.immat));

```

Parcours de tous les avions

Ensemble A de référence

Ensemble B à comparer

Division exacte

Pour programmer le fait qu’un ensemble est strictement égal à un autre (ici $A=B$), il faut qu’il n’existe ni d’élément dans l’ensemble $\{A-B\}$ ni dans l’ensemble $\{B-A\}$. La traduction mathématique est la suivante : $A=B \Leftrightarrow (A-B=\emptyset \text{ et } B-A=\emptyset)$. Les opérateurs se programment de même manière que pour la requête précédente. Le « et » se programme avec un AND.

```

SELECT DISTINCT immat, typeAv FROM Affrètements aliasAff
WHERE NOT EXISTS
  (SELECT comp FROM Compagnie WHERE pays = 'F' '
   AND comp NOT IN
    (SELECT compa FROM Affrètements WHERE immat = aliasAff.immat ))
AND NOT EXISTS
  (SELECT compa FROM Affrètements WHERE immat = aliasAff.immat
   AND compa NOT IN
    (SELECT comp FROM Compagnie WHERE pays = 'F' ));

```

A-B-A

Résultats en HTML ou XML

Les options « --html » et « --xml » (étudiées dans l'introduction) permettent de formater le résultat des extractions au standard du Web. Cela peut être intéressant si vous voulez rapidement publier des pages (qui seront statiques bien sûr) ou composer des fichiers destinés à l'échange de données provenant de votre base. Ainsi la requête précédente fournira les deux sorties suivantes. Concernant XML, la balise de plus haut niveau contient un attribut décrivant la requête.

Tableau 4-50 Résultats formatés pour le Web

 Connexion avec --html	Connexion avec --xml
<pre> <TABLE BORDER=1> <TR><TH>immat</TH><TH>typeAv</TH></TR> <TR><TD>A3</TD><TD>Mercure</TD></TR> </TABLE> </pre>	<pre> <resultset statement="SELECT DISTINCT immat, typeAv FROM Affrètements aliasAff WHERE NOT EXISTS (SELECT comp FROM Compagnie WHERE pays = 'F' ') AND comp NOT IN (SELECT compa FROM Affrètements WHERE immat = aliasAff.immat) AND NOT EXISTS (SELECT compa FROM Affrètements WHERE immat = aliasAff.immat AND compa NOT IN (SELECT comp FROM Compagnie WHERE pays = 'F'))"> <row> <field name="immat">A3</field> <field name="typeAv">Mercure</field> </row> </resultset> </pre>

Mécanisme inverse à LOAD DATA INFILE étudié au chapitre 2, l'exportation de données (au format de fichiers) extraites à l'aide d'une requête peut être programmée à l'aide de la directive INTO OUTFILE de l'instruction SELECT. Une telle requête écrit dans un fichier dans le répertoire du serveur. Le privilège FILE est requis. Le fichier cible doit être inexistant avant d'exécuter son chargement. La syntaxe simplifiée de cette directive est la suivante :

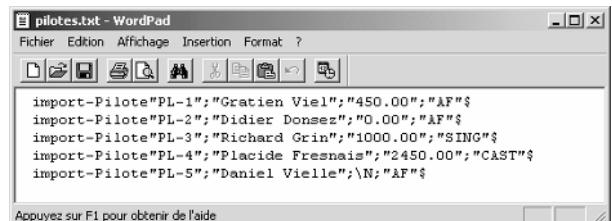
```
SELECT [ { DISTINCT | DISTINCTROW } | ALL ] listeColonnes
      FROM nomTable1 [,nomTable2]. [ WHERE condition ]
      INTO OUTFILE 'cheminEtNomFichier'
          [FIELDS [TERMINATED BY 'string']
           [[OPTIONALLY] ENCLOSED BY 'char']]
           [ESCAPED BY 'char' ] ]
          [LINES [STARTING BY 'string']
           [TERMINATED BY 'string']] ];
```

- FIELDS décrit comment seront formatées dans le fichier les colonnes extraites de(s) table(s). 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 ''.

Créons le fichier « pilotes.txt » situé dans le répertoire « D:\\dev », en exportant la totalité des enregistrements de la table Pilote (SELECT *) décrite au début du chapitre. Le fichier est ensuite ouvert à l'aide du WordPad. Notez l'utilisation du double « \\ » pour désigner une arborescence Windows. Le caractère NULL est exporté par le caractère « \\N ».

Figure 4-23 Crédit d'un fichier

```
SELECT * INTO OUTFILE 'D:\\dev\\pilotes.txt'
          FIELDS TERMINATED BY ';' ENCLOSED BY ''
          LINES STARTING BY 'import-Pilote' TERMINATED BY '$\\n'
          FROM Pilote;
```



Les objectifs de ces exercices sont :

- de créer dynamiquement des tables et leurs données ;
- d'écrire des requêtes monotables et multitablettes ;
- de réaliser des modifications synchronisées ;
- de composer des jointures et des divisions.

Exercice

4.1 Crédation dynamique de tables

Écrire le script `créaDynamique.sql` permettant de créer les tables `Softs` et `PCSeuls` sur la base des enregistrements de la table `Logiciels` que vous avez créée et alimentée précédemment. La table `PCSeuls` doit seulement contenir les enregistrements de la table `Poste`, qui sont de type 'PCWS' ou 'PCNT'. Vérifier :

Figure 4-24 Structures des nouvelles tables

Softs					
nomSoft	version	prix			
PCSeuls					
nP	nomP	seg	ad	typeP	salle

La table `Softs` sera construite sur la base de tous les enregistrements de la table `Logiciels` que vous avez créée et alimentée précédemment. La table `PCSeuls` doit seulement contenir les enregistrements de la table `Poste`, qui sont de type 'PCWS' ou 'PCNT'. Vérifier :

```
SELECT * FROM Softs;  
SELECT * FROM PCSeuls;
```

Exercice

4.2 Requêtes monotables

Écrire le script `requêtes.sql` permettant d'extraire, à l'aide d'instructions `SELECT`, les données suivantes :

- 1 Type du poste 'p8'.
- 2 Noms des logiciels 'UNIX'.
- 3 Noms, adresses IP, numéros de salle des postes de type 'UNIX' ou 'PCWS'.
- 4 Même requête pour les postes du segment '130.120.80' triés par numéros de salles décroissants.
- 5 Numéros des logiciels installés sur le poste 'p6'.
- 6 Numéros des postes qui hébergent le logiciel 'log1'.
- 7 Noms et adresses IP complètes (ex : '130.120.80.01') des postes de type 'TX' (utiliser la fonction de concaténation).

Exercice 4.3

Fonctions et groupements

- 8 Pour chaque poste, le nombre de logiciels installés (en utilisant la table `Installer`).
- 9 Pour chaque salle, le nombre de postes (à partir de la table `Poste`).
- 10 Pour chaque logiciel, le nombre d'installations sur des postes différents.
- 11 Moyenne des prix des logiciels 'UNIX'.
- 12 Plus récente date d'achat d'un logiciel.
- 13 Numéros des postes hébergeant 2 logiciels.
- 14 Nombre de postes hébergeant 2 logiciels (utiliser la requête précédente en faisant un `SELECT` dans la clause `FROM`).

Exercice 4.4

Requêtes multitable

Opérateurs ensemblistes

- 15 Types de postes non recensés dans le parc informatique (utiliser la table `Types`).
- 16 Types existant à la fois comme types de postes et de logiciels.
- 17 Types de postes de travail n'étant pas des types de logiciels.

Jointures procédurales

- 18 Adresses IP complètes des postes qui hébergent le logiciel 'log6'.
- 19 Adresses IP complètes des postes qui hébergent le logiciel de nom 'Oracle 8'.
- 20 Noms des segments possédant exactement trois postes de travail de type 'TX'.
- 21 Noms des salles ou l'on peut trouver au moins un poste hébergeant le logiciel 'Oracle 6'.
- 22 Nom du logiciel acheté le plus récent (utiliser la requête 12).

Jointures relationnelles

Écrire les requêtes 18, 19, 20, 21 avec des jointures de la forme relationnelle. Numéroter ces nouvelles requêtes de 23 à 26.

- 27 Installations (nom segment, nom salle, adresse IP complète, nom logiciel, date d'installation) triées par segment, salle et adresse IP.

Jointures SQL2

Écrire les requêtes 18, 19, 20, 21 avec des jointures SQL2 (`JOIN`, `NATURAL JOIN`, `JOIN USING`). Numéroter ces nouvelles requêtes de 28 à 31.

Exercice 4.5

Modifications synchronisées

Écrire le script modifSynchronisées.sql pour ajouter les lignes suivantes dans la table Installer :

Figure 4-25 Lignes à ajouter

Installer				
nPoste	nLog	numIns	dateIns	délai
...
p2	log6	séquence...	SYSDATE()	NULL
p8	log1		SYSDATE()	NULL
p10	log1		SYSDATE()	NULL

Écrire les requêtes UPDATE synchronisées de la forme suivante :

```
UPDATE table1 alias1  
    SET colonne = (SELECT COUNT(*)  
                    FROM table2 alias2  
                   WHERE alias2.colonneA = alias1.colonneB...);
```

Pour mettre à jour automatiquement les colonnes rajoutées :

- nbSalle dans la table Segment (nombre de salles traversées par le segment) ;
- nbPoste dans la table Segment (nombre de postes du segment) ;
- nbInstall dans la table Logiciel (nombre d'installations du logiciel) ;
- nbLog dans la table Poste (nombre de logiciels installés par poste).

Vérifier le contenu des tables modifiées (Segment, Logiciel et Poste).

Exercice 4.6

Opérateurs existentiels

Rajouter au script requêtes.sql, les instructions SELECT pour extraire les données suivantes :

Sous-interrogation synchronisée

32 Noms des postes ayant au moins un logiciel commun au poste 'p6' (on doit trouver les postes p8 et p10).

Divisions

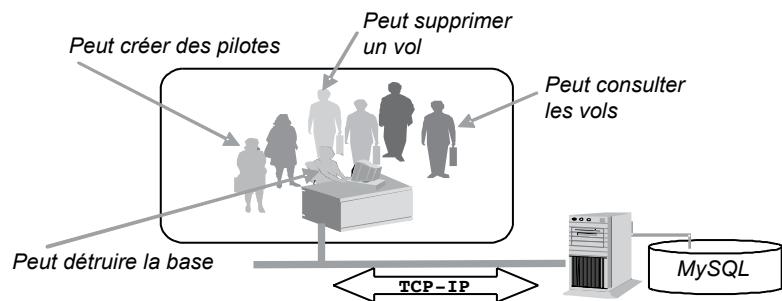
33 Noms des postes ayant les mêmes logiciels que le poste 'p6' (les postes peuvent avoir plus de logiciels que 'p6'). On doit trouver les postes 'p2' et 'p8' (division inexacte).

34 Noms des postes ayant exactement les mêmes logiciels que le poste 'p2' (division exacte), on doit trouver 'p8'.

Contrôle des données

Comme dans tout système multi-utilisateur, l'usager d'un SGBD doit être identifié avant de pouvoir utiliser des ressources. Les accès aux informations et à la base de données doivent être contrôlés à des fins de sécurité et de cohérence. La figure suivante illustre un groupe d'utilisateurs dans lequel existe une classification entre ceux qui peuvent consulter, mettre à jour ou supprimer des enregistrements, voire les tables.

Figure 5-1 Conséquences de l'aspect multi-utilisateur



Nous verrons dans cette section les aspects du langage SQL qui concernent le contrôle des données et des accès. Nous étudierons :

- la gestion des utilisateurs qui manipuleront des bases de données dans lesquelles se trouvent des objets tels que des tables, index, séquences (pour l'instant implémentées par des colonnes AUTO_INCREMENT), vues, procédures, etc. ;
- la gestion des privilèges qui permettent de donner des droits sur la base de données (privileges système) et sur les données de la base (privileges objet) ;
- la gestion des vues ;
- l'utilisation du dictionnaire des données (base de données `information_schema`)

Le chapitre 9 détaille l'outil graphique *MySQL Administrator* qui permet de s'affranchir d'écrire des instructions SQL.

Présenté rapidement à l'introduction, nous verrons qu'un utilisateur (*user*) est identifié par MySQL par son nom et celui de la machine à partir de laquelle il se connecte. Cela fait, il pourra accéder à différents objets (tables, vues, séquences, index, procédures, etc.) d'une ou plusieurs bases sous réserve d'avoir reçu un certain nombre de privilèges.

Classification

Les types d'utilisateurs, leurs fonctions et leur nombre peuvent varier d'une base à une autre. Néanmoins, pour chaque base de données en activité, on peut classifier les utilisateurs de manière suivante :

- Le DBA (*DataBase Administrator*). Il en existe au moins un. Une petite base peut n'avoir qu'un seul administrateur. Une base importante peut en regrouper plusieurs qui se partagent les tâches suivantes :
 - installation et mises à jour de la base et des outils éventuels ;
 - gestion de l'espace disque et des espaces pour les données ;
 - gestion des utilisateurs et de leurs objets (s'ils ne les gèrent pas eux-mêmes) ;
 - optimisation des performances ;
 - sauvegardes, restaurations et archivages ;
 - contact avec le support technique.
- L'administrateur réseau (qui peut être le DBA) se charge de la configuration des couches client pour les accès distants.
- Les développeurs qui conçoivent et mettent à jour la base. Ils peuvent aussi agir sur leurs objets (création et modification des tables, index, séquences, etc.). Ils transmettent au DBA leurs demandes spécifiques (stockage, optimisation, sécurité).
- Les administrateurs d'application qui gèrent les données manipulées par la ou les applications. Pour les petites et les moyennes bases, le DBA joue ce rôle.
- Les utilisateurs qui se connectent et interagissent avec la base à travers les applications ou à l'aide d'outils (interrogations pour la génération de rapports, ajouts, modifications ou suppressions d'enregistrements).

Tous seront des utilisateurs (au sens MySQL) avec des privilèges différents.

Création d'un utilisateur [CREATE USER]

Pour pouvoir créer un utilisateur, vous devez posséder le privilège CREATE USER ou INSERT sur la base système `mysql` (car c'est la table `mysql.user` qui stockera l'existence de ce nouvel arrivant).

La syntaxe de création d'un utilisateur est la suivante :

```
CREATE USER utilisateur
  [ IDENTIFIED BY [PASSWORD] 'motdePasse' ]
  [,utilisateur2 [IDENTIFIED BY [PASSWORD] 'motdePasse2'] ...]
```

- IDENTIFIED BY *motdePasse* permet d'affecter un mot de passe (16 caractères max sensibles à la casse) à un utilisateur (16 caractères maximum, sensibles aussi à la casse).

Le tableau suivant décrit la création d'un utilisateur (à exécuter en étant connecté en local que **root**) :

Tableau 5-1 Crédit d'un utilisateur

Instruction SQL	Résultat
<code>CREATE USER soutou@localhost IDENTIFIED BY 'iut';</code>	<i>soutou</i> est déclaré « utilisateur à accès local », il devra se connecter à l'aide de son mot de passe.

Par défaut, les utilisateurs, une fois créés, n'ont aucun droit sur aucune base de données (sauf en lecture écriture sur la base `test` et en lecture seule sur la base `information_schema`). La section *Privilèges* étudie ces droits.

Un utilisateur bien connu

Lors de l'installation, vous avez dû noter la présence de l'utilisateur `root` (mot de passe root à l'installation). Cet utilisateur est le DBA que MySQL vous offre. Il vous permettra d'effectuer vos tâches administratives en ligne de commande ou par une console graphique (création d'utilisateurs par exemple).

Liste des utilisateurs

À propos de `root`, on le retrouve dans la table `user` de la base `mysql` (`mysql.user`). L'extraction des colonnes `User` et `Host` restitue la liste des utilisateurs connus du serveur. Si `root` n'avait pas sélectionné la base `mysql`, la commande à exécuter aurait été « `SELECT User,Host FROM mysql.user;` ».

```
(root@localhost) [mysql] mysql> SELECT User,Host FROM user;
+-----+-----+
| User   | Host    |
+-----+-----+
|       | %      |
|       | localhost |
| root   | localhost |
| soutou | localhost |
+-----+-----+
```

Vous devez posséder une table similaire. Il apparaît quatre accès potentiels. L'utilisateur vide ' ' correspond à une connexion anonyme. La machine désignée par « % » indique que la connexion est autorisée à partir de tout site (en supposant qu'un client MySQL est installé qu'il est relié au serveur par TCP-IP). La machine désignée par « localhost » spécifie que la connexion est autorisée en local.

Ici, la table fait état que l'accès anonyme (restreint toutefois à la base `test`, voir la section *Table mysql.db*) est permis en local et à partir de tout site, et que `soutou` comme `root` ne peuvent se connecter qu'en local.

Modification d'un utilisateur

Le mot de passe d'un utilisateur peut être modifié sans parler de privilèges. Nous verrons plus tard qu'il est possible de restreindre le nombre de requêtes (SELECT), de modifications (UPDATE), de connexions par heure et de connexions simultanées à un serveur.

Puisqu'il n'existe pas de commande ALTER USER, pour changer un mot de passe, il faut donc modifier la table `user` par la seule commande SQL capable de le faire : UPDATE.

L'instruction suivante modifie le mot de passe de l'utilisateur `soutou` pour l'accès en local. Notez l'utilisation de la fonction `PASSWORD()` qui code le mot de passe à affecter à la colonne `Password` de la table `user`. Il est plus prudent d'utiliser ensuite FLUSH PRIVILEGES qui recharge les tables système de manière à rendre la manipulation effective sur l'instant (un peu comme un COMMIT sur des données).

```
UPDATE mysql.user
    SET Password = PASSWORD('eyrolles')
    WHERE User = 'soutou'
    AND Host = 'localhost';
FLUSH PRIVILEGES;
```

Une fois cette modification réalisée, si `soutou` tente une connexion avec son ancien mot de passe, il vient l'erreur classique : « ERROR 1045 (28000) : Access denied for user 'soutou'@'localhost' (using password: xx) » (xx valant YES si `soutou` se connecte avec son ancien mot de passe, NO s'il n'en donne pas).

Chaque utilisateur peut changer son propre mot de passe à l'aide de cette instruction s'il en a le privilège. Mais attention ! Le fait de lui donner ce droit (nous verrons plus loin comment faire) implique également qu'il puisse aussi modifier les mots de passe de ses copains, ainsi que celui du `root` !

Renommer un utilisateur (RENAME USER)

Pour pouvoir renommer un utilisateur, vous devez posséder le privilège CREATE USER (ou le privilège UPDATE sur la base de données `mysql`). La syntaxe SQL est la suivante :

```
RENAME USER utilisateur TO nouveauNom;
```

Penser à spécifier l'accès complet à renommer (`user@machine`). Les priviléges et le mot de passe ne changent pas. Le tableau suivant décrit trois opérations de renommage d'utilisateur (qui reviennent d'ailleurs à l'état initial).

Tableau 5-2 Renommer un utilisateur

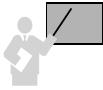
Instruction SQL	Commentaire
<code>RENAME USER soutou@localhost TO christiansoutou@localhost;</code>	L'accès <code>soutou</code> en local est renommé <code>christiansoutou</code> en local.
<code>RENAME USER christiansoutou@localhost TO christiansoutou@194.53.227.12;</code>	L'accès <code>christiansoutou</code> en local est renommé <code>christiansoutou</code> en accès distant.
<code>RENAME USER christiansoutou@194.53.227.12 TO soutou@localhost;</code>	L'accès est renommé complètement.

Suppression d'un utilisateur (DROP USER)

Pour pouvoir supprimer un utilisateur, vous devez posséder le privilège CREATE USER (et le privilège DELETE sur la base de données `mysql`). La syntaxe SQL est la suivante :

■ `DROP USER utilisateur [,utilisateur2 ...];`

Il faut spécifier l'accès à éliminer (`user@machine`). Tous les priviléges relatifs à cet accès sont détruits. Si l'utilisateur est connecté dans le même temps, sa suppression ne sera effectuée qu'à la fin de sa (dernière) session.



Aucune donnée d'aucune table que l'utilisateur aura mis à jour durant toutes ses connexions sera supprimée. Il n'y a pas de notion d'appartenance d'objets (tables, index, procédures etc.) à un utilisateur. Tout ceci est relatif à la base de données (`database`).

Pour supprimer le compte `soutou` en local, la commande à lancer est :

■ `DROP USER soutou@localhost;`

Gestion des bases de données

Abordée brièvement à l'introduction, une base de données (`database`) regroupe essentiellement des tables sur lesquelles l'administrateur affectera des autorisations à des utilisateurs. Cette notion de `database` s'apparente plutôt à celle de `schéma` (connu des utilisateurs d'Oracle). D'ailleurs dans l'instruction de création les deux mots peuvent être utilisés.

Création d'une base [CREATE DATABASE]

Pour pouvoir créer une base de données, vous devez posséder le privilège CREATE sur la nouvelle base (ou au niveau global pour créer toute table).

```
CREATE {DATABASE | SCHEMA} [ IF NOT EXISTS ] nomBase  
[ [DEFAULT] CHARACTER SET nomJeu ]  
[ [DEFAULT] COLLATE nomCollation ];
```

- **IF NOT EXISTS** évite une erreur dans le cas où la base de données existe déjà (auquel cas elle ne sera pas remplacée).
- **nomBase** désigne le nom de la base (64 caractères maximum, caractères compris par le système de gestion de fichier du système d'exploitation, notamment respectant les règles de nommage des répertoires). Les caractères « / », « \ », ou « . » sont proscrits.
- **CHARACTER SET** indique le jeu de caractères associé aux données qui résideront dans les tables de la base.
- **COLLATE** définit la collation¹ du jeu de caractères en question. La collation, dans le jargon informatique, permet de définir la position des caractères dans le jeu. Par exemple, il sera possible de différencier « à » de « a » ou pas (sensibilité diacritique). Le but étant de s'adapter aux différentes règles et langues de notre petite planète.

Une fois créée, vous constaterez l'existence d'un répertoire portant le nom de votre nouvelle base (par défaut sous C:\Program Files\MySQL\MySQL Server 5.n\data\nouvelleBase dans le cas de Windows). Ce répertoire contiendra les données des tables qui seront constituées dans la nouvelle base. Si vous concevez manuellement un répertoire (mkdir rep1 par exemple) dans le répertoire de data de MySQL, rep1 sera considéré comme une base de données avec le jeu de caractères par défaut (visible avec « SHOW DATABASES; »).

N'effacez pas le fichier db.opt qui stocke les caractéristiques de la base. Vous pouvez l'ouvrir avec un éditeur de texte pour connaître le jeu de caractères par défaut que MySQL affectera à vos bases en l'absence de clause CHARACTER SET. Souhaitons que ce ne soit pas gb2312 associé par défaut à la collation gb2312_chinese_ci qui vous ferait dire que je vous parle chinois ! C'est pourtant quelquefois ce que je ressens quand mes étudiants me regardent avec des yeux de poisson en utilisant le langage des carpes...

Le tableau suivant décrit la création de deux bases de données :

Tableau 5-3 Crédit de bases

Instruction SQL	Résultat
<code>CREATE DATABASE bdnouvelle DEFAULT CHARACTER SET ascii COLLATE ascii_general_ci;</code>	La base bdnouvelle est créée, le jeu de caractères par défaut est ASCII.
<code>CREATE DATABASE bdnouvelle2 DEFAULT CHARACTER SET gb2312;</code>	La base bdnouvelle2 est créée pour les Chinois.

1. Action de comparer entre eux des textes, des documents. Petit Larousse.

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 celles tables qui seront constituées dedans, à moins que la table ne soit combinée à un autre jeu 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` (avec jeu par défaut chinoise) en spécifiant que la colonne `col1` sera associée au jeu `cp850 : DOS Latin1 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 base 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 précéder 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
<pre>CREATE TABLE bdnouvelle.testUSE (col3 CHAR(5), col4 CHAR(4)) CHARACTER SET latin1; INSERT INTO bdnouvelle.testUSE VALUES ('ACTMP', 'IUT');</pre>	Création d'une table dans la base. Insertion d'une ligne.
<pre>USE bdnouvelle2;</pre>	Sélection de la base <code>bdnouvelle2</code> .
<pre>SELECT col, bdnouvelle.testUSE.col3 FROM testChap5, bdnouvelle.testUSE WHERE col2 = bdnouvelle.testUSE.col4;</pre>	Jointure de la table <code>testChap5</code> située dans la base active (<code>bdnouvelle2</code>) avec <code>testUSE</code> située dans la base <code>bdnouvelle</code> .
<pre>+-----+-----+ col col3 +-----+-----+ GTR ACTMP +-----+-----+</pre>	

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 contenus 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 ont été supprimées (fichiers à l'extension « .frm »).

Disons à présent *adios* à la base « chinoise » :

```
DROP DATABASE bdnouvelle2;
```

## Privilèges

---

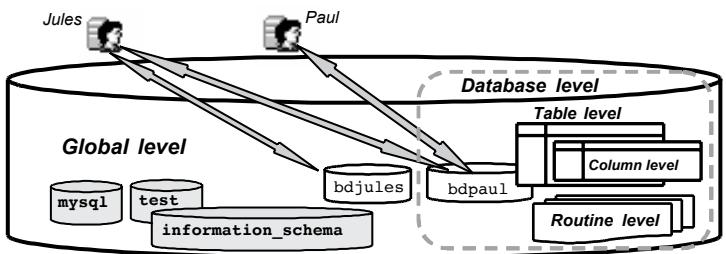
Depuis le début du livre, nous avons parlé de privilèges. Il est temps à présent de préciser ce que recouvre ce terme. Un privilège (sous-entendu *utilisateur*) est un droit d'exécuter une certaine instruction SQL (on parle de privilège *système*), ou un droit relatif aux données de tables situées dans 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.

## Niveaux de privilèges

La figure suivante illustre les différents niveaux de privilèges que l'on peut rencontrer :

Figure 5-2 Niveaux de privilèges

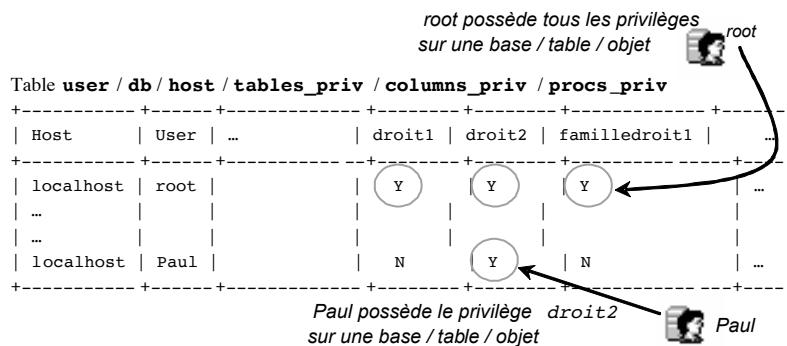


- *Global level* : privilèges s'appliquant à toutes les bases du serveur. Ces privilèges sont stockés dans la table mysql.user (exemple d'attribution d'un privilège global : GRANT CREATE ON \*.\*...).
- *Database level* : privilèges s'appliquant à tous les objets d'une base de données en particulier. Ces privilèges sont stockés dans les tables mysql.db et mysql.host (exemple d'attribution d'un privilège database : GRANT SELECT ON bdpaul.\*...).
- *Table level* : privilèges s'appliquant à la globalité d'une table d'une base de données en particulier. Ces privilèges sont stockés dans la table mysql.tables\_priv (exemple d'attribution d'un privilège table : GRANT INSERT ON bdpaul.Avion...).
- *Column level* : privilèges s'appliquant à une des colonnes d'une table d'une base de données en particulier. Ces privilèges sont stockés dans la table mysql.columns\_priv (exemple d'attribution d'un privilège column : GRANT UPDATE(nomColonne) ON bdpaul.Compagnie...).
- *Routine level* : privilèges globaux ou au niveau d'une base (CREATE ROUTINE, ALTER ROUTINE, EXECUTE, et GRANT) s'appliquant aux procédures cataloguées (étudiée au chapitre 7). Ces privilèges sont stockés dans la table mysql.procs\_priv de la base mysql (exemple d'attribution d'un privilège routine : GRANT EXECUTE ON PROCEDURE bdpaul.sp1...).

## Tables de la base mysql

Cinq tables de la base de données mysql suffisent à MySQL pour stocker les priviléges (système et objet) de tous les utilisateurs. La figure suivante illustre comment MySQL distribue toutes ces prérogatives toujours en fonction des accès (couple utilisateur, machine).

Figure 5-3 Stockage des prérogatives



La colonne Db est en plus dans les tables host, tables\_priv et columns\_priv, car elle est nécessaire pour désigner la base de données sur laquelle portera le droit ou la famille de droits.

Supposons, pour nos exemples, que l'utilisateur Paul (accès en local) et la base de données bdpaul soient créés.

```

CREATE DATABASE bdpaul;
CREATE USER Paul@localhost IDENTIFIED BY 'iut';

```

## Table mysql.user

Présenté brièvement au début du chapitre. Cette table est composée de 37 colonnes qui décrivent les priviléges au niveau global du serveur. Nous détaillons ici la signification des principales.

### Priviléges objet (LMD) sur toutes les bases de données

La requête suivante extrait les prérogatives de Paul (et des autres). Pour l'instant, le caractère 'N' étant dans toutes les colonnes, il ne peut ni interroger une table (Select\_priv), ni insérer dans une table (Insert\_priv), ni en modifier (Update\_priv), ni en supprimer (Delete\_priv), et ce quelle que soit la base de données (excepté les bases système test et information\_schema) sur laquelle il voudra se connecter.

```

SELECT Host, User, Select_priv, Insert_priv, Update_priv, Delete_priv
 FROM mysql.user;
+-----+-----+-----+-----+-----+
| Host | User | Select_priv | Insert_priv | Update_priv | Delete_priv |
+-----+-----+-----+-----+-----+
localhost	root	Y	Y	Y	Y
localhost	%	Y	Y	Y	Y
localhost	Paul	N	N	N	N
+-----+-----+-----+-----+-----+

```

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 colonne (`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 (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 Host,User, Create_view_priv, Show_view_priv FROM mysql.user;
+-----+-----+-----+-----+
| Host | User | Create_view_priv | Show_view_priv |
+-----+-----+-----+-----+
localhost	root	Y	Y
localhost		N	N
%		N	N
localhost	Paul	N	N
+-----+-----+-----+-----+

```

### **Privilèges à propos des procédures cataloguées sur toutes les bases de données**

La requête suivante extrait les prérogatives à propos des procédures cataloguées (détailée dans le chapitre 7). Pour l'instant, le caractère 'N' étant dans toutes les colonnes, Paul ne peut ni créer une procédure (Create\_routine\_priv), ni en modifier (Alter\_routine\_priv), ni en exécuter (Execute\_priv), et ce quelle que soit la base de données.

```

SELECT Host,User, Create_routine_priv, Alter_routine_priv, Execute_priv
 FROM mysql.user;
+-----+-----+-----+-----+
| Host | User | Create_routine_priv | Alter_routine_priv | Execute_priv |
+-----+-----+-----+-----+
localhost	root	Y	Y	Y
localhost		N	N	Y
%		N	N	N
localhost	Paul	N	N	N
+-----+-----+-----+-----+

```

### **Privilèges à propos des restrictions d'utilisateur**

La requête suivante extrait les prérogatives à propos des restrictions qu'on peut définir pour l'accès. Pour l'instant, le chiffre étant à 0 dans toutes les colonnes, aucun accès (utilisateur n'est limité concernant le nombre de requêtes (max\_questions), de modifications (max\_updates), de connexions par heure (max\_connections) et de connexions simultanées (max\_user\_connections) à un serveur.

```

SELECT Host, User, max_questions "Requetes", max_updates "Modifs" ,
 max_connections "Connexions", max_user_connections "Cx simult."
 FROM mysql.user;
+-----+-----+-----+-----+-----+
| Host | User | Requetes | Modifs | Connexions | Cx simult. |
+-----+-----+-----+-----+-----+
localhost	root	0	0	0	0
localhost		0	0	0	0
%		0	0	0	0
localhost	Paul	0	0	0	0
+-----+-----+-----+-----+-----+

```

## Privileges non abordés

D'autres colonnes de la table `mysql.user` sont intéressantes, mais sortent un peu du cadre de ce livre. J'ai donc fait l'impasse sur `Create_tmp_table_priv` (sert à créer des tables temporaires), `Lock_tables_priv` (pose des verrous explicites), `Shutdown_priv` (arrête et redémarre le serveur), `Process_priv` et `Super_priv` (gèrent les processus), `File_priv` (accède aux fichiers du système d'exploitation), `Repl_slave_priv` et `Repl_client_priv` (utilisés pour des aspects de réPLICATION de données).

Nul doute que vous saurez vous servir de ces priviléges en temps voulu, par analogie avec ceux que nous allons étudier.



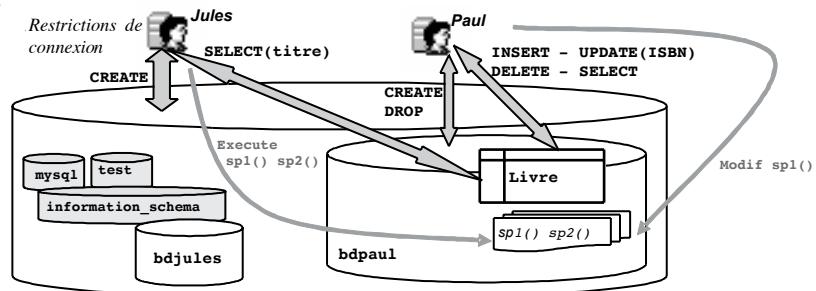
Le privilège `References_priv` n'est pas opérationnel encore. Il permettrait de bénéficier de l'intégrité référentielle entre deux tables appartenant à deux bases distinctes (par exemple une table `Avion` dans `bd1` ferait référence par clé étrangère à la table `Compagnie` dans `bd2`).

Avant de présenter les autres tables (`db, host, tables_priv, columns_priv` et `proxies_priv`) de la base `mysql`, étudions les instructions relatives à l'attribution d'un privilège (`GRANT`), qu'il soit système ou objet, et celles relatives à la révocation d'un privilège (`REVOKE`).

## Attribution de priviléges (GRANT)

La figure suivante illustre le contexte qui va servir d'exemple à l'attribution de prérogatives.

Figure 5-4 Attribution de priviléges



## Syntaxe

L'instruction `GRANT` permet d'attribuer un (ou plusieurs) privilège(s) à propos d'un objet (ou plusieurs) bénéficiaire(s). L'utilisateur qui exécute cette commande doit avoir reçu au moins le droit de transmettre ces priviléges (reçu avec la directive `GRANT OPTION`). Dans le cas de `root`, aucun problème, car il a implicitement tous les droits.

```

GRANT privilège [(col1 [, col2...])] [,privilège2 ...]
 ON [{TABLE | FUNCTION | PROCEDURE}]
 [nomTable | * | *.* | nomBase.*]
 TO utilisateur [IDENTIFIED BY [PASSWORD] 'password']
 [,utilisateur2 ...]
 [WITH [GRANT OPTION]
 [MAX_QUERIES_PER_HOUR nb]
 [MAX_UPDATES_PER_HOUR nb2]
 [MAX_CONNECTIONS_PER_HOUR nb3]
 [MAX_USER_CONNECTIONS nb4]];

```

- *privilège* : description du privilège (ex : SELECT, DELETE, etc.), voir le tableau suivant.
- *col* précise la ou les colonnes sur lesquelles se portent les priviléges SELECT, INSERT ou UPDATE (exemple : UPDATE(*typeAvion*) pour n'autoriser que la modification de la colonne *typeAvion*).
- GRANT OPTION : permet de donner le droit de retransmettre les priviléges reçus à une tierce personne.

Le tableau suivant résume la signification des principaux priviléges à accorder ou à révoquer.

**Tableau 5-5 Priviléges principaux pour GRANT et REVOKE**

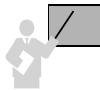
| <b>privilege</b>   | <b>Commentaire</b>                                                                                                                              |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| ALL [ PRIVILEGES ] | Tous les priviléges.                                                                                                                            |
| ALTER              | Modification de base/table.                                                                                                                     |
| ALTER ROUTINE      | Modification de procédure.                                                                                                                      |
| CREATE             | Création de base/table.                                                                                                                         |
| CREATE ROUTINE     | Création de procédure.                                                                                                                          |
| CREATE USER        | Création d'utilisateur.                                                                                                                         |
| CREATE VIEW        | Création de vue.                                                                                                                                |
| DELETE             | Suppression de données de table.                                                                                                                |
| DROP               | Suppression de base/table.                                                                                                                      |
| EXECUTE            | Exécution de procédure.                                                                                                                         |
| INDEX              | Création/Suppression d'index.                                                                                                                   |
| INSERT             | Insertion de données de table.                                                                                                                  |
| SELECT             | Extraction de données de table.                                                                                                                 |
| SHOW DATABASES     | Lister les bases.                                                                                                                               |
| SHOW VIEW          | Lister les vues d'une base.                                                                                                                     |
| SUPER              | Gestion des déclencheurs.                                                                                                                       |
| UPDATE             | Modification de données de table.                                                                                                               |
| USAGE              | Synonyme de « sans privilège », USAGE est utilisé pour conserver les priviléges précédemment définis tout en les restreignant avec des options. |

## Exemples

Le tableau suivant décrit l'affectation de quelques privilèges en donnant les explications ciées.

Tableau 5-6 Affectation de privilèges

| Instruction faite par root                                                                                                                                                                                                            | Explication                                                                                                                                                                                     |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>GRANT CREATE, DROP ON bdpaul.*<br/>      TO 'Paul'@'localhost';</pre>                                                                                                                                                            | Privilège système <i>database level</i> :<br>Paul (en accès local) peut créer et supprimer des tables dans la base bdpaul.                                                                      |
| <pre>GRANT INSERT, SELECT, DELETE, UPDATE(ISBN)<br/>    ON bdpaul.Livre<br/>    TO 'Paul'@'localhost';</pre>                                                                                                                          | Privilège objet <i>table level</i> :<br>Paul peut insérer, extraire, supprimer et modifier la colonne ISBN de la table Livre contenue dans la base bdpaul.                                      |
| <pre>GRANT ALTER ON bdpaul.Livre<br/>      TO 'Paul'@'localhost';</pre>                                                                                                                                                               | Privilège système <i>table level</i> :<br>Paul peut modifier la structure ou les contraintes de la table Livre contenue dans la base bdpaul.                                                    |
| <pre>GRANT SELECT(titre)<br/>    ON bdpaul.Livre<br/>    TO 'Jules'@'localhost'<br/>    WITH GRANT OPTION;</pre>                                                                                                                      | Privilège objet <i>column level</i> :<br>Jules peut extraire seulement la colonne titre de la table Livre contenue dans la base bdpaul. Il pourra par la retransmettre éventuellement ce droit. |
| <pre>GRANT CREATE ON *.*<br/>      TO 'Jules'@'localhost';</pre>                                                                                                                                                                      | Privilège système <i>global level</i> :<br>Jules peut créer des bases de données.                                                                                                               |
| <pre>GRANT USAGE ON bdpaul.*<br/>      TO 'Jules'@'localhost'<br/>      WITH<br/>        MAX_QUERIES_PER_HOUR 50<br/>        MAX_UPDATES_PER_HOUR 20<br/>        MAX_CONNECTIONS_PER_HOUR 6<br/>        MAX_USER_CONNECTIONS 3;</pre> | Privilège système <i>database level</i> :<br>Jules ne peut lancer, chaque heure, 50 SELECT, 20 UPDATE, se connecter (dont 3 connexions simultanées) sur la base données bdpaul.                 |



Tout ce que vous avez le droit de faire doit être explicitement autorisé par la commande GRANT. Ce qui n'est pas dit par GRANT n'est pas permis. Par exemple, Jules peut créer des bases, mais pas en détruire, Paul peut modifier le numéro ISBN d'un livre mais pas son titre, etc.

## Voir les privilèges

La commande SHOW GRANTS FOR liste les différentes instructions GRANT équivalentes à toutes les prérogatives d'un utilisateur donné. C'est bien utile quand vous avez attribué un certain nombre de privilèges à un utilisateur sans avoir pensé à les consigner dans un fichier de commande.

```
SHOW GRANTS FOR utilisateur;
```

Utilisons cette commande pour extraire les profils de Jules, de Paul et de l'administrateur en chef (accès en local). J'avoue avoir un peu retravaillé l'état de sortie (sans en modifier une ligne quand même).

```
SHOW GRANTS FOR 'Jules'@'localhost';
+-----+
| Grants for Jules@localhost
+-----+
| GRANT CREATE ON *.* TO 'Jules'@'localhost' IDENTIFIED BY PASSWORD
 '*6AE163FB9EE8BB011EB2E87316AA5BE563A6CDB7' WITH MAX_QUERIES_PER_HOUR 50
 MAX_UPDATES_PER_HOUR 20 MAX_CONNECTIONS_PER_HOUR 6 MAX_USER_CONNECTIONS 3
| GRANT SELECT (titre) ON `bdpaul`.`Livre` TO 'Jules'@'localhost'
 WITH GRANT OPTION
+-----+
```

On remarque que MySQL a regroupé deux priviléges en une instruction GRANT (CREATE et les restrictions de connexions). Par là même, on se rend compte que les prérogatives d'une connexion sont au niveau *global*, bien qu'on les ait spécifiées au niveau *database*.

```
SHOW GRANTS FOR 'Paul'@'localhost';
+-----+
| Grants for Paul@localhost
+-----+
| GRANT USAGE ON *.* TO 'Paul'@'localhost' IDENTIFIED BY PASSWORD
 '*6AE163FB9EE8BB011EB2E87316AA5BE563A6CDB7'
| GRANT CREATE, DROP ON `bdpaul`.* TO 'Paul'@'localhost'
| GRANT SELECT, INSERT, UPDATE (ISBN), DELETE, ALTER ON `bdpaul`.`Livre`
 TO 'Paul'@'localhost'
+-----+
```

On remarque que MySQL a regroupé tous les priviléges sur la table `Livre` en une instruction GRANT. La première exprime le fait que Paul peut se connecter à toutes les bases (par `US nomBase`), mais qu'il ne pourra travailler en réalité que dans `bdpaul`.

```
SHOW GRANTS FOR 'root'@'localhost';
+-----+
| Grants for root@localhost
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' IDENTIFIED BY PASSWORD
 '*387E25FE2CF7ED941E43A76AD9402825401698FC' WITH GRANT OPTION
+-----+
```

On remarque que MySQL n'attribue qu'un seul droit, mais le plus fort ! Tous les droits (ALL PRIVILEGES) sur toutes les bases (\*.\*), avec en prime la clause GRANT OPTION qui permet de retransmettre n'importe quoi à n'importe qui, ou de tout révoquer.

Interrogeons à nouveau la table `user` de la base `mysql` stockant les prérogatives au niveau global du moment. Le droit de création en local de Jules apparaît sur toutes les bases.

```

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 | Paul | N | N | N | N |
| localhost | Jules | Y | N | N | N |
+-----+-----+-----+-----+-----+

```

Les colonnes suivantes permettent de stocker les restrictions sur les connexions.

```

SELECT Host,User, max_questions "Requetes", max_updates "Modifs" ,
 max_connections "Connexions", max_user_connections "Cx simult."
FROM mysql.user;
+-----+-----+-----+-----+-----+
| Host | User | Requetes | Modifs | Connexions | Cx simult. |
+-----+-----+-----+-----+-----+
| ... |
| localhost | Paul | 0 | 0 | 0 | 0 |
| localhost | Jules | 50 | 20 | 6 | 3 |
+-----+-----+-----+-----+-----+

```

Analysons les autres tables de la base `mysql` pour découvrir les prérogatives des accès à différents niveaux (*database*, *table*, *column* et *routine*).

## Table `mysql.db`

La table `mysql.db` décrit les prérogatives au niveau *database*. Ainsi la colonne `Db` indique le nom de la base de données.

```

SELECT Host, User, Db, Create_priv, Drop_priv, Alter_priv FROM mysql.db
+-----+-----+-----+-----+-----+
| Host | User | Db | Create_priv | Drop_priv | Alter_priv |
+-----+-----+-----+-----+-----+
%		test_%	Y	Y	Y
%		test	Y	Y	Y
localhost	Paul	bdpaul	Y	Y	N
+-----+-----+-----+-----+-----+

```

Notez la possibilité de `Paul`, avec l'accès local, de créer et de supprimer des tables dans la base `bdpaul`. Notez également la possibilité de créer, de supprimer, de modifier des tables par un accès distant anonyme sur la base `test`.

## Table `mysql.host`

Cette table est étudiée à la section *Accès distants*.

## Table mysql.tables\_priv

La table `mysql.tables_priv` décrit les prérogatives objet au niveau *table*. Ainsi la colonne `Table_name` indique la table concernée, la colonne `Grantor` précise l'utilisateur ayant donné le droit. La colonne `Table_priv` est un SET contenant la liste des droits de l'utilisateur sur la table.

```
SELECT CONCAT(User,'@',Host) "Compte",
 CONCAT(Db,'.',Table_name) "Objet", Grantor, Table_priv
 FROM mysql.tables_priv;
```

| Compte          | Objet        | Grantor        | Table_priv                 |
|-----------------|--------------|----------------|----------------------------|
| Jules@localhost | bdpaul.Livre | root@localhost | Grant                      |
| Paul@localhost  | bdpaul.Livre | root@localhost | Select,Insert,Delete,Alter |

On retrouve les quatre privilèges de Paul et celui de Jules (GRANT OPTION de SELECT sur la table).

Cette table possède aussi une colonne de nom `Timestamp` stockant l'instant au cours duquel s'est déroulée l'attribution (ou la révocation).

## Table mysql.columns\_priv

La table `mysql.columns_priv` décrit les prérogatives objet au niveau *column*. Ainsi la colonne `Table_name` indique la table concernée, la colonne `Column_name` précise la colonne concernée par le droit. La colonne `Column_priv` est un SET contenant la liste des droits de l'utilisateur sur la colonne de la table.

```
SELECT CONCAT(User,'@',Host) "Compte", CONCAT(Db,'.',Table_name) "Objet",
 Column_name, Column_priv FROM mysql.columns_priv;
```

| Compte          | Objet        | Column_name | Column_priv |
|-----------------|--------------|-------------|-------------|
| Jules@localhost | bdpaul.Livre | titre       | Select      |
| Paul@localhost  | bdpaul.Livre | ISBN        | Update      |

On retrouve le privilège de Paul et celui de Jules (portant ici sur la même table).

## Table mysql.procs\_priv

La table `mysql.procs_priv` décrit les prérogatives des procédures et des fonctions cataloguées au niveau *routine*.

Les priviléges CREATE ROUTINE, ALTER ROUTINE, EXECUTE, et GRANT s'appliquent aux sous-programmes catalogués et peuvent être attribués au niveau *global* et *database*. Ainsi, CREATE ROUTINE, ALTER ROUTINE, EXECUTE, et GRANT peuvent être assignés aussi au niveau *routine*.

En supposant que la base bdpaul contient la procédure cataloguée sp1() et la fonction sp2(), toutes deux écrites par root, le tableau suivant exprime l'affectation de quelques priviléges en donnant les explications associées.

Tableau 5-7 Affectation de priviléges

| Instruction faite par root                                                            | Explication                                                                                                                                    |
|---------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>GRANT CREATE ROUTINE ON bdpaul.*<br/>TO 'Paul'@'localhost';</pre>                | Privilège système <i>database level</i> :<br>Paul (en accès local) peut créer ou supprimer des sous-programmes catalogués dans la base bdpaul. |
| <pre>GRANT ALTER ROUTINE ON<br/>PROCEDURE bdpaul.sp1<br/>TO 'Paul'@'localhost';</pre> | Privilège système <i>routine level</i> :<br>Paul peut modifier la procédure sp1 contenue dans la base bdpaul.                                  |
| <pre>GRANT EXECUTE ON PROCEDURE bdpaul.sp1<br/>TO 'Jules'@'localhost';</pre>          | Priviléges système <i>routine level</i> :<br>Jules peut exécuter la procédure sp1 contenue dans la base bdpaul.                                |
| <pre>GRANT EXECUTE ON FUNCTION bdpaul.sp2<br/>TO 'Jules'@'localhost';</pre>           |                                                                                                                                                |

La colonne *Routine\_name* de la table mysql.procs\_priv désigne le nom du programme catalogué. La colonne *Routine\_type* précise le type du sous-programme catalogué (fonction ou procédure). La colonne *Grantor* indique l'utilisateur ayant créé le sous-programme. La colonne *Proc\_priv* est un SET contenant la liste des droits de l'utilisateur sur le sous-programme de la base.

Extrayons les priviléges relatifs aux sous-programmes au niveau *database*.

```
SELECT CONCAT(User, '@', Host) "Compte", Db,
 Create_routine_priv "create routine", Alter_routine_priv "alter routine",
 Execute_priv "exec. routine" FROM mysql.db;
```

| Compte         | Db      | create routine | alter routine | exec. routine |
|----------------|---------|----------------|---------------|---------------|
| @%             | test\_% | N              | N             | N             |
| @%             | test    | N              | N             | N             |
| Paul@localhost | bdpaul  | Y              | N             | N             |

On retrouve le privilège de Paul. Extrayons enfin les priviléges relatifs aux sous-programmes au niveau *routine*.

```

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 ce même privilège avec l'option WITH GRANT OPTION.

### Syntaxe

Dans la syntaxe suivante, les options sont les mêmes que pour la commande GRANT.

```

REVOKE privilège [(coll [, coll2...])] [,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                                                                                                                                                                                                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>REVOKE CREATE<br/>ON bdpaul.*<br/>FROM 'Paul'@'localhost';</code>                                                             | Privilège système <i>database level</i> :<br><code>Paul</code> (en accès local) ne peut plus créer de tables dans la base <code>bdpaul</code> .                                                                                                                               |
| <code>REVOKE ALTER, INSERT, UPDATE(ISBN)<br/>ON bdpaul.Livre<br/>FROM 'Paul'@'localhost';</code>                                    | Privilège objet <i>table level</i> :<br><code>Paul</code> ne peut plus modifier la structure (ou les contraintes), insérer et modifier la colonne <code>ISBN</code> de la table <code>Livre</code> contenue dans la base <code>bdpaul</code> .                                |
| <code>GRANT USAGE ON bdpaul.*<br/>TO 'Jules'@'localhost'<br/>WITH<br/>  MAX_QUERIES_PER_HOUR 0<br/>  MAX_UPDATES_PER_HOUR 0;</code> | Privilège système <i>database level</i> :<br><code>Jules</code> n'est plus limité en requêtes SELECT et UPDATE sur la base de données <code>bdpaul</code> . 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 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. A l'extraction du profil actuel de Paul au niveau *table* fait apparaître les deux seuls droits 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. Pour 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 priviléges aux niveaux *global*, *database*, *table* et *column*. Et les priviléges *routine*, me direz-vous. Ils ont dû l'oublier dans la documentation, mais ils sont aussi effacés, ne vous inquiétez pas, j'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.

Ne confondez pas suppression de tous les droits d'un accès et suppression de l'utilisateur. Par analogie, les politiciens qui se voient retirer le droit de vote ne sont pas encore guillotinés (qu'je sache). Énervons-nous contre Jules :

```
| REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'Jules'@'localhost';
```

## Attributions et révocations « sauvages »

Le caractère *open* de MySQL fait des fois bien les choses, mais, dans ce cas précis, je n'en trouve pas. Ici mon billet d'humeur conteste la possibilité qui est donnée de modifier les cinq tables de la base mysql pour affecter tantôt un 'N' par-ci, tantôt un 'Y' par-là, ou encore pour mettre à jour un SET contenant SELECT par exemple, etc. Bref, un UPDATE raté, un INSERT dans la mauvaise colonne, un DELETE sans WHERE, et vous mettez une panique noire (comme la couleur par défaut de l'interface de commande) dans vos bases. Vous pouvez vous même empêcher toute connexion (même celle du root).

Sous Oracle, les commandes GRANT et REVOKE mettent à jour des tables système que vous pouvez interroger, mais que vous ne pouvez pas modifier. C'est heureux.

En conclusion, je ne décrirai aucune de ces manipulations, d'abord parce que je n'ai pas envie de me tromper en faisant des tests et bouleverser ainsi inutilement ma configuration. Ensuite, parce si vous voulez « bidouiller », allez consulter des sites Web ou d'autres ouvrages qui recopient la documentation sans quelquefois changer ni tester les exemples, vous m'en direz des nouvelles.



Vous voulez donner des droits : utilisez GRANT ; les reprendre : utilisez REVOKE, car :

- Ils sont programmés précisément pour ça.
- Les deux instructions sont dans la norme SQL.

Ne pas les employer, c'est comme acheter une quatre cylindres chez BMW (le motoriste est spécialisé des six en ligne) et verser en cachette du colza dans le réservoir chez l'agriculteur du coin en croyant économiser.

La seule utilisation acceptable, parce qu'on n'a pas le choix de faire autrement, concerne la mise à jour de la table mysql.host (décrite dans la section suivante). À configuration avancée, programmeur averti.

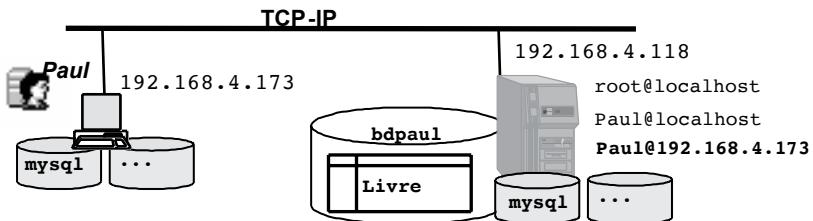
## Accès distants

La figure suivante illustre la configuration de mon test. Un client est en 192.168.4.173 sur lequel sont installées les couches MySQL (*Complete Package* ou *Essentials Package*). Un serveur est en 192.168.4.118 équipé de MySQL *Complete Package*. Sur le serveur, root crée

un accès à Paul, en précisant l'adresse de la machine client, et lui attribue un droit d'écriture sur la table Livre sur la base bdPaul.

```
CREATE USER Paul@192.168.4.173 IDENTIFIED BY 'pauldistant';
GRANT SELECT ON bdpaul.Livre TO 'Paul'@'192.168.4.173';
```

Figure 5-5 Accès distant par l'interface de commande MySQL



## Connexion par l'interface de commande

Sur le client, Paul se connecte au serveur dans une fenêtre de commande, en précisant l'adresse de la machine serveur, puis donne son mot de passe distant. Pensez à enlever le pare-feu Windows sur le client et le serveur (bloquant le port 3306).

```
mysql -h 192.168.4.118 -u Paul -p
```

Paul peut, à présent seulement, interroger la table distante, comme le montre la copie d'écran suivante :

Figure 5-6 Interrogation distante par l'interface de commande MySQL

A screenshot of a terminal window titled "Invite de commandes - mysql -h 192.168.4.118 -u Paul -p". The window shows the MySQL monitor prompt. A user enters the command "select \* from bdpaul.livre;". The response displays a table with columns "titre" and "ISBN". The table has one row: "Objet-relationnel sous Oracle8" and "II". The message "1 row in set <0.15 sec>" is shown at the bottom. The MySQL prompt "mysql>" appears again at the end.

```
C:\Documents and Settings\labat>mysql -h 192.168.4.118 -u Paul -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3 to server version: 5.0.15-nt
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> select * from bdpaul.livre;
+-----+-----+
| titre | ISBN |
+-----+-----+
| Objet-relationnel sous Oracle8 | II |
+-----+-----+
1 row in set <0.15 sec>

mysql>
```

## Table mysql.host

La table mysql.host est utilisée conjointement avec mysql.db et concerne les accès distants (plusieurs machines). Cette table n'est employée que pour les prérogatives au niveau des utilisateurs.

*database*, indépendamment des utilisateurs. La structure est la même que celle de `mysql.db` à l'exception de la colonne `User` qui n'est pas présente. Le couple de colonnes (`Host`, `Db`) est unique.

Tableau 5-9 Tables pour les accès distants

| Caractère         | Signification pour <code>mysql.db</code>   |            | Signification pour <code>mysql.host</code> |            |
|-------------------|--------------------------------------------|------------|--------------------------------------------|------------|
|                   | colonne Host                               | colonne Db | colonne Host                               | colonne Db |
| %                 | toute machine                              | toute base | toute machine                              | toute base |
| ' ' (chaîne vide) | consultez la table <code>mysql.host</code> | toute base | toute machine                              | toute base |

MySQL lit et trie les tables `db` (sur les colonnes `Host`, `Db` et `User`) et `host` (sur les colonnes `Host` et `Db`) en même temps qu'il parcourt la table `user`. Pour les opérations relatives aux bases (`INSERT`, `UPDATE`, etc.), MySQL interroge la table `user`. Si l'accès n'y est pas décrété, la recherche se poursuit dans les tables `db` et `host`. Si la colonne `Host` de la table `db` est renseignée en fonction de l'accès, l'utilisateur reçoit ses priviléges.

Si la colonne `Host` de la table `db` n'est pas renseignée (' '), cela signifie que la table `host` énumère les machines qui sont autorisées à accéder à une base de données en particulier. Si une machine ne correspond pas, l'accès n'est pas permis. Dans le cas contraire, les priviléges sont alors évalués à 'Y' à partir d'une intersection (et pas d'une union) entre les tables `db` et `host` sur le couple (`Host`, `Db`).



La table `mysql.host` n'est mise à jour ni par `GRANT`, ni par `REVOKE`. Il faudra directement insérer (par `INSERT`), modifier (par `UPDATE`) ou supprimer (par `DELETE`) les lignes de cette table. Elle n'est pas utilisée par la plupart des serveurs MySQL, car elle est dédiée à des usages très spécifiques (pour gérer un ensemble de machines à accès sécurisé, par exemple). Elle peut aussi être utilisée pour définir un ensemble de machines à accès non sécurisé.

En supposant que vous déclariez une machine à accès non sécurisé : `cAMPAROLS.GTR.FR`, il est possible d'autoriser l'accès sécurisé à toutes les autres machines du réseau local. Ceci est fait en ajoutant des enregistrements par `INSERT` dans la table `mysql.host` comme suit :

| Host                          | Db | ...                         |
|-------------------------------|----|-----------------------------|
| <code>cAMPAROLS.GTR.FR</code> | %  | (tous les priviléges à 'N') |
| <code>%.GTR.FR</code>         | %  | (tous les priviléges à 'Y') |

Vous déclareriez l'inverse des conditions initiales en remplaçant les 'N' par des 'Y', et réciproquement. Dans tous les cas, il sera nécessaire de mettre à jour les autres tables pour affiner les priviléges.

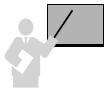
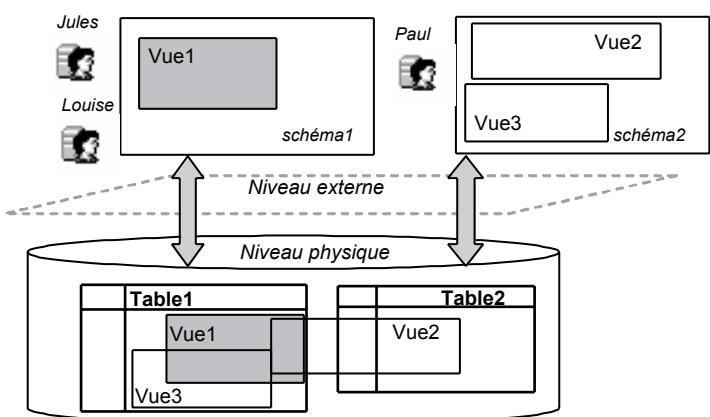
Outre le contrôle de l'accès aux données (privileges), la confidentialité des informations est un aspect important qu'un SGBD relationnel doit prendre en compte. Depuis la version 5 de MySQL, la confidentialité est renforcée par l'utilisation de vues (*views*) qui agissent comme des fenêtres sur la base de données. Cette section décrit les différents types de vues qu'on peut rencontrer.

Les vues correspondent à ce qu'on appelle « le niveau externe » qui reflète la partie visible de la base de données pour chaque utilisateur.

Seules les tables contiennent des données et, pourtant, pour l'utilisateur, une vue apparaît comme une table. En théorie, les utilisateurs ne devraient accéder aux informations directement que par les vues. Cela pose des questions éthiques et pratiques questionnant des vues. Ces dernières masquent la structure des tables interrogées. En pratique, la plupart des applications se passent de ce concept en manipulant directement les tables.

La figure suivante illustre ce qui a été dit en présentant trois utilisateurs. Ils travaillent chacun sur une base de données contenant des vues formées à partir de différentes tables.

**Figure 5-7 Les vues**



Une vue est considérée comme une table virtuelle car elle n'a pas d'existence propre. Sa structure est stockée dans le dictionnaire. Ses données seront extraites de la mémoire à la demande.

Une vue est créée à l'aide d'une instruction `SELECT` appelée « requête de définition ». Une requête interroge une (ou plusieurs) table(s) ou vue(s). Une vue se recharge chaque fois qu'elle est interrogée.

Outre le fait d'assurer la confidentialité des informations, une vue est capable de réaliser des contrôles de contraintes d'intégrité et de simplifier la formulation de requêtes complexes.

Dans certains cas, la définition d'une vue temporaire est nécessaire pour écrire une requête qui n'aurait pas pu être exécutée autrement. Attribuées comme des priviléges (GRANT), les vues améliorent la sécurité des informations stockées.

## Création d'une vue (CREATE VIEW)

Pour pouvoir créer une vue dans une base, vous devez posséder le privilège CREATE VIEW sur les priviléges en SELECT des tables présentes dans la requête de définition de la vue. La syntaxe SQL de création d'une vue est la suivante :

```
CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
 VIEW [nomBase.]nomVue [(listecolonne)]
 AS requêteSELECT
 [WITH [CASCADED | LOCAL] CHECK OPTION];
```

- **OR REPLACE** remplace la vue par la nouvelle définition, même si elle existait déjà (évitant de détruire la vue avant de la recréer). Vous devez avoir le privilège DELETE sur la base pour bénéficier de cette directive.
- **ALGORITHM=MERGE** : la définition de la vue et sa requête sont fusionnées en interne.
- **ALGORITHM=TEMPTABLE** : les résultats sont extraits dans une table temporaire (TEMPORARY) qui est utilisée par la suite. Intéressant si les tables source sont sujettes à de nombreux verrous qui ne gênent plus la manipulation de la vue utilisant, elle, une table temporaire.
- Aucune option ou **ALGORITHM=UNDEFINED** : MySQL choisit la politique à adopter souvent en faveur de **MERGE**, la seule qui convient aux vues modifiables.
- **nomBase** désigne le nom de la base de données qui hébergera la vue. En l'absence de ce paramètre, la vue est créée dans la base en cours d'utilisation.
- **requeteSELECT** : requête de définition interrogeant une (ou des) table(s) ou vue(s) pour charger les données dans la vue.



- La requête de définition ne peut interroger une table temporaire, ni contenir de paramètres ou de variables de session.
- Si la requête de définition sélectionne toutes les colonnes d'un objet source (SELECT FROM...), et si des colonnes sont ajoutées par la suite à cet objet, la vue ne contiendra pas ces colonnes définies ultérieurement à elle. Il faudra recréer la vue pour prendre en compte l'évolution structurelle de l'objet source.
- **WITH CHECK OPTION** garantit que toute mise à jour de la vue par INSERT ou UPDATE s'effectuera conformément au prédictat contenu dans la requête de définition.
- Les paramètres **LOCAL** et **CASCADED** (par défaut) déterminent la portée de la vérification quand une vue est définie à partir d'une autre vue. **LOCAL** restreint la vérification du prédictat à la vue elle-même. **CASCADED** permet d'étendre éventuellement les vérifications aux autres vues source de la vue qui vient d'être définie.

# Classification

On distingue les vues simples des vues complexes en fonction de la nature de la requête de définition. Le tableau suivant résume ce que nous allons détailler au cours de cette section.

Tableau 5-10 Classification des vues

| Requête de définition    | Vue simple | Vue complexe   |
|--------------------------|------------|----------------|
| Nombre de tables         | 1          | 1 ou plusieurs |
| Fonction                 | non        | oui            |
| Regroupement             | non        | oui            |
| Mises à jour possibles ? | oui        | pas toujours   |



Une vue monotable est définie par une requête SELECT ne comportant qu'une seule table dans sa clause FROM.



## Vues monotables

Les mécanismes présentés ci-après s'appliquent aussi, pour la plupart, aux vues multitable (étudiées plus loin). Considérons les deux vues illustrées par la figure suivante et dérivées de la table Pilote. La vue PilotesAF décrit les pilotes d'Air France à l'aide d'une restriction (éléments du WHERE). La vue Etat\_civil est constituée par une projection de certaines colonnes (éléments du SELECT).

Figure 5-8 Deux vues d'une table

| Pilote |         |        |                  |       |
|--------|---------|--------|------------------|-------|
| brevet | nom     | nbHVol | adresse          | compa |
| PL-1   | Soutou  | 890    | Castanet         | CAST  |
| PL-2   | Laroche | 500    | Montauban        | CAST  |
| PL-3   | Lamothe | 1200   | Ramonville       | AF    |
| PL-4   | Albaric | 500    | Vieille-Toulouse | AF    |
| PL-5   | Bidal   | 120    | Paris            | ASO   |
| PL-6   | Labat   | 120    | Pau              | ASO   |
| PL-7   | Tauzin  | 100    | Bas-Mauco        | ASO   |

```
CREATE VIEW PilotesAF
AS SELECT *
FROM Pilote
WHERE compa = 'AF';
```

```
CREATE VIEW Etat_civil
AS SELECT nom, nbHVol, adresse,
compa FROM Pilote;
```

Une fois créée, une vue s'interroge comme une table par tout utilisateur, sous réserve qu'il ait obtenu le privilège en lecture directement (GRANT SELECT ON *nomVue* TO...). Le tableau suivant présente une interrogation des deux vues :

Tableau 5-11 Interrogation d'une vue

| Web                                               | Besoin et requête                  | Résultat                                                        |
|---------------------------------------------------|------------------------------------|-----------------------------------------------------------------|
| Somme des heures de vol des pilotes d'Air France. | SELECT SUM(nbHVol) FROM PilotesAF; | +-----+<br>  SUM(nbHVol)  <br>+-----+<br>  1700.00  <br>+-----+ |
| Nombre de pilotes.                                | SELECT COUNT(*) FROM Etat_civil;   | +-----+<br>  COUNT(*)  <br>+-----+<br>  7  <br>+-----+          |

À partir de cette table et de ces vues, nous allons étudier d'autres options de l'instruction CREATE VIEW.

### Alias

Les alias, s'ils sont utilisés, désignent le nom de chaque colonne de la vue. Ce mécanisme permet de mieux contrôler les noms de colonnes. Quand un alias n'est pas présent, la colonne prend le nom de l'expression renvoyée par la requête de définition. Ce mécanisme sert à masquer les noms des colonnes de l'objet source.

Les vues suivantes sont créées avec des alias qui masquent le nom des colonnes de la table source. Les deux écritures sont équivalentes.

Tableau 5-12 Vue avec alias

| Web     | Écriture 1                                                                                                                                                                | Écriture 2                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |            |         |           |            |         |      |        |        |          |      |      |         |        |           |      |      |       |        |       |     |      |       |        |     |     |      |        |        |           |     |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|---------|-----------|------------|---------|------|--------|--------|----------|------|------|---------|--------|-----------|------|------|-------|--------|-------|-----|------|-------|--------|-----|-----|------|--------|--------|-----------|-----|
|         | <pre>CREATE OR REPLACE VIEW     PilotesPasAF     (codepil,nomPil,heuresPil,      adressePil, societe)     AS SELECT *     FROM Pilote     WHERE NOT (compa = 'AF');</pre> | <pre>CREATE OR REPLACE VIEW     PilotesPasAF     AS SELECT brevet codepil, nom nomPil,     nbHVol heuresPil, adresse adressePil,     compa societe     FROM Pilote     WHERE NOT (compa = 'AF');</pre>                                                                                                                                                                                                                                                                                                                                                                                                               |            |         |           |            |         |      |        |        |          |      |      |         |        |           |      |      |       |        |       |     |      |       |        |     |     |      |        |        |           |     |
|         | Contenu de la vue :                                                                                                                                                       | <table border="1"> <thead> <tr> <th>codepil</th> <th>nomPil</th> <th>heuresPil</th> <th>adressePil</th> <th>societe</th> </tr> </thead> <tbody> <tr> <td>PL-1</td> <td>Soutou</td> <td>890.00</td> <td>Castanet</td> <td>CAST</td> </tr> <tr> <td>PL-2</td> <td>Laroche</td> <td>500.00</td> <td>Montauban</td> <td>CAST</td> </tr> <tr> <td>PL-5</td> <td>Bidal</td> <td>120.00</td> <td>Paris</td> <td>ASO</td> </tr> <tr> <td>PL-6</td> <td>Labat</td> <td>120.00</td> <td>Pau</td> <td>ASO</td> </tr> <tr> <td>PL-7</td> <td>Tauzin</td> <td>100.00</td> <td>Bas-Mauco</td> <td>ASO</td> </tr> </tbody> </table> | codepil    | nomPil  | heuresPil | adressePil | societe | PL-1 | Soutou | 890.00 | Castanet | CAST | PL-2 | Laroche | 500.00 | Montauban | CAST | PL-5 | Bidal | 120.00 | Paris | ASO | PL-6 | Labat | 120.00 | Pau | ASO | PL-7 | Tauzin | 100.00 | Bas-Mauco | ASO |
| codepil | nomPil                                                                                                                                                                    | heuresPil                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | adressePil | societe |           |            |         |      |        |        |          |      |      |         |        |           |      |      |       |        |       |     |      |       |        |     |     |      |        |        |           |     |
| PL-1    | Soutou                                                                                                                                                                    | 890.00                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Castanet   | CAST    |           |            |         |      |        |        |          |      |      |         |        |           |      |      |       |        |       |     |      |       |        |     |     |      |        |        |           |     |
| PL-2    | Laroche                                                                                                                                                                   | 500.00                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Montauban  | CAST    |           |            |         |      |        |        |          |      |      |         |        |           |      |      |       |        |       |     |      |       |        |     |     |      |        |        |           |     |
| PL-5    | Bidal                                                                                                                                                                     | 120.00                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Paris      | ASO     |           |            |         |      |        |        |          |      |      |         |        |           |      |      |       |        |       |     |      |       |        |     |     |      |        |        |           |     |
| PL-6    | Labat                                                                                                                                                                     | 120.00                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Pau        | ASO     |           |            |         |      |        |        |          |      |      |         |        |           |      |      |       |        |       |     |      |       |        |     |     |      |        |        |           |     |
| PL-7    | Tauzin                                                                                                                                                                    | 100.00                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Bas-Mauco  | ASO     |           |            |         |      |        |        |          |      |      |         |        |           |      |      |       |        |       |     |      |       |        |     |     |      |        |        |           |     |

### Vue d'une vue

L'objet source d'une vue est en général une table, mais peut aussi être une vue. La vue suivante est définie à partir de la vue PilotesPasAF précédemment créée. Notez qu'il aurait été possible d'utiliser des alias pour renommer les colonnes de la nouvelle vue.

Tableau 5-13 Vue d'une vue

| Web | Création                                                                                                         | Contenu de la vue                                                                                                                                                                                                                                                 |
|-----|------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>CREATE OR REPLACE VIEW EtatCivilPilotesPasAF AS SELECT nomPil,heuresPil,adressePil FROM PilotesPasAF;</pre> | <pre>+-----+-----+-----+   nomPil   heuresPil   adressePil +-----+-----+-----+   Soutou      890.00   Castanet   Laroche      500.00   Montauban   Bidal      120.00   Paris   Labat      120.00   Pau   Tauzin      100.00   Bas-Mauco +-----+-----+-----+</pre> |

### Vues en lecture seule

L'option ALGORITHM=TEMPTABLE déclare la vue non modifiable par INSERT, UPDATE ou DELETE.

Redéfinissons une vue des pilotes n'étant pas d'Air France à l'aide de cette option. Les messages d'erreur induits par la clause de lecture seule, générés par MySQL, sont très parlants.

Tableau 5-14 Vue en lecture seule

| Web | Création                                                                                                               | Opérations impossibles                                                                                                                                                                                                                                                                                                                                                                                         |
|-----|------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>CREATE OR REPLACE ALGORITHM=TEMPTABLE VIEW PilotesPasAFRO AS SELECT * FROM Pilote WHERE NOT (compa = 'AF');</pre> | <pre>INSERT INTO PilotesPasAFRO VALUES ('PL-8','Ferry',5,'Paris','ASO'); ERROR 1288 (HY000): The target table PilotesPasAFRO of the INSERT is not updatable UPDATE PilotesPasAFRO SET nbHvol=nbHvol+2; ERROR 1288 (HY000): The target table PilotesPasAFRO of the UPDATE is not updatable DELETE FROM PilotesPasAFRO; ERROR 1288 (HY000): The target table PilotesPasAFRO of the DELETE is not updatable</pre> |

### Vues modifiables



Lorsqu'il est possible d'exécuter des instructions INSERT, UPDATE ou DELETE sur une vue, cette dernière est dite « modifiable » (*updatable view*). Vous pouvez créer une vue qui est modifiable intrinsèquement.



Pour mettre à jour une vue, il doit exister une correspondance biunivoque entre les lignes de la vue et celles de l'objet source. De plus certaines conditions doivent être remplies.

Si une vue n'est pas modifiable en soi, il n'est pas encore possible de programmer un déclencheur de type *instead of* qui prenne le pas sur l'instruction de modification de la vue, en émettant un bloc d'instructions à effectuer à la place. Les mises à jour de la vue seraient alors automatiquement répercutées au niveau d'une ou de plusieurs tables. Notons que ce type de déclencheur n'est arrivé qu'assez tardivement sous Oracle, DB2 et SQL Server.



Pour qu'une vue simple soit modifiable, sa requête de définition doit respecter les critères suivants

- pas de directive DISTINCT, de fonction (AVG, COUNT, MAX, MIN, SUM, ou VARIANCE) d'expression dans le SELECT ;
- pas de GROUP BY, ORDER BY ou HAVING.

Dans notre exemple, nous constatons qu'il sera quand même possible d'ajouter un pilote à la vue `Etat_civil`, bien que la clé primaire de la table source ne soit pas renseignée. MySQL insère à la place, en l'absence de valeur par défaut de la clé primaire, la chaîne vide (' ') ; si la clé avait été une séquence, les insertions se feraient normalement. Cette opération ne pourra donc se faire qu'une seule fois, après, cela sera contradictoire avec la condition de correspondance biunivoque.

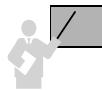
En revanche, il sera possible de modifier les colonnes de cette vue. On pourra aussi ajouter, modifier (sous réserve de respecter les éventuelles contraintes issues des colonnes de la table source), ou supprimer des pilotes en passant par la vue `PilotesAF`.

La dernière instruction est paradoxale, car elle permet d'ajouter un pilote de la compagnie 'ASO' en passant par la vue des pilotes de la compagnie 'AF'. La directive WITH CHECK OPTION permet d'éviter ces effets de bord indésirables pour l'intégrité de la base.

Tableau 5-15 Mises à jour de vues

| Web | Opérations possibles                                                                                                                                                                 | Opérations non conseillées et impossibles                                                                                                                       |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | Suppression des pilotes de ASO<br><br><pre>DELETE FROM Etat_civil       WHERE compa = 'ASO';</pre>                                                                                   | Ajout d'un pilote (pas conseillé)<br><pre>INSERT INTO Etat_civil VALUES ('Raffarin',10,'Poitiers','ASO');</pre>                                                 |
|     | Le pilote <i>Lamothe</i> double ses heures<br><br><pre>UPDATE Etat_civil       SET nbHVol = nbHVol*2       WHERE nom = 'Lamothe';</pre>                                              | Ajout d'un autre pilote impossible<br><pre>INSERT INTO Etat_civil VALUES ('Lebur',20,'Bordeaux','ASO'); ERROR 1062 (23000): Duplicate entry ' ' for key 1</pre> |
|     | Ajout d'un pilote<br><br><pre>INSERT INTO PilotesAF VALUES ('PL-8','Ferry',5,'Paris','AF');</pre>                                                                                    | Toute mise à jour qui ne respecterait pas les contraintes de la table Pilote.                                                                                   |
|     | Modification<br><br><pre>UPDATE PilotesAF SET nbHVol = nbHVol*2;</pre>                                                                                                               |                                                                                                                                                                 |
|     | Suppression<br><br><pre>DELETE FROM PilotesAF WHERE nom='Ferry'; Ajout d'un pilote qui n'est pas de 'AF'! INSERT INTO PilotesAF VALUES ('PL-9','Caboche', 600,'Rennes','ASO');</pre> |                                                                                                                                                                 |

## Directive CHECK OPTION



La directive WITH CHECK OPTION empêche un ajout ou une modification non conforme à la définition de la vue.

Interdisons l'ajout (ou la modification de la colonne compa) d'un pilote au travers de la vue PilotesAF, si le pilote n'appartient pas à la compagnie de code 'AF'. Il est nécessaire de redéfinir la vue PilotesAF. Le script suivant décrit la redéfinition de la vue, l'ajout d'un pilote et les tentatives d'ajout et de modification ne respectant pas les caractéristiques de la vue. Les messages sont très clairs.

Tableau 5-16 Vue avec CHECK OPTION

| Web | Opérations possibles                                                                                                                            | Opérations impossibles                                                                                                                                                            |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | Recréation de la vue<br><pre>CREATE OR REPLACE VIEW PilotesAF<br/>AS SELECT * FROM pilote<br/>WHERE compa = 'AF'<br/>WITH CHECK OPTION;</pre>   | Ajout d'un pilote<br><pre>INSERT INTO PilotesAF VALUES<br/>( 'PL-9', 'Caboche', 600, 'Rennes', 'ASO' )<br/>ERROR 1369 (HY000): CHECK OPTION failed<br/>'bdsoutou.PilotesAF'</pre> |
|     | Nouveau pilote<br><pre>INSERT INTO PilotesAF VALUES<br/>( 'PL-11', 'Teste', 900, 'Revel', 'AF' );<br/>Query OK, 1 row affected (0.03 sec)</pre> | Modification de pilotes<br><pre>UPDATE PilotesAF SET compa='ASO';<br/>ERROR 1369 (HY000): CHECK OPTION failed<br/>'bdsoutou.PilotesAF'</pre>                                      |

## Vues complexes

Une vue complexe est caractérisée par le fait qu'elle contient, dans sa définition, plusieurs tables (jointures) et une fonction appliquée à des regroupements ou à des expressions. La mise à jour de telles vues n'est pas toujours possible.



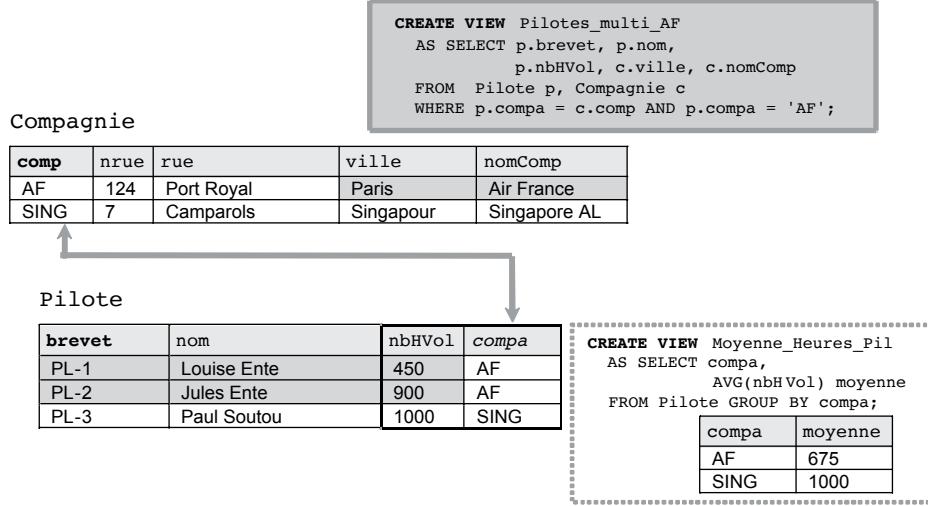
Pour pouvoir modifier une vue complexe, les restrictions sont les suivantes :

- La requête de définition ne doit pas contenir de sous-interrogation (jointure procédurale).
- Il n'est pas possible d'utiliser d'opérateur ensembliste (sauf UNION [ALL]).



La figure suivante présente deux vues complexes qui ne sont pas modifiables. La vue multiple Pilotes\_multi\_AF est créée à partir d'une jointure entre les tables Compagnie et Pilote. La vue Moyenne\_Heures\_Pil est créée à partir d'un regroupement de la table Pilote.

Figure 5-9 Vues complexes



### Mises à jour

Il semblerait qu'on ne puisse insérer aucun enregistrement dans ces vues du fait de la cohérence qu'il faudrait établir du sens *vue vers tables*. Les messages d'erreur générés par MySQL sont différents suivant la nature de la vue (monotable ou multitable). Nous verrons comment résoudre l'erreur du deuxième cas.

Tableau 5-17 Tentatives d'insertions dans des vues complexes

| Web | Vue monotable                                                                                                                                                                                         | Vue multitable                                                                                                                                                                                         |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre> INSERT INTO Moyenne_Heures_Pil VALUES ('TAT',50); </pre> <p>ERROR 1288 (HY000): The target table Moyenne_Heures_Pil of the view 'bdsoutou.Pilotes_multi_AF' without INSERT is not updatable</p> | <pre> INSERT INTO Pilotes_multi_AF VALUES ('PL-4','Test',400,'Castanet','Castanet AL') </pre> <p>ERROR 1394 (HY000): Can not insert into join view 'bdsoutou.Pilotes_multi_AF' without fields list</p> |

On pourrait croire qu'il en est de même pour les modifications. Ce n'est pas le cas. Alors que la vue monotable *Moyenne\_Heures\_Pil* n'est pas modifiable, ni par *UPDATE* ni par *DELETE* (message d'erreur 1288), la vue multitable *Pilotes\_multi\_AF* est transformable dans une certaine mesure, car la table *Pilote* (qui entre dans sa composition) est dite « protégée par clé » (*key preserved*). Nous verrons dans le prochain paragraphe la signification de cette notion.



Les colonnes de la vue correspondant à la table protégée par clé ne sont pas les seules à voir être modifiées. Ici, nbHVOL peut être mise à jour car elle appartient à la table protégée ville qui n'appartient pas à une table protégée peut aussi être modifiée !

Les suppressions ne se répercutent pas sur les enregistrements de la table protégée (Pilote).

Modifions et tentons de supprimer des enregistrements à travers la vue multitable Pilotes\_multi\_AF.

Tableau 5-18 Mise à jour d'une vue multitable

| Web | Mise à jour                                                                                         | Résultats                                                                                                                                                                                                                 |
|-----|-----------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | -- Multiplie par 2 le nombre d'heures<br><b>UPDATE Pilotes_multi_AF</b><br>SET nbHVOL = nbHVOL * 2; | SELECT brevet,nom,nbHVOL<br>FROM Pilotes_multi_AF;<br>+-----+-----+-----+<br>  brevet   nom   nbHVOL  <br>+-----+-----+-----+<br>  PL-1   Louise Ente   900.00  <br>  PL-2   Paul Ente   1800.00  <br>+-----+-----+-----+ |
|     | Query OK, 2 rows affected<br>Rows matched: 2 Changed: 2 Warnings: 0                                 |                                                                                                                                                                                                                           |
|     | -- Modif de la ville de la compagnie<br><b>UPDATE Pilotes_multi_AF</b><br>SET ville = 'Orly';       | SELECT brevet,nom,ville<br>FROM Pilotes_multi_AF;<br>+-----+-----+-----+<br>  brevet   nom   ville  <br>+-----+-----+-----+<br>  PL-1   Louise Ente   orly  <br>  PL-2   Paul Ente   orly  <br>+-----+-----+-----+        |
|     | --Pas possible :<br><b>DELETE FROM Pilotes_multi_AF;</b>                                            | SELECT comp,ville,nomComp<br>FROM Compagnie;<br>+-----+-----+-----+<br>  comp   ville   nomComp  <br>+-----+-----+-----+<br>  AF   Orly   Air France  <br>  SING   Singapour   Singapore AIR  <br>+-----+-----+-----+     |
|     |                                                                                                     | ERROR 1395 (HY000): Can not delete<br>from join view 'bdsoutou.Pilotes_<br>multi_AF'                                                                                                                                      |

## 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 extra de la vue, alors que la colonne `comp` ne le fait pas.

Tableau 5-19 Vue multitable

| Web | Création de la vue                 | Résultats                                          |
|-----|------------------------------------|----------------------------------------------------|
|     | <code>CREATE VIEW</code>           | +-----+-----+-----+-----+                          |
|     | <code>Vue_Multi_Comp_Pil</code>    | comp   nomComp   brevet   nom   nbHVol             |
|     | <code>AS SELECT c.comp,</code>     | +-----+-----+-----+-----+                          |
|     | <code>    c.nomComp,</code>        | AF   Air France   PL-1   Louise Ente   450.00      |
|     | <code>    p.brevet, p.nom,</code>  | AF   Air France   PL-2   Paul Ente   900.00        |
|     | <code>    p.nbHVol</code>          | SING   Singapore AL   PL-3   Paul Soutou   1000.00 |
|     | <code>FROM Pilote p,</code>        | +-----+-----+-----+-----+                          |
|     | <code>    Compagnie c</code>       |                                                    |
|     | <code>WHERE p.compa=c.comp;</code> |                                                    |

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.

Tableau 5-20 Mises à jour

| Web | Mise à jour                                                                                                                                    | Résultats                                                                                                                                                                                                                                                                                     |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <b>DELETE</b> FROM Vue_Multi_Comp_Pil<br>WHERE comp='AF';                                                                                      | ERROR 1395 (HY000): Can not delete<br>from join view<br>'bdsoutou.Vue_Multi_Comp_Pil'                                                                                                                                                                                                         |
|     | <b>UPDATE</b> Vue_Multi_Comp_Pil<br>SET nbHVol = nbHVol * 3;<br>Query OK, 3 rows affected (0.10 sec)<br>Rows matched: 3 Changed: 3 Warnings: 0 | SELECT brevet, nbHVol FROM Pilote<br>+-----+-----+<br>  brevet   nbHVol  <br>+-----+-----+<br>  PL-1    1350.00  <br>  PL-2    2700.00  <br>  PL-3    3000.00  <br>+-----+-----+                                                                                                              |
|     | <b>UPDATE</b> Vue_Multi_Comp_Pil<br>SET nomComp = 'Dupond';<br>Query OK, 2 rows affected (0.38 sec)<br>Rows matched: 3 Changed: 2 Warnings: 0  | (SELECT comp,ville,nomComp<br>FROM Compagnie;<br>+-----+-----+-----+<br>  comp   ville     nomComp  <br>+-----+-----+-----+<br>  AF     Paris      Dupond   <br>  SING   Singapour   Dupond   <br>+-----+-----+-----+                                                                         |
|     | <b>INSERT</b> INTO Vue_Multi_Comp_Pil<br>(brevet,nom,nbHVol)<br>VALUES ('PL-5', 'Jean',2500);;                                                 | SELECT brevet,nom,nbHVol<br>FROM Pilote;<br>+-----+-----+-----+<br>  brevet   nom       nbHVol  <br>+-----+-----+-----+<br>  PL-1    Louise Ente   1350.0  <br>  PL-2    Paul Ente     2700.0  <br>  PL-3    Paul Soutou   3000.0  <br>  PL-5    Jean         2500.0  <br>+-----+-----+-----+ |
|     | <b>INSERT</b> INTO Vue_Multi_Comp_Pil<br>(comp,nomComp)<br>VALUES ('TAT', 'Test');                                                             | SELECT comp,ville,nomComp<br>FROM Compagnie;<br>+-----+-----+-----+<br>  comp   ville     nomComp  <br>+-----+-----+-----+<br>  AF     Paris      Dupond   <br>  SING   Singapour   Dupond   <br>  TAT    Paris      Test     <br>+-----+-----+-----+                                         |

## Autres utilisations de vues

Les vues peuvent également servir pour simplifier l'écriture de requêtes complexes, renforçant la confidentialité et une certaine intégrité.

## Simplifier les noms

Une vue permet de simplifier la manipulation d'une table ayant un nom long ou des colonnes portant des noms compliqués. Considérons par exemple la table `COLLATION_CHARACTER_SET_APPLICABILITY[COLLATION_NAME,CHARACTER_SET_NAME]` qui décrit les collations des jeux de caractères disponibles, et qui est située dans la base de données stockant le dictionnaire des données (`INFORMATION_SCHEMA`). Nous étudierons dans la prochaine section les différentes tables de cette base système. Supposons que l'on désire souvent accéder à cette table pour connaître les différentes collations possibles pour les jeux de caractères latin.

Créons la vue `CollationsLatines` qui simplifie l'accès à cette table au niveau du nom mais aussi au niveau des colonnes. Interrogeons cette vue de manière à connaître les collations spécifiques à la langue de Molière ou à celle de Goethe.

Tableau 5-21 Vue pour simplifier les noms

| Web | Création                                                                                                                                                                                        | Interrogation                                                                                                                                                                                                                              |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>CREATE VIEW CollationsLatines     (collation,jeu) AS SELECT * FROM INFORMATION_SCHEMA.J_ COLLATION_CHARACTER_SET_APPLICABILITY WHERE     CHARACTER_SET_NAME           LIKE 'Latin%';</pre> | <pre>SELECT * FROM CollationsLatines   WHERE collation LIKE '%french%'     OR collation LIKE '%german%'; +-----+-----+   collation   jeu   +-----+-----+   latin1_german1_ci   latin1     latin1_german2_ci   latin1   +-----+-----+</pre> |

On dira que MySQL est plus « branché » par la nouveauté, Goethe étant né 76 ans après le décès de Molière. Aucun `french` dans la base, donc.

## Contraintes de vérification

Nous avons décrit au chapitre 1 les contraintes de vérification (`CHECK`) qui ne sont pas encore totalement prises en charge. Il est possible de programmer ce type de contraintes par des vues.

Considérons la table `Pilote` illustrée ci-après et programmons, par l'intermédiaire de la vue `VueGradePilotes`, la contrainte vérifiant qu'un pilote :

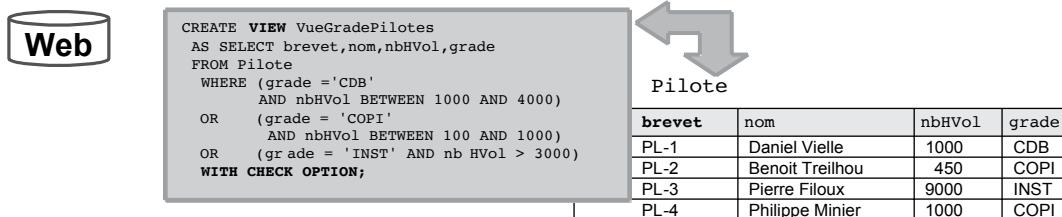
- ne peut être commandant de bord qu'à la condition qu'il ait entre 1 000 et 4 000 heures de vol ;
- ne peut être copilote qu'à la condition qu'il ait entre 100 et 1 000 heures de vol ;
- ne peut être instructeur qu'à partir de 3 000 heures de vol.



Les règles conseillées pour manipuler les enregistrements d'une vue `v1` décrivant des contraintes de vérification sur une table `t1` sont les suivantes :

- modification et insertion par la vue `v1` ;
- suppression et lecture par la table `t1`.

Figure 5-10 Vue simulant la contrainte CHECK



Manipulons à présent la vue de notre exemple :

Tableau 5-22 Manipulations des vues pour l'intégrité référentielle

| Mises à jour possibles                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Mises à jour non valides : ERROR 1369 (HY000): CHECK OPTION failed 'bdsoutou.VueGradePilotes'                                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| INSERT INTO VueGradePilotes<br>(brevet, nom, nbHVol, grade) VALUES<br>( 'PL-1', 'Daniel Vielle', 1000, 'CDB' );<br>INSERT INTO VueGradePilotes<br>(brevet, nom, nbHVol, grade) VALUES<br>( 'PL-2', 'Benoit Treihlou', 450, 'COPI' );<br>INSERT INTO VueGradePilotes<br>(brevet, nom, nbHVol, grade) VALUES<br>( 'PL-3', 'Pierre Filoux', 9000, 'INST' );<br>INSERT INTO VueGradePilotes<br>(brevet, nom, nbHVol, grade) VALUES<br>( 'PL-4', 'Philippe Minier', 1000, 'COPI' ); | INSERT INTO VueGradePilotes<br>(brevet, nom, nbHVol, grade) VALUES<br>( 'PL-5', 'Trop jeune', 100, 'CDB' );<br>INSERT INTO VueGradePilotes<br>(brevet, nom, nbHVol, grade) VALUES<br>( 'PL-6', 'Inexperimenté', 2999, 'INST' ); |
| UPDATE VueGradePilotes<br>SET grade = 'CDB'<br>WHERE brevet = 'PL-4';                                                                                                                                                                                                                                                                                                                                                                                                          | UPDATE VueGradePilotes<br>SET nbHVol = 50<br>WHERE brevet = 'PL-3';                                                                                                                                                             |

### Confidentialité

La confidentialité est une des vocations premières des vues. Outre l'utilisation de variables d'environnement, il est possible de restreindre l'accès à des tables en fonction de mots précis.

Les vues suivantes limitent temporellement les accès en lecture et en écriture à des table



Notez qu'il est possible, en plus, de limiter l'accès à un utilisateur particulier en utilisant une variable d'environnement (exemple : rajout de la condition AND CURRENT\_USER('Paul@localhost' à une vue).

Tableau 5-23 Vues pour restreindre l'accès à des moments précis

| Web | Définition de la vue                                                                                                                                                                                      | Accès                                                                                                                          |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>CREATE VIEW VueDesCompagniesJoursFeries AS SELECT * FROM Compagnie WHERE DATE_FORMAT(SYSDATE(), '%W') IN ('Sunday', 'Saturday');</pre>                                                               | Restriction en lecture de la table Compagnie, les samedis et dimanches. Mises à jour possibles tout moment.                    |
|     | <pre>CREATE VIEW VueDesPilotesJoursOuvrables AS SELECT * FROM Pilote WHERE CURTIME() + 0 BETWEEN 83000 AND 173000 AND DATE_FORMAT(SYSDATE(), '%W') NOT IN ('Sunday', 'Saturday') WITH CHECK OPTION;</pre> | Restriction, en lecture et en écriture (à cause du WITH CHECK OPTION), de la table Pilote les jours ouvrables de 8h30 à 17h30. |

## Transmission de droits

Les mécanismes de transmission et de révocation de privilèges que nous avons étudiés s'appliquent également aux vues. Ainsi, si un utilisateur désire transmettre des droits sur une partie d'une de ses tables, il utilisera une vue. Seules les données appartenant à la vue seront accessibles aux bénéficiaires.

Les privilèges objet qu'il est possible d'attribuer sur une vue sont les mêmes que ceux applicables sur les tables (SELECT, INSERT, UPDATE sur une ou plusieurs colonnes, DELETE).

Tableau 5-24 Privilèges sur les vues

| Attribution du privilège                                                       | Signification                                                                             |
|--------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|
| <pre>GRANT SELECT ON VueDesCompagniesJoursFeries TO 'Paul'@'localhost';</pre>  | Accès en local de l'utilisateur Paul en lecture sur la vue VueDesCompagniesJoursFériés.   |
| <pre>GRANT INSERT ON VueDesPilotesJoursOuvrables TO 'Jules'@'localhost';</pre> | Accès en local de l'utilisateur Jules en écriture sur la vue VueDesCompagniesJoursFériés. |

## Modification d'une vue [ALTER VIEW]

Vous devez au moins posséder les privilèges CREATE VIEW et DELETE au niveau d'une vue pour pouvoir la modifier. La syntaxe SQL est la suivante :

```
ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
VIEW [nomBase.]nomVue [(listecolonomies)]
AS requêteSELECT
[WITH [CASCADED | LOCAL] CHECK OPTION];
```

## 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 :

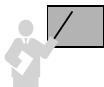
```
SHOW CREATE VIEW [nomBase.]nomVue;
```

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`
| | 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.

Le dictionnaire des données (*metadata* ou *data dictionary*) est une partie majeure d'une base de données MySQL qu'on peut assimiler à une structure centralisée.

Pour MySQL, 16 vues sont issues de tables système non visibles. Ces vues, qui sont appelées *tables* par abus de langage (dans la documentation officielle, dans les livres et sur les forums) et qui sont situées dans la base `INFORMATION_SCHEMA`, suffisent à stocker toutes les informations décrivant tous les objets contenus dans toutes les bases de données. MySQL se rapproche davantage de DB2 et de SQL Server que d'Oracle (qui possède 600 vues). Les noms des vues sont similaires et se rapprochent de la spécification ANSI/ISO SQL:2003 standard Part 11 *Schemata*.

## Constitution

Le dictionnaire des données contient :

- la définition des tables, vues, index, séquences, procédures, fonctions et déclencheurs ;
- la description de l'espace disque alloué et occupé par chaque objet ;
- les valeurs par défaut des colonnes (`DEFAULT`) ;
- la description des contraintes d'intégrité référentielle (de vérification à venir) ;
- le nom des utilisateurs de la base ;
- les priviléges pour chaque utilisateur ;
- des informations d'audit (accès aux objets) et d'autre nature (commentaires, par exemple).



Toutes les tables du dictionnaire des données ne sont accessibles qu'en lecture seulement. Elles appartiennent à la base de données `INFORMATION_SCHEMA`. L'interrogation du dictionnaire des données est permise à tout utilisateur (qui ne verra que les objets qui lui sont toutefois accessibles avec ses propres priviléges) et peut se faire au travers de requêtes `SELECT` ou par le biais de la commande `SHOW`.

Toutes les informations contenues dans les tables du dictionnaire des données sont codées en minuscules.

Le dictionnaire des données est mis automatiquement à jour après chaque instruction SQL ou LMD (`INSERT`, `UPDATE`, `DELETE`, `LOCK TABLE`).

Les avantages d'interroger le dictionnaire des données par des requêtes sont les suivants :

---

1. Je conserve le vocable de « vue » pour être plus près de la réalité. Cependant, parler de *table* ou de *vue* est équivalent, puisqu'elles sont interrogables de la même manière : par `SELECT`.

- Conforme aux règles d'E.F. Codd (le père du modèle relationnel) ; toutes les manipulations sont réalisées à l'aide des opérateurs relationnels sur des tables.
- Inutile d'apprendre de nouvelles instructions (`SHOW paramètres`) propriétaire MySQL. La migration vers un autre SGBD est ainsi facilitée.
- Les possibilités d'extraction sont quasiment illimitées du fait du grand nombre de tables de jointures (ou d'autres opérateurs) possible.
- Vous avez tellement souffert au chapitre 4, que vous avez ici l'occasion de mettre à l'épreuve vos connaissances dans un contexte plus « système ».

## Modèle graphique du dictionnaire des données

Le diagramme suivant – qui s'apparente plus au niveau logique, car les clés étrangères apparaissent –, tiré de <http://mysqldevelopment.com>, décrit la structure des vues du dictionnaire des données.

## Démarche à suivre

La démarche à suivre afin d'interroger correctement le dictionnaire des données à propos d'un objet est la suivante :

- trouver le nom de la vue (ou des vues) pertinente(s) à partir du schéma précédent ;
- choisir les colonnes de la vue (ou des vues) à sélectionner (soit à partir du graphique en affichant la structure de la vue par la commande `DESCRIBE`) ;
- interroger la vue (ou les vues) en exécutant une instruction `SELECT` contenant les colonnes intéressantes.

### *Recherche du nom d'une vue*

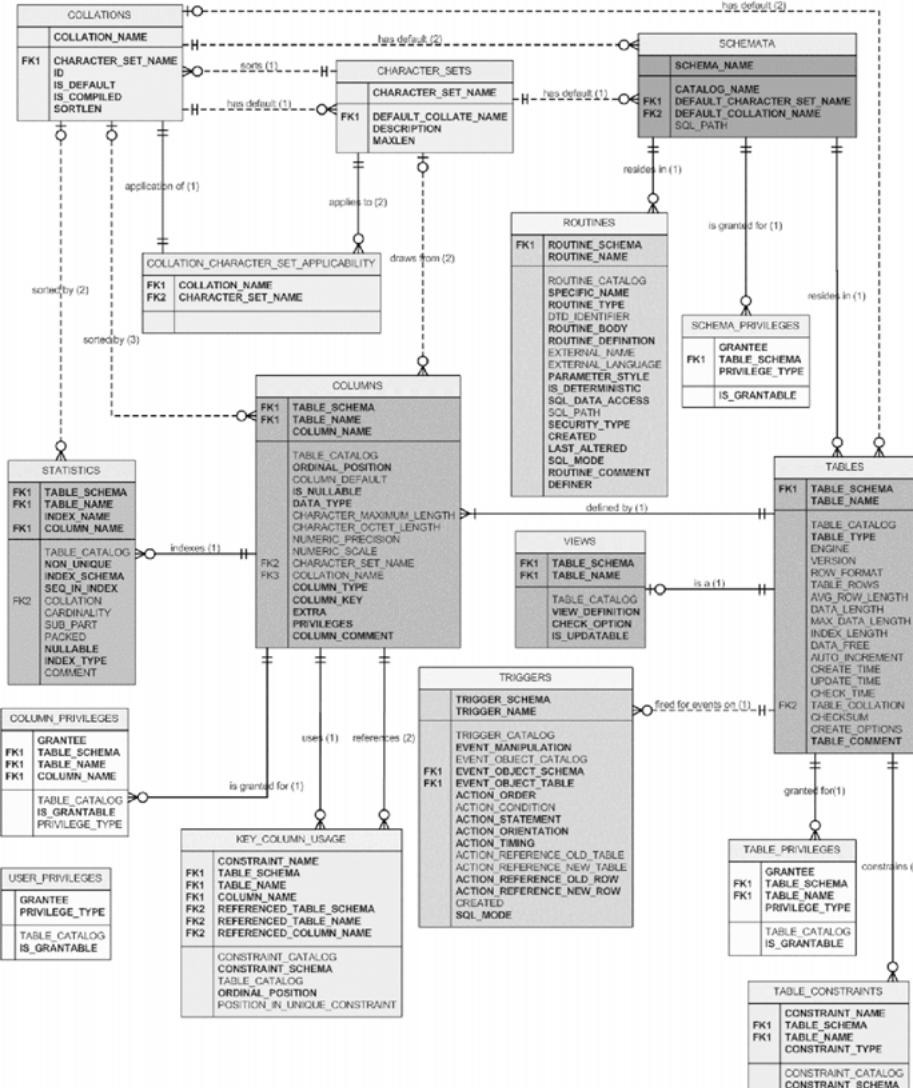
Il n'y a pas de moyen automatique de trouver le nom de la vue pertinente. Cela dit, il n'y a que 16, et vous en aurez vite fait le tour.

### *Choisir les colonnes*

Le choix des colonnes d'une vue du dictionnaire des données s'effectue après avoir inspecté la structure de cette vue (par `DESCRIBE`). Le nom de la colonne est en général assez parlant. Dans notre exemple, la vue contient huit colonnes. Il apparaît que la clause de définition de chaque vue est contenue dans la colonne `VIEW_DEFINITION`. La colonne `CHECK_OPTION` doit indiquer en principe le fait que la vue est déclarée avec une contrainte de vérification. Mais quelle colonne renseigne donc le nom de ladite vue ?

Figure 5-11 Modèle graphique du dictionnaire des données

Conceptual model of the MySQL INFORMATION\_SCHEMA database



Applies to MySQL version: 5.0.15.

Last updated: 25-10-2005. For earlier versions, check out [http://www.xcdsql.org/Misc/MySQL\\_INFORMATION\\_SCHEMA\\_CHANGES.html](http://www.xcdsql.org/Misc/MySQL_INFORMATION_SCHEMA_CHANGES.html) visit <http://www.mysqldevelopment.com> for info on all these cool MySQL 5 features.

For bugs, omissions or suggestions, mail: R\_P\_Bouman@hotmail.com

```
DESCRIBE INFORMATION_SCHEMA.VIEWS;
+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+
TABLE_CATALOG	varchar(512)	YES		NULL	
TABLE_SCHEMA	varchar(64)	NO			
TABLE_NAME	varchar(64)	NO			
VIEW_DEFINITION	longtext	NO			
CHECK_OPTION	varchar(8)	NO			
IS_UPDATABLE	varchar(3)	NO			
DEFINER	varchar(77)	NO			
SECURITY_TYPE	varchar(7)	NO			
+-----+-----+-----+-----+
```

### ***Interroger la vue***

L’interrogation de la vue sur les colonnes choisies est l’étape finale de la recherche de données dans le dictionnaire. Il convient d’écrire une requête monotable ou multitable (jointures) extrait des données contenues dans la vue. Ces données sont en fait contenues dans des tables du système qui ne sont pas accessibles pour des raisons sécuritaires.

Supposons que je suis root en local et que je désire connaître le nom, l’emplacement et le caractère contraint de toutes les vues existantes :

```
SELECT TABLE_SCHEMA, TABLE_NAME, CHECK_OPTION FROM INFORMATION_SCHEMA.VIEWS;
+-----+-----+-----+
| TABLE_SCHEMA | TABLE_NAME | CHECK_OPTION |
+-----+-----+-----+
| bdnouvelle | VueDesSocietes | NONE |
| bdsoutou | VueDesPilotesJoursOuvrables | CASCDED |
+-----+-----+-----+
```

Si j’avais voulu connaître les vues contenues seulement dans la base bdsoutou, il suffirait d’ajouter la condition (WHERE TABLE\_SCHEMA='bdsoutou').

Vous pouvez noter que MySQL utilise :

- La colonne TABLE\_SCHEMA pour désigner une *database*.
- La colonne TABLE\_NAME pour stocker le nom de chaque vue des différents schémas. La norme SQL doit y être pour quelque chose (Oracle nomme la colonne VIEW\_NAME).
- La colonne CHECK\_OPTION pour indiquer le caractère restreint de chaque vue (la première n’est pas restreinte, la seconde l’est).

## **Classification des vues**

Le tableau suivant classe les vues selon leur fonctionnalité. Notez qu’aucune redondance de synonyme n’existe (si vous voulez réaliser une extraction pour découvrir quelque chose, il n’y aura pas beaucoup de requêtes différentes possibles).

Tableau 5-25 Vues du dictionnaire des données

| Nature de l'objet   | Vues                                                                                                                                                                                                                                                                                      |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Serveur             | SCHEMATA : caractéristiques du serveur (jeux de caractères utilisés).<br>CHARACTER_SETS : informations sur les colonnes pour lesquelles l'utilisateur a reçu une autorisation.<br>COLLATIONS et COLLATION_CHARACTER_SET_APPLICABILITY : relatifs aux jeux de caractères.                  |
| Privilèges          | SCHEMA_PRIVILEGES : liste des prérogatives au niveau <i>database</i> .<br>TABLE_PRIVILEGES : liste des prérogatives au niveau <i>table</i> .<br>USER_PRIVILEGES : liste des prérogatives au niveau <i>user</i> .<br>COLUMN_PRIVILEGES : liste des prérogatives au niveau <i>columns</i> . |
| Tables et séquences | TABLES : caractéristiques des tables (et séquences) dans les bases.                                                                                                                                                                                                                       |
| Colonnes            | COLUMNS : colonnes des tables et vues.                                                                                                                                                                                                                                                    |
| Index               | STATISTICS : description des index.                                                                                                                                                                                                                                                       |
| Contraintes         | TABLE_CONSTRAINTS : définition des contraintes de tables.<br>KEY_COLUMN_USAGE : composition des contraintes (colonnes).                                                                                                                                                                   |
| Vues                | VIEWS : description des vues.                                                                                                                                                                                                                                                             |
| Sous-programmes     | ROUTINES : description des sous-programmes stockés.<br>TRIGGERS : description des déclencheurs.                                                                                                                                                                                           |

Interrogeons à présent quelques-unes de ces vues dans le cadre d'exemples concrets.

## Bases de données du serveur

La requête suivante interroge la vue SCHEMATA et permet de retrouver les caractéristiques (jeux de caractères) des bases de données hébergées par le serveur.

```
SELECT SCHEMA_NAME AS 'Base de donnees',
 DEFAULT_CHARACTER_SET_NAME AS 'Jeu caracteres',
 DEFAULT_COLLATION_NAME AS 'Collation'
 FROM INFORMATION_SCHEMA.SCHEMATA;
+-----+-----+-----+
| Base de donnees | Jeu caracteres | Collation |
+-----+-----+-----+
information_schema	utf8	utf8_general_ci
bdnouvelle	ascii	ascii_general_ci
bdsoutou	latin1	latin1_swedish_ci
mysql	latin1	latin1_swedish_ci
test	latin1	latin1_swedish_ci
+-----+-----+-----+
```

Notez que MySQL utilise :

- la colonne `DEFAULT_CHARACTER_SET_NAME` pour désigner le jeu de caractères de la *database*. Le dictionnaire est lui-même codé en `utf8`.
- la colonne `DEFAULT_COLLATION_NAME` pour désigner la collation du jeu de caractères.



Les colonnes `CATALOG_NAME` et `SQL_PATH` ne sont pas encore renseignées. Elles proviennent toutes deux de la spécification de la norme SQL. La première est relative au concept de schéma (qu'on peut assimiler à une collection de bases), et la seconde concerne un nom binaire qu'on pourrait associer à une routine (sous-programme).

## Composition d'une base

La requête suivante interroge la vue `TABLES` et décrit la composition des bases de données utilisateurs (j'ai filtré volontairement les lignes qui correspondent aux bases de MySQL).

```
SELECT TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE, DATE(CREATE_TIME)
 FROM INFORMATION_SCHEMA.TABLES
 WHERE TABLE_SCHEMA NOT IN ('information_schema', 'test', 'mysql');
```

| TABLE_SCHEMA | TABLE_NAME                  | TABLE_TYPE | DATE(CREATE_TIME) |
|--------------|-----------------------------|------------|-------------------|
| bdbnouvelle  | VueDesSocietes              | VIEW       | NULL              |
| bdsoutou     | Installer                   | BASE TABLE | 2005-11-30        |
| bdsoutou     | Logiciel                    | BASE TABLE | 2005-11-30        |
| bdsoutou     | PCSeuls                     | BASE TABLE | 2005-11-30        |
| bdsoutou     | Pilote                      | BASE TABLE | 2005-11-30        |
| bdsoutou     | Poste                       | BASE TABLE | 2005-11-30        |
| bdsoutou     | Salle                       | BASE TABLE | 2005-11-30        |
| bdsoutou     | Segment                     | BASE TABLE | 2005-11-30        |
| bdsoutou     | Softs                       | BASE TABLE | 2005-11-30        |
| bdsoutou     | Types                       | BASE TABLE | 2005-11-30        |
| bdsoutou     | VueDesPilotesJoursOuvrables | VIEW       | NULL              |

Vous pouvez remarquer que MySQL utilise :

- la colonne `TABLE_TYPE` pour désigner le type de la structure de stockage (les tables temporaires, si elles existent, n'apparaissent pas).
- la colonne `CREATE_TIME` pour désigner la date de création de l'objet.

## Détail de stockage d'une base

En utilisant la même vue du dictionnaire, intéressons-nous à la table `Installer` dans la base `bdsoutou` qui fait partie du schéma des exercices de ce livre. La requête suivante extrait quelques informations intéressantes.

```

SELECT ENGINE,AUTO_INCREMENT,TABLE_ROWS,AVG_ROW_LENGTH,DATA_LENGTH
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'bdsoutou' AND TABLE_NAME='Installer';
+-----+-----+-----+-----+-----+
| ENGINE | AUTO_INCREMENT | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+-----+-----+-----+-----+-----+
| InnoDB | 15 | 14 | 1170 | 16384 |
+-----+-----+-----+-----+-----+

```

Notez que MySQL utilise :

- la colonne `ENGINE` pour désigner le type de moteur de stockage de la table en question.
- la colonne `AUTO_INCREMENT` pour désigner la prochaine valeur de la séquence.
- la colonne `TABLE_ROWS` pour donner le nombre d'enregistrements de la table (ici c'est bien cohérent avec la séquence qui fait office de clé primaire).
- la colonne `AVG_ROW_LENGTH` pour désigner la taille moyenne d'une ligne en octets.
- la colonne `DATA_LENGTH` pour désigner la taille de la table en octets.

Citons, pour en terminer avec cette vue, les colonnes :

- `TABLE_COLLATION` qui indique le jeu de caractères de la table.
- `TABLE_COMMENT` qui renseigne notamment à propos des références entre tables par les clés étrangères.

## Structure d'une table

Intéressons-nous à présent à la vue `COLUMNS` qui décrit la structure des tables au niveau *column level*.

### Structure au premier niveau

La requête suivante décrit en partie la table `Installer`. Notez que ça ressemble assez à `DESCRIBE` (normal car l'instruction a été programmée à l'aide d'une requête analogue).

```

SELECT COLUMN_NAME,DATA_TYPE,ORDINAL_POSITION,COLUMN_DEFAULT,COLUMN_KEY
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'bdsoutou' AND TABLE_NAME='Installer';
+-----+-----+-----+-----+-----+
| COLUMN_NAME | DATA_TYPE | ORDINAL_POSITION | COLUMN_DEFAULT | COLUMN_KEY |
+-----+-----+-----+-----+-----+
nPoste	varchar	1	NULL	
nLog	varchar	2	NULL	
numIns	int	3	NULL	PRI
dateIns	timestamp	4	CURRENT_TIMESTAMP	
delai	time	5	NULL	
+-----+-----+-----+-----+-----+

```

Remarquez que MySQL utilise :

- COLUMN\_NAME pour renseigner le nom de chaque colonne.
- DATA\_TYPE pour donner le typeMySQL.
- ORDINAL\_POSITION pour renseigner l'ordre des colonnes dans la table (utilisé en SELECT \*).
- COLUMN\_DEFAULT pour préciser la valeur par défaut de chaque colonne.
- COLUMN\_KEY pour donner la composition de la clé primaire.

### ***Extraction des colonnes caractères***

La requête suivante décrit en détail les colonnes chaînes de caractères de la table Installer.

```
SELECT COLUMN_NAME,DATA_TYPE,CHARACTER_OCTET_LENGTH AS 'Taille max',IS_NULLABLE
 FROM INFORMATION_SCHEMA.COLUMNS
 WHERE TABLE_SCHEMA = 'bdsoutou' AND TABLE_NAME='Installer'
 AND NUMERIC_PRECISION IS NULL
 AND DATA_TYPE NOT IN ('timestamp','time','date');
+-----+-----+-----+-----+
| COLUMN_NAME | DATA_TYPE | Taille max | IS_NULLABLE |
+-----+-----+-----+-----+
| nPoste | varchar | 7 | YES |
| nLog | varchar | 5 | YES |
+-----+-----+-----+-----+
```

Vous pouvez noter que MySQL utilise :

- IS\_NULLABLE pour renseigner le fait qu'une colonne puisse être nulle.
- CHARACTER\_OCTET\_LENGTH pour renseigner la taille des chaînes de caractères de chaque colonne.

### ***Extraction des colonnes numériques***

La requête suivante détaille les colonnes numériques de la table Installer.

```
SELECT COLUMN_NAME,DATA_TYPE,NUMERIC_PRECISION AS 'Taille max',
 NUMERIC_SCALE AS 'Précision'
 FROM INFORMATION_SCHEMA.COLUMNS
 WHERE TABLE_SCHEMA = 'bdsoutou' AND TABLE_NAME='Installer'
 AND CHARACTER_MAXIMUM_LENGTH IS NULL
 AND DATA_TYPE NOT IN ('timestamp','time','date');
+-----+-----+-----+-----+
| COLUMN_NAME | DATA_TYPE | Taille max | Précision |
+-----+-----+-----+-----+
| numIns | int | 10 | 0 |
+-----+-----+-----+-----+
```

Vous pouvez noter que MySQL utilise :

- NUMERIC\_PRECISION pour renseigner la taille des numériques pour chaque colonne.
- NUMERIC\_SCALE pour renseigner la précision des numériques.

### **Extraction des colonnes date-heure**

La requête suivante extrait toutes les colonnes de type date-heure de la table Installer.

```
SELECT COLUMN_NAME, DATA_TYPE, COLUMN_DEFAULT
 FROM INFORMATION_SCHEMA.COLUMNS
 WHERE TABLE_SCHEMA = 'bdsoutou' AND TABLE_NAME='Installer'
 AND CHARACTER_MAXIMUM_LENGTH IS NULL
 AND DATA_TYPE IN ('timestamp','time','date');
+-----+-----+-----+
| COLUMN_NAME | DATA_TYPE | COLUMN_DEFAULT |
+-----+-----+-----+
| dateIns | timestamp | CURRENT_TIMESTAMP |
| delai | time | NULL |
+-----+-----+-----+
```

Citons, pour en terminer avec cette vue, les colonnes :

- CHARACTER\_SET\_NAME et COLLATION\_NAME qui renseignent sur le jeu de caractères pour chaque colonne de la table.
- COLUMN\_COMMENT qui renseigne sur les éventuels commentaires sur chaque colonne.

### **Recherche des contraintes d'une table**

La vue TABLE\_CONSTRAINTS décrit la nature des contraintes. La requête suivante détaillera les contraintes contenues dans la table Installer de la base bdsoutou.

```
SELECT CONSTRAINT_SCHEMA,CONSTRAINT_NAME,CONSTRAINT_TYPE
 FROM INFORMATION_SCHEMA.TABLE_CONSTRAINTS
 WHERE TABLE_SCHEMA = 'bdsoutou' AND TABLE_NAME='Installer';
+-----+-----+-----+
| CONSTRAINT_SCHEMA | CONSTRAINT_NAME | CONSTRAINT_TYPE |
+-----+-----+-----+
bdsoutou	PRIMARY	PRIMARY KEY
bdsoutou	un_installation	UNIQUE
bdsoutou	fk_Installer_nLog_Logiciel	FOREIGN KEY
bdsoutou	fk_Installer_nPoste_Poste	FOREIGN KEY
+-----+-----+-----+
```

MySQL utilise :

- CONSTRAINT\_SCHEMA pour indiquer la base de données qui contient la contrainte (qui peut être située dans une autre base de données que la table elle-même).

- **CONSTRAINT\_NAME** pour renseigner le nom de la contrainte. Notez ici que MySQL n'extrait pas le nom de ma contrainte (**pk\_Installer**), sans doute est-ce le fait qu'elle est déclarée également **AUTO\_INCREMENT**.
- **CONSTRAINT\_TYPE** pour renseigner le type de chaque contrainte.



La valeur **CHECK** dans la colonne **CONSTRAINT\_TYPE** n'est pas encore prise en charge. Vous pouvez toutefois créer des tables avec des contraintes **CHECK** ; rien ne se passera si vous insérez des données non valides et le dictionnaire restera cohérent en n'extrayant pas d'informations.

## Composition des contraintes d'une table

La vue **KEY\_COLUMN\_USAGE** décrit la composition des contraintes.

### Positions

La requête suivante permet d'extraire la composition des contraintes de la table **Installer** dans la base **bdsoutou**, et en particulier celle de l'unicité du couple (**nPoste,nLog**).

```
SELECT CONSTRAINT_NAME, COLUMN_NAME, ORDINAL_POSITION AS 'Position',
 POSITION_IN_UNIQUE_CONSTRAINT AS 'Position index'
 FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
 WHERE TABLE_SCHEMA = 'bdsoutou' AND TABLE_NAME='Installer';
+-----+-----+-----+-----+
| CONSTRAINT_NAME | COLUMN_NAME | Position | Position index |
+-----+-----+-----+-----+
PRIMARY	numIns	1	NULL
un_installation	nPoste	1	NULL
un_installation	nLog	2	NULL
fk_Installer_nLog_Logiciel	nLog	1	NULL
fk_Installer_nPoste_Poste	nPoste	1	NULL
+-----+-----+-----+-----+
```

MySQL utilise :

- **ORDINAL\_POSITION** qui indique la position de la colonne dans la contrainte (début à 1).
- **POSITION\_IN\_UNIQUE\_CONSTRAINT** est évaluée à **NULL** pour les index (unique primaire). Pour les clés étrangères composites, elle indique la position de la colonne dans la contrainte.

### Détails des contraintes référentielles

Cette même vue permet également de retrouver la nature de la référence pour chaque clé étrangère.

```

SELECT CONSTRAINT_NAME, COLUMN_NAME AS 'Cle',
 REFERENCED_TABLE_SCHEMA AS 'Base cible',
 REFERENCED_TABLE_NAME AS 'Table pere',
 REFERENCED_COLUMN_NAME AS 'Col pere'
 FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
 WHERE TABLE_SCHEMA = 'bdsoutou' AND TABLE_NAME='Installer'
 AND REFERENCED_TABLE_SCHEMA IS NOT NULL;
+-----+-----+-----+-----+
| CONSTRAINT_NAME | Cle | Base cible | Table pere | Col pere
+-----+-----+-----+-----+
| fk_Installer_nLog_Logiciel | nLog | bdsoutou | logiciel | nLog
| fk_Installer_nPoste_Poste | nPoste | bdsoutou | poste | nPoste
+-----+-----+-----+-----+

```

MySQL utilise :

- REFERENCED\_TABLE\_SCHEMA qui indique la base de données hébergeant la table « père ». Ici la base données contient les tables « fils » et « pères », mais il se peut que ces tables soient dans deux bases distinctes.
- REFERENCED\_TABLE\_NAME qui indique le nom de la table « père ».
- REFERENCED\_COLUMN\_NAME qui indique le nom de la colonne référencée dans la table « père ». Ici, les noms des colonnes des tables « fils » et « père » sont identiques, mais il se peut qu'ils diffèrent.

## Recherche du code source d'un sous-programme

La vue ROUTINES décrit la composition des sous-programmes (procédures, et fonctions cataloguées). La requête suivante permet d'extraire le code source des sous-programmes stockés dans la base `test`.

```

SELECT ROUTINE_NAME,ROUTINE_TYPE,ROUTINE_DEFINITION
 FROM INFORMATION_SCHEMA.ROUTINES
 WHERE ROUTINE_SCHEMA = 'test';
+-----+-----+-----+
| ROUTINE_NAME | ROUTINE_TYPE | ROUTINE_DEFINITION
+-----+-----+-----+
| sp1 | PROCEDURE | BEGIN
 DECLARE v_brevet CHAR(6);
 SET v_brevet := 'PROC';
 END
| sp2 | FUNCTION | BEGIN
 DECLARE v_brevet CHAR(3);
 SET v_brevet := 'FCT';
 RETURN v_brevet ;
 END
+-----+-----+-----+

```

MySQL utilise :

- `ROUTINE_SCHEMA` qui indique le nom de la base hébergeant le sous-programme.
- `ROUTINE_NAME` qui indique le nom du sous-programme.
- `ROUTINE_TYPE` qui indique la nature du sous-programme.
- `ROUTINE_DEFINITION` qui liste le code MySQL du sous-programme.

Citons, pour en terminer avec cette vue, les colonnes :

- `SECURITY_TYPE` qui renseigne sur les privilèges associés à la vue lors de son exécution (soit les privilèges de l'utilisateur créateur : `definer` ; soit ceux de l'utilisateur qui l'exécution : `invoker`).
- `CREATED` et `LAST_ALTERED` pour stocker la date de création du sous-programme et l'instant de la dernière compilation.
- `DEFINER` qui indique l'identité de l'utilisateur qui a créé le sous-programme.
- `ROUTINE_COMMENT` qui stocke un éventuel commentaire relatif au sous-programme (initialisé lors de la compilation).

## Privilèges des utilisateurs d'une base de données

On retrouve les différents niveaux de privilèges étudiés en début de chapitre.

### Au niveau global

La vue `USER_PRIVILEGES` liste les privilèges des accès utilisateurs au niveau global (les données viennent de la table `mysql.user`). La requête suivante extrait les privilèges `Paul` et de `Jules` (en accès distant ou en local). Non, vous ne rêvez pas, trois simples qu sont nécessaires pour tester la valeur de la colonne GRANTEE.

```
SELECT GRANTEE,PRIVILEGE_TYPE,IS_GRANTABLE
 FROM INFORMATION_SCHEMA.USER_PRIVILEGES
 WHERE GRANTEE LIKE '''Paul%'' OR GRANTEE LIKE '''Jules%'';
```

| GRANTEE                | PRIVILEGE_TYPE | IS_GRANTABLE |
|------------------------|----------------|--------------|
| 'Paul'@'localhost'     | USAGE          | NO           |
| 'Jules'@'localhost'    | USAGE          | NO           |
| 'Paul'@'192.168.4.173' | SELECT         | NO           |
| 'Paul'@'192.168.4.173' | CREATE         | NO           |

MySQL utilise :

- GRANTEE qui indique le nom de l'accès utilisateur.
- PRIVILEGE\_TYPE qui indique le type du privilège.
- IS\_GRANTABLE qui renseigne sur la possibilité que l'accès utilisateur puisse retransmettre le privilège acquis (reçu avec WITH GRANT OPTION).

### Au niveau database

La vue SCHEMA\_PRIVILEGES possède la même structure que la précédente, en ajoutant colonne TABLE\_SCHEMA. Celle-ci donne le nom de la base de données concernée par les priviléges des accès utilisateurs (les données viennent de la table db.user). La requête suivante extrait les priviléges au niveau *database* de Paul, en accès distant ou local.

```
SELECT TABLE_SCHEMA ,GRANTEE,PRIVILEGE_TYPE,IS_GRANTABLE
 FROM INFORMATION_SCHEMA.SCHEMA_PRIVILEGES
 WHERE GRANTEE LIKE '''Paul%';
```

| TABLE_SCHEMA | GRANTEE                | PRIVILEGE_TYPE | IS_GRANTABLE |
|--------------|------------------------|----------------|--------------|
| bdpaul       | 'Paul'@'192.168.4.173' | SELECT         | NO           |
| bdpaul       | 'Paul'@'localhost'     | DROP           | NO           |
| bdpaul       | 'Paul'@'localhost'     | CREATE ROUTINE | NO           |

### Au niveau table

La vue TABLE\_PRIVILEGES possède la même structure que la précédente, en ajoutant colonne TABLE\_NAME. Celle-ci donne le nom de la table concernée par les priviléges d'accès utilisateurs (les données viennent de la table mysql.tables\_priv). La requête suivante extrait les priviléges au niveau *table* de Paul, en accès distant ou local.

```
SELECT CONCAT(TABLE_SCHEMA,'.',TABLE_NAME) AS 'Base.Table',
 GRANTEE,PRIVILEGE_TYPE AS 'Privilege'
 FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
 WHERE GRANTEE LIKE '''Paul%';
```

| Base.Table                           | GRANTEE                | Privilege |
|--------------------------------------|------------------------|-----------|
| bdsoutou.VueDesCompagniesJoursFeries | 'Paul'@'localhost'     | SELECT    |
| bdpaul.Livre                         | 'Paul'@'192.168.4.173' | SELECT    |
| bdpaul.Livre                         | 'Paul'@'localhost'     | SELECT    |
| bdpaul.Livre                         | 'Paul'@'localhost'     | DELETE    |

## Au niveau column

La vue COLUMN\_PRIVILEGES possède la même structure que la précédente, en ajoutant une colonne COLUMN\_NAME. Celle-ci précise le nom de la colonne concernée par les priviléges des accès utilisateurs (les données viennent de la table mysql columns\_priv). La requête suivante extrait les priviléges au niveau *column* de Paul, en accès distant ou local sur la base de données bdpaul.

```
SELECT CONCAT(TABLE_NAME, '.', COLUMN_NAME) AS 'Table.colonne',
 GRANTEE,PRIVILEGE_TYPE AS 'Privilege'
 FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
 WHERE TABLE_SCHEMA='bdpaul';
+-----+-----+-----+
| Table.colonne | GRANTEE | Privilege |
+-----+-----+-----+
| Livre.ISBN | 'Paul'@'localhost' | UPDATE |
+-----+-----+-----+
```

## Au niveau routine

Se reporter au niveau *database*.

## Commande SHOW

La commande SHOW permet d'extraire facilement des informations provenant du dictionnaire des données. Elle est bien sûr, à l'inverse, plus limitée que l'écriture d'une requête SELECT qui pourra toujours extraire les mêmes informations, mais en interrogant les vues adéquates. La copie d'écran suivante illustre la commande SHOW TABLES qui restitue une réponse à la question : « Quelles sont les tables et les vues présentes dans la base de données en cours d'utilisation ? ».

Figure 5-12 Exemple de SHOW pour lister les tables d'un schéma



The screenshot shows a terminal window titled "MySQLClient" with the command "SHOW TABLES;" entered. The output lists several tables and views:

```
<root@localhost> [bdsoutou] mysql> SHOW TABLES;
+-----+
| Tables_in_bdsoutou |
+-----+
| Installer |
| Logiciel |
| PCSeuls |
| Poste |
| Salle |
| Segment |
| Softs |
| Types |
| VueDesPilotesJoursOuvrables |
+-----+
9 rows in set <0.00 sec>
<root@localhost> [bdsoutou] mysql>
```

Le tableau suivant décrit quelques exemples qui vous seront peut-être utiles.

Tableau 5-26 Exemples de SHOW

| Commande                                                | Résultat                                                                                                                           |
|---------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| SHOW COLUMNS FROM Installer<br>FROM bdsoutou LIKE 'n%'; | Liste des colonnes dont le nom commence par 'n' dans la table Installer de la base bdsoutou.                                       |
| SHOW CREATE DATABASE bdsoutou;                          | Options de création de la base bdsoutou.                                                                                           |
| SHOW CREATE TABLE bdsoutou.Installer;                   | Description totale de l'instruction permettant de créer la table Installer de la base bdsoutou.                                    |
| SHOW DATABASES;                                         | Liste des bases présentes sur le serveur.                                                                                          |
| SHOW ENGINES;                                           | Liste des moteurs de stockage utilisables sur le serveur.                                                                          |
| SHOW ERRORS;                                            | Libellé de l'erreur SQL courante.                                                                                                  |
| SHOW GRANTS FOR 'Paul'@'localhost';                     | Pour un accès utilisateur, liste de ses priviléges aux niveaux <i>global</i> , <i>database</i> , <i>column</i> et <i>routine</i> . |
| SHOW INDEX FROM Installer FROM bdsoutou;                | Description des index de la table Installer de la base bdsoutou.                                                                   |
| SHOW PRIVILEGES;                                        | Liste de tous les priviléges possibles.                                                                                            |
| SHOW TABLE STATUS FROM bdsoutou LIKE 'S%';              | Caractéristiques physiques des tables dont le nom commence par 'S' dans la base bdsoutou.                                          |
| SHOW TABLES FROM mysql;                                 | Liste des tables de la base mysql.                                                                                                 |
| SHOW TRIGGERS;                                          | Caractéristiques des déclencheurs présent sur le serveur.                                                                          |

Les objectifs de ces exercices sont :

- de créer des vues monatables et multitable ;
- d'insérer des enregistrements dans des vues ;
- d'effectuer une mise à jour conditionnée via une vue.

## Exercice

### 5.1 Vues monatables

#### *Vues sans contraintes*

Écrire le script `vues.sql`, permettant de créer :

- La vue `LogicielsUnix` qui contient tous les logiciels de type 'UNIX' (toutes les colonnes conservées). Vérifier la structure et le contenu de la vue (`DESCRIBE` et `SELECT`).
- La vue `Poste_0` de structure (`nPos0, nomPoste0, nSalle0, TypePoste0, indIP, ad`) contient tous les postes du rez-de-chaussée (`etage=0` au niveau de la table `Segment`). une jointure procédurale, sinon la vue sera considérée comme une vue multitable. Vérifier la structure et le contenu de la vue.

Insérer deux nouveaux postes dans la vue tels qu'un poste soit connecté au segment du rez-de-chaussée, et l'autre à un segment d'un autre étage. Vérifier le contenu de la vue et celui de la table `Segment`. Conclusion ?

Supprimer ces deux enregistrements de la table `Poste`.

#### *Résoudre une requête complexe*

Créer la vue `SallePrix` de structure (`nSalle, nomSalle, nbPoste, prixLocation`) qui contient les salles et leur prix de location pour une journée (en fonction du nombre de postes). Le montant de la location d'une salle à la journée sera d'abord calculé sur la base de 100 € par poste. Servez-vous de l'expression `100*nbPoste` dans la requête de définition.

Vérifier le contenu de la vue, puis afficher les salles dont le prix de location dépasse 150 €.

Ajouter la colonne `tarif` de type `SMALLINT(4)` à la table `Types`. Mettre à jour cette table de manière à insérer les valeurs suivantes :

Tableau 5-27 Tarifs des postes

| Type du poste | Tarif en € |
|---------------|------------|
| TX            | 50         |
| PCWS          | 100        |
| PCNT          | 120        |
| UNIX          | 200        |
| NC            | 80         |
| BeOS          | 400        |

Créer la vue `SalleIntermediaire` de structure (`nSalle, typePoste, nombre, tarif`), de telle sorte que le contenu de la vue reflète le tarif ajusté des salles, en fonction du nombre et du type des postes de travail. Il s'agit de grouper par salle, type et tarif (tout en faisant une jointure avec la table `Types` pour les tarifs), et de compter le nombre de postes pour avoir le résultat suivant :

| <code>nSalle</code> | <code>typePoste</code> | <code>nombre</code> | <code>tarif</code> |
|---------------------|------------------------|---------------------|--------------------|
| s01                 | TX                     | 2                   | 50                 |
| s01                 | UNIX                   | 2                   | 200                |
| s02                 | PCWS                   | 2                   | 100                |
| s03                 | TX                     | 1                   | 50                 |
| ...                 |                        |                     |                    |

À partir de la vue `SalleIntermediaire`, créer la vue `SallePrixTotal(nSalle, PrixReel)` qui reflète le prix réel de chaque salle (par exemple, la s01 sera facturée  $2*50 + 1*200 = 300$  €). Vérifier le contenu de cette vue.

Afficher les salles les plus économiques à la location.

#### Vues avec contraintes

Remplacer la vue `Poste0` en rajoutant l'option de contrôle (CHECK OPTION). Tenter d'insérer un poste appartenant à un étage différent du rez-de-chaussée.

Créer la vue `Installer0` de structure (`nPoste, nLog, dateIns`) ne permettant de travailler qu'avec les postes du rez-de-chaussée, tout en interdisant l'installation d'un logiciel de type 'PCNT'. Tenter d'insérer deux postes, dans cette vue, ne correspondant pas à ces deux contraintes : un poste d'un étage, puis un logiciel de type 'PCNT'. Insérer l'enregistrement 'p6', 'log2' qui doit passer à travers la vue.

---

## Exercice 5.2 Vue multitable

Créer la vue `SallePoste` de structure (`nomSalle, nomPoste, adrIP, nomTypePoste`), permettant d'extraire toutes les installations sous la forme suivante :

```
SELECT * FROM SallePoste;
```

| <code>nomSalle</code> | <code>nomPoste</code> | <code>adrIP</code> | <code>nomTypePoste</code> |
|-----------------------|-----------------------|--------------------|---------------------------|
| Salle 1               | Poste 1               | 130.120.80.01      | Terminal X-Window         |
| Salle 1               | Poste 2               | 130.120.80.02      | Système Unix              |
| Salle 1               | Poste 3               | 130.120.80.03      | Terminal X-Window         |
| ...                   |                       |                    |                           |

# **Partie II**

# Programmation

# procédurale



# Bases du langage de programmation

Ce chapitre décrit les caractéristiques générales du langage procédural de programmation MySQL :

- structure d'un programme ;
- déclaration et affectation de variables ;
- structures de contrôle (*si, tant que, répéter, pour*) ;
- mécanismes d'interaction avec la base ;
- programmation de transactions.

## Généralités

---

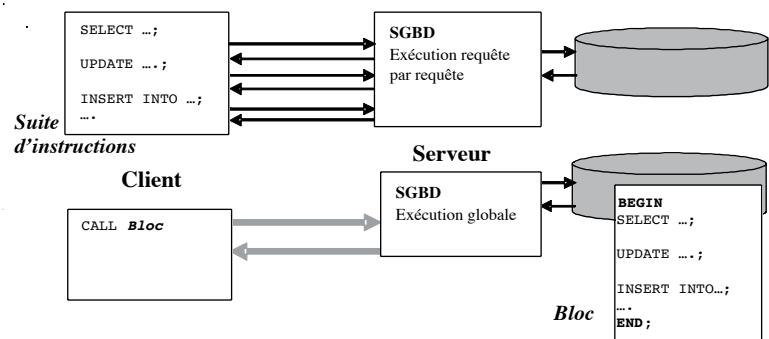
Les structures de contrôle habituelles d'un langage (**IF, WHILE...**) ne font pas partie intégrante de la norme SQL. Elles apparaissent dans une sous-partie optionnelle de la norme ISO/IEC 9075-5:1996. *Flow-control statements*). MySQL les prend en compte.

Le langage procédural de MySQL est une extension de SQL, car il permet de faire cohabiter les habituelles structures de contrôle (*si, pour et tant que* pour les plus connues) avec des instructions SQL (principalement **SELECT, INSERT, UPDATE et DELETE**).

## Environnement client-serveur

Dans un environnement client-serveur, chaque instruction SQL donne lieu à l'envoi d'un message du client vers le serveur suivi de la réponse du serveur vers le client. Il est préférable de travailler avec un sous-programme (qui sera stocké, en fait, côté serveur) plutôt qu'une suite d'instructions SQL susceptibles d'encombrer le trafic réseau. En effet, un bloc d'instructions donnera lieu à un seul échange sur le réseau entre le client et le serveur. Les résultats intermédiaires sont traités côté serveur et seul le résultat final est retourné au client.

Figure 6-1 Trafic sur le réseau d'instructions SQL



## Avantages

Les principaux avantages d'utiliser des sous-programmes (procédures ou fonctions cataloguées qui sont stockées côté serveur) sont :

- La modularité : un sous-programme peut être composé d'autres blocs d'instructions. Un sous-programme peut aussi être réutilisable, car il peut être appelé par un autre.
- La portabilité : un sous-programme est indépendant du système d'exploitation qui héberge le serveur MySQL. En changeant de système, les applicatifs n'ont pas à être modifiés.
- L'intégration avec les données des tables : on retrouvera avec ce langage procédural tous les types de données et d'instructions disponibles sous MySQL, des mécanismes pour parcourir des résultats de requêtes (cursor), pour traiter des erreurs (*handlers*) et pour programmer des transactions (COMMIT, ROLLBACK, SAVEPOINT).
- La sécurité, car les sous-programmes s'exécutent dans un environnement *a priori* sécurisé (SGBD) où il est plus facile de garder la maîtrise sur les ordres SQL exécutés et donc sur les agissements des utilisateurs.

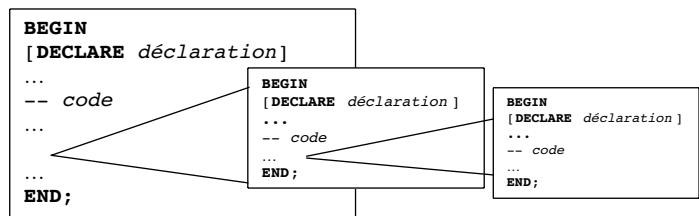
## Structure d'un bloc

Un bloc d'instructions est composé de :

- BEGIN (section obligatoire) contient le code incluant ou non des directives SQL se terminant par le symbole « ; » ;
- DECLARE (directive optionnelle) déclare une variable, un curseur, une exception, etc. ;
- END ferme le bloc.

Un bloc peut être imbriqué dans un autre bloc. Pour tester un bloc, nous verrons dans section *Tests des exemples*, qu'il faut l'inclure dans une procédure cataloguée. MySQL ne prend pas encore en charge les procédures anonymes (sans nom).

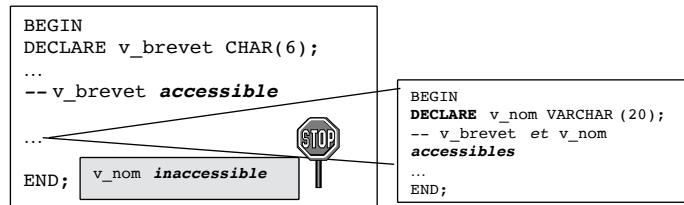
Figure 6-2 Structure d'un bloc d'instructions MySQL



## Portée des objets

La portée d'un objet (variable, curseur ou exception) est la zone du programme qui peut l'accéder. Un objet déclaré dans un bloc est accessible dans les sous-blocs. En revanche, un objet déclaré dans un sous-bloc n'est pas visible du bloc supérieur (principe des accolades de langages C et Java).

Figure 6-3 Visibilité des objets



## Casse et lisibilité

Comme SQL, les sous-programmes sont capables d'interpréter les caractères alphanumériques du jeu de caractères sélectionné. Aucun objet manipulé par programme n'est sensible à la casse (*not case sensitive*). Ainsi `numeroBrevet` et `NumeroBREVET` désignent le même identificateur (tout est traduit en minuscules au niveau du dictionnaire des données). Les règles d'écriture classiques concernant l'indentation et les espaces entre variables, mots réservés et instructions doivent être respectées dans un souci de lisibilité.

Tableau 6-1 Lisibilité du code

| Peuisible                                                    | C'est mieux                                                                |
|--------------------------------------------------------------|----------------------------------------------------------------------------|
| <pre>IF x&gt;y THEN SET max:=x;ELSE SET max:=y;END IF;</pre> | <pre>IF x&gt;y THEN       SET max:=x; ELSE       SET max:=y; END IF;</pre> |

## 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 : monolignes, 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                                                                                                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>-- 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</pre> | <pre>/* 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;</pre> |

## 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.

## Variables scalaires

La déclaration d'une variable scalaire est de la forme suivante :

```
■ DECLARE nomVariable1[, nomVariable2...] typeMySQL [DEFAULT expression]
```

- DEFAULT permet d'initialiser la (ou les) variable(s) – pas forcément à l'aide d'une tante. Le tableau suivant décrit quelques exemples :

Tableau 6-4 Déclarations

| Déclarations                                               | Commentaires                                                                         |
|------------------------------------------------------------|--------------------------------------------------------------------------------------|
| DECLARE v_dateNaissance DATE;                              | Déclare la variable sans l'initialiser.<br>Équivalent à SET v_dateNaissance := NULL; |
| DECLARE v_capacite SMALLINT(4) DEFAULT 999;                | Initialise la variable à 999.                                                        |
| DECLARE v_trouve BOOLEAN DEFAULT TRUE;                     | Initialise la variable à vrai (1).                                                   |
| DECLARE v_Dans2jours DATE<br>DEFAULT ADDDATE(SYSDATE(),2); | Initialise la variable à dans 2 jours.                                               |

## Affectations

Il existe plusieurs possibilités pour affecter une valeur à une variable :

- l'affectation comme on la connaît dans les langages de programmation (SET *variable* := *expression*). Vous pouvez aussi utiliser le symbole « = », mais il est plus préférable de le réserver à la programmation de conditions ;
- la directive DEFAULT ;
- la directive INTO d'une requête (SELECT ... INTO *variable* FROM ...).

## Restrictions



Le type tableau (*array*) n'est pas encore présent dans le langage de MySQL. Cela peut être pénalisant quand on désire travailler en interne avec des résultats d'extraction de moyenne.

Il est impossible d'utiliser un identificateur dans une expression, s'il n'est pas déclaré au tableau. Ici, la déclaration de la variable v\_maxi est incorrecte :

```
■ DECLARE v_maxi INT DEFAULT 2 * v_mini;
```

```
■ DECLARE v_mini INT DEFAULT 15;
```

Comme la plupart des langages récents, les déclarations multiples sont permises. Celle qui suit est juste :

```
| DECLARE i, j, k INT;
```

## Résolution de noms

Lors des conflits potentiels de noms (variables ou colonnes) dans des instructions SQL ( principalement `INSERT`, `UPDATE`, `DELETE` et `SELECT`), le nom de la variable est prioritairement interprété au détriment de la colonne de la table (de même nom).

Dans l'exemple suivant, l'instruction `DELETE` supprime tous les pilotes de la table (et non pas seulement le pilote de nom 'Placide Fresnais'), car MySQL considère les deux identificateurs comme étant la même variable, et non pas comme colonne de la table et variable.

```
| DECLARE nom VARCHAR(16) DEFAULT 'Placide Fresnais';
| DELETE FROM Pilote WHERE nom = nom ;
```

Pour se prémunir de tels effets de bord, une seule solution existe : elle consiste à nommer toutes les variables différemment des colonnes (en utilisant un préfixe, par exemple). Une autre solution serait d'utiliser une étiquette de bloc (*block label*) pour lever d'éventuelles ambiguïtés. Bien qu'il soit possible d'employer des étiquettes de blocs (aussi disponibles pour les structures de contrôle), on ne peut pas encore préfixer des variables pour en distinguer une de même nom entre différents blocs.

Tableau 6-5 Éviter les ambiguïtés

| Préfixer les variables                                                                                                                                        | Étiquette de bloc (préfixe pas opérationnel)                                                                                                             |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>DECLARE v_nom VARCHAR(16)       DEFAULT 'Placide Fresnais'; ... DELETE FROM Pilote WHERE nom = v_nom ; --ou DELETE FROM Pilote WHERE v_nom = nom ;</pre> | <pre>principal: BEGIN DECLARE nom VARCHAR(16)       DEFAULT 'Placide Fresnais'; DELETE FROM Pilote       WHERE principal.nom = nom; END principal;</pre> |

## Opérateurs

Les opérateurs SQL étudiés au chapitre 4 (logiques, arithmétiques, de concaténation...) sont disponibles au sein d'un sous-programme. Les règles de priorité sont les mêmes que dans le cas de SQL.



L'opérateur `IS NULL` permet de tester une formule avec la valeur `NULL`. Toute expression arithmétique contenant une valeur nulle est évaluée à `NULL`.

Le tableau suivant illustre quelques utilisations possibles d'opérateurs logiques :

Tableau 6-6 Utilisation d'opérateurs

| Code MySQL                              | Commentaires                                                          |
|-----------------------------------------|-----------------------------------------------------------------------|
| DECLARE v_compteur INT(3) DEFAULT 0;    | Trois déclarations dont une avec initialisation                       |
| DECLARE v_boolean BOOLEAN;              |                                                                       |
| DECLARE v_nombre INT(3);                |                                                                       |
| SET v_compteur := v_compteur+1;         | incrémentation de v_compteur (opérateur +)                            |
| SET v_boolean := (v_compteur=v_nombre); | v_boolean reçoit <b>NULL</b> , car la condition est fausse.           |
| SET v_boolean := (v_nombre IS NULL);    | v_boolean reçoit <b>TRUE</b> (1 en fait), car la condition est vraie. |

## Variables de session

Il est possible de passer en paramètres d'entrée d'un bloc des variables externes. Ces variables sont dites de session (*user variables*). Elles n'existent que durant la session. On déclare ces variables en ligne de commande à l'aide du symbole « @ ».

```
SET @var1 = expression1 [, @var2 = expression2] ...
```

Le tableau suivant illustre un exemple de deux variables de session : on extrait le nom et le nombre d'heures de vol d'un pilote (table décrite au début du chapitre 4) augmenté d'un autre nombre en paramètre. Son numéro de brevet et la durée du vol sont lus au clavier. Ces variables de session ne sont bien sûr pas à déclarer dans le bloc.

Tableau 6-7 Variables de session

| Web | Code MySQL                                                                                                                                                                                                                                                                                                                                           | Résultat                                                                                             |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
|     | <pre>SET @vs_num = 'PL-4'\$<br/>SET @vs_hvol = 15\$<br/>...<br/>BEGIN<br/>    DECLARE v_nom CHAR(16);<br/>    DECLARE v_nbHVol DECIMAL(7,2);<br/>    SELECT nom, nbHVol<br/>        INTO v_nom, v_nbHVol FROM Pilote<br/>        WHERE brevet = @vs_num;<br/>    SET v_nbHVol := v_nbHVol + @vs_hvol;<br/>    SELECT v_nom, v_nbHVol;<br/>END;</pre> | <pre>+-----+<br/>  v_nom   v_nbHVol  <br/>+-----+<br/>  Placide Fresnais   2465.00<br/>+-----+</pre> |

## Conventions recommandées

Adoptez les conventions d'écriture suivantes pour que vos programmes MySQL soient facilement lisibles et maintenables :



| Objet                         | Convention            | Exemple           |
|-------------------------------|-----------------------|-------------------|
| Variable                      | <b>v_nomVariable</b>  | <b>v_compteur</b> |
| Constante                     | <b>c_nomConstante</b> | <b>c_pi</b>       |
| Variable de session (globale) | <b>vs_nomVariable</b> | <b>vs_brevet</b>  |

## Test des exemples

Parce qu'il n'est pas encore possible d'exécuter des blocs anonymes (sous-programme sans nom et qui n'est pas stocké dans la base), vous devez les inclure dans une procédure cataloguée que vous appellerez dans l'interface de commande.

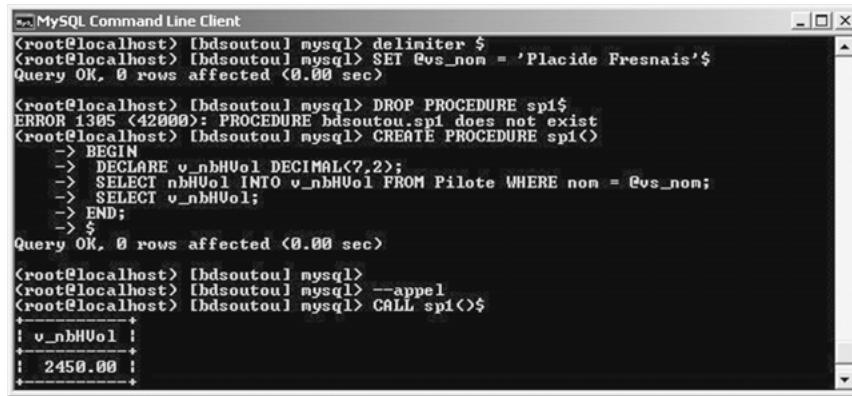
L'exemple suivant extrait le nombre d'heures de vol du pilote de nom 'Placide Fresnais'. Pensez à redéfinir le délimiteur à « \$ » (par exemple) pour pouvoir utiliser, dans le bloc, le symbole « ; » pour terminer chaque instruction.

Tableau 6-9 Tester un exemple de bloc

| Web | Préfixer les variables             | Commentaire                                             |
|-----|------------------------------------|---------------------------------------------------------|
|     | delimiter \$                       | Déclaration du délimiteur et d'une variable de session. |
|     | SET @vs_nom = 'Placide Fresnais'\$ |                                                         |
|     | DROP PROCEDURE sp1\$               | Suppression de la procédure.                            |
|     | CREATE PROCEDURE sp1()             | Création de la procédure.                               |
|     | BEGIN                              | Bloc d'instructions.                                    |
|     | DECLARE v_nbHVol DECIMAL(7,2);     |                                                         |
|     | SELECT nbHVol INTO v_nbHVol        |                                                         |
|     | FROM Pilote WHERE nom = @vs_nom;   |                                                         |
|     | SELECT v_nbHVol;                   | Trace du résultat.                                      |
|     | END;                               | Fin du bloc                                             |
|     | \$                                 |                                                         |
|     | CALL sp1()\$                       | Appel de la procédure.                                  |

Le résultat dans l'interface de commande est le suivant. Allez-y tester vos exemples, maintenant.

Figure 6-4 Exécution d'un bloc



```
MySQL Command Line Client
<root@localhost> [bdssoutou] mysql> delimiter $
<root@localhost> [bdssoutou] mysql> SET @vs_nom = 'Placide Fresnais'$
Query OK, 0 rows affected (0.00 sec)

<root@localhost> [bdssoutou] mysql> DROP PROCEDURE sp1$
ERROR 1305 (42000): PROCEDURE bdssoutou.sp1 does not exist
<root@localhost> [bdssoutou] mysql> CREATE PROCEDURE sp1()
 -> BEGIN
 -> DECLARE v_nbHVol DECIMAL(7,2);
 -> SELECT nbHVol INTO v_nbHVol FROM Pilote WHERE nom = @vs_nom;
 -> SELECT v_nbHVol;
 -> END;
 -> $
Query OK, 0 rows affected (0.00 sec)

<root@localhost> [bdssoutou] mysql> --appel
<root@localhost> [bdssoutou] mysql> CALL sp1();$
+-----+
| v_nbHVol |
+-----+
| 2450.00 |
+-----+
```

## Structures de contrôle

En tant que langage procédural, MySQL offre la possibilité de programmer :

- les structures conditionnelles *si* et *cas* (IF... et CASE) ;
- des structures répétitives *tant que*, *répéter* et *boucle sans fin* (WHILE, REPEAT et LOO



Pas de structure FOR pour l'instant. Deux directives supplémentaires qui sont toutefois à utiliser avec modération : LEAVE qui sort d'une boucle (ou d'un bloc étiqueté) et ITERATE qui force le programme à refaire un tour de boucle depuis le début.

## Structures conditionnelles

MySQL propose deux structures pour programmer des actions conditionnées : la structure IF et la structure CASE.

### Trois formes de IF

Suivant les tests à programmer, on peut distinguer trois formes de structure IF : IF-THEN-THEN (alors), IF-THEN-ELSE (avec le *sinon* à programmer), et IF-THEN-ELSEIF (imbrication de conditions).

Le tableau suivant donne l'écriture des différentes structures conditionnelles IF. N'oubliez pas « END IF » en fin de structure, et non pas « ENDIF ». L'exemple affiche un message différent selon la nature du numéro de téléphone contenu dans la variable v\_telephone.

Tableau 6-10 Structures IF

| IF-THEN                                                                                                                                                                                                                                                   | IF-THEN-ELSE                                                                                       | IF-THEN-ELSEIF                                                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>IF</b> condition <b>THEN</b><br>instructions;<br><b>END IF;</b>                                                                                                                                                                                        | <b>IF</b> condition <b>THEN</b><br>instructions;<br><b>ELSE</b><br>instructions;<br><b>END IF;</b> | <b>IF</b> condition1 <b>THEN</b><br>instructions;<br><b>ELSEIF</b> condition2 <b>THEN</b><br>instructions2;<br><b>ELSE</b><br>instructions3;<br><b>END IF;</b> |
| <hr/>                                                                                                                                                                                                                                                     |                                                                                                    |                                                                                                                                                                |
| <pre>BEGIN     DECLARE v_telephone CHAR(14)         DEFAULT '06-76-85-14-89';     <b>IF</b> SUBSTR(v_telephone,1,2)='06' <b>THEN</b>         SELECT "C'est un portable";     <b>ELSE</b>         SELECT "C'est un fixe...";     <b>END IF;</b> END;</pre> |                                                                                                    |                                                                                                                                                                |

### Conditions booléennes

Les tableaux suivants précisent le résultat d'opérateurs logiques qui mettent en jeu des variables booléennes pouvant prendre trois valeurs (TRUE, FALSE, NULL). Bien sûr, en l'absence d'un vrai type booléen, MySQL représente TRUE avec 1, et FALSE avec 0. Il est à noter que la négation de NULL (NOT NULL) renvoie une valeur nulle.

Tableau 6-11 Opérateur AND

| AND   | TRUE  | FALSE | NULL  |
|-------|-------|-------|-------|
| TRUE  | TRUE  | FALSE | NULL  |
| FALSE | FALSE | FALSE | FALSE |
| NULL  | NULL  | FALSE | NULL  |

Tableau 6-12 Opérateur OR

| OR    | TRUE | FALSE | NULL |
|-------|------|-------|------|
| TRUE  | TRUE | TRUE  | TRUE |
| FALSE | TRUE | FALSE | NULL |
| NULL  | TRUE | NULL  | NULL |

### Structure CASE

Comme l'instruction IF, la structure CASE permet d'exécuter une séquence d'instructions en fonction de différentes conditions. La structure CASE est utile lorsqu'il faut évaluer une même expression et proposer plusieurs traitements pour diverses conditions.

Selon la nature de l'expression et des conditions, une des deux écritures suivantes peut être utilisée :

Tableau 6-13 Structures CASE

| CASE                                                                                                                                                                                                                                         | <i>searched</i> CASE                                                                                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CASE</b> <i>variable</i><br>WHEN <i>expr1</i> THEN <i>instructions1</i> ;<br>WHEN <i>expr2</i> THEN <i>instructions2</i> ;<br>...<br>WHEN <i>exprN</i> THEN <i>instructionsN</i> ;<br>[ELSE <i>instructionsN+1</i> ;]<br><b>END CASE;</b> | <b>CASE</b><br>WHEN <i>condition1</i> THEN <i>instructions1</i> ;<br>WHEN <i>condition2</i> THEN <i>instructions2</i> ;<br>...<br>WHEN <i>conditionN</i> THEN <i>instructionsN</i> ;<br>[ELSE <i>instructionsN+1</i> ;]<br><b>END CASE;</b> |

Le tableau suivant nous livre l'écriture avec IF d'une programmation qu'il est plus rationnel d'effectuer avec une structure CASE (de type *searched*) :

Tableau 6-14 Différentes programmations

| IF                                                                                                                                                                                                                                                                                                                                                                                                                                                   | CASE                                                                                                                                                                                                                                                         |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <br><pre> IF BEGIN DECLARE v_mention CHAR(2); DECLARE v_note DECIMAL(4,2) DEFAULT 9.8; ...  IF v_note &gt;= 16 THEN     SET v_mention := 'TB'; ELSEIF v_note &gt;= 14 THEN     SET v_mention := 'B'; ELSEIF v_note &gt;= 12 THEN     SET v_mention := 'AB'; ELSEIF v_note &gt;= 10 THEN     SET v_mention := 'P'; ELSE     SET v_mention := 'R'; END IF; ... </pre> | <b>CASE</b><br><pre> WHEN v_note &gt;= 16 THEN SET v_mention := 'TB'; WHEN v_note &gt;= 14 THEN SET v_mention := 'B'; WHEN v_note &gt;= 12 THEN SET v_mention := 'AB'; WHEN v_note &gt;= 10 THEN SET v_mention := 'P'; ELSE     SET v_mention := 'R'; </pre> |

## Structures répétitives

Étudions à présent les trois structures répétitives *tant que*, *répéter* et *boucle sans fin*.

### Structure *tant que*

La structure *tant que* se programme à l'aide de la syntaxe suivante. Avant chaque itération (notamment avant la première), la condition est évaluée. Si elle est vraie, la séquence d'instructions est exécutée, puis la condition est réévaluée pour un éventuel nouveau passage dans la boucle.

boucle. Ce processus continue jusqu'à ce que la condition soit fausse pour passer en séquence après le END WHILE. Quand la condition n'est jamais fausse, on dit que le programme boucle.

```
[etiquette:] WHILE condition DO
 instructions;
END WHILE [etiquette];
```

Le tableau suivant décrit la programmation de deux *tant que*. Le premier calcule la somme des 100 premiers entiers. Le second recherche le premier numéro 4 dans une chaîne de caractères.

Tableau 6-15 Structures tant que

| Web | Condition simple                                                                                                                                                                                                                                                  | Condition composée                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>DECLARE v_somme INT DEFAULT 0; DECLARE v_entier SMALLINT DEFAULT 1;  WHILE (v_entier &lt;= 100) DO     SET v_somme := v_somme+v_entier;     SET v_entier := v_entier+1; END WHILE; SELECT v_somme;</pre> <p>+-----+   v_somme   +-----+   5050   +-----+</p> | <pre>DECLARE v_telephone CHAR(14)         DEFAULT '06-76-85-14-89'; DECLARE v_trouve BOOLEAN         DEFAULT FALSE; DECLARE v_indice SMALLINT DEFAULT 1; WHILE (v_indice &lt;= 14 AND NOT v_trouve) DO     IF SUBSTR(v_telephone,v_indice,1) = '4'         THEN             SET v_trouve := TRUE;         ELSE             SET v_indice := v_indice + 1;         END IF; END WHILE; IF v_trouve THEN     SELECT CONCAT('Trouvé 4 à l''indice :',     ',v_indice); END IF;</pre> <p>Trouvé 4 à l'indice : 11</p> |

Cette structure est la plus puissante, car elle permet de programmer aussi un *répéter*, un *boucle sans fin* et même un *pour* (qui n'est pas encore opérationnel). Elle doit être utilisée quand il est nécessaire de tester une condition avant d'exécuter les instructions contenues dans la boucle.

### Structure répéter

La structure *répéter* se programme à l'aide de la syntaxe REPEAT... UNTIL.



Enfin un *répéter* qui se programme comme il faut (à savoir « répéter... jusqu'à condition »). Les langages C et Java nous avaient déformé cette traduction par do {...} while(condition) qui nécessite d'écrire l'inverse de la condition du *jusqu'à* de l'algorithme. Ouf, MySQL (comme Oracle) a bien programmé la structure *répéter* en traduisant ce fameux *jusqu'à* par directive *until*, et non plus par ce fâcheux *while*.

```
[etiquette:] REPEAT
 instructions;
UNTIL condition END REPEAT [etiquette];
```

La particularité de cette structure est que la première itération est effectuée quelles que soient les conditions initiales. La condition n'est évaluée qu'en fin de boucle.

- Si la condition est fausse, la séquence d'instructions est de nouveau exécutée. Ce processus continue jusqu'à ce que la condition soit vraie pour passer en séquence après le END REPEAT.
- Quand la condition n'est jamais vraie, on dit aussi que le programme boucle...

Le tableau suivant décrit la programmation de la somme des 100 premiers entiers et de la recherche du premier numéro 4 dans une chaîne de caractères, à l'aide de la structure répétitive. Les variables sont les mêmes qu'au tableau précédent.

Tableau 6-16 Structures répéter

| Web | Condition simple                                                                                                               | Condition composée                                                                                                                                                                                                                                                                                               |
|-----|--------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>REPEAT     SET v_somme := v_somme + v_entier;     SET v_entier := v_entier + 1; UNTIL v_entier &gt; 100 END REPEAT;</pre> | <pre>REPEAT     IF SUBSTR(v_telephone,v_indice,1) =         THEN             SET v_trouve := TRUE;         ELSE             SET v_indice := v_indice + 1;         END IF; UNTIL (v_indice &gt; 14 OR v_trouve) END REPEAT; IF v_trouve THEN     SELECT CONCAT('Trouvé 4 à l''indice', ',v_indice); END IF;</pre> |

Cette structure doit être utilisée quand il n'est pas nécessaire de tester la condition avec les données initiales, avant d'exécuter les instructions contenues dans la boucle.

### Structure boucle sans fin

La syntaxe générale de cette structure est programmée par la directive LOOP. Elle devient sans fin si vous n'utilisez pas l'instruction LEAVE qui passe en séquence du END LOOP.

```
[etiquette:] LOOP
 instructions;
END LOOP [etiquette];
```

Le tableau suivant donne l'écriture du calcul de la somme des 100 premiers entiers en utilisant deux boucles sans fin (qui se terminent toutefois, car *tout a une fin*, mais celles-là jusqu'à la fin du programme avec LEAVE). J'en profite pour présenter ITERATE qui force à reprendre l'exécution au début de la boucle.

| Web | Avec LEAVE                                                                                                                                                                           | Avec ITERATE                                                                                                                                                                                               |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>boucle1: LOOP     SET v_somme := v_somme + v_entier;     SET v_entier := v_entier + 1;     IF v_entier &gt; 100 THEN         LEAVE boucle1;     END IF; END LOOP boucle1;</pre> | <pre>boucle1: LOOP     SET v_somme := v_somme + v_entier;     SET v_entier := v_entier + 1;     IF v_entier &lt;= 100 THEN         ITERATE boucle1;     END IF;     LEAVE boucle1; END LOOP boucle1;</pre> |

Il est à noter que LEAVE peut être aussi utilisé pour sortir d'un bloc (s'il est étiqueté). LEAVE et ITERATE peuvent aussi être employés au sein de structures REPEAT ou WHILE.

### Redirection (GOTO)

Célèbre pour faire tendre un programme vers une configuration plutôt de feu d'artifice que de cours d'eau tranquille, l'instruction GOTO est bien connue, mais souvent mal utilisée. Elle peut être pratique dans certains cas, pour sortir d'une boucle ou d'un bloc. Il n'est pas souhaitable que vous utilisez GOTO, à moins que vous écriviez vos algorithmes avec des organigrammes.



Dans son livre blanc (<http://dev.mysql.com/tech-resources/articles/mysql-storedprocedures.html>) Peter Gulutzan parle de « *GOTO étiquette;* » et de « *LABEL étiquette;* ». Cet article est sorti alors que la version *bêta* de MySQL 5.0 n'en était qu'à ses débuts. Cette fonctionnalité semble avoir été supprimée dans la version de production. À suivre, donc.

### Structure pour

Renommée pour les parcours de vecteurs, tableaux et matrices en tout genre, la structure *pour* se caractérise par la connaissance a priori du nombre d'itérations que le programmeur souhaite faire effectuer à son algorithme. La syntaxe générale de cette structure est programmée dans tous les langages par l'instruction *for*.



Absente pour l'instant de MySQL, elle peut se programmer par un *répéter*, un *tant que* ou encore par une *boucle sans fin*. Dans tous ces cas, il faudra définir un indice allant d'une valeur initiale à une valeur finale, tout en incrémentant ce même indice en fin de boucle.

## Interactions avec la base

Cette section décrit les mécanismes que MySQL offre pour interfaçer un sous-programme avec une base de données.

# Extraire des données

La principale instruction capable d'extraire des données contenues dans des tableaux est la commande **SELECT**. Étudiée au chapitre 4 dans un contexte SQL, la particularité de cette instruction à l'intérieur d'un sous-programme est la directive **INTO** qui permet de charger des variables à partir de valeurs de colonnes, comme le montre la syntaxe suivante :

```
SELECT col1 [,col2 ...] INTO variable1 [,variable2 ...]
 FROM nomTable ...;
```

Cette instruction peut aussi être utilisée à l'extérieur d'un bloc pour charger une variable dans une session, par exemple.



Veuillez à ne récupérer qu'un seul enregistrement à l'aide du **WHERE** de la requête. C'est que, puisque vous désirez ne charger qu'une valeur par variable.

- Si vous en extrayez plusieurs, vous verrez l'erreur : « **ERROR 1172 (42000): Result set consisted of more than one row** ».
- Si vous n'en extrayez aucun (*no data found*), aucune erreur n'est soulevée et la variable reste inchangée (elle reste initialisée à la valeur présente avant la requête).

## Colonnes simples

Le tableau suivant décrit l'extraction de la colonne **compa** pour le pilote de code 'PL-2' dans différents contextes :

Tableau 6-18 Extraction de données

| Code MySQL                                                                                                                                                                                                                                               | Commentaires                                                                                                              |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|
| BEGIN<br>DECLARE v_comp VARCHAR(15);<br>SELECT compa <b>INTO</b> v_comp<br>FROM Pilote WHERE brevet='PL-2';<br>...<br>END;                                                                                                                               | Chargement d'une variable locale à un niveau de session.<br>Nécessité d'appeler par la suite cette procédure (CALL).      |
| SET @vs_compa=' '\$<br>SELECT compa <b>INTO</b> @vs_compa FROM Pilote<br>WHERE brevet='PL-2'\$<br>...<br>SET @vs_compa=' '\$<br>CREATE PROCEDURE sp1()<br>BEGIN<br>SELECT compa <b>INTO</b> @vs_compa<br>FROM Pilote WHERE brevet='PL-2';<br>...<br>END; | Chargement d'une variable de session d'un sous-programme.<br>Chargement d'une variable de session dans un sous-programme. |

Pour traiter des requêtes renvoyant plusieurs enregistrements, il faudra utiliser des curseurs (étudiés au chapitre suivant).

## Fonctions SQL

Il est naturel que les fonctions SQL (mono et multilignes) étudiées au chapitre 4 soient également disponibles dans un sous-programme, à condition de les utiliser au sein d'une instruction SELECT. Deux exemples sont décrits dans le tableau suivant : le premier changera la variable avec le nom du pilote de code 'PL-1' en majuscules (table décrise au début du chapitre 4) ; le second affectera à la variable le maximum du nombre d'heures de vol de tous pilotes confondus.

Tableau 6-19 Utilisation de fonctions

| Web | Monoligne                                                                                                                                                                                                                                                                            | Multiligne                                                                                                                                                                                                                                          |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>BEGIN     DECLARE v_nomEnMAJUSCULES CHAR(20);     SELECT <b>UPPER</b>(nom)         INTO v_nomEnMAJUSCULES     FROM Pilote WHERE brevet = 'PL-1';     SELECT v_nomEnMAJUSCULES; END;</pre> <p>+-----+<br/>  v_nomEnMAJUSCULES  <br/>+-----+<br/>  GRATIEN VIEL  <br/>+-----+</p> | <pre>BEGIN     DECLARE v_plusGrandHVol DECIMAL(7,2);     SELECT <b>MAX</b>(nbHVol) INTO v_plusGrandHVol         FROM Pilote;     SELECT v_plusGrandHVol ; END;</pre> <p>+-----+<br/>  v_plusGrandHVol  <br/>+-----+<br/>  2450.00  <br/>+-----+</p> |

## Manipuler des données

Les principales instructions disponibles pour manipuler, par un sous-programme, les éléments d'une base de données sont les mêmes que celles proposées par SQL, à savoir INSERT, UPDATE et DELETE. Pour libérer les verrous au niveau d'un enregistrement (ou des tables), il faudra ajouter les instructions COMMIT ou ROLLBACK (aspects étudiés en fin de chapitre).

### Insertions

Le tableau suivant décrit l'insertion de différents enregistrements sous plusieurs écritures (il est aussi possible d'utiliser des variables de session) :

Comme sous SQL, il faut respecter les noms, types et domaines de valeurs des colonnes. De même, les contraintes de vérification (CHECK qui n'est pas encore opérationnel et NOT NULL et d'intégrité (PRIMARY KEY et FOREIGN KEY) doivent être valides.

Dans le cas inverse, une exception qui précise la nature du problème est levée et peut être interceptée par la directive HANDLER (voir chapitre suivant). Si une telle directive n'existe pas,

Tableau 6-20 Insertion d'enregistrements

| Code MySQL                                                                                                                                                                                                                                                                                                                                                                                                 | Commentaires                                                                                                                                                                                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> BEGIN     DECLARE v_brevet VARCHAR(6) DEFAULT 'PL-7';     DECLARE v_nom      VARCHAR(6);     DECLARE v_HVol     DECIMAL(7,2) DEFAULT 0;     DECLARE v_comp     VARCHAR(6);     INSERT INTO Pilote VALUES         ('PL-6', 'Jules Ente', 3000, 'AF');     SET v_nom := 'Fabrice Peyrard';     SET v_comp := 'SING';     INSERT INTO Pilote VALUES         (v_brevet,v_nom,v_HVol,v_comp); END; </pre> | <p>Déclaration des variables locales.</p> <p>Insertion d'un enregistrement en renseignant les colonnes par des constantes.</p> <p>Insertion d'un enregistrement en renseignant les colonnes par des variables locales.</p> |

dans le bloc qui contient l'instruction `INSERT`, la première exception fera s'interrompre le programme.

### ***Modifications***

Concernant la mise à jour de colonnes par `UPDATE`, la clause `SET` peut être ambiguë dans le sens où l'identificateur à gauche de l'opérateur d'affectation est toujours une colonne de données, alors que celui à droite de l'opérateur peut correspondre à une colonne ou à une variable.

```

UPDATE nomTable
 SET col1 = { variable1 | expression1 | autrecol | (requête) }
 [,col2 = ...]
 [WHERE ...];

```



Si aucun enregistrement n'est modifié, aucune erreur ne se produit et aucune exception n'est levée.

Alors que les affectations dans le code MySQL (`SET ...`) peuvent s'écrire par les symboles « `:=` » ou « `=` », les comparaisons ou affectations SQL nécessitent le symbole « `=` ».

Le tableau suivant décrit la modification de différents enregistrements (il est aussi possible d'employer des variables de session).

Tableau 6-21 Modifications d'enregistrements

| Code MySQL                                                                                                                                | Commentaires                                                                                   |
|-------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| BEGIN<br>DECLARE v_dureeVol DECIMAL(3,1) DEFAULT 4.8;<br><b>UPDATE</b> Pilote<br>SET nbHVol= nbHVol + v_dureeVol<br>WHERE brevet= 'PL-6'; | Déclaration.<br>Modification d'un enregistrement de la table Pilote en utilisant une variable. |
| <b>UPDATE</b> Pilote<br>SET nbHVol= nbHVol + 10<br>WHERE compa = 'AF';                                                                    | Modification de plusieurs enregistrements de la table Pilote en utilisant une constante.       |
| END;                                                                                                                                      |                                                                                                |

### Suppressions

La suppression par DELETE peut être ambiguë (même raison que pour l'instruction UPDATE au niveau de la clause WHERE).

```
DELETE FROM nomTable
[WHERE col1 = { variable1 | expression1 | autrecol | (requête) }
[,col2 = ...]];
```



Si aucun enregistrement n'est modifié, aucune erreur ne se produit et aucune exception n'est levée.

Le tableau suivant décrit la suppression de différents enregistrements (il est aussi possible d'utiliser des variables de session).

Tableau 6-22 Suppression d'enregistrements

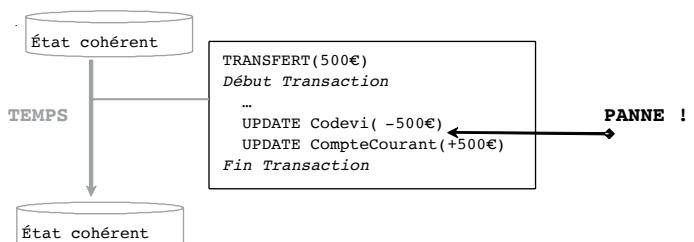
| Code MySQL                                                | Commentaires                                                                                          |
|-----------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| BEGIN<br>DECLARE v_hVolMini DECIMAL(7,2) DEFAULT 1000.00; |                                                                                                       |
| <b>DELETE FROM</b> Pilote WHERE nbHVol < v_hVolMini;      | Supprime les enregistrements de la table Pilote dont le nombre d'heures de vol est inférieur à 1 000. |
| DELETE FROM Pilote WHERE brevet = 'PL-3';                 | Supprime un pilote.                                                                                   |
| DELETE FROM Pilote WHERE brevet = NULL ;                  | Ne supprime aucun pilote.                                                                             |
| END;                                                      |                                                                                                       |

Une transaction est un bloc d'instructions LMD faisant passer la base de données d'un état cohérent à un autre état cohérent. Si un problème logiciel ou matériel survient au cours d'une transaction, aucune des instructions contenues dans la transaction n'est effectuée, que ce soit l'endroit de la transaction où est intervenue l'erreur.

On peut supposer que la majorité des transactions sous MySQL sont programmées dans le langage du serveur. Les langages plus évolués permettent aussi de développer des transactions à travers des API (par exemple la méthode `commit` est comprise dans le package `java.sql`).

L'exemple typique d'une transaction est celui du transfert d'un compte épargne vers un compte courant. Imaginez qu'après une panne votre compte épargne a été débité de la somme de 500 €, sans que votre compte courant soit crédité du même montant ! Vous ne seriez pas très content des services de votre banque (à moins que l'erreur ne soit intervenue dans l'autre sens). Le mécanisme transactionnel empêche un tel scénario en invalidant toutes les opérations faites depuis le début de la transaction, si une panne survient au cours de cette transaction.

Figure 6-5 Transaction



## Caractéristiques

Une transaction assure :

- l'atomicité des instructions qui sont considérées comme une seule opération (principe tout ou rien) ;
- la cohérence (passage d'un état cohérent de la base à un autre état cohérent) ;
- l'isolation des transactions entre elles (lecture consistante, mécanisme décrit plus loin) ;
- la durabilité des opérations (les mises à jour perdurent même si une panne se produit pendant la transaction).

## Début et fin d'une transaction

Deux instructions sont disponibles pour marquer le début d'une transaction : START TRANSACTION ou BEGIN. Ainsi, entre BEGIN et END d'un programme MySQL, il est possible d'écrire plusieurs transactions. Le fait de commencer une transaction termine implicitement celle qui précédait ladite transaction.



Une transaction se termine explicitement par les instructions SQL COMMIT ou ROLLBACK. Elle se termine implicitement :

- à la première commande SQL du LDD ou du LCD rencontrée (CREATE, ALTER, DROP...);
- à la fin normale d'une session utilisateur avec déconnexion ;
- à la fin anormale d'une session utilisateur (sans déconnexion).

Nous détaillons ici les principes de base d'une transaction MySQL sans entrer dans des détails plus techniques (vêrouillages, accès concurrents et transactions réparties) qui sortent du cadre de cet ouvrage.

### Mode de validation

Deux modes de fonctionnement sont possibles : celui par défaut (*autocommit*) qui valide systématiquement toutes les instructions reçues par la base. Dans ce mode point de salut, car vous sera impossible de revenir en arrière afin d'annuler une instruction. Le mode à utiliser pour programmer des transactions est celui inverse (*autocommit off*) qui se déclare à l'aide du paramètre 0 dans l'instruction suivante :

```
SET AUTOCOMMIT = {0 | 1}
```

Le tableau suivant précise la validité de la transaction en fonction des événements possibles.

Tableau 6-23 Validité d'une transaction

| Événement                                                                    | Validité                 |
|------------------------------------------------------------------------------|--------------------------|
| <b>COMMIT</b>                                                                | Transaction validée.     |
| <b>ROLLBACK</b><br>Commande SQL (LDD ou LCD).<br>Fin anormale d'une session. | Transaction non validée. |

### Votre première transaction

Vous pouvez tester rapidement une transaction en écrivant le bloc suivant qui insère une ligne dans une de vos tables.

```

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$
```

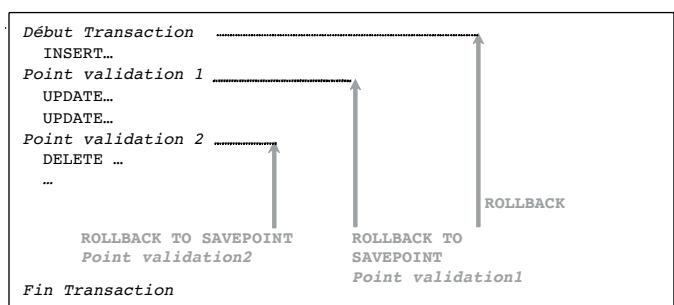
Exécutez ce bloc dans l'interface, puis déconnectez-vous soit en cassant la fenêtre (icône haut à droite), soit proprement avec `exit`. Reconnectez-vous et constatez que l'enregistrement n'est pas présent dans votre table. Même quand la fin du programme est normale, la transaction n'est pas validée (car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

## Contrôle des 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 la 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**



Le tableau suivant décrit une transaction MySQL découpée en trois parties. Le programmeur aura le choix entre les instructions ROLLBACK TO SAVEPOINT indiquées en commentaires pour valider tout ou partie de la transaction. Il faudra finalement se décider entre COMMIT et ROLLBACK.



L'instruction SAVEPOINT déclare un point de validation.

Tableau 6-24 Transaction découpée

| Code MySQL                                      | Commentaires                        |
|-------------------------------------------------|-------------------------------------|
| BEGIN                                           | Première partie de la transaction.  |
| SET AUTOCOMMIT = 0;                             |                                     |
| INSERT INTO Compagnie VALUES                    |                                     |
| ('C2',2,'Place Brassens','Blagnac','Easy Jet'); |                                     |
| SAVEPOINT P1;                                   | Deuxième partie de la transaction   |
| UPDATE Compagnie SET nrue = 125                 |                                     |
| WHERE comp = 'AF';                              |                                     |
| UPDATE Compagnie SET ville = 'Castanet'         |                                     |
| WHERE comp = 'C1';                              |                                     |
| SAVEPOINT P2;                                   | Troisième partie de la transaction. |
| DELETE FROM Compagnie WHERE comp = 'C1';        |                                     |
| -- ROLLBACK TO SAVEPOINT P1;                    | Première partie à valider.          |
| -- ROLLBACK TO SAVEPOINT P2;                    | Deuxième partie à valider.          |
| -- ROLLBACK TO SAVEPOINT P3                     | Troisième partie à valider.         |
| ROLLBACK;                                       | Tout à invalider.                   |
| COMMIT;                                         | Valide la ou les sous-parties.      |
| END;                                            |                                     |

## Transactions imbriquées



Il n'est pas possible d'imbriquer plusieurs transactions se déroulant dans différents blocs.

Il n'est pas possible d'invalider par ROLLBACK une commande SQL du LDD ou du LCD rencontrée (CREATE, ALTER, DROP...).

L'objectif de ces exercices est d'écrire des blocs puis des transactions manipulant des tables du schéma *Parc Informatique*. Vous utiliserez une procédure pour tester vos blocs, comme c'est indiqué dans la section *Test des exemples*.

## Exercice

### 6.1 Extraction de données

Écrire le bloc MySQL qui affiche les détails de la dernière installation de logiciel sous la forme suivante (les champs en gras sont à extraire) :

```
+-----+
| Resultat 1 exo 1 |
+-----+
| Derniere installation en salle : numérodeSalle |
+-----+
+-----+
| Resultat 2 exo 1 |
+-----+
| Poste : numéroPoste Logiciel : nomLogiciel en date du dateInstallation |
+-----+
```

Vous utiliserez SELECT ... INTO pour extraire ces valeurs. Ne tenez pas compte, pour le moment, des erreurs qui pourraient éventuellement se produire (aucune installation de logiciel, poste ou logiciel référencés dans la base, etc.).

## Exercice

### 6.2 Variables de session

Écrire le bloc MySQL qui affecte hors d'un bloc, par des variables session, un numéro de salle, un type de poste, et qui retourne des variables session permettant de composer un message indiquant les nombres de postes et d'installations de logiciels correspondants :

```
+-----+
| Resultat exo2 |
+-----+
| x poste(s) installe(s) en salle y, z installation(s) de type t |
+-----+
```

Essayez pour la salle s01 et le type UNIX. Vous devez extraire 1 poste et 3 installations. Ne tenez pas compte pour le moment d'éventuelles erreurs (aucun poste trouvé ou aucune installation réalisée).

## Exercice

### 6.3 Transaction

Écrire une transaction permettant d'insérer un nouveau logiciel dans la base après avoir passé ses paramètres, par des variables de session, toutes ses caractéristiques (numéro, nom, version et type de logiciel). La date d'achat doit être celle du jour. Tracer l'insertion du logiciel (message Logiciel inséré dans la base).

Il faut ensuite procéder à l'installation de ce logiciel sur le poste de code p7 (utiliser une variable pour pouvoir plus facilement modifier ce paramètre). L'installation doit se faire aussi à la date du jour. Pensez à actualiser correctement la colonne délai qui mesure le délai (TIME) entre l'achat et l'installation. Pour ne pas que ce délai soit nul (les deux insertions se feraient dans la même seconde dans cette transaction), placer une attente de 5 secondes entre l'ajout dans la table Logiciel et celui dans la table Installer à l'aide de l'instruction SELECT SLEEP(5). Utiliser la fonction TIMEDIFF pour calculer ce délai.

Insérer par exemple le logiciel log15, de nom MySQL Query, version 1.4, typePCWS coûtant 95 €.

Tracer la transaction comme suit :

```
+-----+
| message1 |
+-----+
| Logiciel insere dans la base |
+-----+
1 row in set (0.01 sec)

+-----+
| message2 |
+-----+
| Date achat : 2005-11-23 19:16:04 |
+-----+

+-----+
| SLEEP(5) |
+-----+
| 0 |
+-----+

+-----+
| message3 |
+-----+
| Date installation : 2005-11-23 19:16:10 |
+-----+

+-----+
| message4 |
+-----+
| Logiciel installe sur le poste |
+-----+
```

← Attente de 5 secondes à ce niveau

Vérifiez l'état des tables mises à jour après la transaction. Ne tenez pas compte pour le moment d'éventuelles erreurs (numéro du logiciel déjà référencé, type du logiciel incorrect, installation déjà réalisée, etc.).

# Programmation avancée

Ce chapitre est consacré à des caractéristiques avancées du langage procédural de MySQL.

- écriture et appel de sous-programmes ;
- programmation des curseurs ;
- gestion des exceptions ;
- mise en place de déclencheurs ;
- utilisation du SQL dynamique.

## Sous-programmes

---

Les sous-programmes sont des blocs nommés qui sont compilés et qui résident dans la base de données. Dans le vocabulaire des bases de données, on appelle les sous-programmes *functions*, *procedures* ou *stored routines*. Ce sont des fonctions ou procédures « cataloguées » ou « stockées ») capables d'inclure des paramètres en entrée. Comme dans tous les langages de programmation, les fonctions retournent un unique résultat, alors que les procédures réalisent plusieurs actions sans en donner (sauf éventuellement en paramètre de sortie).

Étant un des plus « jeunes » des SGBD, MySQL tend au plus près (en ajoutant toutefois quelques extensions) de la syntaxe normative de SQL:2003 (sections *Stored Modules* et *Computational Completeness*). L'autre SGBD se rapprochant le plus de la norme est DB2 d'IBM. Oracle et SQL Server de Microsoft ont un grand nombre de caractéristiques absentes de la norme.



Pour l'instant, seules les procédures sont capables d'inclure des paramètres en sortie.

Un sous-programme ne se recompile pas automatiquement suite à la modification d'un objet de la base manipulé dans son code (ajout d'une colonne dans une table par exemple).

## Généralités

Il est possible de retrouver le code d'un sous-programme au niveau du dictionnaire de données (voir la fin du chapitre 5). Le sous-programme peut être ainsi partagé dans

contexte multi-utilisateur. Les avantages d'utiliser des sous-programmes ont été soulignés au chapitre 6 (modularité, portabilité, extensibilité, réutilisabilité, intégrité et confidentialité).

Comme les blocs, nous verrons que les sous-programmes ont une partie de déclaration de variables, une autre contenant les instructions et éventuellement une partie pour gérer les exceptions (erreurs produites durant l'exécution).

Une procédure peut être appelée à l'aide de l'interface de commande (par **CALL**), dans un programme externe (Java, PHP, C...), par d'autres procédures ou fonctions, ou dans le corps d'un déclencheur. Les fonctions peuvent être invoquées dans une instruction SQL (**SELECT**, **INSERT**, et **UPDATE**) ou dans une expression (affectation de variable ou calcul).

Le cycle de vie d'un sous-programme est le suivant : création de la procédure ou de la fonction (compilation et stockage dans la base), appel et éventuellement suppression du sous-programme de la base.

## Procédures cataloguées

La syntaxe de création d'une procédure cataloguée est la suivante. Le privilège **CREATE ROUTINE** est requis sur la base de données (ou au niveau global) en question (**ALTER ROUTINE** et **EXECUTE** sont affectés par la suite automatiquement).

```
CREATE PROCEDURE [nomBase.]nomProcédure(
 [[IN | OUT | INOUT] param typeMySQL
 [, [IN | OUT | INOUT] param2 typeMySQL]] ...)
 [LANGUAGE SQL
 | [NOT] DETERMINISTIC
 | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
 | SQL SECURITY { DEFINER | INVOKER }
 | COMMENT 'commentaire'
]
BEGIN
 [DECLARE ... ;]
 bloc d'instructions SQL et MySQL ... ;
END;
délimiteur
```

- Par défaut, la procédure est créée dans la base de données courante (sélectionnée). Si un nom est spécifié (*nomBase*), la procédure appartiendra à cette base de données.
- **IN** désigne un paramètre d'entrée (par défaut), **OUT** un paramètre de sortie et **INOUT** un paramètre d'entrée et de sortie.
- **LANGUAGE SQL** (par défaut) détermine le langage de programmation de la procédure. MySQL n'est pas encore compatible avec d'autres langages que le sien.

- **DETERMINISTIC** est simplement informationnel (l'optimiseur s'en servira dans les versions ultérieures) et décrit le caractère déterministe de la procédure. Si vous interagissez avec la base, il serait plus naturel d'utiliser NOT DETERMINISTIC, car on ne sait pas à priori ce que l'on va extraire (comme dans la boîte de chocolats de *Forrest Gump*).
- **CONTAINS SQL** renseigne sur le fait que la procédure interagit avec la base. NO SQL indique l'inverse. READS SQL DATA précise que les interactions sont en lecture seulement. MODIFIES SQL DATA signifie que des mises à jour de la base sont possibles.
- **SQL SECURITY** détermine si la procédure s'exécute avec les priviléges du créateur (option par défaut : *definer-rights procedure*) ou ceux de l'utilisateur qui appelle la procédure (*invoker-rights procedure*).
- **COMMENT** permet de commenter la procédure au niveau du dictionnaire des données (voir le chapitre 5).
- *bloc d'instructions SQL et MySQL* contient les déclarations et les instructions de la procédure écrite dans le langage de MySQL (voir le chapitre précédent).
- **délimiteur** : délimiteur de commandes différent de « ; » (symbole utilisé obligatoirement en fin de chaque déclaration et instruction du langage procédural de MySQL).

## Fonctions cataloguées

La syntaxe de création d'une fonction cataloguée est **CREATE FUNCTION**. Les prérogatives et les options sont les mêmes que pour les procédures. N'oubliez pas l'instruction « **RETURN variable;** » qui termine la fonction et retourne le résultat (de même type que celui déclaré dans la clause **RETURNS**).

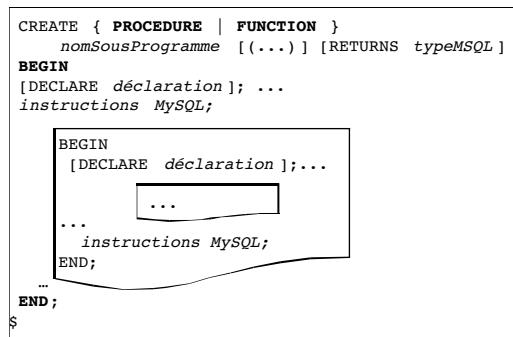
```
CREATE FUNCTION [nomBase.]nomFonction(
 [param typeMySQL
 [,param2 typeMySQL]] ...)

RETURNS typeMySQL
[LANGUAGE SQL
| [NOT] DETERMINISTIC
| { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'commentaire'
]
BEGIN
[DECLARE ... ;]
bloc d'instructions SQL et MySQL ... ;
contenant un « RETURN variable ; »
END;
délimiteur
```

## 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



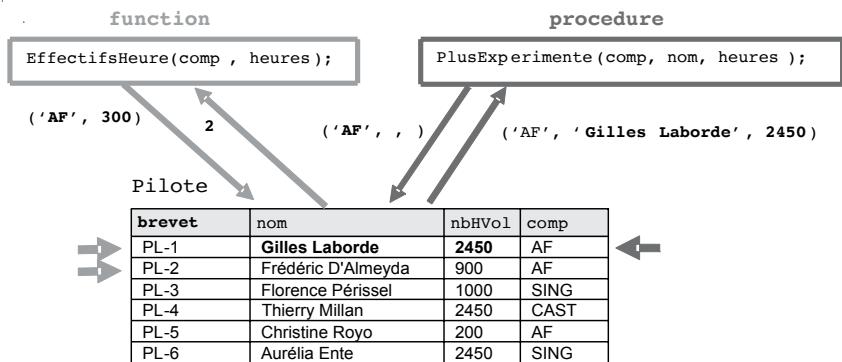
## Exemples

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

- La fonction `EffectifsHeure(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.

Figure 7-2 Procédures



### Une fonction

La création de la fonction est réalisée à l'aide du script suivant (`EffectifsHeure.sql`). Notez que les deux paramètres d'entrée ne sont pas définis par la directive IN et que la clause RETURN doit être présente en fin de codage.



```

CREATE FUNCTION bdsoutou.EffectifsHeure(pcomp VARCHAR(4),
 pheuresVol DECIMAL(7,2)) RETURNS SMALLINT
BEGIN
 DECLARE resultat SMALLINT;
 IF (pcomp IS NULL) THEN
 SELECT COUNT(*) INTO resultat FROM Pilote WHERE nbHVol > pheuresVol;
 ELSE
 SELECT COUNT(*) INTO resultat FROM Pilote WHERE nbHVol > pheuresVol
 AND comp = pcomp;
 END IF;
 RETURN resultat;
END;

```

### Une procédure

La création de la procédure est réalisée à l'aide du script suivant (`PlusExperimente.sql`). Notez les deux derniers paramètres de sortie définis par la directive OUT, et le premier se d'entrée ou de sortie avec la directive INOUT. On peut assimiler le passage d'un paramètre à référence à l'utilisation de la directive INOUT.

```

CREATE PROCEDURE bdsoutou.PlusExperimente
 (INOUT pcomp VARCHAR(4), OUT pnomPil VARCHAR(20), OUT pheuresVol DECIMAL(7,2))
BEGIN
 DECLARE p1 SMALLINT;
 IF (pcomp IS NULL) THEN
 SELECT COUNT(*) INTO p1 FROM Pilote
 WHERE nbHVol=(SELECT MAX(nbHVol) FROM Pilote);
 ELSE
 SELECT COUNT(*) INTO p1 FROM Pilote
 WHERE nbHVol=(SELECT MAX(nbHVol) FROM Pilote WHERE comp=pcomp)
 AND comp = pcomp;
 END IF;
 IF (p1 = 0) THEN
 SELECT ('Aucun pilote n''est le plus expérimenté') AS resultat;
 ELSEIF p1 > 1 THEN
 SELECT('Plusieurs pilotes sont les plus expérimentés') AS resultat;
 ELSE
 IF (pcomp IS NULL) THEN
 SELECT nom, nbHVol, comp INTO pnomPil, pheuresVol, pcomp
 FROM Pilote WHERE nbHVol=(SELECT MAX(nbHVol) FROM Pilote);
 ELSE
 SELECT nom, nbHVol INTO pnomPil, pheuresVol FROM Pilote
 WHERE nbHVol=(SELECT MAX(nbHVol) FROM Pilote WHERE comp=pcomp)
 AND comp = pcomp;
 END IF;
 END IF;
END;

```

## Fonction n'interagissant pas avec la base

La fonction `EcritureComplexe` renvoie une chaîne de caractères désignant l'écriture d'un nombre complexe sous la forme «  $a + bi$  » ou «  $a - bi$  » (`EcritureComplexe.sql`), en fonction du signe des deux paramètres  $a$  et  $b$  définissant la partie réelle et la partie imaginaire du complexe.

```

CREATE FUNCTION bdsoutou.EcritureComplexe
 (reel DECIMAL(7,2), imaginaire DECIMAL(7,2))
 RETURNS VARCHAR(80)
BEGIN
 DECLARE result VARCHAR(80);
 IF (imaginaire < 0) THEN
 SET result := CONCAT('Complexe : ',reel,'-',-imaginaire,'i');
 END IF;
END;

```

```
 ELSE
 SET result := CONCAT('Complexe : ',reel,'+',imaginaire,'i');
 END IF;
 RETURN result ;
END;
```

## Compilation

Pour compiler ces sous-programmes à partir de l'interface de commande, il faut ajouter un délimiteur après chaque dernier END comme suit :

```
delimiter $
Sous programme ;
$
```

Si un message d'erreur apparaît, il indique la ligne concernée ; souvent l'erreur peut être signalée juste avant cette ligne. Le même résultat peut être obtenu par la commande SHOW ERRORS.

Une fois que le message « `Query OK, 0 rows affected (... sec)` » apparaît, le programme est correctement compilé.

## Appel d'un sous-programme

Le créateur d'un sous-programme peut exécuter ce dernier à la demande et sans aucune condition préalable. Pour exécuter un sous-programme à partir d'un autre accès, les conditions suivantes doivent être respectées :

- détenir le privilège `EXECUTE` sur la procédure en question ou sur la base qui contient le sous-programme, ou au niveau global ;
- mentionner le nom de la base (du schéma) contenant le sous-programme à l'exécution de ce dernier (exemple d'appel sous l'interface de commande de la procédure `AugmenteCapacite` de la base `bdjean`, pour l'avion d'immatriculation 'F-GLFS' : « `CALL bdjean.AugmenteCapacite('F-GLFS');` »).

Décrivons l'appel d'un sous-programme sous l'interface de commande, dans un programme MySQL et dans une instruction SQL. Le chapitre suivant détaillera un tel appel à partir d'un programme externe (Java ou PHP).

### *Sous l'interface de commande*

En phase de tests, il est intéressant de pouvoir déclencher un sous-programme directement dans l'interface de commande. La commande `CALL` permet d'appeler une procédure. La fonction est exécutée par son nom dans une instruction SQL.

Le tableau suivant décrit l'appel et le résultat des trois sous-programmes. La fonction est appelée ici dans un `SELECT` et dans un `INSERT` de deux manières différentes.

Tableau 7-1 Appels dans l'interface de commande

| Appel d'un sous-programme                                                                                                                                                                                                  | Résultat                                                                                                                                                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| delimiter ;<br><b>SELECT bdsoutou.EffectifsHeure</b><br>('AF',300);                                                                                                                                                        | +-----+<br>  bdsoutou.EffectifsHeure('AF',300)  <br>+-----+<br>  2  <br>+-----+                                                                                        |
| delimiter ;<br>SET @vs_compa = 'AF';<br>SET @vs_nompil = '';<br>SET @vs_heures = '';<br><br><b>CALL bdsoutou.PlusExperimente</b><br>(@vs_compa, @vs_nompil,<br>@vs_heures);                                                | SELECT @vs_compa,@vs_nompil,@vs_heures;<br>+-----+-----+<br>  @vs_compa   @vs_nompil   @vs_heures<br>+-----+-----+<br>  AF   Gilles Laborde   2450.00<br>+-----+-----+ |
| delimiter ;<br><b>SELECT</b><br><b>bdsoutou.EcritureComplexe(2,-5);</b>                                                                                                                                                    | +-----+<br>  bdsoutou.EcritureComplexe(2,-5)  <br>+-----+<br>  Complexe : 2-5i  <br>+-----+                                                                            |
| CREATE TABLE test.Trace<br>(col VARCHAR(80));<br>INSERT INTO test.Trace<br>SELECT bdsoutou.EcritureComplexe<br>bdsoutou.EcritureComplexe<br>(-2,-5);<br>INSERT INTO test.Trace VALUES<br>bdsoutou.EcritureComplexe(3,-7)); | SELECT * FROM test.Trace;<br>+-----+<br>  col  <br>+-----+<br>  Complexe : -2-5i  <br>  Complexe : 3-7i  <br>+-----+                                                   |

### Dans un sous-programme

Invoquons les sous-programmes à présent à partir d'un autre sous-programme. Le même principe peut être adopté pour l'appel dans un déclencheur. La procédure s'appelle toujours `CALL`, la fonction par son nom (ici elle est appelée dans l'affectation d'une variable).

Tableau 7-2 Appels dans un sous-programme

| Web | Codage                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Appels                                                                                                                                                                                                                                                                                                                                                       |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre> CREATE PROCEDURE test.sp1() BEGIN     DECLARE v_compa VARCHAR(4)         DEFAULT 'AF';     DECLARE v_heures DECIMAL(7, 2)         DEFAULT 300;     DECLARE v_nb pil SMALLINT;     SET v_nb pil := bdsoutou.EffectifsHeure         (v_compa, v_heures);     SELECT v_nb pil; END;  SET @vs_compa = NULL\$  SET @vs_nompil = ''\$  SET @vs_heures = ''\$  CREATE PROCEDURE test.sp2() BEGIN     CALL bdsoutou.PlusExperimente         (@vs_compa, @vs_nompil, @vs_heures); END;  SET @vs_resultat = ''\$  CREATE PROCEDURE test.sp3() BEGIN     SET @vs_resultat :=         bdsoutou.EcritureComplexe(7, -2); END; </pre> | <pre> CALL test.sp1()\$  +-----+   v_nb pil   +-----+        2     +-----+  CALL test.sp2()\$  +-----+   resultat   +-----+   Plusieurs pilotes sont les plus   expérimentés   +-----+ -- Les variables de session sont -- toutes à NULL  CALL test.sp3()\$  SELECT @vs_resultat AS 'Résulta +-----+   Résultat   +-----+   Complexe : 7-2i   +-----+ </pre> |
|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                              |

## Récurivité



La récursivité n'est pour l'instant pas permise dans MySQL au niveau des sous-programmes.

Comme dans tout programme récursif, il ne faudrait pas oublier la condition de terminaison. L'exemple suivant décrit la programmation à l'aide d'une fonction récursive du calcul factorielle d'un entier positif. La compilation se déroule sans erreur, l'appel, lui, nous ramène à l'ordre.

Tableau 7-3 Récursivité

| Code MySQL                                                                                                                                                                       | Commentaires                              |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| delimiter \$<br>CREATE FUNCTION factorielle(n INT)<br>RETURNS INT<br>BEGIN<br>IF n = 1 THEN<br>RETURN (1);<br>ELSE<br>RETURN (n * factorielle(n - 1));<br>END IF;<br>END;<br>\$; | Condition de terminaison.                 |
| SELECT factorielle(10) AS 'Factorielle de 10'\$<br>ERROR 1424 (HY000): Recursive stored routines are not allowed.                                                                | Appel récursif.<br>Appel de la fonction ! |

## Sous-programmes imbriqués



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 imbriquée. 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

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

La compilation renvoie un message d'erreur très clair :

```
ERROR 1303 (2F003): Can't create a PROCEDURE from within another
stored routine
```

La solution est de créer les deux procédures à part ou d'inclure un bloc BEGIN... END dans la procédure de plus haut niveau.

## Modification d'un sous-programme

La modification d'un sous-programme s'exécute par la commande ALTER. Pour pouvoir changer un sous-programme, si vous n'êtes pas son créateur, vous devez détenir le privilège ALTER ROUTINE sur la base de données (ou au niveau global). Plusieurs caractéristiques peuvent être corrigées en une seule instruction (mais ni le code, ni les paramètres). La syntaxe générale est la suivante :

```
ALTER {PROCEDURE | FUNCTION} [nomBase.]nomSousProg
[{ CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA
| SQL SECURITY { DEFINER | INVOKER }
| COMMENT 'commentaire'
...]
```

L'instruction suivante commente le sous-programme Mouchard présent dans la base bdsoutou en indiquant qu'il interagit avec la base en écriture, et qu'il s'exécutera avec les privilèges de l'accès utilisateur qui l'a créé.

```
ALTER PROCEDURE bdsoutou.Mouchard
MODIFIES SQL DATA
SQL SECURITY DEFINER
COMMENT 'Traces de qui lance quoi';
```

## Destruction d'un sous-programme

Pour supprimer un sous-programme, si vous n'êtes pas son créateur, le privilège ALTER ROUTINE est requis sur la base de données (ou au niveau global). La syntaxe de suppression d'un sous-programme est la suivante :

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] [nomBase.]nomSousProg
```

- IF EXISTS évite un message de warning si le sous-programme n'existe pas.

Les instructions suivantes suppriment la procédure cataloguée Mouchard de la base bdsoutou.

```
delimiter ;
DROP PROCEDURE bdsoutou.Mouchard;
```

## 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) fourni par 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, sous 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.

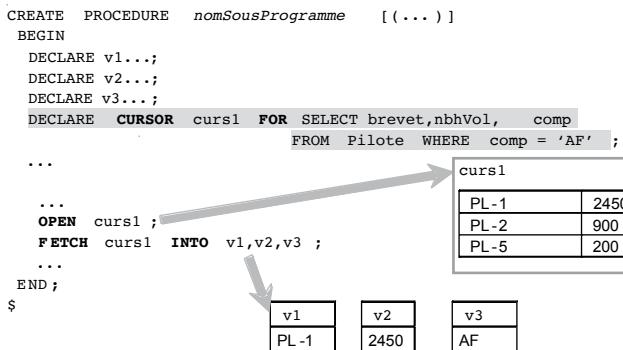
## Généralités

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 d'extrait 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.

Figure 7-3 Principes d'un curseur



Il n'existe pour l'instant qu'une seule manière de parcourir un curseur : du premier au dernier enregistrement.

## Instructions

Les instructions disponibles pour travailler avec des curseurs sont définies dans le tableau suivant :

Tableau 7-5 Instructions pour les curseurs

| Instructions                                 | Commentaires et exemples                                                                                                                                          |
|----------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CURSOR FOR requête;</b>                   | Déclaration du curseur.<br>DECLARE curs1 CURSOR FOR<br>SELECT brevet,nbhVol,comp<br>FROM bdsoutou.Pilote WHERE comp = 'AF';                                       |
| <b>OPEN nomCurseur;</b>                      | Ouverture du curseur (chargement des lignes). Aucune exception levée si la requête ne ramène aucune ligne.<br><b>OPEN curs1;</b>                                  |
| <b>FETCH nomCurseur INTO listeVariables;</b> | Positionnement sur la ligne suivante et chargement de l'enregistrement courant dans une ou plusieurs variables.<br><b>FETCH curs1 INTO var1,var2,var3;</b>        |
| <b>CLOSE nomCurseur;</b>                     | Ferme le curseur. L'exception « ERROR 1326 (24000): Cursor not open » se déclenche si des accès au curseur sont opérés après sa fermeture.<br><b>CLOSE curs1;</b> |

## Parcours d'un curseur

Vous pouvez choisir d'utiliser une structure répétitive *tant que* ou *répéter*. Le tableau suivant présente le parcours d'un curseur à l'aide de ces deux techniques. Ici, il s'agit de faire somme des heures de vol des pilotes de la compagnie de code 'AF'.

Notez l'utilisation obligatoire d'une exception (*handler* dans le vocabulaire de MySQL) qui force le programme à continuer si on arrive en fin du curseur (au-delà du dernier enregistrement), tout en positionnant la variable `curs1` à *vrai* (1). Je dis « obligatoire », car MySQL ne propose pas, pour l'heure, de fonctions sur les curseurs (FOUND, NOT\_FOUND, IS\_CLOSE, etc.). Nous étudierons plus en détail les exceptions dans la prochaine section.

Tableau 7-6 Parcours d'un curseur

| Web | Tant que                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Répéter                                                                                                                            |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>DECLARE fincurs1 BOOLEAN DEFAULT 0; DECLARE v_nbHv    DECIMAL(7,2); DECLARE v_tot     DECIMAL(8,2) DEFAULT 0;  DECLARE curs1 CURSOR FOR         SELECT nbHVol FROM bdsoutou.Pilote WHERE comp = 'AF'; DECLARE CONTINUE HANDLER FOR NOT FOUND SET fincurs1 := 1; OPEN curs1;  REPEAT     FETCH curs1 INTO v_nbHv;     IF NOT fincurs1 THEN         SET v_tot := v_tot+v_nbHv;     END IF;     UNTIL fincurs1 END REPEAT;  CLOSE curs1; SELECT v_tot AS 'Total heures pour les pilotes de ''AF''';</pre> | <pre>FETCH curs1 INTO v_nbHv; WHILE (NOT fincurs1) DO     SET v_tot := v_tot+v_nbHv;     FETCH curs1 INTO v_nbHv; END WHILE;</pre> |

Personnellement, je préfère la programmation avec un *tant que*. L'appel de cette procédure (avec l'une ou l'autre des techniques) sur la table de la figure 7-2 produira le résultat suivant :

```
+-----+
| Total heures pour les pilotes de 'AF' |
+-----+
| 3550.00 |
+-----+
```

## Accès concurrents (FOR UPDATE)

Si vous voulez verrouiller les lignes d'une table interrogée par un curseur dans le but de la 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

| Code MySQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Commentaires                                        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
| <p> BEGIN</p> <pre>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; 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     IF (v_comp='AF') THEN         UPDATE bdsoutou.Pilote             SET nbHVol = nbHVol + 100             WHERE brevet = v_brevet;     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;</pre> | Déclaration du curseur avec verrouillage explicite. |



- Une validation (COMMIT) avant la fermeture d'un curseur FOR UPDATE aura des effets sur le 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).

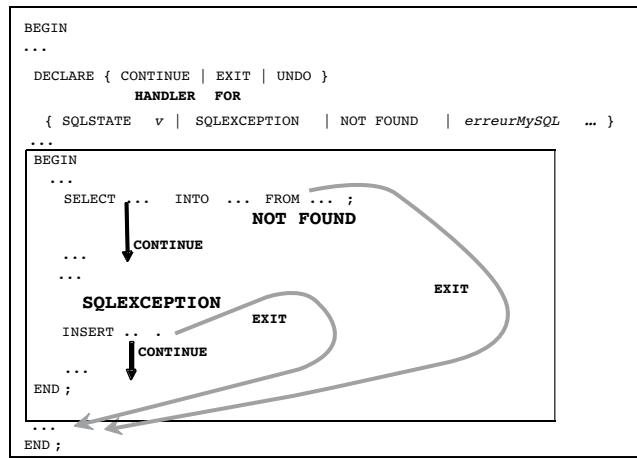
Dans le langage des informaticiens, on dit qu'on *garde la main* pendant l'exécution d'un programme. Le mécanisme des exceptions (*handling errors*) est largement utilisé par tous les développeurs, car il est prépondérant dans la mise en œuvre des transactions. Les exceptions peuvent se paramétriser 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 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 (bloque d'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 interrogant 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 une 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**).

| Code MySQL                                                                                                                                                                       | Commentaires                                   |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|
| <code>CREATE PROCEDURE bdsoutou.procException1<br/>    (IN p_comp VARCHAR(4))<br/>BEGIN<br/>    DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;<br/>    DECLARE var1 VARCHAR(20);</code> | Déclaration de la procédure et des variables.  |
| <code>BEGIN<br/>    DECLARE EXIT HANDLER FOR NOT FOUND<br/>        SET flagNOTFOUND :=1;<br/>    SELECT nom INTO var1 FROM bdsoutou.Pilote<br/>        WHERE comp=p_comp;</code> | Bloc qui déclare l'exception.                  |
| <code>    SELECT CONCAT('Le seul pilote de la compagnie ',<br/>        p_comp,' est ',var1)<br/>        AS 'Resultat procException1';</code>                                     | Requête pouvant déclencher l'exception prévue. |
| <code>END;</code>                                                                                                                                                                | Affichage du résultat.                         |
| <code>IF flagNOTFOUND THEN<br/>    SELECT CONCAT('Il n''y a pas de pilote pour la<br/>        compagnie ',p_comp)<br/>        AS 'Resultat procException1';</code>               | Fin du bloc.<br>Gestion de l'erreur.           |
| <code>END IF;<br/>END;</code>                                                                                                                                                    | Fin de la procédure.                           |

La trace d'une exécution correcte de cette procédure (si la requête retourne une seule ligne) procédure ne passe pas dans le si) est la suivante :

```
CALL bdsoutou.procException1('CAST')$
+-----+
| Resultat procException1 |
+-----+
| Le seul pilote de la compagnie CAST est Sonia Dietrich |
+-----+
```

Une autre exécution correcte de cette procédure (qui se déroute hors du bloc du fait de NOT FOUND) est la suivante :

```
CALL bdsoutou.procException1('RIEN')$
+-----+
| Resultat procException1 |
+-----+
| Il n'y a pas de pilote pour la compagnie RIEN |
+-----+
```

## 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 117 (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

| Code MySQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | Commentaires                                                                                                                                                                                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Web</b></p> <pre>CREATE PROCEDURE bdsoutou.procException1     (IN p_comp VARCHAR(4)) BEGIN     DECLARE flagPlusDun BOOLEAN DEFAULT 0;     DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;     DECLARE var1 VARCHAR(20);      BEGIN         DECLARE EXIT HANDLER FOR 1172             SET flagPlusDun :=1;         DECLARE EXIT HANDLER FOR NOT FOUND             SET flagNOTFOUND :=1;          SELECT nom INTO var1 FROM bdsoutou.Pilote         WHERE comp=p_comp;          SELECT CONCAT('Le seul pilote de la compagnie ',                      p_comp, ' est ',var1)         AS 'Resultat procException1';     END;      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; END;</pre> | <p>Déclaration de la procédure et des variables.</p> <p>Bloc qui déclare les deux exceptions.</p> <p>Requête pouvant déclencher l'exception prévue.</p> <p>Affichage du résultat.</p> <p>Fin du bloc.</p> <p>Gestion des erreurs.</p> <p>Fin de la procédure.</p> |

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

```

CALL bdsoutou.procException1('AF')$+
+-----+
| Resultat procException1 |
+-----+
| Il y a plusieurs pilotes pour la compagnie AF |
+-----+

```

### *Appel d'une procédure dans l'exception*

Le tableau suivant décrit l'appel d'une procédure dans le cas de l'erreur NOT FOUND. La procédure principale exécute le sous-programme, puis sort du bloc principal et se termine correctement.

Tableau 7-10 Exception NOT FOUND traitée avec EXIT et CALL

| Code MySQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | Commentaires                                                             |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
| <b>Web</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                          |
| <pre> CREATE PROCEDURE bdsoutou.pasTrouve     (IN p_comp VARCHAR(4)) BEGIN     SELECT CONCAT('Il n''y a pas de pilote pour     la compagnie ',p_comp)     AS 'Resultat procpasTrouve'; END;  CREATE PROCEDURE bdsoutou.procException1     (IN p_comp VARCHAR(4)) BEGIN     DECLARE var1 VARCHAR(20);     DECLARE EXIT HANDLER FOR NOT FOUND         CALL bdsoutou.pasTrouve(p_comp);     SELECT nom INTO var1 FROM bdsoutou.Pilote         WHERE comp = p_comp;     SELECT CONCAT('Le seul pilote de la compagnie     ',p_comp,' est ',var1)     AS 'Resultat procException1'; END; </pre> | Codage du sous-programme appelé lors de l'exception.                     |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Procédure qui déclare l'exception.                                       |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Requête pouvant déclencher l'exception prévue.<br>Affichage du résultat. |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Fin de la procédure principale.                                          |

La trace d'une exécution correcte de cette procédure (qui se déroute vers le sous-programme appelé du fait de NOT FOUND) est la suivante :

```

+-----+
| Resultat procpasTrouve |
+-----+
| Il n'y a pas de pilote pour la compagnie RIEN |
+-----+

```

Il est aisément de transposer ce raisonnement à plusieurs exceptions appelant différents programmes.

## Exceptions avec CONTINUE

Étudions à présent la clause CONTINUE de l'instruction DECLARE HANDLER.

### Gestion de NOT FOUND

Le tableau suivant décrit la même procédure gérant l'erreur NOT FOUND. La procédure se termine correctement si la requête retourne une seule ligne, mais pas lorsqu'elle en renvoie plusieurs (erreur Result consisted of more than one row).

Tableau 7-11 Exception NOT FOUND traitée avec CONTINUE

| Code MySQL                                                                                                                                                                                                                                                                                                                                                                  | Commentaires                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|
|  CREATE PROCEDURE bdsoutou.procException1<br>(IN p_comp VARCHAR(4))<br>BEGIN<br>DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;<br>DECLARE var1 VARCHAR(20);<br>DECLARE CONTINUE HANDLER FOR NOT FOUND<br>SET flagNOTFOUND :=1;<br>SELECT nom INTO var1 FROM bdsoutou.Pilote<br>WHERE comp=p_comp; | Déclaration de la procédure et des variables.  |
|                                                                                                                                                                                                                                                                                                                                                                             | Bloc qui déclare l'exception.                  |
|                                                                                                                                                                                                                                                                                                                                                                             | Requête pouvant déclencher l'exception prévue. |
| IF flagNOTFOUND THEN<br>SELECT CONCAT('Il n''y a pas de pilote pour la<br>compagnie ',p_comp)<br>AS 'Resultat procException1';<br>ELSE<br>SELECT CONCAT('Le seul pilote de la compagnie ',<br>p_comp,' est ',var1)<br>AS 'Resultat procException1';<br>END IF;<br>END;                                                                                                      | Test de gestion de l'erreur.                   |
|                                                                                                                                                                                                                                                                                                                                                                             | Affichage du résultat.                         |
|                                                                                                                                                                                                                                                                                                                                                                             | Fin de la procédure.                           |

### Gestion d'une erreur particulière

Le tableau suivant décrit la même procédure qui gère en plus l'erreur Result consisted of more than one row. La procédure se termine correctement si la requête retourne une seule ou plusieurs lignes.

Tableau 7-12 Exceptions 1172 et NOT FOUND traitées avec CONTINUE

| Web | Code MySQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Commentaires                                   |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------|
|     | CREATE PROCEDURE bdsoutou.procException1<br>(IN p_comp VARCHAR(4))<br>BEGIN<br>DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;<br>DECLARE flagPlusDun BOOLEAN DEFAULT 0;<br>DECLARE var1 VARCHAR(20);<br>DECLARE <b>CONTINUE</b> HANDLER FOR <b>1172</b><br>SET flagPlusDun :=1;<br>DECLARE <b>CONTINUE</b> HANDLER FOR <b>NOT FOUND</b><br>SET flagNOTFOUND :=1;<br>SELECT nom INTO var1 FROM bdsoutou.Pilote<br>WHERE comp=p_comp;<br><br>IF flagNOTFOUND THEN<br>SELECT CONCAT('Il n''y a pas de pilote pour la<br>compagnie ',p_comp)<br>AS 'Resultat procException1';<br>ELSEIF flagPlusDun THEN<br>SELECT CONCAT('Il y a plusieurs pilotes<br>pour la compagnie ',p_comp)<br>AS 'Resultat procException1';<br>ELSE<br>SELECT CONCAT('Le seul pilote de la compagnie Affichage du résultat.<br>',p_comp,' est ',var1)<br>AS 'Resultat procException1';<br>END IF;<br>END; | Déclaration de la procédure et des variables.  |
|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Bloc qui déclare les deux exceptions.          |
|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Requête pouvant déclencher l'exception prévue. |
|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Test de gestion des erreurs.                   |
|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        | Fin de la procédure.                           |

## Gestion des autres erreurs (SQLEXCEPTION)

Si une erreur non prévue en tant qu'exception (dans les clauses `DECLARE HANDLER`) produisait, le programme se terminerait anormalement en renvoyant l'erreur en question. La directive `SQLEXCEPTION` couvre toutes les erreurs qui ne sont administrées ni par `SQLWARNING` ni par `NOT FOUND`.

Dans notre exemple (sélection d'une colonne d'une table), seule une erreur « interne » pourrait éventuellement se produire (base de données ou table inexistante, colonne de la table inexistante ou d'un type différent de `VARCHAR(4)` ou autre erreur système).



MySQL ne permet pas, pour l'instant, de récupérer dynamiquement, au sein d'un sous-programme, le code et le message de l'erreur associée à une exception levée suite à une instruction SQL, et qui n'a pas été prévue dans un *handler*.

En d'autres termes, il faudra soit :

- interrompre brusquement votre procédure avec différents cas d'erreurs, pour lister les codes erreur générés en sortie ;
- vous contenter de gérer globalement les autres exceptions, tout en ne sachant pas de quelles erreurs il s'agit.

Le tableau suivant décrit la précédente procédure qui contrôle en plus toutes les autres erreurs non prévues, en appelant le sous-programme `autreErreur()`.

Tableau 7-13 Exceptions toutes traitées avec EXIT et SQLEXCEPTION

| Code MySQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Commentaires                                                                                                                                                                                                                                                                                        |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Web</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                     |
| <pre>CREATE PROCEDURE bdsoutou.tropdeLignes     (IN p_comp VARCHAR(4)) BEGIN     SELECT CONCAT('Il y a plusieurs pilotes pour                   la compagnie ',p_comp)               AS 'Resultat tropdeLignes'; END;</pre>                                                                                                                                                                                                                                                                                                                                                                   | Codage des sous-programmes appelés lors des exceptions.                                                                                                                                                                                                                                             |
| <pre>CREATE PROCEDURE bdsoutou.pasTrouve     (IN p_comp VARCHAR(4)) BEGIN     SELECT CONCAT('Il n''y a pas de pilote pour                   la compagnie ',p_comp)               AS 'Resultat pasTrouve'; END;</pre>                                                                                                                                                                                                                                                                                                                                                                          |                                                                                                                                                                                                                                                                                                     |
| <pre>CREATE PROCEDURE bdsoutou.autreErreur() BEGIN     SELECT 'Erreur mais laquelle?'               AS 'Resultat autreErreur'; END;</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                     |
| <pre>CREATE PROCEDURE bdsoutou.procException1     (IN p_comp VARCHAR(4)) BEGIN     DECLARE var1 VARCHAR(20);     DECLARE EXIT HANDLER FOR 1172         CALL bdsoutou.tropdeLignes(p_comp);     DECLARE EXIT HANDLER FOR NOT FOUND         CALL bdsoutou.pasTrouve(p_comp);     DECLARE EXIT HANDLER FOR SQLEXCEPTION         <b>CALL</b> bdsoutou.autreErreur();     SELECT nom INTO var1 FROM bdsoutou.Pilote         WHERE comp = p_comp;     SELECT CONCAT('Le seul pilote de la compagnie',                   p_comp,' est ',var1)               AS 'Resultat procException1'; END;</pre> |  <p>3 cas</p> <p>Procédure principale qui déclare les deux exceptions et toutes les autres.</p> <p>Requête pouvant déclencher l'exception prévue.</p> <p>Affichage du résultat.</p> <p>Fin de la procédure.</p> |

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 en paramètre. Le sous-programme se termine correctement si les deux requêtes ne retournent qu'un seul enregistrement. Les autres erreurs potentielles ne sont pas prises en compte.

Le principe est d'utiliser une variable indiquant quelle est la requête qui a fait sortir du programme 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 7-14 Une exception NOT FOUND traitée pour deux instructions

| Code MySQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Commentaires                                                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> CREATE PROCEDURE bdsoutou.procException2     (IN p_brevet VARCHAR(6), IN p_heures DECIMAL(7,2)) BEGIN     DECLARE v_nom          VARCHAR(20);     DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;     DECLARE v_requete      TINYINT;      BEGIN         DECLARE EXIT HANDLER FOR NOT FOUND SET flagNOTFOUND :=1;         SET v_requete := 1;         SELECT nom INTO v_nom FROM bdsoutou.Pilote                     WHERE brevet = p_brevet;         SELECT CONCAT('Le pilote de code ',p_brevet,                       ' est ',v_nom);         SET v_requete := 2;         SELECT nom INTO v_nom FROM bdsoutou.Pilote                     WHERE nbHVol = p_heures;         SELECT CONCAT('Le pilote ayant ',p_heures,                       ' heures de vol est ',v_nom);     END;      IF flagNOTFOUND THEN         IF v_requete = 1 THEN             SELECT CONCAT('Pas de pilote de brevet : ',p_brevet);         ELSEIF v_requete = 2 THEN             SELECT CONCAT('Pas de pilote ayant ce nombre d''heures de vol : ',p_heures);         END IF;     END IF; END; </pre> | <p>Bloc avec les requêtes déclenchant potentiellement une exception prévue.</p> <p>Traitement pour savoir quelle requête a provoqué l'exception.</p> |

Dans cette procédure, une erreur sur la première requête interrompt le programme (après avoir traité l'exception), et de ce fait la deuxième requête n'est pas évaluée. Pour cela, il est intéressant d'utiliser des blocs imbriqués pour poursuivre le traitement après avoir traité une ou plusieurs exceptions.

### Gestion de plusieurs exceptions

Le tableau suivant décrit une procédure qui traite toutes les mêmes exceptions en séquence. Ce mécanisme permet de continuer l'exécution après que MySQL a levé une exception. Dans cette procédure, les deux requêtes sont évaluées indépendamment du résultat retourné par chacune d'elles.

Tableau 7-15 Exceptions NOT FOUND toutes traitées

| Web | Code MySQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Commentaires                                            |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
|     | <pre> CREATE PROCEDURE bdsoutou.procException3     (IN p_brevet VARCHAR(6), IN p_heures DECIMAL(7,2)) BEGIN     DECLARE v_nom          VARCHAR(20);     DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;     DECLARE CONTINUE HANDLER FOR NOT FOUND         SET flagNOTFOUND := 1;     SELECT nom INTO v_nom FROM bdsoutou.Pilote         WHERE brevet = p_brevet;     IF flagNOTFOUND THEN         SELECT CONCAT('Pas de pilote de brevet : ',p_brevet); Gestion de l'exception         SET flagNOTFOUND := 0;     ELSE         SELECT CONCAT('Le pilote de code ',p_brevet,                       ' est ',v_nom);     END IF;     SELECT nm INTO v_nom FROM bdsoutou.Pilote         WHERE nbHVol = p_heures;     IF flagNOTFOUND THEN         SELECT CONCAT('Pas de pilote ayant ce nombre d''heures de vol : ',p_heures); Gestion de l'exception         SET flagNOTFOUND := 0;     ELSE         SELECT CONCAT('Le pilote ayant ',p_heures,                       ' heures de vol est ',v_nom);     END IF; END;</pre> | <p>la première requête.</p> <p>la deuxième requête.</p> |

L'exécution suivante de cette procédure déclenche les deux exceptions (ce qui n'était pas dans la procédure précédente).

```

CALL bdsoutou.procException3('PL-0', 500)$
+-----+
| CONCAT('Pas de pilote de brevet : ',p_brevet) |
+-----+
| Pas de pilote de brevet : PL-0 |
+-----+
+-----+
| CONCAT('Pas de pilote ayant ce nombre d''heures de vol : ',p_heures) |
+-----+
| Pas de pilote ayant ce nombre d'heures de vol : 500 |
+-----+
```

Il resterait à programmer l'exception ERROR 1172 (Result consisted of more than one row) pour gérer, au niveau de la seconde requête, l'extraction de plusieurs pilotes.

ayant un même nombre d'heures de vol. Deux solutions s'offrent à vous : dans un sous-bloc avec EXIT ou dans le même bloc avec un CONTINUE, et une variable pour tester l'éventuelle redirection.



Afin de déclarer d'autres exceptions, il faut consulter la liste des erreurs dans la documentation officielle (*Appendix B. Error Codes and Messages*) de manière à connaître le numéro d'erreur MySQL (paramètre code\_error\_mysql dans la syntaxe DECLARE HANDLER).

Vous pouvez aussi écrire un bloc qui programme volontairement l'erreur pour voir, sous l'interface de commande, le numéro que MySQL renvoie.

## Exceptions nommées

Pour intercepter une erreur MySQL et lui attribuer au passage un identificateur, il faut utiliser la clause DECLARE CONDITION. La syntaxe est la suivante :

### Déclaration

```
DECLARE nomException CONDITION FOR
 {SQLSTATE [VALUE] 'valeur_sqlstate' | code_error_mysql}
```

Il est ainsi possible de regrouper plusieurs types d'erreurs (avec SQLSTATE ou cibler une erreur en particulier en indiquant le code erreur de MySQL). Une fois l'exception nommée, il est possible de l'utiliser dans la déclaration de l'événement associé via la directive DECLARE HANDLER.

### Déclenchement

Considérons les deux tables suivantes. La colonne comp de la table Pilote est une clé étrangère vers la table Compagnie. Programmons une procédure qui supprime une compagnie par code passé en paramètre.

Figure 7-5 Deux tables

| Compagnie  |           |              | Pilote |                     |        |      |
|------------|-----------|--------------|--------|---------------------|--------|------|
| comp       | ville     | nomComp      | brevet | nom                 | nbHVol | comp |
| AF         | Paris     | Air France   | PL-1   | Gilles Laborde      | 2450   | AF   |
| SING       | Singapour | Singapore AL | PL-2   | Frédéric D'Almeysda | 900    | AF   |
| CAST       | Blagnac   | Castanet AL  | PL-3   | Florence Périsse    | 1000   | SING |
| EJET       | Dublin    | Easy Jet     | PL-4   | Thierry Millan      | 2450   | CAST |
| à détruire |           |              | PL-5   | Christine Royo      | 200    | AF   |
|            |           |              | PL-6   | Aurélia Ente        | 2450   | SING |

Le tableau suivant décrit la procédure procExceptionNommee qui intercepte une erreur référentielle (SQLSTATE à 23 000). Il s'agit de contrôler le programme si la compagnie détruire est encore rattachée à un enregistrement référencé dans la table Pilote.

Tableau 7-16 Programmation d'une exception nommée

| Web | Code MySQL                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Commentaires                                                                                                                  |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
|     | <pre> CREATE PROCEDURE bdsoutou.procExceptionNommee     (IN p_comp VARCHAR(4)) BEGIN     DECLARE flagerr BOOLEAN DEFAULT 0;     BEGIN         DECLARE erreur_ilResteUnPilote CONDITION             FOR SQLSTATE '23000';         DECLARE EXIT HANDLER             FOR erreur_ilResteUnPilote SET flagerr :=1;         SET AUTOCOMMIT=0;         DELETE FROM Compagnie WHERE comp = p_comp;         SELECT CONCAT('Compagnie ',p_comp, ' détruite')             AS 'Resultat procExceptionNommee';     END;     IF flagerr THEN         ROLLBACK;         SELECT CONCAT('Désolé, il reste encore un pilote à             la compagnie ',p_comp)             AS 'Resultat procExceptionNommee';     ELSE         COMMIT;     END IF; END; </pre> | <p>Déclaration de l'exception nommée.</p> <p>Corps du traitement (vérification et action).</p> <p>Gestion de l'exception.</p> |

Les traces de l'exécution de cette procédure sont les suivantes. Notez que si on appelle la procédure en passant en paramètre une compagnie inexisteante, le sous-programme se termine normalement sans passer dans la section d'erreur.

```

CALL bdsoutou.procExceptionNommee('AF')$+
-----+
| Resultat procExceptionNommee |
-----+
| Désolé, il reste encore un pilote à la compagnie AF |
-----+
CALL bdsoutou.procExceptionNommee('EJET')$+
-----+
| Resultat procExceptionNommee |
-----+
| Compagnie EJET détruite |
-----+

```

Concernant MySQL, les déclencheurs n'existent que depuis la version 5. Comme nous verrons, ils sont limitatifs en termes de fonctionnalités et relativement instables (P. Gulutzan citait, en parlant de la version bêta dans son livre blanc toujours présent (au moment de l'impression de ce livre) en page d'accueil du site de MySQL, *MySQL 5.0 Triggers* : « *Triggers are very new. There are bugs. ... Do not try triggers with a database that has important data in it...* »).

Bien que beaucoup d'améliorations aient été apportées, les déclencheurs qui modifient les tables ne sont pas encore à mon sens tout à fait fiables. Prudence donc avec vos données ! Toutes les limitations que nous allons détailler seront sans doute résolues au fur et à mesure des prochaines versions majeures du serveur. Songez qu'avant la version 5.0.10, les déclencheurs ne pouvaient même pas accéder à la base !

## Généralités

D'un point de vue général et sans parler de MySQL, la plupart des déclencheurs (*triggers*) peuvent être vus comme des sous-programmes résidents associés à un événement particulier (insertion, modification d'une ou de plusieurs colonnes, suppression) sur une table (ou une vue). Une table (ou vue) peut « héberger » plusieurs déclencheurs ou aucun. Pour certains SGBD, il existe d'autres types de déclencheurs que ceux associés à une table (ou vue) afin de répondre à des événements qui ne concernent pas les données (exemple : connexion d'un utilisateur particulier, suppression d'une table, démarrage ou arrêt du serveur, déconnexion d'un utilisateur, etc.).

À la différence des sous-programmes, l'exécution d'un déclencheur n'est pas explicite (par `CALL` par exemple), c'est l'événement de mise à jour de la table qui lance automatiquement le code programmé dans le déclencheur. On dit que le déclencheur « se déclenche » (l'anglais traduit mieux : *fired trigger*).

## À quoi sert un déclencheur ?

En théorie, un déclencheur permet de :

- Programmer toutes les règles de gestion qui n'ont pas pu être mises en place par des contraintes au niveau des tables. Par exemple, la condition : *une compagnie ne fait voler un pilote que s'il a totalisé plus de 60 heures de vol dans les 2 derniers mois, sur le type d'appareil du vol en question*, ne pourra pas être programmée par une contrainte et nécessitera l'utilisation d'un déclencheur.
- Déporter des contraintes au niveau du serveur et alléger ainsi la programmation client.
- Renforcer des aspects de sécurité et d'audit.
- Programmer l'intégrité référentielle et la réPLICATION dans des architectures distribuées avec l'utilisation de liens de bases de données.

En théorie, les événements déclencheurs peuvent être :

- une instruction **INSERT**, **UPDATE**, ou **DELETE** sur une table (ou vue). On parle de déclencheurs LMD ;
- une instruction **CREATE**, **ALTER**, ou **DROP** sur un objet (table, vue, index, etc.). On parle de déclencheurs LDD ;
- le démarrage ou l'arrêt de la base (*startup* ou *shutdown*), une erreur spécifique (*not found*, *duplicate key*, etc.), une connexion ou une déconnexion d'un utilisateur. On parle de déclencheurs d'instances.

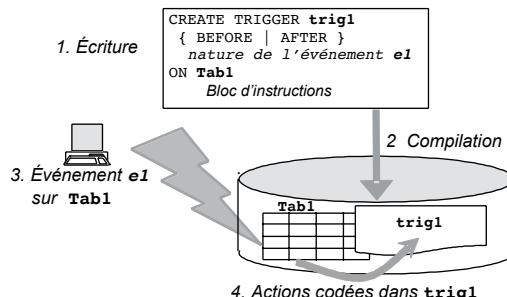
En pratique, dans MySQL, seuls les événements de la première catégorie sont pris en compte.

## Mécanisme général

Auparavant considérés au niveau *table*, les déclencheurs sont désormais attachés, à plus titre, au niveau *database*. En conséquence, le nom d'un déclencheur doit être unique dans chaque base de données. En revanche, différentes bases peuvent héberger un déclencheur avec le même nom.

La figure suivante illustre les quatre étapes à suivre pour mettre complètement en œuvre un déclencheur (de la création à son test). Il faut d'abord le coder (comme un sous-programme), puis le compiler (il est pour l'instant stocké indépendamment de la table, mais il devrait y être inclus dans les prochaines versions). Par la suite, au cours du temps, chaque événement qui caractérise le déclencheur aura pour conséquence son exécution.

**Figure 7-6** Mécanisme des déclencheurs



## Syntaxe

Pour pouvoir créer un déclencheur, vous devez disposer du privilège **SUPER** (le privilège **CREATE TRIGGER** est à venir). Un déclencheur est composé de deux parties : la description de l'événement traqué et celle de l'action à réaliser lorsque l'événement se produit. La syntaxe de création d'un déclencheur est la suivante :

```

CREATE TRIGGER nomDéclencheur
 { BEFORE | AFTER } { DELETE | INSERT | UPDATE }
 ON nomTable
 FOR EACH ROW
 { instruction; /
 [etiquette:] BEGIN
 instructions;
 END [etiquette]; }

```

Les options de cette commande sont les suivantes :

- BEFORE | AFTER précise la chronologie entre l'action à réaliser par le déclencheur LMD la réalisation de l'événement (BEFORE INSERT exécutera le déclencheur avant de réaliser l'insertion).
- DELETE | INSERT | UPDATE précise la nature de l'événement pour les déclencheurs LMD
  - Pour DELETE, le déclencheur examine les événements DELETE et REPLACE.
  - Pour INSERT, le déclencheur prend en compte les événements suivants : INSERT... CREATE... SELECT, LOAD DATA, et REPLACE.
  - Pour UPDATE, le déclencheur considère seulement l'événement UPDATE.
- ON nomTable spécifie la table associée au déclencheur LMD.
- FOR EACH ROW différencie les déclencheurs LMD au niveau ligne (le niveau état n'est pas encore pris en charge).
- *instruction* ou *instructions* compose le corps du code du déclencheur.



Il n'est pas possible de définir deux déclencheurs distincts sur le même événement d'une table donnée. En revanche, il est possible d'avoir deux déclencheurs distincts sur la même action dans une table donnée (un déclencheur pour BEFORE UPDATE et un autre pour AFTER UPDATE par exemple).

Attention à ne pas créer de déclencheurs recursifs (exemple d'un déclencheur qui exécute une instruction lançant elle-même le déclencheur, ou deux déclencheurs s'appelant en cascade jusqu'à l'occupation de toute la mémoire réservée).

Étudions à présent plus précisément les caractéristiques du seul type de déclencheur qu'il est actuellement possible de programmer.

## Déclencheurs LMD (de lignes)

Pour ce type de déclencheur, l'événement à déterminer est une mise à jour particulière de base (ajout, modification ou suppression dans une table ou une vue).

L'exécution est, pour l'heure, dépendante du nombre de lignes touchées par l'événement. Seuls les déclencheurs de lignes (*row trigger*) sont permis, car la directive FOR EACH est obligatoire. Ils sont pratiques quand on désire utiliser autant de fois le déclencheur qu'il y a de lignes concernées par une mise à jour.

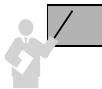


Si on désirait exécuter une seule fois le déclencheur, quel que soit le nombre de lignes concernées, il faudrait employer un déclencheur d'état (*statement trigger*) qui n'est pas encore reconnu. Pour ce faire, il faudrait ne pas indiquer la directive FOR EACH ROW, ou préciser EACH STATEMENT à la place (à confirmer avec une prochaine version incluant cette extension).

Dans l'exemple d'une table *t1* ayant cinq enregistrements, si on programme un déclencheur au niveau ligne avec l'événement AFTER DELETE, et qu'on lance DELETE FROM *t1*, le déclencheur effectuera cinq fois ses instructions (une fois après chaque suppression). Le tableau suivant explique ce mécanisme :

Tableau 7-17 Exécution d'un déclencheur LMD

| Nature de l'événement | État ( <i>statement trigger</i> ) sans FOR EACH ROW | Ligne ( <i>row trigger</i> ) avec FOR EACH ROW |
|-----------------------|-----------------------------------------------------|------------------------------------------------|
| BEFORE                | Exécution une fois avant la mise à jour.            | Exécution avant chaque ligne mise à jour.      |
| AFTER                 | Exécution une fois après la mise à jour.            | Exécution après chaque ligne mise à jour.      |



Seuls les déclencheurs de lignes peuvent accéder aux anciennes et aux nouvelles valeurs des colonnes de la ligne affectée par la mise à jour prévue par l'événement. Les identificateurs OLD et NEW sont programmés pour cela. Ce sont des extensions de MySQL à la norme SQL (MySQL suit ainsi Oracle dans ce domaine qui propose :OLD et :NEW).

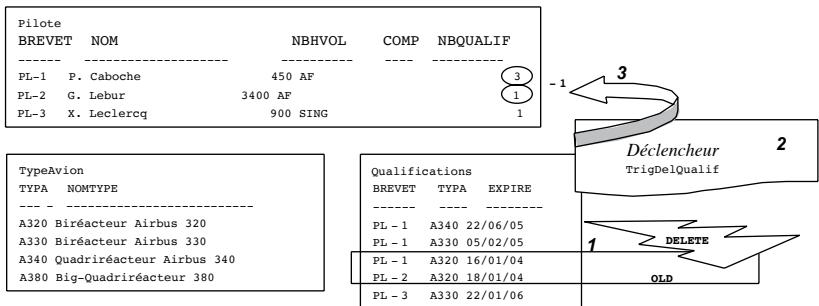
Un déclencheur de type AFTER ne se lance sur une table *t* que si le déclencheur de type BEFORE de la table *t* (s'il existe) et que l'instruction associée à l'événement se sont correctement déroulés.

### Quand utiliser la directive OLD ?

Chaque enregistrement qui tente d'être supprimé d'une table, qui inclut un déclencheur de type DELETE FOR EACH ROW, est désigné par OLD au niveau du code du déclencheur. L'accès aux colonnes de ce pseudo-enregistrement dans le corps du déclencheur se fait par notation pointée.

Considérons l'exemple suivant, et programmons la règle de gestion *tout pilote qui perd une qualification doit voir son compteur automatiquement décrémenter*. Programmons le déclencheur TrigDelQualif qui surveille les suppressions de la table Qualifications, et qui diminue de 1 la colonne nbQualif pour le pilote concerné par la suppression de sa qualification.

Figure 7-7 Principe du déclencheur TrigDelQualif



L'événement déclencheur est ici AFTER DELETE, car il faudra s'assurer que la suppression n'est pas entravée par d'éventuelles contraintes référentielles. On utilise un déclencheur FOR EACH ROW, car s'il se produit une suppression de toute la table (DELETE FROM Qualifications ;), on exécutera autant de fois le déclencheur qu'il y a de lignes détruites.

Chaque enregistrement qui va être supprimé de la table Qualifications est désigné par OLD au niveau du code du déclencheur. L'accès aux colonnes de ce pseudo-enregistrement dans le corps du déclencheur se fait par la notation pointée.

Le code minimal de ce déclencheur (on ne prend pas en compte le fait qu'il n'existe pas de pilote de ce brevet) est décrit dans le tableau suivant :

Tableau 7-18 Déclencheur après suppression

| Code MySQL                                                                                                                                                                 | Commentaires                                                                                                           |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------|
| <pre>CREATE TRIGGER TriDelQualif AFTER DELETE ON Qualifications FOR EACH ROW BEGIN     UPDATE Pilote SET nbQualif = nbQualif - 1     WHERE brevet = OLD.brevet; END;</pre> | Déclaration de l'événement déclencheur.<br>Corps du déclencheur.<br>Mise à jour du pilote concerné par la suppression. |

En considérant les données initiales des tables, le test de ce déclencheur sous l'interface commande est le suivant. Par ailleurs, la table Qualifications ne contient plus que trois enregistrements.

Tableau 7-19 Test du déclencheur AFTER DELETE

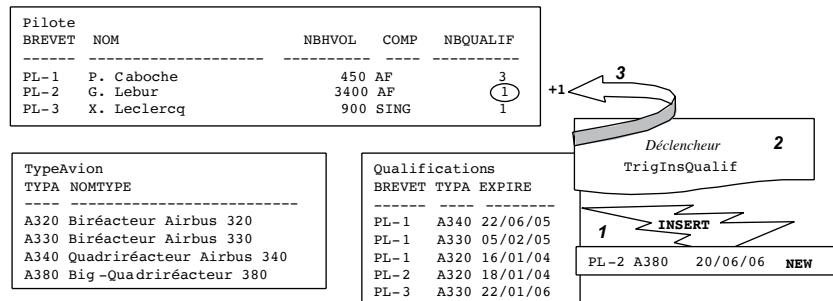
| Événement déclencheur                                            | Résultat                                                                                                                                                                                                                                                                                               |
|------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>DELETE FROM Qualifications<br/>WHERE typa = 'A320';</code> | <pre>SELECT * FROM Pilote;</pre> <pre>+-----+-----+-----+-----+   brevet   nom        nbHVol   compa   nbQualif   +-----+-----+-----+-----+   PL-1     P. Caboche   450.00   AF        PL-2     G. Lebur     3400.00  AF        PL-3     X. Leclercq  900.00   SING    +-----+-----+-----+-----+</pre> |

### Quand utiliser la directive NEW ?

Chaque enregistrement qui tente d'être ajouté dans la table Qualifications est déclenché par NEW au niveau du code du déclencheur. L'accès aux colonnes de ce pseudo-enregistrement dans le corps du déclencheur se fait par la notation pointée.

Considérons le même exemple et écrivons la règle de gestion *tout pilote qui gagne une qualification doit voir son compteur automatiquement incrémenter*. Programmons le déclencheur TrigInsQualif qui surveille les insertions sur la table Qualifications et augmente d'un 1 la colonne nbQualif pour le pilote concerné.

Figure 7-8 Principe du déclencheur TrigInsQualif



L'événement déclencheur est ici AFTER INSERT, car il faudra s'assurer, avant de faire l'insertion, que le code du pilote et celui de l'avion sont corrects (existant dans les tables « père »). On utilise un déclencheur FOR EACH ROW car on désire qu'il s'exécute plusieurs fois qu'il y a de lignes concernées par l'événement déclencheur.

Le code minimal de ce déclencheur (on ne prend en compte aucune erreur potentiellement décrite dans le tableau suivant :

Tableau 7-20 Déclencheur après insertion

| Déclencheur                                                                                                                                                                   | Commentaires                                                                                                                            |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------|
| <pre> CREATE TRIGGER TrigInsQualif AFTER INSERT ON Qualifications FOR EACH ROW BEGIN     UPDATE Pilote SET nbQualif = nbQualif + 1     WHERE brevet = NEW.brevet; END; </pre> | <p>Déclaration de l'événement déclencheur.</p> <p>Corps du déclencheur.</p> <p>Mise à jour du pilote concerné par la qualification.</p> |

En considérant les données initiales des tables, le test de ce déclencheur (réalisé 20 décembre 2005) sous l'interface de commande est le suivant :

Tableau 7-21 Test du déclencheur AFTER INSERT

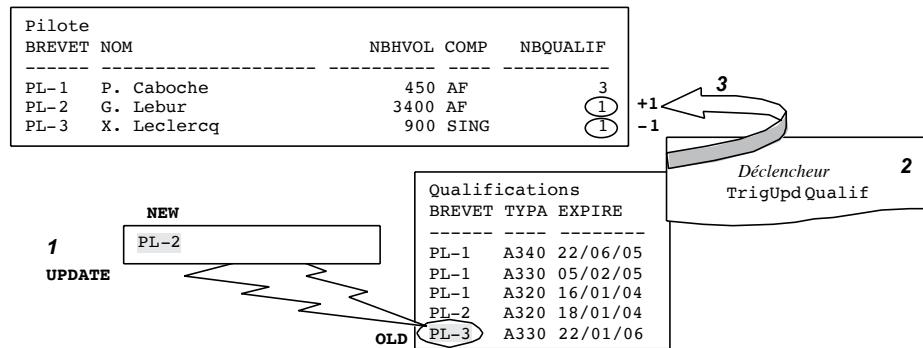
| Web | Événement déclencheur                                                       | Résultat                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-----|-----------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre> INSERT INTO Qualifications VALUES ('PL-2', 'A380', SYSDATE()); </pre> | <pre> SELECT * FROM Qualifications \$  +-----+-----+-----+   brevet   typa   expire     +-----+-----+-----+   PL-1    A340   2005-06-22     PL-1    A330   2005-02-05     PL-1    A320   2004-01-16     PL-2    A320   2004-01-18     PL-3    A330   2006-01-22     PL-2    A380   2005-12-20   +-----+-----+-----+ SELECT * FROM Pilote\$  +-----+-----+-----+-----+   brevet   nom        nbHVol   compa   nbQualif   +-----+-----+-----+-----+   PL-1    P. Caboche   450.00   AF      3            PL-2    G. Lebur     3400.00   AF      2            PL-3    X. Leclercq   900.00   SING    1          +-----+-----+-----+-----+ </pre> |

### Quand utiliser à la fois les directives NEW et OLD ?

Seuls les déclencheurs de type UPDATE FOR EACH ROW permettent de manipuler à la fois les directives NEW et OLD. En effet, la mise à jour d'une ligne dans une table fait intervenir une nouvelle donnée qui en remplace une ancienne. L'accès aux anciennes valeurs se fera par la notation pointée du pseudo-enregistrement OLD. L'accès aux nouvelles valeurs se fera par NEW.

La figure suivante illustre ce mécanisme dans le cas de la modification de la colonne brevet du dernier enregistrement de la table Qualifications. Le déclencheur doit programmer deux mises à jour dans la table Pilote.

Figure 7-9 Principe du déclencheur TrigUpdQualif



L'événement déclencheur est ici AFTER UPDATE, car il faudra s'assurer que la suppression n'est pas entravée par d'éventuelles contraintes référentielles. Le code minimal de ce déclencheur est décrit dans le tableau suivant :

Tableau 7-22 Déclencheur après modification

| Web | Déclencheur                                                                                                                                                                                                                                                                        | Commentaires                                                                                                                                                |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre> CREATE TRIGGER TrigUpdQualif AFTER UPDATE ON Qualifications FOR EACH ROW BEGIN     UPDATE Pilote         SET nbQualif = nbQualif + 1         WHERE brevet = NEW.brevet;     UPDATE Pilote         SET nbQualif = nbQualif - 1         WHERE brevet = OLD.brevet; END; </pre> | <p>Déclaration de l'événement déclenché.</p> <p>Corps du déclencheur.</p> <p>Mise à jour des pilotes concernés par la modification de la qualification.</p> |

En considérant les données présentées à la figure précédente, le test de ce déclencheur suivant :

Tableau 7-23 Test du déclencheur AFTER UPDATE

| Événement déclencheur                                                                                        | Résultat                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>UPDATE Qualifications<br/>SET brevet = 'PL-2'<br/>WHERE brevet = 'PL-3'<br/>AND typa = 'A330'\$</code> | <pre>SELECT * FROM Pilote\$</pre> <pre>+-----+-----+-----+-----+   brevet   nom        nbHVol   compa     nbQualif   +-----+-----+-----+-----+   PL-1    P. Caboche   450.00   AF                 3     PL-2    G. Lebur     3400.00   AF                 2     PL-3    X. Leclercq   900.00   SING              0   +-----+-----+-----+-----+</pre> <pre>SELECT * FROM Qualifications\$</pre> <pre>+-----+-----+-----+   brevet   typa      expire    +-----+-----+-----+   PL-1    A340     2005-06-22     PL-1    A330     2005-02-05     PL-1    A320     2004-01-16     PL-2    A320     2004-01-18     PL-2    A330     2006-01-22   +-----+-----+-----+</pre> |

### Bilan de NEW et OLD

Le tableau suivant résume les valeurs contenues dans les pseudo-enregistrements OLD et NEW pour les déclencheurs FOR EACH ROW. Retenez que seuls les déclencheurs UPDATE peuvent manipuler à bon escient les deux types de directives.

Tableau 7-24 Valeurs de OLD et NEW

| Nature de l'événement | OLD.colonne      | NEW.colonne      |
|-----------------------|------------------|------------------|
| INSERT                | Impossible.      | Nouvelle valeur. |
| UPDATE                | Ancienne valeur. | Nouvelle valeur. |
| DELETE                | Ancienne valeur. | Impossible.      |

MySQL prévient clairement, à la compilation, que vous utilisez une variable OLD dans un déclencheur INSERT – ou NEW dans un déclencheur DELETE – par deux messages de même code, mais de libellés différents, suivant les cas :

- ERROR 1363 (HY000): There is no NEW row in on DELETE trigger
- ERROR 1363 (HY000): There is no OLD row in on INSERT trigger

Une colonne prefixée de OLD est en lecture seule dans le corps d'un déclencheur.

Une colonne prefixée de NEW ne peut être accessible qu'à l'aide du privilège SELECT associé.

## Appel de sous-programmes

Un déclencheur peut appeler directement par `CALL` (ou dans son corps) un sous-programme MySQL.



- Un déclencheur ne peut pas lancer une procédure cataloguée retournant des données client (traces avec `SELECT`) ou utilisant du SQL dynamique (étudié plus loin). En revanche les procédures employées peuvent renvoyer des résultats via leurs paramètres de sortie (`OUT`).
- Les procédures appelées ne peuvent constituer aucune transaction (oubliez donc de faire faire `START TRANSACTION`, `COMMIT`, et `ROLLBACK`).

Le tableau suivant décrit l'utilisation d'un sous-programme (`procTrigg`) dans un déclencheur (`espionAjoutPilote`) qui s'exécutera avant chaque ajout d'un nouveau pilote. Le sous-programme ajoute simplement une ligne dans la table `Trace`.

```
CREATE PROCEDURE bdsoutou.procTrigg(IN param DATETIME)
BEGIN
 INSERT INTO Trace VALUES
 (CONCAT('Insertion pilote, appel de bdsoutou.procTrigg le ',param))
END;
```

Tableau 7-25 Appel d'un sous-programme dans un déclencheur

| Web | Déclencheur                                                                                                                                          | Commentaire                                                                                |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
|     | <pre>CREATE TRIGGER bdsoutou.espionAjoutPilote     BEFORE INSERT ON Pilote     FOR EACH ROW BEGIN     CALL bdsoutou.procTrigg(SYSDATE()); END;</pre> | Appel dans le corps du déclencheur d'une procédure MySQL en passant un paramètre d'entrée. |

La trace d'exécution en considérant les données initiales des tables est la suivante :

```
mysql> (INSERT INTO Pilote VALUES ('PL-4', 'C. Soutou', 100, 'AF')
Query OK, 1 row affected (0.04 sec)
mysql> SELECT * FROM Pilote$
+-----+-----+-----+-----+
| brevet | nom | nbHVol | compa | nbQualif |
+-----+-----+-----+-----+
PL-1	P. Caboche	450.00	AF	3
PL-2	G. Lebur	3400.00	AF	1
PL-3	X. Leclercq	900.00	SING	1
PL-4	C. Soutou	100.00	AF	0
+-----+-----+-----+-----+
```

```

mysql> SELECT * FROM Trace$
```

+-----  
| col  
+-----  
| Insertion pilote, appel de bdsoutou.procTrigg le 2005-12-20 08:27:20  
+-----

## Dictionnaire des données

Étudiée au chapitre 5, la base INFORMATION\_SCHEMA inclut la vue TRIGGERS qui renseigne les caractéristiques des déclencheurs qui étaient auparavant considérés au niveau *table*. Ils sont désormais reconnus à juste titre au niveau *database*. Il faut détenir le privilège SUPER pour accéder à cette vue.

La requête suivante interroge cette vue et permet de retrouver les noms et les caractéristiques relatives aux événements déclencheurs des trois *triggers* de la base de données bdsoutou.

```

SELECT TRIGGER_NAME,EVENT_OBJECT_TABLE "Table",
 EVENT_MANIPULATION "Evenement", ACTION_TIMING,
 EVENT_OBJECT_SCHEMA "Base"
 FROM INFORMATION_SCHEMA.TRIGGERS
 WHERE TRIGGER_SCHEMA='bdsoutou';
```

| TRIGGER_NAME  | Table          | Evenement | ACTION_TIMING | Base     |
|---------------|----------------|-----------|---------------|----------|
| TrigInsQualif | Qualifications | INSERT    | AFTER         | bdsoutou |
| TrigUpdQualif | Qualifications | UPDATE    | AFTER         | bdsoutou |
| TriDelQualif  | Qualifications | DELETE    | AFTER         | bdsoutou |

Notez que MySQL utilise :

- la colonne TRIGGER\_NAME pour désigner le nom du déclencheur d'une *database* ;
- la colonne TRIGGER\_SCHEMA pour désigner le nom de la base de données à laquelle appartient ;
- les colonnes EVENT\_OBJECT\_TABLE et EVENT\_OBJECT\_SCHEMA pour désigner respectivement le nom de la table qui accueille ce déclencheur ainsi que la base de données qui la contient (elle peut être différente de celle du déclencheur, ici nous raisonnons sur même) ;
- la colonne EVENT\_MANIPULATION pour désigner l'événement déclencheur ;
- la colonne ACTION\_TIMING pour préciser la chronologie de l'événement déclencheur.

La requête suivante interroge cette même vue pour extraire le code du déclencheur de type *after update* hébergé par la table Qualifications, dans la base de données bdsoutou :

```

SELECT ACTION_STATEMENT FROM INFORMATION_SCHEMA.TRIGGERS
 WHERE TRIGGER_SCHEMA='bdsoutou' AND EVENT_OBJECT_TABLE='Qualification'
 AND EVENT_MANIPULATION='UPDATE' AND ACTION_TIMING='AFTER';
+
+-----+
| ACTION_STATEMENT
+-----+
| BEGIN
| UPDATE Pilote SET nbQualif = nbQualif + 1 WHERE brevet = NEW.brevet;
| UPDATE Pilote SET nbQualif = nbQualif - 1 WHERE brevet = OLD.brevet;
| END
+-----+

```

Remarquons que MySQL utilise la colonne ACTION\_STATEMENT pour contenir le corps du déclencheur (visible aussi par SHOW TRIGGERS).



La colonne ACTION\_ORIENTATION est pour l'instant toujours évaluée à 'ROW' (déclencheur d'état pas encore opérationnel).

Les colonnes ACTION\_REFERENCE\_OLD\_ROW et ACTION\_REFERENCE\_NEW\_ROW contiennent pour l'instant, toujours respectivement 'OLD' et 'NEW' (il n'est pas encore possible de renommer ces identificateurs).

Dans le but d'être davantage en phase avec la norme dans les prochaines versions, les colonnes suivantes contiennent, pour l'heure, toujours la valeur NULL : TRIGGER\_CATALOG, EVENT\_OBJECT\_CATALOG, ACTION\_CONDITION, ACTION\_REFERENCE\_OLD\_TABLE, ACTION\_REFERENCE\_NEW\_TABLE, et CREATED. Les deux premières colonnes sont relatives à la table de catalogue, la suivante à la possibilité de conditionner un déclencheur (clause WHEN dans le code), la dernière contiendrait le moment de création du déclencheur.

## Programmation d'une contrainte de vérification

Nous avons vu que les contraintes de vérification (CHECK) ne sont pas encore prises en charge. Nous avons étudié au chapitre 5 la possibilité d'en programmer à l'aide de vues. Nous allons créer un déclencheur s'en chargeant. Attention, il n'est pas toujours possible d'utiliser un déclencheur pour valider une contrainte de vérification.

Considérons l'exemple du chapitre 5 (*Figure 5-10. Vue simulant la contrainte CHECK*) qui décrit la table Pilote et la contrainte vérifiant qu'un pilote :

- ne peut être commandant de bord qu'à la condition qu'il ait entre 1 000 et 4 000 heures de vol ;
- ne peut être copilote qu'à la condition qu'il ait entre 100 et 1 000 heures de vol ;
- ne peut être instructeur qu'à partir de 3 000 heures de vol.

Le tableau suivant décrit le code du déclencheur. Ici, on choisit de forcer la valeur de la colonne grade pour conserver la cohérence avec les conditions initiales.

| Déclencheur                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Commentaires                                                                                                                                                                                    |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>CREATE TRIGGER TrigInsGrade BEFORE INSERT ON Pilote FOR EACH ROW  BEGIN     IF (<b>NEW.grade</b> = 'CDB' AND (<b>NEW.nbHVol</b>&lt;1000)) THEN         SET <b>NEW.grade</b> := 'COPI';     END IF;     IF (<b>NEW.grade</b> = 'CDB' AND (<b>NEW.nbHVol</b>&gt;4000)) THEN         SET <b>NEW.grade</b> := 'INST';     END IF;     IF (<b>NEW.grade</b> = 'COPI' AND (<b>NEW.nbHVol</b>&gt;1000)) THEN         SET <b>NEW.grade</b> := 'CDB';     END IF;     IF (<b>NEW.grade</b> = 'INST' AND (<b>NEW.nbHVol</b>&lt;3000))         OR (<b>NEW.nbHVol</b>&lt;100) THEN         SET <b>NEW.grade</b> := NULL;     END IF; END;</pre> | <p>Déclaration de l'événement déclencheur.</p> <p>Corps du déclencheur.</p> <p>Test des conditions et mise à jour éventuelle de la nouvelle valeur à insérer au niveau de la colonne grade.</p> |

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                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <pre>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');  SELECT * FROM Pilote;</pre> | <pre>INSERT INTO Pilote VALUES ('PL-5','Trop jeune',100,'CDB'); INSERT INTO Pilote VALUES ('PL-6','Inexperimente',2999,'INST');</pre> |

| Insertions valides                                                                                                                                                                                                                                                                                  | Insertions non valides                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <pre>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');  SELECT * FROM Pilote;</pre> | <pre>INSERT INTO Pilote VALUES ('PL-5','Trop jeune',100,'CDB'); INSERT INTO Pilote VALUES ('PL-6','Inexperimente',2999,'INST');</pre> |

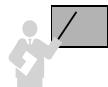
| Insertions valides                                                                                                                                                                                                                                                                                  | Insertions non valides                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <pre>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');  SELECT * FROM Pilote;</pre> | <pre>INSERT INTO Pilote VALUES ('PL-5','Trop jeune',100,'CDB'); INSERT INTO Pilote VALUES ('PL-6','Inexperimente',2999,'INST');</pre> |

## Invalidation dans le déclencheur

Une fonctionnalité importante des déclencheurs consiste à pouvoir invalider l'événement déclenché l'action. En d'autres termes, c'est pouvoir faire dire au déclencheur *non* à une insertion, une modification ou à une suppression. Ces cas concernent les déclencheurs de type BEFORE, puisqu'il faudra vérifier des conditions dans le corps du déclencheur et éventuellement d'accepter l'événement.

Dans un déclencheur de type BEFORE :

- Il est possible de modifier une colonne préfixée de NEW à la condition de détenir le privilège UPDATE associé. Cela signifie que l'on peut changer un enregistrement avant l'insérer.
- La valeur d'une colonne AUTO\_INCREMENT préfixée par NEW est 0 (et ne suit pas la séquence existante). La valeur actualisée de la séquence ne sera effective que lors de l'insertion.



Dans tout déclencheur (de type BEFORE ou AFTER), une erreur lors de l'exécution et toutes les instructions du bloc sont invalidées.

### Principe

Sans parler de MySQL, l'invalidation dans un déclencheur se traduit en général par le déclenchement d'une exception (qui fait avorter l'instruction LMD), et par le retour d'un message d'erreur personnalisé.



Les procédures et déclencheurs MySQL ne permettent pour l'instant ni de provoquer une exception système ni de retourner un code SQL personnalisé.

Il n'est pas non plus possible d'utiliser ROLLBACK dans un déclencheur (ERROR (HY000): Explicit or implicit commit is not allowed in stored function or trigger).

Une seule solution, qui n'est pas du tout satisfaisante, comme nous allons le voir, consiste à provoquer artificiellement une erreur (mais pas une erreur système, par exemple accéder à une table inexistante). Il faut une erreur sémantiquement correcte qui pose problème à l'exécution (NULL dans une clé primaire).



Si vous programmez une erreur système (SELECT d'une table inexistante) elle sera reçue dans tous les cas d'exécution du déclencheur !

L'inconvénient majeur de cette astuce est que le message d'erreur ne sera jamais explicitement fourni, il ne sera pas en rapport avec la contrainte qui n'est pas satisfaite dans le déclencheur.

## Exemple

Considérons à nouveau l'exemple précédent et programmons la contrainte que *tout pilote n'a peut être qualifié sur plus de trois types d'appareils*. Ici, il s'agit d'assurer la cohérence entre la valeur de la colonne nbQualif de la table Pilote et les enregistrements de la table Qualifications.

L'événement déclencheur est ici BEFORE INSERT, car il faudra s'assurer de la condition avant d'autoriser. Le code minimal de ce déclencheur est décrit dans le tableau suivant en supposant qu'on dispose d'une table de travail : CREATE TABLE Trace(column VARCHAR(80) PRIMARY KEY).

Tableau 7-28 Déclencheur avant insertion

| Web | Déclencheur                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Commentaires                                                                                                                                                                                                                                   |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>CREATE TRIGGER TrigInsQualif     BEFORE INSERT ON Qualifications     FOR EACH ROW BEGIN     DECLARE v_compteur TINYINT(1);     DECLARE v_nom VARCHAR(30);     SELECT nbQualif, nom INTO v_compteur, v_nom         FROM Pilote WHERE brevet = NEW.brevet;     IF v_compteur &lt; 3 THEN         UPDATE Pilote             SET nbQualif = nbQualif + 1             WHERE brevet = NEW.brevet;     ELSE         INSERT INTO TRACE VALUES             (CONCAT('Le pilote ',v_nom,                     ' a déjà 3 qualifications!'));         INSERT INTO TRACE VALUES (NULL);     END IF; END;</pre> | <p>Déclaration de l'événement déclencheur.</p> <p>Corps du déclencheur.</p> <p>Test de la condition.</p> <p>Mise à jour du pilote concerné par l'ajout de la qualification.</p> <p>Instruction jamais réalisée !</p> <p>Erreur volontaire.</p> |

En considérant les données initiales, le test de ce déclencheur est le suivant. On remarque qu'avec le premier INSERT est bien évité (le pilote PL-1 a déjà trois qualifications). Bizarremennt, l'ajout du message dans la table Trace n'est pas effectué, car le déclencheur invalide tout son traitement en cas d'erreur. La seconde insertion, en revanche, est bien effectuée (le pilote PL-3 n'a qu'une seule qualification).

Tableau 7-29 Test du déclencheur BEFORE INSERT

| Événement déclencheur                                                                           | Résultat                                                                                                                      |
|-------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| --ajout incorrect<br>INSERT INTO Qualifications<br>VALUES<br>( 'PL-1' , 'A380' , :SYSDATE() )\$ | ERROR 1048 (23000): Column 'col' cannot be null<br>-- ne fait pas l'INSERT dans Qualifications<br>-- ni dans Trace !          |
| -- ajout correct<br>INSERT INTO Qualifications<br>VALUES<br>( 'PL-3' , 'A380' , SYSDATE() )\$   | Query OK, 1 row affected (0.09 sec)<br>-- fait l'INSERT dans Qualifications et met à<br>-- 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 RAISER EXCEPTION APPLICATION\_ERROR. Cette fonction permet de retourner un message personnalisé à partir du corps d'un déclencheur (<http://rpbooman.blogspot.com/2005/11/using-udf-to-raise-error-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 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 in the same 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                                                                                                                                                                                                                                                          |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>SET @vs_nombre=0\$<br/><br/>CREATE TRIGGER TrigMutant<br/>  AFTER INSERT ON Trace FOR EACH ROW<br/>BEGIN<br/>  SELECT COUNT(*) INTO @vs_nombre<br/>    FROM Trace;<br/>END;<br/>\$</pre> | <pre>SELECT @vs_nombre\$<br/>+-----+<br/>  @vs_nombre  <br/>+-----+<br/>  0  <br/>+-----+<br/><br/>INSERT INTO Trace<br/>          VALUES ('Test TrigMutant')\$<br/>SELECT @vs_nombre\$<br/>+-----+<br/>  @vs_nombre  <br/>+-----+<br/>  1  <br/>+-----+</pre> |

## Restrictions

Pour en finir avec les déclencheurs, je vais terminer la « litanie »...



- Pas de déclencheur sur une table temporaire ou une vue, ou sur un événement système (connexion, arrêt de la base, etc.).
- Pas de possibilité de combiner plusieurs événements par 2 ou par 3 (INSERT OR UPDATE, INSERT OR DELETE, etc.).
- Les déclencheurs état (*statement trigger*) ne sont pas encore reconnus.
- Il n'est pas possible de désactiver un déclencheur sans le détruire.
- Les déclencheurs ne peuvent être écrits qu'avec le langage procédural de MySQL (qui se limite aussi aux procédures et aux fonctions cataloguées).
- Un déclencheur ne peut constituer aucune transaction, ainsi les instructions suivantes sont interdites : COMMIT, ROLLBACK, SAVEPOINT, START TRANSACTION et SET CONSTRAINTS.
- Les déclencheurs ne sont pas activés par des actions en cascade résultant d'opérations sur des clés primaires ou étrangères (CASCADE voir le chapitre 2).

## Suppression d'un déclencheur

Pour pouvoir supprimer un déclencheur, vous devez disposer du privilège SUPER (le privilège DROP TRIGGER est à venir). La syntaxe de l'instruction DROP TRIGGER est la suivante :

**DROP TRIGGER [nomBase.]nomDéclencheur;**

Si le nom de la base est omis, MySQL cherchera à détruire le déclencheur dans la base de données en cours d'utilisation.



Le fait de détruire une table a pour conséquence d'effacer aussi tous les déclencheurs qui lui sont associés.

Le fait de détruire une base supprime toutes les tables. Par conséquent, les déclencheurs partent ainsi tous « à la casserole ».

## SQL dynamique

MySQL parle de *server-side prepared statements* (états préparés) pour évoquer le fait de pouvoir programmer en SQL dynamique. En plus des directives SQL (LMD, LID), il est possible de construire automatiquement certaines instructions SQL du LDD.

Il est bien sûr possible de construire des instructions SQL à partir d'un programme C (*MySQL API client library*), Java (*MySQL Connector/J*), .Net (*MySQL Connector/NET*) ou PHP une API écrite en C (*mysqli extension*).

Par exemple, on pourra créer une table dont le nom passera en paramètre et qui aura un nombre variable de colonnes. Il sera aussi permis de construire automatiquement une requête SQL en fonction des choix d'un utilisateur. En plus des ordres simples, on pourra également paramétriser une suite d'instructions dans un bloc MySQL.

Une instruction SQL dynamique est stockée en tant que chaîne de caractères qui sera évaluée à l'exécution et non à la compilation (en opposition aux instructions SQL statiques qui peuplent la majorité des sous-programmes).



Seules les instructions suivantes peuvent être construites dynamiquement (dans un *prepared statement*) : CREATE TABLE, DELETE, DO, INSERT, REPLACE, SELECT, SET, UPDATE, et la plupart des commandes SHOW.

Les états préparés ne peuvent être utilisés dans un déclencheur (seuls les sous-programmes peuvent en bénéficier pour l'heure).

L'instruction :

```
| DO expression1 [,expression2] ...;
```

exécute l'expression (ou les expressions) sans retourner aucun résultat. On peut l'assimiler à un raccourci de SELECT *expression1,...* mais sans renvoi de résultat. Elle peut être utile pour des fonctions à effet de bord (comme la relâche de verrou : RELEASE\_LOCK(

## Syntaxe

La construction dynamique d'instructions SQL (*prepared statements*) est basée sur les directives suivantes :

```
| PREPARE nomEtat FROM étatPréparé;
| EXECUTE nomEtat [USING @var1 [, @var2] ...];
| {DEALLOCATE | DROP} PREPARE nomEtat;
```

- L'instruction PREPARE associe un nom (insensible à la casse) à une instruction dynamique.
- *étatPréparé* est soit une chaîne soit une variable de session contenant le texte de l'instruction SQL construite (instruction simple, pas d'instructions multiples). Dans cette chaîne, le caractère « ? » (appelé *placeholder*) permet de se substituer à un paramètre.
- L'instruction EXECUTE lance l'ordre paramétré avec éventuellement la clause USING qui reliera les paramètres aux variables de session.
- Pour en terminer avec un ordre préparé, utilisez DEALLOCATE PREPARE qui supprime le contenu de l'ordre (une fin de session désalloue tous les ordres ouverts).

## Exemples

Considérons la table Avion contenant deux enregistrements.

```
CREATE TABLE Avion
(immat VARCHAR(6), typeAv CHAR(8), nbHVol DECIMAL(7,2), comp VARCHAR(4))
INSERT INTO Avion VALUES ('F-GLFS', 'A320', 1000, 'AF');
INSERT INTO Avion VALUES ('F-WOWW', 'A380', 1500, 'AF');
```

### Instruction DELETE

Le tableau suivant décrit la construction dynamique de l'ordre de suppression des avions dont le nombre d'heures de vol est supérieur à un paramètre spécifié par une variable de session (ici évaluée à 1 000).

Tableau 7-31 Utilisation de DELETE

| Web | Code MySQL                                                                                                                                                                                                                                             | Commentaires                                                                        |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
|     | <pre>SET @vs_nbhVol = 1000 \$<br/>CREATE PROCEDURE bdsoutou.sousProg()<br/>BEGIN<br/>    PREPARE etat FROM<br/>        'DELETE FROM Avion WHERE nbHVol &gt; ?';<br/>    EXECUTE etat USING @vs_nbhVol;<br/>    DEALLOCATE PREPARE etat;<br/>END;</pre> | Déclaration de la variable de session.<br><br>Préparation de l'ordre.<br>Exécution. |

L'appel (CALL bdsoutou.sousProg()) de cette procédure aura pour conséquence de détruire l'avion immatriculé 'F-WOWW'.

### Instruction SELECT

Le tableau suivant décrit la construction dynamique de l'extraction des avions dont le nombre d'heures de vol est égal à un paramètre spécifié par une variable de session (ici évaluée à 1 000). La requête est elle-même stockée dans une variable de session.

Tableau 7-32 Utilisation de SELECT

| Web | Code MySQL                                                                                                                                                                                                                                                                       | Commentaires                                                                       |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
|     | <pre>SET @vs_chaine =<br/>    'SELECT * FROM Avion WHERE nbHVol=?\$'<br/>SET @vs_nbhVol = 1000\$<br/>CREATE PROCEDURE bdsoutou.sousProg()<br/>BEGIN<br/>    PREPARE etat FROM @vs_chaine;<br/>    EXECUTE etat USING @vs_nbhVol;<br/>    DEALLOCATE PREPARE etat;<br/>END;</pre> | Déclaration des variables de session.<br><br>Préparation de l'ordre.<br>Exécution. |

L'appel de cette procédure aura pour conséquence d'extraire l'avion immatriculé 'F-GLFS'.

## Instruction UPDATE

Le tableau suivant décrit la construction dynamique de l'instruction de modification (augmentation du nombre d'heures de vol d'un pourcentage passé en premier paramètre) d'un avion dont l'immatriculation passe en deuxième paramètre. Notez qu'il n'est pas besoin de doubler le guillemet dans la spécification du deuxième *placeholder* : paramètre *immat* (bien s'agit d'une chaîne de caractères).

Tableau 7-33 Utilisation de UPDATE

| Code MySQL                                                                                                                                                                                                                                                                                                | Commentaires                                                                  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| <pre>SET @vs_chaine =     'UPDATE Avion SET nbHVol=nbHVol*?       WHERE immat=?\$' SET @vs_immat    = 'F-GLFS '\$ SET @vs_pourcent = 1.1\$  CREATE PROCEDURE bdsoutou.sousProg() BEGIN     PREPARE etat FROM @vs_chaine;     EXECUTE etat USING @vs_pourcent,@vs_immat;     DROP PREPARE etat; END;</pre> | Déclaration des variables de session<br>Préparation de l'ordre.<br>Exécution. |

L'appel de cette procédure aura pour conséquence d'aumenter de 10 % le nombre d'heures de vol de l'avion immatriculé 'F-GLFS'.



Si l'immatriculation avait été une constante, il aurait fallu doubler le guillemet dans l'affectation de la variable de session :

```
SET @vs_chaine ='UPDATE Avion SET nbHVol=nbHVol*? WHERE immat=' 'F-GLFS'
```

## Restrictions

Les *placeholders* (points d'interrogation) des états préparés ne peuvent pas remplacer les noms de tables, de vues, d'index, de colonnes, etc., dans une instruction, de sorte à considérer un ordre dynamiquement. Ils ne peuvent que remplacer des données :

- dans la clause WHERE pour des SELECT, UPDATE (dans la clause SET aussi) ou DELETE
- dans la clause VALUES pour un INSERT.

## Utilisations

Le tableau suivant résume quelques cas permis et les cas non valides associés. N'oubliez pas de doubler chaque guillemet pour affecter une telle chaîne de caractères dans une variable de session.

Tableau 7-34 Utilisation des placeholders

| Possibles                                | Impossibles                                 |
|------------------------------------------|---------------------------------------------|
| SELECT * FROM table WHERE col = ? ...    | SELECT ? FROM table WHERE ...               |
|                                          | SELECT * FROM ? WHERE ...                   |
|                                          | SELECT * FROM table WHERE ? > 1000          |
| INSERT INTO table VALUES (?, ?, ?, ?, ?) | INSERT INTO ? VALUES ('F-FRTY', ?, ?, ?, ?) |
| UPDATE table SET col=? WHERE col=? ...   | UPDATE table SET ?=? WHERE col=? ...        |
|                                          | UPDATE ? SET ...                            |
| DELETE FROM table WHERE col=? ...        | DELETE FROM table WHERE ? = ...             |
|                                          | DELETE FROM ? ...                           |

Afin de pallier cette limitation, il faut construire l'instruction dynamique à l'aide de la fonction CONCAT en incluant éventuellement des *placeholders* aux endroits permis.

### Exemple sans placeholder

La procédure cataloguée suivante crée dynamiquement, dans la base bdsoutou, une table dont le nom passé en premier paramètre. Le nom de la seconde colonne de la table (ici de type INT) est passé en second paramètre de la procédure.

Tableau 7-35 Crédation dynamique d'une table

| Web | Code MySQL                                                                                                                                                                                                                                                                                                                                                 | Commentaires                                                                                                                                                   |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre> CREATE PROCEDURE bdsoutou.sousProg     (IN v_param1 VARCHAR(10),      IN v_param2 VARCHAR(10)) BEGIN     SET @vs_chaine := CONCAT         ('CREATE TABLE IF NOT EXISTS',          bdsoutou.',v_param1,' (immat CHAR(4), ',          ',v_param2, INT)');     PREPARE etat FROM @vs_chaine;     EXECUTE etat;     DEALLOCATE PREPARE etat; END; </pre> | <p>Construction de la chaîne :</p> <p>'CREATE TABLE IF NOT EXISTS<br/>bdsoutou.v_param1(immat<br/>CHAR(4),<br/>v_param2 INT)'</p> <p>Création de la table.</p> |

L'appel suivant de cette procédure aura pour effet de créer la table Helico. La commande DESCRIBE confirme la structure de la nouvelle table.

```

CALL bdsoutou.sousProg('Helico','turbine')$
```

Query OK, 0 rows affected (0.21 sec)

```
DESCRIBE bdsoutou.Helico $
```

```
+-----+-----+-----+-----+-----+
| 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 nombre d'heures de vol (colonnes de noms passés en premier et en deuxième paramètre) la table de nom passé en troisième paramètre, en fonction d'une condition sur une colonne 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                                                                                                                                                               |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>CREATE PROCEDURE bdsoutou.sousProg (IN v_param1 CHAR(6), IN v_param2 CHAR(6), IN v_param3 CHAR(5), IN v_param4 CHAR(5)) BEGIN     SET @vs_immat := 'F-GLFS';     SET @vs_chaine := CONCAT('SELECT ',v_param1,',',v_param2,       ' FROM bdsoutou.',v_param3,       ' WHERE ',v_param4,' = ?');     PREPARE etat FROM @vs_chaine;     EXECUTE etat USING @vs_immat;     DEALLOCATE PREPARE etat; END;</pre> | <p>Avec l'appel suivant, construction de la chaîne :</p> <pre>'SELECT typeAv,nbHVol   FROM bdsoutou.Avion  WHERE immat=?'</pre> <p>Exécution de la requête paramétrée.</p> |

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)
```

L'objectif de ces exercices est d'écrire des sous-programmes MySQL manipulant des curseurs et gérant des exceptions sur la base de données *Parc informatique*.

## Exercice 7.1

### Curseur

On désire connaître, pour chaque logiciel installé, le temps (nombre de jours entier décimal) passé entre l'achat et l'installation. Ce calcul devra renseigner la colonne `delai` de la table `Installé` pour l'instant nulle.

Utiliser une table `test.Trace` (`message` `VARCHAR(80)`) (et l'afficher en fin de sous-programme) pour stocker :

- les incohérences (date d'installation antérieure à la date d'achat, date d'installation ou date d'achat inconnue) ;
- le nombre entier de jours séparant l'achat de l'installation (utiliser `DATEDIFF`) ;
- une chaîne simulant un format *TIME* étendu qui représente le nombre de jours décimal séparant l'achat de l'installation (par exemple, si le nombre de jours décimal vaut « 14,5 », il faudra construire la chaîne : « 14 j 12:00:00 »).

Écrire la procédure `calculTemps` pour programmer ce processus. Un exemple de tableau `test.Trace` à produire en sortie :

```
+-----+
| message |
+-----+
| Logiciel Oracle 6 sur Poste 2 attende 2924 jour(s). |
| En format TIME étendu 2924 j 00:00:00 |
| Logiciel Oracle 8 sur Poste 2 attende 1463 jour(s). |
| ... |
| Logiciel I. I. S. installé sur Poste 7 11 jour(s) avant l'achat! |
| Date d'achat inconnue pour le logiciel SQL*Net sur Poste 2 |
| Logiciel Oracle 6 sur Poste 8 attende 3876 jour(s). |
| En format TIME étendu 3876 j 10:59:17 |
| ... |
+-----+
```

## Exercice 7.2

### Transaction

Écrire la procédure `installLogSeg` permettant d'effectuer une installation groupée sur tous les postes d'un même segment d'un nouveau logiciel. La transaction doit enregistrer dans un premier temps le nouveau logiciel, puis les différentes installations sur tous les postes du segment de même type que celui du logiciel acheté. L'installation se fera à la date du jour. Penser à mettre à jour colonne `delai` comme programmé précédemment.

Ne pas encore tenir compte des éventuelles exceptions et tracer chaque insertion dans la test.Trace. Utiliser les paramètres ci-dessous pour tester votre procédure. L'état de sortie doit être le suivant. Vérifier aussi la présence des deux nouveaux enregistrements dans la table Instal. Ne programmer le COMMIT qu'une fois la procédure bien testée.

```
CALL installLogSeg('130.120.80', 'log99', 'Blaster', '2005-09-05',
'9.9', 'PCWS', 999.9)$
+-----+
| message |
+-----+
| Blaster stocké dans la table Logiciel |
| Installation sur Poste 4 dans Salle 2 |
| Installation sur Poste 5 dans Salle 2 |
+-----+
```

## Exercice

### 7.3

#### Exceptions

Modifier la procédure installLogSeg afin de prendre en compte quelques-unes des exceptions potentielles :

- numéro de segment inconnu (erreur NOT FOUND) ;
- numéro de logiciel déjà présent (ERROR 1062 *Duplicate entry*) ;
- type du logiciel inconnu (ERROR 1452 *Cannot add or update a child row*) ;
- date d'achat postérieure à celle du jour (se servir du même calcul que pour la colonne de l'exercice précédent) ;
- aucune installation réalisée, car pas de poste de travail de ce type (erreur utilisateur *install\_possible*).

Vérifier chacun de ces cas avec le jeu de tests suivant :

```
--test segment
CALL installLogSeg('toto', 'log99', 'Blaster', '2005-09-05', '9.9',
'PCWS', 999.9)$
--test logiciel déjà présent
CALL installLogSeg('130.120.80', 'log1', 'Blaster', '2005-09-05',
'9.9', 'PCWS', 999.9)$
--test type du logiciel
CALL installLogSeg('130.120.80', 'log98', 'Mozilla', '2005-11-04',
'toto', 100.0)$
--date d'achat plus grande que celle du jour ?
CALL installLogSeg('130.120.80', 'log98', 'Mozilla', '2010-11-04',
'PCWS', 100.0)$
--aucune install
CALL installLogSeg('130.120.81', 'log55', 'Eudora', '2005-12-06',
'PCWS', 540)$
--bonne installation
CALL installLogSeg('130.120.80', 'log77', 'Blog Up', '2005-12-05',
'1.3', 'PCWS', 90)$
```

*Mises à jour de colonnes*

Écrire les déclencheurs Trig\_AD\_Installer et Trig\_AI\_Installer sur la table Installer permettant de faire la mise à jour automatique des colonnes nbLog de la table Poste, et nbInstall de la table Logiciel. Prévoir les cas de désinstallation d'un logiciel (AFTER DELETE) sur un poste, et d'installation (AFTER INSERT) d'un logiciel sur un autre.

Écrire les déclencheurs Trig\_AI\_Poste et Trig\_AD\_Poste sur la table Poste permettant d'actualiser la colonne nbPoste de la table Salle à chaque ajout ou suppression d'un nouveau poste.

Écrire le déclencheur Trig\_AU\_Salle sur la table Salle qui met à jour automatiquement la colonne nbPoste de la table Segment après la modification de la colonne nbPoste.

Ces deux derniers déclencheurs vont s'enchaîner : l'ajout ou la suppression d'un poste entraîne l'actualisation de la colonne nbPoste de la table Salle, qui conduira à la mise à jour de la colonne nbPoste de la table Segment. Ajouter un poste pour vérifier le rafraîchissement des deux tables (Salle et Segment). Supprimer ce poste puis vérifier à nouveau la cohérence des deux tables.

*Programmation de contraintes*

Écrire le déclencheur Trig\_BI\_Installer sur la table Installer permettant de contrôler, avant chaque nouvelle installation, que le type du logiciel correspond au type du poste, et que la date d'installation est soit nulle soit postérieure à la date d'achat.

# **Partie III**

# Langages

# et outils



# Utilisation avec Java

MySQL offre, sur son site, différents pilotes pour rendre compatibles des applications une base de données sur différents systèmes.

- *Connector/ODBC (Open DataBase Connectivity)* pour Windows, Linux, Mac OS Unix ;
- *Connector/J* pour toute plate-forme Java en utilisant JDBC (*Java DataBase Connectivity*) ;
- *Connector/Net* pour toute plate-forme .Net ;
- *Connector/MXJ* : composant qui encapsule le moteur MySQL dans une application Java.

Ce chapitre explique l'utilisation de l'API JDBC 3.0 pour manipuler une base MySQL via un programme Java.

## JDBC avec Connector/J

---

L'interface JDBC initialement programmée par Sun, appelée aussi « passerelle » ou « API de base de données », est composée d'un ensemble de classes permettant le dialogue entre une application Java et une source de données compatible SQL (tables relationnelles en général, mais aussi données issues d'un fichier texte ou d'un classeur Excel par exemple). L'API JDBC 3.0 que MySQL fournit gratuitement est appelée *Connector/J*.

L'interface JDBC est conforme au niveau d'entrée de la norme SQL2 (*entry level*) et précharge la programmation *multithread*. La communication est réalisée en mode client-serveur déconnecté et s'effectue en plusieurs étapes :

- connexion à la base de données ;
- émissions d'instructions SQL et exploitation des résultats provenant de la base de données ;
- déconnexion de la base.

Le spectre de JDBC est large, car l'applicatif Java peut être une classe ou une *applet*, un client, une *servlet*, un EJB (*Enterprise Java Beans*) ou une procédure cataloguée côté serveur.

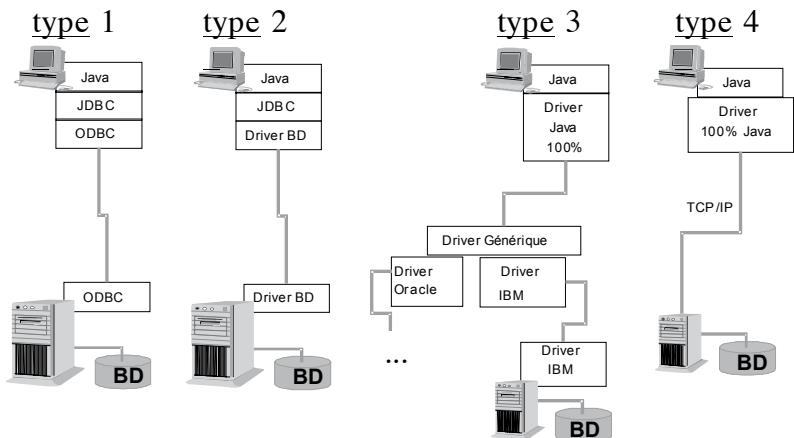
## Classification des pilotes (drivers)

Un pilote (*driver*) JDBC est une couche logicielle chargée d'assurer la liaison entre l'application Java (cliente) et le SGBD (serveur). La classification des pilotes JDBC distingue quatre types

- Les pilotes de type 1 (*JDBC-ODBC Bridge*) utilisent la couche logicielle de Microsoft appelée ODBC. Le client est dit « épais », puisque le pilote JDBC convertit les appels Java en appels ODBC avant de les exécuter. Cette approche convient bien pour des sources de données Windows ou si l'interface cliente est écrite dans un langage natif de Microsoft.
- Les pilotes de type 2 (*Native-API Partly-Java Driver*) utilisent un pilote fourni par le constructeur de la base de données (natif). Le pilote n'étant pas développé en Java, le client est aussi dit « épais » pour cette approche. En effet, les commandes JDBC sont toutes converties en appels natifs du SGBD considéré. Cette approche convient pour les applications qui manipulent des sources de données uniques (tout Oracle ou IBM, etc.).
- Les pilotes de type 3 (*Net Protocol All-Java Driver*) utilisent un pilote générique natif écrit en Java. Le client est plus « léger » car les appels JDBC sont transformés par un protocole indépendant du SGBD. Cette approche convient pour des sources de données hétérogènes.
- Les pilotes de type 4 (*Native Protocol All-Java Driver*) sont écrits en Java. Le client est léger car il ne nécessite aucune couche logicielle supplémentaire. Les appels JDBC sont traduits en *sockets* exploités par le SGBD. Cette approche est la plus simple, mais pas forcément la plus puissante ; elle convient pour tous les types d'architectures.

La figure suivante schématisé le principe mis en œuvre au travers des quatre types de pilotes JDBC

**Figure 8-1** Types de pilotes JDBC



Le choix du pilote n'a pas d'influence majeure sur la programmation. Seules les phases de chargement du pilote et de connexion aux bases sont spécifiques, les autres instructions sont indépendantes du pilote. En d'autres termes, si vous avez une application déjà écrite et qu'il

vous décidez de changer le type du pilote – soit que la source de données migre de MySQL, à Oracle ou à SQL Server par exemple, soit que vous optiez pour un autre pilote conservant votre source de données –, seules quelques instructions devront être réécrites. Avec MySQL, vous pouvez travailler avec l’API de Sun, mais vous n’avez pas trop le choix pour le type du pilote (*Connector/J* est un pilote JDBC de type 4).

## Le paquetage `java.sql`

La version 3.0 de JDBC est composée de classes et d’interfaces situées dans le paquetage `java.sql` du JDK. MySQL propose également une API propriétaire (qui redéfinit et étend celle de Sun). Le tableau suivant résume la composition de ce paquetage.

Tableau 8-1 L’API JDBC 3.0 standard

| Classe/interface                        | Description                                                                      |
|-----------------------------------------|----------------------------------------------------------------------------------|
| <code>java.sql.Driver</code>            | Pilotes JDBC pour les connexions aux sources de données SQL.                     |
| <code>java.sql.Connection</code>        |                                                                                  |
| <code>java.sql.Statement</code>         | Construction d’ordres SQL.                                                       |
| <code>java.sql.PreparedStatement</code> |                                                                                  |
| <code>java.sql.CallableStatement</code> |                                                                                  |
| <code>java.sql.ResultSet</code>         | Gestion des résultats des requêtes SQL.                                          |
| <code>java.sql.DriverManager</code>     | Gestion des pilotes de connexion.                                                |
| <code>java.sql.SQLException</code>      | Gestion des erreurs SQL.                                                         |
| <code>java.sql.DatabaseMetaData</code>  | Gestion des métainformations (description de la base de données, des tables...). |
| <code>java.sql.ResultSetMetaData</code> |                                                                                  |
| <code>java.sql.SavePoint</code>         | Gestion des transactions et des sous-transactions.                               |

## Structure d’un programme

La structure d’un programme Java utilisant JDBC comprend successivement les phases suivantes :

- d’importation de paquetages ;
- de chargement d’un pilote ;
- de création d’une ou de plusieurs connexions ;
- de création d’un ou de plusieurs états ;
- d’émission d’instructions SQL sur ces états ;
- de fermeture des objets créés.

Le code suivant (`JDBCTest.java`) décrit la syntaxe du plus simple programme JDBC. Nous inscrivons toutes les phases dans un même bloc (le `main`), mais elles peuvent se trouver dans différents blocs ou méthodes de diverses classes.

Tableau 8-2 Programme de test de connexion JDBC

| Code Java                                     | Commentaires                     |
|-----------------------------------------------|----------------------------------|
| Web                                           | Importation du paquetage.        |
| import java.sql.*;                            | Classe ayant une méthode main    |
| public class JDBCtest                         |                                  |
| {public static void main(String[] args)       |                                  |
| throws SQLException, Exception                |                                  |
| {                                             |                                  |
| try                                           |                                  |
| { System.out.println                          |                                  |
| ("Initialisation de la connexion");           |                                  |
| Class.forName                                 | Chargement du pilote JDBC        |
| ("com.mysql.jdbc.Driver").newInstance();      | MySQL.                           |
| } catch (ClassNotFoundException ex)           |                                  |
| { System.out.println                          |                                  |
| ("Problème au chargement"+ex.toString()); }   |                                  |
| try                                           |                                  |
| { Connection cx = DriverManager.getConnection | Création d'une connexion.        |
| ("jdbc:mysql://localhost/bdsoutou?            |                                  |
| user=soutou&password=iut") ;                  |                                  |
| Statement etat = cx.createStatement () ;      | Création d'un état de connexion. |
| ResultSet rset = etat.executeQuery            | Extraction de la date courante.  |
| ("SELECT SYSDATE());                          |                                  |
| while (rset.next ())                          |                                  |
| System.out.println("Nous sommes le : "+       |                                  |
| rset.getString (1));                          | Affichage du résultat.           |
| System.out.println("JDBC correctement         |                                  |
| configuré");                                  |                                  |
| }                                             |                                  |
| catch(SQLException ex)                        | Gestion des erreurs.             |
| { System.err.println("Erreur : "+ex); }       |                                  |
| }                                             |                                  |
| }                                             |                                  |

Le dernier bloc permet de récupérer les erreurs renvoyées par le SGBD. Nous détaillerons en fin de section le traitement des exceptions.

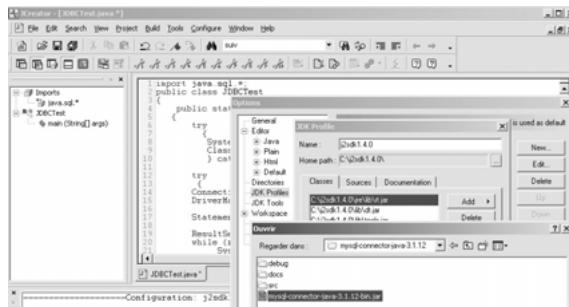
## Test de votre configuration

L'environnement JDBC sous MySQL nécessite la configuration d'un certain nombre de variables.

- La variable PATH doit contenir le chemin de la machine virtuelle Java pour compiler et exécuter des classes. Le JDK est en général installé dans C:\j2sdk1.XXX, les fichiers javac et java se trouvent dans le sous-répertoire bin.
- La variable CLASSPATH doit inclure le paquetage JDBC (fichier .jar) pour MySQL (téléchargeable sur le site de MySQL). Pour ma part, j'ai dézippé le fichier mysql-connector-java-3.1.12.zip dans le répertoire C:\temp.

Vous pouvez tester votre environnement en utilisant le fichier `JDBCTest.java`. Si vous utilisez l'outil *JCreator*, configurez la variable `CLASSPATH` de la manière suivante : Configure/Options/JDK Profiles, clic sur la version du JDK, puis Edit, onglet Classpath, faire Add Archive et choisir le fichier jar (pour mon cas `mysql-connector-java-3.1.12-bin.jar`).

Figure 8-2 Interface JCreator



Cet exemple décrit le code nécessaire à la connexion à votre base (il faudra modifier le nom de la base, le nom et le mot de passe de l'utilisateur) et doit renvoyer les messages suivants :

#### Initialisation de la connexion

Nous sommes le : date et heure courante  
JDBC correctement configuré

## Connexion à une base

La connexion à une base de données est rendue possible par l'utilisation de la classe `DriverManager` et de l'interface `Connection`.



Deux étapes sont nécessaires pour qu'un programme Java puisse se connecter à une base de données :

- Le chargement du pilote par appel de la méthode `java.lang.Class.forName`.
- L'établissement de la connexion en créant un objet (ici `cx`) de l'interface `Connection` par l'instruction suivante : `cx = DriverManager.getConnection(chaineConnexion)`

Pour MySQL, nous verrons que le paramètre `chaineConnexion` représente une variable dont une syntaxe simplifiée est de type :

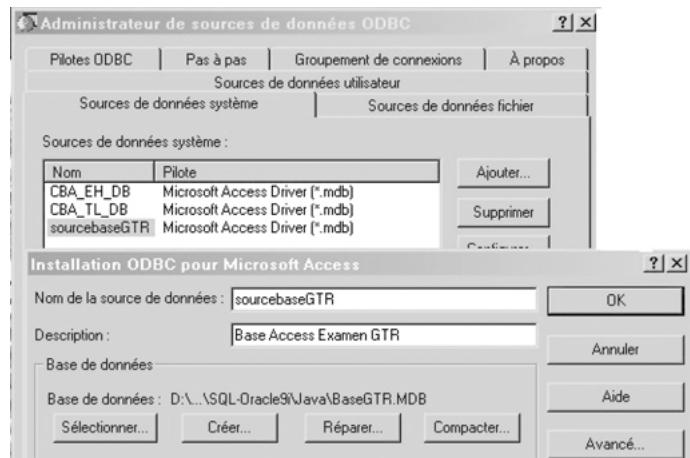
`jdbc:mysql://[host][:port]/[database][?user][=nomUtil][&password][=motPass]`

Cette chaîne permettra de désigner la base et d'identifier l'utilisateur.

## Base Access

Étudions, pour information, l'établissement de la connexion d'un pilote de type 1 pour mettre en rapport avec une base Access via une source de données ODBC. La figure suivante illustre les parties du panneau de configuration Windows qui permettent de désigner une base Access. Dans notre exemple, la source (**BaseGTR.MDB**) est située dans un répertoire sur l'unité de disque D:\ et désignée par le DSN (*Data Source Name*) **sourcebaseGTR** :

Figure 8-3 Source de données ODBC



Le code suivant (**TestJDBCODBC.java**) charge un pilote de type 1, puis se connecte à une source ODBC précitée (inutile de préciser le nom et le mot de passe de l'utilisateur du fait d'une base Access). Le DSN est noté en gras dans le script.

Tableau 8-3 Programme JDBC

| Code Java                                                                                                                                                                                                                                                                                                                                                                                                          | Commentaires                                                                                                                               |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>Web import java.sql.*; class TestJDBCODBC { public static void main (String args [])     throws SQLException {try {Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");}  catch (ClassNotFoundException ex) { System.out.println ("Problème au chargement"); }  try {Connection conn = DriverManager.getConnection           ("jdbc:odbc:<b>sourcebaseGTR</b>","","");  ... catch(SQLException ex){ ... } } }</pre> | Importation.<br>Classe ayant une méthode main.<br>Chargement d'un pilote JDBC/ODBC.<br>Connexion à la base Access.<br>Gestion des erreurs. |

## Base MySQL

Seules les phases de chargement de pilote et de création de la connexion changent. Afin de transférer un pilote MySQL, il faut utiliser l'interface `DriverManager` implémentée par l'appel de la méthode `Class.forName()`. Sous `Connector/J`, le nom de la classe à charger est `com.mysql.jdbc.Driver`. La connexion s'effectue par la méthode `getConnectio`

Tableau 8-4 Chargement du pilote MySQL

| Code Java                                                                                                                                                                                                   | Commentaires                                         |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------|
| try<br>{ <b>Class.forName("com.mysql.jdbc.Driver").newInstance();</b><br>} catch (ClassNotFoundException ex)<br>{ System.out.println<br>("Problème au chargement"+ex.toString()); }                         | Chargement du pilote MySQL.                          |
| try<br>{ <b>Connection cx = DriverManager.getConnection</b><br>("jdbc:mysql://localhost/bdsoutou?<br>user=soutou&password=iut"); ... }<br>catch(SQLException ex)<br>{ System.err.println("Erreur : "+ex); } | Déclaration d'une connexion.<br>Gestion des erreurs. |

## Interface Connection

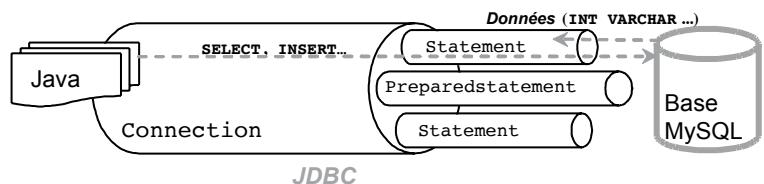
Le tableau ci-après présente les principales méthodes disponibles de l'interface `Connection`. Nous détaillerons l'invocation de certaines de ces méthodes à l'aide des exemples de sections suivantes.

Tableau 8-5 Méthodes de l'interface Connection

| Méthode                                  | Description                                                                                                                           |
|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>createStatement()</code>           | Création d'un objet destiné à recevoir un ordre SQL statique non paramétrisé.                                                         |
| <code>prepareStatement(String)</code>    | Précompile un ordre SQL acceptant des paramètres et pouvant être exécuté plusieurs fois.                                              |
| <code>prepareCall(String)</code>         | Appel d'une procédure cataloguée (certains pilotes attendent <code>execute</code> ou ne reconnaissent pas <code>prepareCall</code> ). |
| <code>void setAutoCommit(boolean)</code> | Positionne ou non le <i>commit</i> automatique.                                                                                       |
| <code>void commit()</code>               | Valide la transaction.                                                                                                                |
| <code>void rollback()</code>             | Invalide la transaction.                                                                                                              |
| <code>void close()</code>                | Ferme la connexion.                                                                                                                   |

Une fois la connexion établie, il est nécessaire de définir des états qui permettront l'encapsulation d'instructions SQL dans du code Java. Un état permet de faire passer plusieurs instructions SQL sur le réseau. On peut affecter à un état une ou plusieurs instruction SQL. Si on désire exécuter plusieurs fois la même instruction, il est intéressant de réserver l'utilisation d'un état à cet effet.

**Figure 8-4 Connexion et états**



## Interfaces disponibles

Différentes interfaces sont prévues à cet effet :

- **Statement** pour les ordres SQL statiques. Ces états sont construits par la méthode `createStatement` appliquée à la connexion.
- **PreparedStatement** pour les ordres SQL paramétrés. Ces états sont construits par méthode `prepareStatement` appliquée à la connexion.
- **CallableStatement** pour les procédures ou fonctions cataloguées. Ces états sont construits par la méthode `prepareCall` appliquée à la connexion.

S'il ne doit plus être utilisé dans la suite du code Java, chaque objet de type `Statement`, `PreparedStatement` ou `CallableStatement` devra être fermé à l'aide de la méthode `close`.

## Méthodes génériques pour les paramètres

Une fois qu'un état est créé, il est possible de lui passer des paramètres par des méthodes génériques (étudiées plus en détail par la suite) :

- `setxxx` où `xxx` désigne le type de la variable (exemple : `setString` ou `setInt`) dans le sens Java vers MySQL (*setter methods*). Il s'agit ici de paramétriser un ordre SQL (instruction ou appel d'un sous-programme) ;

- `getxxx` (exemple : `getString` ou `getInt`) du sens MySQL vers Java. Il s'agit d'extraire des données de la base dans des variables hôtes Java via un curseur Java (*get methods*) ;
- `updatexxx` (exemple : `updateString` ou `updateInt`) du sens Java vers MySQL. Il s'agit ici de mettre à jour des données de la base via un curseur Java (*update methods*). Ces méthodes sont disponibles seulement depuis la version 2 de JDBC (SDK 1.2).

## États simples (interface Statement)

Nous décrivons ici l'utilisation d'un état simple (interface `Statement`). Nous étudierons la suite les instructions paramétrées (interface `PreparedStatement`) et appels de programmes (interface `CallableStatement`). Le tableau suivant décrit les principales méthodes de l'interface `Statement`.

Tableau 8-6 Méthodes de l'interface Statement

| Méthode                                     | Description                                                                                                                                                                         |
|---------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ResultSet executeQuery(String)</code> | Exécute une requête et retourne un ensemble de lignes ( <code>ResultSet</code> ).                                                                                                   |
| <code>int executeUpdate(String)</code>      | Exécute une instruction SQL et retourne le nombre de lignes traitées (INSERT, UPDATE ou DELETE) ou 0 pour les instructions ne renvoyant aucun résultat (LDD).                       |
| <code>boolean execute(String)</code>        | Exécute une instruction SQL et renvoie <code>true</code> si c'est une instruction SELECT, <code>false</code> sinon (instructions LMD ou plus de résultats <code>ResultSet</code> ). |
| <code>Connection getConnection()</code>     | Retourne l'objet de la connexion.                                                                                                                                                   |
| <code>void setMaxRows(int)</code>           | Positionne la limite du nombre d'enregistrements à extraire dans toute requête issue de cet état.                                                                                   |
| <code>int getUpdateCount()</code>           | Nombre de lignes traitées par l'instruction SQL (-1 si c'est une requête ou si l'instruction n'affecte aucune ligne).                                                               |
| <code>void close()</code>                   | Ferme l'état.                                                                                                                                                                       |

Le code suivant (`Etats.java`) présente quelques exemples d'utilisation de ces méthodes. On crée d'abord un état (objet `etatSimple`). Nous supposons qu'un pilote JDBC est chargé et que la connexion `cx` a été créée. Nous verrons en fin de chapitre comment traiter proprement les exceptions.

| Code Java                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Commentaires                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>Statement etatSimple = cx.createStatement();     etatSimple.executeUpdate("CREATE TABLE IF NOT EXISTS Compagnie(comp VARCHAR(4), nomComp VARCHAR(30), CONSTRAINT pk_Compagnie PRIMARY KEY(comp))");     int j = etatSimple.executeUpdate("CREATE TABLE IF NOT EXISTS Avion (immat VARCHAR(6),typeAvion VARCHAR(15), cap SMALLINT, compa VARCHAR(4), CONSTRAINT pk_Avion PRIMARY KEY(immat), CONSTRAINT fk_Avion_comp_Compagnie FOREIGN KEY(compa) REFERENCES Compagnie(comp))");     int k = etatSimple.executeUpdate ("INSERT INTO Compagnie VALUES ('AF','Air France')");     etatSimple.execute("INSERT INTO Avion VALUES         ('F-WTSS','Concorde',90,'AF')");     etatSimple.execute ("INSERT INTO Avion VALUES         ('F-FGFB','A320',148,'AF')");     etatSimple.setMaxRows (10);</pre> | Création de l'état.<br>Ordre LDD.                                                                                                                             |
| <pre>ResultSet curseurJava =     etatSimple.executeQuery("SELECT * FROM Avion");     etatSimple.execute("DELETE FROM Avion");     int l = etatSimple.getUpdateCount();</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Ordre LDD (autre écriture), <i>j</i> contient 0<br>(aucune ligne n'est concernée).                                                                            |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Ordre LMD, <i>k</i> contient 1<br>(une ligne est concernée).<br>Ordres LMD (autres écritures).                                                                |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Pas plus de 10 lignes<br>retournées par les prochaines extractions.<br>Chargement d'un curseur Java.<br>Ordre LMD, <i>l</i> contient 2<br>(avions supprimés). |

## Méthodes à utiliser

Le tableau suivant indique la méthode préférentielle à utiliser sur l'état courant (objet Statement) en fonction de l'instruction SQL à émettre :

Tableau 8-8 Méthodes Java pour les ordres SQL

| Instruction SQL      | Méthode       | Type de retour |
|----------------------|---------------|----------------|
| CREATE ALTER DROP    | executeUpdate | int            |
| INSERT UPDATE DELETE | executeUpdate | int            |
| SELECT               | executeQuery  | ResultSet      |

## Correspondances de types

Les échanges de données entre variables Java et colonnes des tables Oracle impliquent de prévoir des conversions de types. D'une manière générale, tout type de donnée MySQL peut être convertit en un type `java.lang.String`. Les types numériques trouvent aussi une correspondance dans les types numériques Java (attention toutefois aux arrondis, dépassages de capacité ou perte de précision).

Les tableaux suivants présentent les principales correspondances existantes :

**Tableau 8-9** Conversions possibles entre types

| Les types MySQL                                                                                           | Peuvent créer les classes Java                                                                                     |
|-----------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|
| CHAR, VARCHAR, BLOB, TEXT,<br>ENUM et SET                                                                 | java.lang.String, java.io.InputStream,<br>java.io.Reader, java.sql.Blob et java.sql.Cl                             |
| FLOAT, REAL, DOUBLE PRECISION,<br>NUMERIC, DECIMAL, TINYINT,<br>SMALLINT, MEDIUMINT, INTEGER<br>et BIGINT | java.lang.String, java.lang.Short,<br>java.lang.Integer, java.lang.Long, java.lang.<br>ble et java.math.BigDecimal |
| DATE, TIME, DATETIME et<br>TIMESTAMP                                                                      | java.lang.String, java.sql.Date et<br>java.sql.Timestamp                                                           |

La méthode `getObject()` de l’interface `ResultSet` (que nous allons étudier plus tard) réalise implicitement les conversions suivantes :

**Tableau 8-10** Correspondances entre types

| Types MySQL                                                                                   | Types Java                                                                                                                                           |
|-----------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| BOOL et BOOLEAN                                                                               | alias de TINYINT(1)                                                                                                                                  |
| BLOB, BINARY( <i>n</i> ), BIT(>1), LONGBLOB,<br>MEDIUMBLOB, TINYBLOB et VARBINARY( <i>n</i> ) | byte[]                                                                                                                                               |
| BIT(1) et TINYINT                                                                             | java.lang.Boolean si la configuration<br>tinyInt1isBit est mise à <i>true</i> (par défaut) et la ta<br>de la variable = 1 (java.lang.Integer sinon). |
| DOUBLE[( <i>n,p</i> )]                                                                        | java.lang.Double                                                                                                                                     |
| FLOAT[( <i>n,p</i> )]                                                                         | java.lang.Float                                                                                                                                      |
| SMALLINT[( <i>n</i> )] [UNSIGNED]                                                             | java.lang.Integer (sans contrôle du signe).                                                                                                          |
| INT et INTEGER[( <i>n</i> )] [UNSIGNED]                                                       | java.lang.Integer (si UNSIGNED<br>java.lang.Long).                                                                                                   |
| MEDIUMINT[( <i>n</i> )] [UNSIGNED]                                                            | java.lang.Integer (si UNSIGNED<br>java.lang.Long).                                                                                                   |
| BIGINT[( <i>n</i> )] [UNSIGNED]                                                               | java.lang.Long, (si UNSIGNED<br>java.math.BigInteger).                                                                                               |
| TINYTEXT, TEXT, MEDIUMTEXT,<br>LONGTEXT, ENUM(... ) et SET(... )                              | java.lang.String                                                                                                                                     |
| CHAR( <i>M</i> ), VARCHAR( <i>M</i> ) [BINARY]                                                | java.lang.String (si BINARY, byte[]).                                                                                                                |
| DECIMAL[( <i>n[,p</i> ])]                                                                     | java.math.BigDecimal                                                                                                                                 |
| DATE                                                                                          | java.sql.Date                                                                                                                                        |
| YEAR[(2 4)]                                                                                   | java.sql.Date (date initialisée au 1 <sup>er</sup> Janvier 0h)                                                                                       |
| TIME                                                                                          | java.sql.Time                                                                                                                                        |
| DATETIME, TIMESTAMP[( <i>n</i> )]                                                             | java.sql.Timestamp                                                                                                                                   |



Il est possible de connaître le type de la variable (ou de l'objet Java) que vous devez utiliser dans votre programme JDBC pour travailler avec une colonne *col* d'une table *tab*. Pour cela, utiliser la méthode Class getClass() appliquée à l'objet résultant de l'extraction de la colonne, dans le ResultSet, par getObject("col"). Une fois cette classe instanciée, il suffit d'utiliser String getName() pour trouver son nom.

Le code suivant (*CorresTypes.java*) présente la manière d'extraire le type Java nécessaire pour travailler avec la colonne *cap* de la table *Avion* (ici SMALLINT).

Tableau 8-11 Déduction du type Java à utiliser

| Code Java                                                                                                                                                                                                                                                                                                                                                                          | Commentaires                                                                                                                                                         |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>ResultSet curseurJava = etatSimple.executeQuery         ("SELECT cap FROM Avion LIMIT 1"); while (curseurJava.next ()) {     Object obj = curseurJava.getObject("cap");     System.out.println("Valeur : "         +curseurJava.getObject("cap"));     Class clJava = obj.getClass();     System.out.println ("Classe Java équivalente : "         +clJava.getName());}</pre> | Extraction dans un curseur de la capacité du premier avion.<br>Ouverture et lecture du curseur.<br>Extraction de l'objet Java équivalent.<br>Déduction de sa classe. |

La trace de ce programme est la suivante :

```
| Valeur : 148
| Classe Java équivalente : java.lang.Integer
```

## Manipulations avec la base

Détaillons à présent les différents scénarios que l'on peut rencontrer lors d'une manipulation de la base de données par un programme Java. Les tableaux suivants répertorient les conséquences les plus fréquentes. Les autres cas (relatifs aux contraintes référentielles et aux problèmes de syntaxe) seront étudiés dans la section *Traitement des exceptions*.

## Suppression de données

Tableau 8-12 Enregistrements présents dans la table

| Code Java                                    | Résultat                                                                                     |
|----------------------------------------------|----------------------------------------------------------------------------------------------|
| etat.executeUpdate("DELETE FROM Avion");     | Fait la suppression et passe en séquence.                                                    |
| j = etat.executeUpdate("DELETE FROM Avion"); | Fait la suppression, affecte à j le nombre d'enregistrements supprimés et passe en séquence. |

Tableau 8-13 Aucun enregistrement dans la table

| Code Java                                                 | Résultat                                                            |
|-----------------------------------------------------------|---------------------------------------------------------------------|
| <code>etat.executeUpdate("DELETE FROM Avion");</code>     | Aucune action sur la base et passe en séquence.                     |
| <code>j = etat.executeUpdate("DELETE FROM Avion");</code> | Aucune action sur la base, affecte à valeur 0 et passe en séquence. |

## Ajout d'enregistrements

Tableau 8-14 Différentes écritures d'un INSERT

| Code Java                                                                                                | Résultat                                                   |
|----------------------------------------------------------------------------------------------------------|------------------------------------------------------------|
| <code>etat.executeUpdate("INSERT INTO Compagnie<br/>VALUES ('TAF', 'Toulouse Air Free')");</code>        | Fait l'insertion et passe en séquence.                     |
| <code>int j= etat.executeUpdate("INSERT INTO Compagnie<br/>VALUES ('TAF', 'Toulouse Air Free')");</code> | Fait l'insertion, affecte à j le nombre passé en séquence. |

## Modification d'enregistrements

Tableau 8-15 Différentes écritures d'un UPDATE

| Code Java                                                                                                           | Résultat                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>etat.executeUpdate("UPDATE Compagnie SET<br/>nomComp = 'Air France Compagny' WHERE comp =<br/>'AF' ");</code> | Fait la modification et passe en séquence. Si aucun enregistrement concerné, aucune exception n'est levée.                                      |
| <code>int j= etat.executeUpdate("UPDATE Avion SET<br/>capacite=capacite*1.2");</code>                               | Fait la (les) modification(s), affecte à j le nombre d'enregistrements modifiés et passe en séquence (0 si aucun enregistrement n'est modifié). |

## Extraction de données

Étudions ici la gestion des résultats d'une instruction SELECT.



Le résultat d'une requête est placé dans un objet de l'interface `ResultSet` qui s'apparente à un curseur Java.

Le tableau suivant présente les principales méthodes disponibles de l'interface `ResultSet`. Les méthodes relatives aux curseurs navigables seront étudiées par la suite. Le parcours du curseur s'opère par la méthode `next`. Initialement (après création et chargement du curseur) il est positionné avant la première ligne. Bien qu'un objet de l'interface `ResultSet` soit automatiquement fermé quand son état est fermé ou recréé, il est préférable de le fermer explicitement par la méthode `close` s'il ne doit pas être réutilisé.

Tableau 8-16 Méthodes principales de l'interface ResultSet

| Méthode                         | Description                                                                                                                                                                                   |
|---------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| boolean next()                  | Charge l'enregistrement suivant en renvoyant true, renvoie false lorsqu'il n'y a plus d'enregistrement suivant.                                                                               |
| void close()                    | Ferme le curseur.                                                                                                                                                                             |
| getxxx(int)                     | Récupère, au niveau de l'enregistrement, la valeur de la colonne numérotée de type xxx. Exemple : getInt(1), getString(1), getDate(1), etc., pour récupérer la valeur de la première colonne. |
| updatexxx(...)                  | Modifie, au niveau de l'enregistrement, la valeur de la colonne numérotée de type xxx. Exemple : updateInt(1,i), updateString(1,nom), etc.                                                    |
| ResultSetMetaData getMetaData() | Retourne un objet ResultSetMetaData correspondant au curseur.                                                                                                                                 |
| Object getObject(String)        | Retourne la valeur de la colonne désignée par le paramètre dans une variable Java de type adéquat.                                                                                            |

Distinguons l'instruction SELECT qui génère un curseur statique (objet Resultset utilisant sans option particulière) de celle qui produit un curseur navigable ou modifiable (objet Resultset employé avec des options disponibles depuis la version 2 de JDBC).

## Curseurs statiques

Le code suivant (SELECTstatique.java) extrait les avions de la compagnie 'Air France' par l'intermédiaire du curseur curseurJava. Notez l'utilisation des différentes méthodes get pour récupérer des valeurs issues de colonnes.

Tableau 8-17 Extraction de données dans un curseur statique

| Code Java                                                | Commentaires                       |
|----------------------------------------------------------|------------------------------------|
| try { ...                                                |                                    |
| Statement etatSimple = cx.createStatement();             | Création de l'état.                |
| ResultSet curseurJava =                                  | Création et chargement du curseur. |
| etatSimple.executeQuery("SELECT immat, cap FROM Avion    |                                    |
| WHERE comp = (SELECT comp FROM Compagnie WHERE nom-      |                                    |
| Comp='Air France')");                                    |                                    |
| float moyenneCapacité =0;                                |                                    |
| int nbAvions = 0;                                        |                                    |
| while (curseurJava.next())                               | Parcours du curseur.               |
| {System.out.print("Immat : "+curseurJava.getString(1));  |                                    |
| System.out.println("Capacité : "+curseurJava.getInt(2)); | Extraction de colonnes.            |
| moyenneCapacité += curseurJava.getInt(2);                |                                    |
| nbAvions ++; }                                           |                                    |
| moyenneCapacité /= nbAvions;                             |                                    |
| System.out.println("Capacité moy : "+moyenneCapacité);   |                                    |
| curseurJava.close();                                     | Fermeture du curseur.              |
| } catch(SQLException ex) { ... }                         | Gestion des erreurs.               |

# 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 dans la méthode `createStatement` :

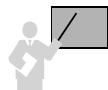
```
Statement createStatement(int typeCurseur, int modifCurseur)
```

## Constantes

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

Tableau 8-18 Constantes de navigation d'un curseur

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



Un curseur est sensible dès que des mises à jour de la table sont automatiquement répétées 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 partagent le sens de parcours pour tous les curseurs de l'état donné.

Tableau 8-19 Méthodes de navigation dans un curseur

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

## Parcours

Le code suivant (`SELECTnavigable.java`) présente une utilisation du curseur navigable `curseurNaviJava`. Le deuxième test renvoie `false`, car, après l'ouverture, le curseur n'est pas positionné sur le premier enregistrement, et la méthode `next` le place selon le parcours du curseur.

Tableau 8-20 Parcours d'un curseur navigable

| Web | Code Java                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Commentaire                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>try {<br/>    Statement etatSimple =createStatement<br/>    (ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);<br/><br/>    ResultSet curseurNaviJava = etatSimple.executeQuery("SELECT<br/>    immat,typeAvion,cap FROM Avion");<br/>    if ( curseurNaviJava.isBeforeFirst() )<br/>        System.out.println("Curseur positionné au début");<br/>    if ( curseurNaviJava.isFirst() )<br/>        System.out.println("Curseur positionné sur le 1er déjà");<br/>    while( curseurNaviJava.next() )<br/>        {if ( curseurNaviJava.isFirst() )<br/>            System.out.println("1er avion : ");<br/>        if ( curseurNaviJava.isLast() )<br/>            System.out.println("Dernier avion : ");<br/>        System.out.print("Immat: "+curseurNaviJava.getString(1));<br/>        System.out.println(" type : "+curseurNaviJava.getString(2));}<br/>    if ( curseurNaviJava.isAfterLast() )System.out.println("Curseur<br/>    positionné après la fin");<br/>    if ( curseurNaviJava.previous() )<br/>        if ( curseurNaviJava.previous() )<br/>            {System.out.println("Avant dernier avion : "+<br/>            curseurNaviJava.getString(1));}<br/>    if ( curseurNaviJava.first() )<br/>        {System.out.println("First avion : "+<br/>        curseurNaviJava.getString(1));}<br/>    if ( curseurNaviJava.last() )<br/>        {System.out.println("Last avion : "+<br/>        curseurNaviJava.getString(1));}<br/>    curseurNaviJava.close();<br/>}<br/>} catch(SQLException ex) { ... }<br/></pre> | <p>Création de l'état.</p> <p>Création et chargement du curseur.</p> <p>Test renvoyant true.</p> <p>Test renvoyant false.</p> <p>Parcours du curseur en chant les premiers et derniers enregistrements.</p> <p>Test renvoyant true.</p> <p>Affiche l'avant dernier enregistrement.</p> <p>Affiche le premier enregistrement.</p> <p>Affiche le dernier enregistrement.</p> <p>Ferme le curseur.</p> <p>Gestion des erreurs.</p> |



Créez des curseurs non navigables quand vous voulez rapatrier de très gros volumes de données (taille du cache limitative côté client). Fragmentez vos requêtes quand vous voulez empêcher des curseurs navigables. Les prochaines versions de MySQL et *Connector/J* devraient prendre en charge une gestion côté serveur des curseurs navigables.

## Positionnements

Des méthodes assurent l'accès direct à un curseur navigable. Notez que `absolute(1)` équivaut à `first()`, de même `absolute(-1)` équivaut à `last()`. Concernant la méthode `relative`, il faut l'utiliser dans un test pour s'assurer qu'elle s'applique à un enregistrement existant (voir l'exemple suivant). D'autre part, l'utilisation de `relative(0)` n'a aucun effet. Considérons la table suivante qui est interrogée au niveau des trois premières colonnes par curseur navigable `curseurPosJava` :

Figure 8-5 Curseur navigable

| Avion          |        |           |     |      |
|----------------|--------|-----------|-----|------|
|                | immat  | typeAvion | cap | comp |
| absolute(1) →  | F-FGFB | Concorde  | 95  | AF   |
| relative (2) → | F-GKUB | A330      | 240 | AERI |
|                | F-GLFS | A320      | 140 | TAT  |
|                | F-GLKT | A340      | 300 | AERI |
|                | F-GLZV | A330      | 250 | AERI |
| absolute(-1) → | F-WTSS | Concorde  | 90  | AF   |
| curseurPosJava |        |           |     |      |

Le code suivant (`SELECTPositions.java`) présente les méthodes qui permettent d'accéder directement à des enregistrements de ce curseur :

Tableau 8-21 Positionnements dans un curseur navigable

| Web | Code Java                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Commentaires                                                                                                                                                                                     |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | try {<br>Statement etatSimple =createStatement<br>( <b>ResultSet.TYPE_SCROLL_INSENSITIVE</b> ,<br><b>ResultSet.CONCUR_READ_ONLY</b> );<br>ResultSet curseurPosJava =<br>etatSimple.executeQuery("SELECT immat,typeAvion,cap<br>FROM Avion");<br><b>curseurPosJava.absolute(1);</b><br>if ( <b>curseurPosJava.relative(2)</b> )<br>System.out.println("relative(2): "+<br><b>curseurPosJava.getString(1)</b> );<br>else<br>System.out.println("Pas de 3ème avion!");<br>if ( <b>curseurPosJava.relative(-2)</b> )<br>System.out.println("relative(-2) : "+<br><b>curseurPosJava.getString(1)</b> );<br>else<br>System.out.println("Pas retour -2 possible !"); | Création de l'état avec curseurs insensibles et non modifiables.<br>Création et chargement du curseur.<br>Curseur sur le premier avion.<br>Accès au troisième avion.<br>Retour au premier avion. |

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

| Code Java                                                                                                                                                                                                                                                                                                                               | Commentaires                            |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------|
| if ( <b>curseurPosJava.absolute(-2)</b><br>System.out.println("absolute(-2) : "+<br>curseurPosJava.getString(1));<br>else<br>System.out.println("Pas d'avant dernier avion");<br><b>curseurPosJava.afterLast();</b><br>while( <b>curseurPosJava.previous()</b> ) { ... }<br>curseurPosJava.close();<br>} catch(SQLException ex) { ... } | Accès à l'avant-dernier enregistrement. |
|                                                                                                                                                                                                                                                                                                                                         | Parcours du curseur en inverse.         |
|                                                                                                                                                                                                                                                                                                                                         | Fermeture du curseur.                   |
|                                                                                                                                                                                                                                                                                                                                         | 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                      |
|-----------------------------------------|----------------------------------|
| <code>ResultSet.CONCUR_READ_ONLY</code> | Le curseur ne peut être modifié. |
| <code>ResultSet.CONCUR_UPDATABLE</code> | 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 sur une 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 exposer des colonnes (les fonctions monolignes et multilignes sont interdites).
- Écrivez « `SELECT a.* FROM table a...` » à la place de « `SELECT * FROM table...` ».

Il est aussi possible de définir un curseur par une requête de type `SELECT...` `FOR UPDATE`. Les principales méthodes relatives aux curseurs modifiables sont les suivantes :

Tableau 8-23 Méthodes de navigation dans un curseur

| Méthode                                    | Fonction                                                                                                                                             |
|--------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>int getResultSetType()</code>        | Renvoie le caractère navigable des curseurs d'un état donné ( <code>ResultSet.TYPE_FORWARD_ONLY...</code> ).                                         |
| <code>int getResultSetConcurrency()</code> | Renvoie le caractère modifiable des curseurs d'un état donné ( <code>ResultSet.CONCUR_READ_ONLY</code> ou <code>ResultSet.CONCUR_UPDATABLE</code> ). |
| <code>int getType()</code>                 | Renvoie le caractère navigable d'un curseur donné.                                                                                                   |
| <code>int getConcurrency()</code>          | Renvoie le caractère modifiable d'un curseur donné.                                                                                                  |
| <code>void deleteRow()</code>              | Supprime l'enregistrement courant.                                                                                                                   |
| <code>void updateRow()</code>              | Modifie la table avec l'enregistrement courant.                                                                                                      |
| <code>void cancelRowUpdates()</code>       | Annule les modifications faites sur l'enregistrement courant.                                                                                        |
| <code>void moveToInsertRow()</code>        | Déplace le curseur vers un nouvel enregistrement.                                                                                                    |
| <code>void insertRow()</code>              | Insère dans la table l'enregistrement courant.                                                                                                       |
| <code>void moveToCurrentRow()</code>       | Retour vers l'enregistrement courant (à utiliser éventuellement après <code>moveToInsertRow()</code> ).                                              |

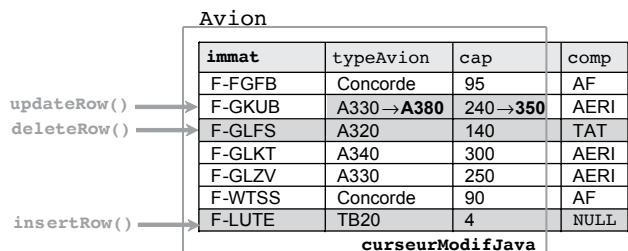


Les opérations de modification et d'insertion (`UPDATE` et `INSERT`) à travers un curseur se réalisent en deux temps : mise à jour du curseur puis propagation à la table de la base de données. Il suffit ainsi de ne pas exécuter la deuxième étape pour ne pas opérer la mise à jour de la base.

La suppression d'enregistrements (`DELETE`) à travers un curseur s'opère en une seule instruction qui n'est pas forcément validée par la suite : il faudra programmer explicitement le `commit` ou laisser le paramètre d'autocommit à `true` (par défaut).

La figure suivante illustre les modifications effectuées sur la table `Avion` par l'intermédiaire du curseur `CurseurModifJava` utilisé par les trois programmes Java suivants :

Figure 8-6 Mises à jour d'un curseur



## Suppressions

Le code suivant (`ResultDELETE.java`) supprime le troisième enregistrement du curseur et répercute la mise à jour au niveau de la table `Avion` du schéma connecté. Nous déclarons ce curseur « navigable » :

Tableau 8-24 Suppression d'un enregistrement

| Web | Code Java                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Commentaires                                                                                                                                                                                                         |
|-----|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>try {...<br/>    Statement etatSimple =<br/>        cx.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,<br/>                           ResultSet.CONCUR_UPDATABLE);<br/>    cx.setAutoCommit(false);<br/><br/>    ResultSet curseurModifJava = etatSimple.executeQuery<br/>        ("SELECT immat,typeAvion,cap FROM Avion");<br/>    if (curseurModifJava.absolute(3))<br/>    { curseurModifJava.deleteRow();<br/>        cx.commit(); }<br/>    else<br/>        System.out.println("Pas de 3ème avion!");<br/>    curseurModifJava.close();<br/>} catch(SQLException ex) { ... }</pre> | Création de l'état<br>désactivation de la transaction automatique<br>Création du curseur<br><br>Accès direct au deuxième avion, suppression de l'enregistrement.<br><br>Fermeture du curseur<br>Gestion des erreurs. |

Le code suivant (`ResultDELETE2.java`) supprime le même enregistrement en supposant son indice a priori inconnu. Nous déclarons ici ce curseur « non navigable ». Notez l'utilisation de la méthode `equals` pour comparer deux chaînes de caractères :

Tableau 8-25 Suppression d'un enregistrement

| Web | Code Java                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | Commentaires                                                                                                                                                                                  |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>try {...<br/>    Statement etatSimple =<br/>        cx.createStatement(ResultSet.TYPE_FORWARD_ONLY,<br/>                           ResultSet.CONCUR_UPDATABLE);<br/>    cx.setAutoCommit(false);<br/><br/>    ResultSet curseurModifJava = etatSimple.executeQuery<br/>        ("SELECT immat,typeAvion,cap FROM Avion");<br/>    String p_immat = "F-GLFS";<br/>    while(curseurModifJava.next())<br/>    {if (curseurModifJava.getString(1).equals(p_immat))<br/>        {curseurModifJava.deleteRow(); ; }<br/>    }<br/>    curseurModifJava.close();<br/>} catch(SQLException ex) { ... }</pre> | Création de l'état<br>désactivation de la transaction automatique<br>Création du curseur.<br><br>Accès à l'enregistrement et suppression.<br><br>Fermeture du curseur<br>Gestion des erreurs. |

## Modifications

La modification de colonnes d'un enregistrement au niveau de la base de données s'opère en deux étapes : mise à jour du curseur par les méthodes `updateXXX` (*update methods*), puis propagation des mises à jour dans la table par la méthode `updateRow()`.

Les méthodes `updateXXX` ont chacune deux signatures. Par exemple, la méthode de modification d'une chaîne de caractères (valable pour les colonnes CHAR et VARCHAR) est disponible en raisonnant en fonction soit de la position soit du nom de la colonne du curseur :

```
void updateString(int positionColonne, String chaîne)
void updateString(String nomColonne, String chaîne)
```

Le code suivant (`ResultUPDATE.java`) change, au niveau de la table `Avion`, deux colonnes du deuxième enregistrement du curseur. Nous déclarons ici ce curseur « sensible » pour pouvoir éventuellement visualiser la transformation réalisée dans le même programme.

Tableau 8-26 Modifications d'un enregistrement

| Web | Code Java                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Commentaires                                                                                                                                                                                                                 |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>try ... Statement etatSimple =     cx.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,                       ResultSet.CONCUR_UPDATABLE); cx.setAutoCommit(false);  ResultSet curseurModifJava = etatSimple.executeQuery     ("SELECT immat,typeAvion,cap FROM Avion"); if (curseurModifJava.absolute(2)) { curseurModifJava.updateString(2,"A380");   curseurModifJava.updateInt(3,350);    curseurModifJava.updateRow();   cx.commit(); } else   System.out.println("Pas de 2ème avion!"); curseurModifJava.close(); } catch(SQLException ex) { ... }</pre> | Création de l'état et désactivation de la validation automatique.<br>Création du curseur.<br>Accès à l'enregistrement.<br>Première étape.<br>Deuxième étape.<br>Validation.<br>Fermeture du curseur.<br>Gestion des erreurs. |

## Insertions

L'insertion d'un enregistrement au niveau de la base de données s'opère en trois étapes : préparation à l'insertion dans le curseur par la méthode `moveToInsertRow`, mise à jour du curseur par les méthodes `updateXXX`, puis propagation des actualisations dans la table par la méthode `insertRow`. L'éventuel retour à l'enregistrement courant se programme à l'aide de la méthode `moveToCurrentRow`.

Le code suivant (`ResultINSERT.java`) insère un nouvel enregistrement au niveau de la table `Avion`. La quatrième colonne de la table n'est pas indiquée dans le curseur, elle est

donc passée à NULL au niveau de la table, en l'absence de valeur par défaut définie dans la colonne.

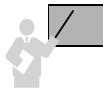
Tableau 8-27 Insertion d'un enregistrement

| Code Java                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | Commentaires                                                                                                                                                                                                                                                                 |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>Web try {...<br/>    Statement etatSimple =<br/>        cx.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,<br/>                           ResultSet.CONCUR_UPDATABLE);<br/>    cx.setAutoCommit(false);<br/><br/>    ResultSet curseurModifJava = etatSimple.executeQuery<br/>        ("SELECT immat,typeAvion,cap FROM Avion");<br/>    curseurModifJava.moveToInsertRow();<br/><br/>    curseurModifJava.updateString(1,"F-LUTE");<br/>    curseurModifJava.updateString(2,"TB20");<br/>    curseurModifJava.updateInt(3,4);<br/><br/>    curseurModifJava.insertRow();<br/>    cx.commit();<br/>    curseurModifJava.close();<br/><br/>} catch(SQLException ex) { ... }</pre> | <p>Création de l'état de curseur.<br/>Désactivation de la transaction automatique.</p> <p>Création du curseur modifiable.</p> <p>Première étape.</p> <p>Deuxième étape.</p> <p>Troisième étape.<br/>Validation.</p> <p>Fermeture du curseur.</p> <p>Gestion des erreurs.</p> |

## Gestion des séquences

Avant la version 3.0 de l'API JDBC de Sun, il n'y avait pas de possibilité « standard » d'extraire la valeur courante d'une séquence (colonne AUTO\_INCREMENT). Avec des anciens pilotes JDBC pour MySQL, il était possible d'utiliser une méthode spécifique de l'interface Statement, ou d'exécuter une requête du type SELECT LAST\_INSERT\_ID() après avoir inséré un enregistrement. Le premier mécanisme n'assure pas la portabilité, le second n'est pas efficace, puisqu'il oblige à insérer une ligne au préalable (qu'on peut toutefois annuler avec un rollback).

À présent, JDBC 3.0 offre deux nouveaux mécanismes afin d'extraire la valeur courante d'une séquence AUTO\_INCREMENT : la méthode getGeneratedKeys() ou l'exploitation d'un curseur modifiable via la méthode insertRow().



Dans tous les cas, il n'est pas nécessaire d'insérer réellement un nouvel enregistrement (action possible avec rollback). En revanche, l'exécution de l'ajout (par INSERT) est obligatoire pour que MySQL évalue une nouvelle valeur de la séquence. Cette action est irréversible dans le sens où la séquence sera incrémentée qu'on valide ou non l'ajout de l'enregistrement.

## Méthode getGeneratedKeys

Le code suivant (`Sequence1.java`) insère un nouvel enregistrement dans la table `Affreter` (décrise au chapitre 2, section *Séquences*) et récupère la valeur courante de la séquence à l'aide de la méthode `getGeneratedKeys()`.

Tableau 8-28 Récupération d'une séquence à l'aide d'un état

| Web | Code Java                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Commentaires                                                                                                                                                                                                                                                       |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>try {<br/>    Statement etat = cx.createStatement<br/>        (java.sql.ResultSet.TYPE_FORWARD_ONLY,<br/>         java.sql.ResultSet.CONCUR_UPDATABLE);<br/>    cx.setAutoCommit(false);<br/><br/>    etat.execute("INSERT INTO bdsoutou.Affreter<br/>    (comp, immat, dateAff, nbPax)VALUES('AF', 'A1', NOW(), 100)",<br/>    Statement.RETURN_GENERATED_KEYS);<br/><br/>    ResultSet curseur = etat.getGeneratedKeys();<br/>    if (curseur.next())<br/>        System.out.println("Valeur de la sequence "+<br/>                           curseur.getInt(1));<br/>    else System.out.println("Pb sequence ");<br/>    cx.rollback();<br/>    curseur.close(); etat.close(); cx.close();<br/>}<br/>catch(SQLException ex) { ... }</pre> | <p>Création de l'état et désactivation de la validation automatique.</p> <p>Insertion avec l'option de récupération de clé générée.</p> <p>Extraction de la séquence.</p> <p>Invalidation de l'ajout.</p> <p>Fermeture des objets.</p> <p>Gestion des erreurs.</p> |

## Curseur modifiable

Le code suivant (`Sequence2.java`) insère un nouvel enregistrement dans la table `Affreter` par un curseur modifiable, et récupère la valeur courante de la séquence à l'aide de la méthode `getInt()` appliquée à la colonne `AUTO_INCREMENT`.

Tableau 8-29 Récupération d'une séquence à l'aide d'un curseur modifiable

| Web | Code Java                                                                                                                                                                                           | Commentaires                                                             |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|
|     | <pre>try {<br/>    Statement etat = cx.createStatement<br/>        (java.sql.ResultSet.TYPE_FORWARD_ONLY,<br/>         java.sql.ResultSet.CONCUR_UPDATABLE);<br/>    cx.setAutoCommit(false);</pre> | <p>Création de l'état et désactivation de la validation automatique.</p> |

Tableau 8-29 Récupération d'une séquence à l'aide d'un curseur modifiable (suite)

| Code Java                                                                                               | Commentaires              |
|---------------------------------------------------------------------------------------------------------|---------------------------|
| ResultSet curseur = etat.executeQuery("SELECT<br>numAff, immat, comp, dateAff FROM bdsoutou.Affreter"); |                           |
| curseur.moveToInsertRow();                                                                              | Insertion via le curseur  |
| curseur.insertRow();                                                                                    |                           |
| curseur.last();                                                                                         | Extraction de la séquence |
| System.out.println("Valeur de la sequence "+<br>curseur.getInt(1));                                     |                           |
| cx.rollback();                                                                                          |                           |
| curseur.close(); etat.close(); cx.close();                                                              | Invalidation de l'ajout.  |
| } catch(SQLException ex) { ... }                                                                        | Fermeture des objets.     |
|                                                                                                         | Gestion des erreurs.      |

## Interface ResultSetMetaData

L'interface `ResultSetMetaData` est utile pour retrouver dynamiquement des propriétés tables qui sont manipulées par des curseurs `ResultSet`. Cette interface est intéressante pour programmer dynamiquement des requêtes ou d'autres instructions SQL. Ces fonctions extraire de manière transparente des informations par l'intermédiaire du dictionnaire des données.

Une fois un curseur `ResultSet` programmé, il suffit de lui appliquer la méthode `getMetaData()` pour disposer d'un objet `ResultSetMetaData`. Le tableau suivant présente les principales méthodes disponibles de l'interface `ResultSetMetaData` :

Tableau 8-30 Méthodes principales de l'interface `ResultSetMetaData`

| Méthode                                    | Description                                                                                                                                                                                                                                                        |
|--------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>int getColumnCount()</code>          | Retourne le nombre de colonnes du curseur.                                                                                                                                                                                                                         |
| <code>String getColumnName(int)</code>     | Retourne le nom de la colonne d'un indice donné du curseur.                                                                                                                                                                                                        |
| <code>int getColumnType(int)</code>        | Retourne le code du type (selon la classification <code>java.sql.Types</code> ) de la colonne d'un indice donné du curseur.                                                                                                                                        |
| <code>String getColumnTypeName(int)</code> | Retourne le nom du type SQL de la colonne d'un indice donné du curseur.                                                                                                                                                                                            |
| <code>int isNullable(int)</code>           | Indique si la colonne d'un indice donné du curseur peut contenir une valeur nulle ( constantes renvoyées : <code>ResultSetMetaData.columnNoNulls</code> , <code>ResultSetMetaData.columnNullable</code> ou <code>ResultSetMetaData.columnNullableUnknown</code> ). |
| <code>int getPrecision(int)</code>         | Nombre de chiffres avant la virgule de la colonne désignée.                                                                                                                                                                                                        |
| <code>int getScale(int)</code>             | Nombre de décimales de la colonne désignée.                                                                                                                                                                                                                        |
| <code>String getSchemaName(int)</code>     | Nom du schéma propriétaire de la colonne.                                                                                                                                                                                                                          |
| <code>String getTableName(int)</code>      | Nom de la table de la colonne.                                                                                                                                                                                                                                     |



Comme nous l'avons vu au chapitre 5, MySQL ne renseigne pas encore le nom de catalogue. Ainsi, la méthode `getSchemaName()` n'est pas encore reconnue.

Le code suivant (`ResultSetMeta.java`) utilise des méthodes de l'interface `ResultSetMetaData` sur la base de la requête extrayant trois colonnes dans la table `Avion` :

**Tableau 8-31 Extraction de métainformations au niveau d'un curseur**

| Web | Code Java                                                                                                            | Commentaires                                                      |
|-----|----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
|     | try { ...                                                                                                            | Création du curseur.                                              |
|     | ResultSet curseurJava=etatSimple.executeQuery<br>("SELECT <b>immat</b> , <b>typeAvion</b> , <b>cap</b> FROM Avion"); |                                                                   |
|     | ResultSetMetaData rsmd =<br><b>curseurJava.getMetaData();</b>                                                        | Création d'un objet<br>ResultSetMetaData.<br>nbCol contient 3.    |
|     | int nbCol = rsmd.getColumnCount();                                                                                   | nom2emeCol contient typeAvion.                                    |
|     | String nom2emeCol = rsmd.getColumnName(2);                                                                           | type2emeCol contient VARCHAR.                                     |
|     | String type2emeCol = rsmd.getColumnTypeName(2);                                                                      | codeType2emeCol contient 12<br>(code pour VARCHAR).               |
|     | int codeType2emeCol = rsmd.getColumnType(2);                                                                         |                                                                   |
|     | if (rsmd.isNullable(1) ==<br>ResultSetMetaData.columnNoNulls)                                                        | Test renvoyant vrai (la première<br>colonne est la clé primaire). |
|     | ..."                                                                                                                 |                                                                   |
|     | int p1 = rsmd.getPrecision(3);                                                                                       | Taille de la colonne cap (renvoie 6<br>pour un SMALLINT).         |
|     | int t1 = rsmd.getScale(3);                                                                                           | Décimales de la colonne cap<br>(renvoie 0 pour un SMALLINT).      |
|     | curseurJava.close();                                                                                                 | Fermeture du curseur.                                             |
|     | } catch(SQLException ex) { ... }                                                                                     | Gestion des erreurs.                                              |

## Interface DatabaseMetaData

L'interface `DatabaseMetaData` est utile pour connaître des aspects plus généraux de base de données cible (version, éditeur, prise en charge des transactions...) ou des informations sur la structure de la base (structures des tables et vues, prérogatives...).

Plus de quarante méthodes sont proposées par l'interface `DatabaseMetaData`. Le tableau suivant en présente quelques-unes. Consultez la documentation du JDK pour en savoir plus.

**Tableau 8-32 Méthodes principales de l'interface ResultSetMetaData**

| Méthode                                                           | Description                                                       |
|-------------------------------------------------------------------|-------------------------------------------------------------------|
| <code>ResultSet getColumns(String, String, String, String)</code> | Description de toutes les colonnes d'une table d'un schéma donné. |
| <code>String getDatabaseProductName()</code>                      | Nom de l'éditeur de la base de données utilisée.                  |
| <code>String getDatabaseProductVersion()</code>                   | Numéro de la version de la base utilisée.                         |

Tableau 8-32 Méthodes principales de l'interface ResultSetMetaData (suite)

| Méthode                                               | Description                                                 |
|-------------------------------------------------------|-------------------------------------------------------------|
| ResultSet getTables(String, String, String, String[]) | Description des tables d'un schéma donné.                   |
| String getUserName()                                  | Nom de l'utilisateur connecté (schéma courant).             |
| boolean supportsSavepoints()                          | Renvoie true si la base reconnaît les points de validation. |
| boolean supportsTransactions()                        | Renvoie true si la base reconnaît les transactions.         |

Le code suivant (`MetaData.java`) emploie ces méthodes pour extraire des informations propos de la base cible et des objets (tables, vues, séquences...) du schéma courant.

Tableau 8-33 Extraction de métainformations au niveau d'un schéma

| Code Java                                                                                                                                                                                                                                                                                                                                       | Commentaires                                                                                                                                                  |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  Web                                                                                                                                                                                                                                                           |                                                                                                                                                               |
| try {<br>...<br>DatabaseMetaData infoBase = cx.getMetaData();                                                                                                                                                                                                                                                                                   | Création d'un objet DatabaseMetaData.                                                                                                                         |
| ResultSet toutesLesTables = infoBase.getTables("",<br>infoBase.getUserName(), null, null);                                                                                                                                                                                                                                                      |                                                                                                                                                               |
| System.out.println("Objets du schema "+<br>infoBase.getUserName());                                                                                                                                                                                                                                                                             | Création d'un objet ResultSet contenant caractéristiques du schéma courant.                                                                                   |
| while (toutesLesTables.next())<br>{ System.out.print("Nom de l'objet: "+<br>toutesLesTables.getString(3));<br>System.out.println(" Type : "+<br>toutesLesTables.getString(4)); }<br>System.out.println("Nom base : "+<br>infoBase.getDatabaseProductName());<br>System.out.println("Version base : "+<br>infoBase.getDatabaseProductVersion()); | Parcours du curseur en échantillonnant quelques caractéristiques.                                                                                             |
| if (infoBase.supportsTransactions())<br>System.out.println("Supporte les Transactions");<br>toutesLesTables.close();<br>} catch(SQLException ex) { ... }                                                                                                                                                                                        | Affichage du nom de la base.<br>Affichage de la version de la base.<br>Transactions prises en charge ou pas.<br>Fermeture du curseur.<br>Gestion des erreurs. |

La trace de ce programme est la suivante (dans notre jeu d'exemples) :

```
Objets du schema soutou@localhost
Nom de l'objet: Avion Type : TABLE
Nom de l'objet: Compagnie Type : TABLE
Nom base : MySQL
Version base : 5.0.15-nt
Supporte les SelectForUpdate
Supporte les Transactions
```

L'interface `PreparedStatement` hérite de l'interface `Statement`, et la spécialise en permettant de paramétriser des objets (états préparés) représentant des instructions SQL précompilées. Ces états sont créés par la méthode `prepareStatement` de l'interface `Connection` décrite ci-après. La chaîne de caractères contient l'ordre SQL dont les paramètres, s'il en possède, doivent être indiqués par le symbole « ? ».

## ■ `PreparedStatement prepareStatement(String)`

Une fois créés, ces objets peuvent être aisément réutilisés pour exécuter à la demande l'instruction SQL, en modifiant éventuellement les valeurs des paramètres d'entrée à l'aide des méthodes `setxxx` (*setter methods*). Le tableau suivant décrit les principales méthodes de l'interface `PreparedStatement` :

Tableau 8-34 Méthodes de l'Interface `PreparedStatement`

| Méthode                               | Description                                                                                                                                                         |
|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ResultSet executeQuery()</code> | Exécute la requête et retourne un curseur ni navigable, ni modifiable par défaut.                                                                                   |
| <code>int executeUpdate()</code>      | Exécute une instruction LMD (INSERT, UPDATE ou DELETE) et retourne le nombre de lignes traitées, ou 0 pour les instructions SQL ne retournant aucun résultat (LDD). |
| <code>boolean execute()</code>        | Exécute une instruction SQL et renvoie <code>true</code> , si c'est une instruction SELECT, <code>false</code> sinon.                                               |
| <code>void setNull(int, int)</code>   | Affecte la valeur <code>NULL</code> au paramètre de numéro et de type (classification <code>java.sql.Types</code> ) spécifiés.                                      |
| <code>void close()</code>             | Ferme l'état.                                                                                                                                                       |

Décrivons à présent un exemple d'appel pour chaque méthode de compilation d'un ordre paramétré. On suppose la connexion `cx` créée :

## Extraction de données (`executeQuery`)

Le code suivant (`PrepareSELECT.java`) illustre l'utilisation de la méthode `executeQuery` pour extraire les enregistrements de la table `Avion` :

Tableau 8-35 Extraction de données par un ordre préparé

| Web | Code Java                                                                                                                                                                                                                                                                                                                                                          | Commentaires                                                                                                                                                                                                          |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>try { ...     String ordreSQL =         "SELECT immat, typeAvion, cap FROM Avion";     PreparedStatement étatPréparé =         cx.prepareStatement(ordreSQL);     ResultSet curseurJava = étatPréparé.executeQuery();      while(curseurJava.next())         { ... }     curseurJava.close();     étatPréparé.close(); } catch(SQLException ex) { ... }</pre> | <p>Création d'un état préparé.</p> <p>Création du curseur résultant de la compilation de l'état.</p> <p>Parcours du curseur.</p> <p>Fermeture du curseur.</p> <p>Fermeture de l'état.</p> <p>Gestion des erreurs.</p> |

## Mises à jour (executeUpdate)

Le code suivant (`PrepareINSERT.java`) illustre l'utilisation de la méthode `executeUpdate` pour insérer l'enregistrement (F-NEW, A319, 178, AF) dans la table Avion composée de quatre colonnes de types CHAR(6), VARCHAR(15), SMALLINT et VARCHAR(4).

Tableau 8-36 Insertion d'un enregistrement par un ordre préparé

| Web | Code Java                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Commentaires                                                                                                                                                 |
|-----|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
|     | <pre>try { ...     String ordreSQL =         "INSERT INTO Avion VALUES (?, ?, ?, ?)";     PreparedStatement étatPréparé =         cx.prepareStatement(ordreSQL);     étatPréparé.setString(1, "F-NEW");     étatPréparé.setString(2, "A319");     étatPréparé.setInt(3, 178);     étatPréparé.setString(4, "AF");     System.out.println(étatPréparé.executeUpdate()         + " avion inséré.");     étatPréparé.close(); } catch(SQLException ex) { ... }</pre> | <p>Création d'un état préparé.</p> <p>Passage des paramètres.</p> <p>Exécution de l'instruction.</p> <p>Fermeture de l'état.</p> <p>Gestion des erreurs.</p> |

## Instruction LDD (execute)

Le code suivant (`PrepareDELETE.java`) illustre l'utilisation de la méthode `executeDelete` pour supprimer un avion dont l'immatriculation passe en paramètre :

Tableau 8-37 Insertion d'un enregistrement par un ordre préparé

| Code Java                                                                                                                                                                                                                                                                                                                                                                                                                | Commentaires                                                                                                                                               |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> Web try { ...     cx.setAutoCommit(false);     String ordreSQL =         "DELETE FROM Avion WHERE immat = ?";     PreparedStatement étatPréparé =         cx.prepareStatement(ordreSQL);     étatPréparé.setString(1, "F-NEW ");     if (! étatPréparé.execute())         { System.out.println("Enregistrement sup- primé");         cx.commit();     étatPréparé.close(); } catch(SQLException ex) { ... } </pre> | <p>Création d'un état préparé.</p> <p>Passage du paramètre.</p> <p>Exécution de l'instruction.</p> <p>Fermeture de l'état.</p> <p>Gestion des erreurs.</p> |



Il n'est pas possible de paramétriser des instructions SQL du LDD (CREATE, ALTER...). Pour résoudre ce problème, il faut construire dynamiquement la chaîne (String) qui contient l'instruction à l'aide de l'opérateur de concaténation Java (+). Cette chaîne sera ensuite l'unique paramètre de la méthode `prepareStatement`.

## Procédures cataloguées

L'interface `CallableStatement` permet d'appeler des sous-programmes (fonctions ou procédures cataloguées), en passant d'éventuels paramètres en entrée et en en récupérant en sortie. L'interface `CallableStatement` spécialise l'interface `PreparedStatement`. Les paramètres d'entrée sont affectés par les méthodes `setxxxx`. Les paramètres de sortie (définis OUT au niveau du sous-programme) sont extraits à l'aide des méthodes `getxxxx`. Ces états qui permettent d'appeler des sous-programmes sont créés par la méthode `prepareCall` de l'interface `Connection`, décrite ci-après :

■ `CallableStatement prepareCall(String)`

Le tableau suivant décompose le paramètre de cette méthode (deux écritures sont possibles). Chaque paramètre est indiqué par un symbole « ? » :

Tableau 8-38 Paramètre de `prepareCall`

| Type du sous-programme | Paramètre                            |
|------------------------|--------------------------------------|
| Fonction               | {? = call nomFonction( ?, ?, ... ) } |
| Procédure              | {call nomProcédure( ?, ?, ... ) }    |

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 pour analyser les résultats. Le tableau suivant décrit les principales méthodes de l'interface `CallableStatement` :

Tableau 8-39 Méthodes de l'interface `CallableStatement`

| Méthode                                          | Description                                                                                                                                              |
|--------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ResultSet executeQuery()</code>            | Idem <code>PreparedStatement</code> .                                                                                                                    |
| <code>int executeUpdate()</code>                 | Idem <code>PreparedStatement</code> .                                                                                                                    |
| <code>boolean execute()</code>                   | Idem <code>PreparedStatement</code> .                                                                                                                    |
| <code>void registerOutParameter(int, int)</code> | Transfère un paramètre de sortie à un indice donné du type Java (classification <code>java.sql.Types</code> ).                                           |
| <code>boolean wasNull()</code>                   | Détermine si le dernier paramètre de sortie extrait est NULL. Cette méthode doit être seulement invoquée après une méthode de type <code>getxxx</code> . |

## Exemple

Le programme JDBC suivant (`CallableProcedure.java`) décrit l'appel de la procédure `leNomCompagnieEst` (ayant 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'.

Tableau 8-40 Appel d'une procédure (paramètre en entrée et sortie)

| Code Java                                                                  | Commentaires                        |
|----------------------------------------------------------------------------|-------------------------------------|
| String ordreSQL =<br>" {call bdsoutou.leNomCompagnieEst(?,?)}";            | Création d'un état appelable.       |
| CallableStatement étatAppelable =<br>cx.prepareCall(ordreSQL);             | Déclaration du paramètre de sortie. |
| étatAppelable.registerOutParameter<br>(2, java.sql.Types.VARCHAR);         | Passage du paramètre d'entrée.      |
| étatAppelable.setString(1, "F-GLFS");                                      | Exécution de la procédure.          |
| étatAppelable.execute();                                                   |                                     |
| System.out.print("Compagnie de F-GLFS : "+<br>étatAppelable.getString(2)); |                                     |
| étatAppelable.close();                                                     | Extraction du résultat.             |
| } catch(SQLException ex) { ... }                                           | Fermeture de l'état.                |
|                                                                            | Gestion des erreurs.                |

La trace de l'appel de cette procédure est la suivante :

| Compagnie de F-GLFS : Transport Air Tour



Pensez, sous *root*, à donner les priviléges nécessaires à l'utilisateur qui va lancer le sous-programme via Java (pour mon test, j'ai dû lancer GRANT EXECUTE ON bdsoutou.\* TO 'soutou'@'localhost' et GRANT SELECT ON mysql.proc TO 'soutou'@'localhost'). Si vous ne le faites pas, JDBC vous rappellera clairement à l'ordre.

## Transactions

JDBC supporte le mode transactionnel qui consiste à valider tout ou une partie d'un ensemble d'instructions. Nous avons déjà décrit, à la section *Interface Connection*, les méthodes qui permettent à un programme Java de coder des transactions (`setAutoCommit`, `commit` et `rollback`).

Par défaut, chaque instruction SQL est validée (on parle *d'autocommit*). Lorsque ce mode est désactivé, il faut gérer manuellement les transactions avec `commit` ou `rollback`.



Quand le mode *autocommit* est désactivé :

- La déconnexion d'un objet `Connection` (par la méthode `close`) valide implicitement la transaction (même si `commit` n'a pas été invoqué avant la déconnexion).
- Chaque instruction du LDD (CREATE, ALTER, DROP) valide implicitement la transaction.

## Points de validation

Depuis la version 3.0 de JDBC (JDK 1.4), on peut inclure des points de validation et ainsi la programmation des transactions. Les interfaces Connection et Savepoint rendent possible cette programmation.

### Interface Connection

Le tableau suivant présente les méthodes de l'interface Connection qui sont relatives au principe des points de validation :

Tableau 8-41 Méthodes concernant les points de validation de l'interface Connection

| Méthode                                       | Description                                                              |
|-----------------------------------------------|--------------------------------------------------------------------------|
| <code>Savepoint setSavepoint()</code>         | Positionne un point de validation anonyme et retourne l'objet Savepoint. |
| <code>Savepoint setSavepoint(String)</code>   | Positionne un point de validation nommé et retourne l'objet Savepoint.   |
| <code>void releaseSavepoint(Savepoint)</code> | Supprime le point de validation de la transaction courante.              |
| <code>void rollback(Savepoint)</code>         | Invalide la transaction à partir du point de validation.                 |

### Interface Savepoint

Les points de validation sont anonymes (identifiés toutefois par un entier) ou nommés. Le tableau suivant présente les deux seules méthodes de l'interface Savepoint :

Tableau 8-42 Méthodes de l'interface Savepoint

| Méthode                                | Description                                                         |
|----------------------------------------|---------------------------------------------------------------------|
| <code>int getSavepointId()</code>      | Retourne l'identifiant du point de validation de l'objet Savepoint. |
| <code>String getSavepointName()</code> | Retourne le nom du point de validation de l'objet Savepoint.        |

Le code suivant (`TransactionJDBC.java`) illustre une transaction découpée en phases par deux points de validation. Dans notre exemple, nous validons seulement la première partie (seul l'avion 'F-NEW2' sera inséré dans la table). On suppose la connexion créée.

Tableau 8-43 Points de validation

| Code Java                                                                                                                                                                                                                                                 | Commentaires                                                                                                                 |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <code>try { ...<br/>cx.setAutoCommit(false);<br/>String ordreSQL =<br/>    "INSERT INTO Avion VALUES (?, ?, ?, ?, ?)";<br/>PreparedStatement étatPréparé =<br/>    cx.prepareStatement(ordreSQL);<br/><b>Savepoint p1 = cx.setSavepoint("P1");</b></code> | Désactivation de l'auto-commit.                                                                                              |
| <code>étatPréparé.setString(1, "F-NEW2");<br/>...<br/>if (! étatPréparé.execute() )<br/>    System.out.println("F-NEW2 inséré");<br/><b>Savepoint p2 = cx.setSavepoint("P2");</b></code>                                                                  | Création d'un état appelable.<br>Création du point de validation P1.<br>Passage de paramètres et première insertion.         |
| <code>étatPréparé.setString(1, "F-NEW3");<br/>...<br/>if (! étatPréparé.execute() )<br/>    System.out.println("F-NEW3 inséré");<br/><b>cx.rollback(p2);</b></code>                                                                                       | ↑<br>Création du point de validation P2.<br>Passage de paramètres et deuxième insertion.                                     |
| <code>cx.commit();<br/>cx.close();<br/>} catch(SQLException ex) { ... }</code>                                                                                                                                                                            | Annulation de la deuxième partie.<br>Validation de la première partie.<br>Fermeture de la connexion.<br>Gestion des erreurs. |

## TraITEMENT DES EXCEPTIONS

Les exceptions qui ne sont pas traitées dans les sous-programmes appelés, ou celles que les sous-programmes ou déclencheurs peuvent retourner, doivent être prises en compte au niveau du code Java (dans un bloc `try...catch...`). Le bloc d'exceptions permet de programmer des traitements en fonction des codes d'erreur renvoyés par la base Oracle. Plusieurs blocs d'exceptions peuvent être imbriqués dans un programme JDBC.

Afin de gérer les erreurs renvoyées par le SGBD, JDBC propose la classe `SQLException` qui hérite de la classe `Exception`. Chaque objet (automatiquement créé dès la première erreur) de cette classe dispose des méthodes suivantes :

Tableau 8-44 Méthodes de la classe `SQLException`

| Méthode                                      | Description                                                                 |
|----------------------------------------------|-----------------------------------------------------------------------------|
| <code>String getMessage()</code>             | Message décrivant l'erreur.                                                 |
| <code>String getSQLState()</code>            | Code erreur SQL Standard (XOPEN ou SQL99).                                  |
| <code>int getErrorCode()</code>              | Code erreur SQL de la base.                                                 |
| <code>SQLException getNextException()</code> | Chaînage à l'exception suivante (si une erreur renvoie plusieurs messages). |

## Affichage des erreurs

Le code suivant (`Exceptions1.java`) illustre une manière d'afficher explicitement toutes les erreurs sans effectuer d'autres instructions :

Tableau 8-45 Affichage des erreurs

| Web | Code Java                                              | Commentaires         |
|-----|--------------------------------------------------------|----------------------|
|     | import java.sql.*;                                     |                      |
|     | public class Exceptions1                               |                      |
|     | { public static void main                              |                      |
|     | (String args []) throws SQLException, Exception        |                      |
|     | {try                                                   |                      |
|     | {Class.forName                                         | Chargement du pilote |
|     | ("com.mysql.jdbc.Driver").newInstance();}              |                      |
|     | catch (ClassNotFoundException ex)                      |                      |
|     | {System.out.println                                    |                      |
|     | ("Problème au chargement"+ex.toString()); }            |                      |
|     | try                                                    | Connexion.           |
|     | { Connection cx =                                      |                      |
|     | DriverManager.getConnection(...                        |                      |
|     | cx.close(); }                                          | Instructions...      |
|     | catch(SQLException ex)                                 | Gestion des erreurs  |
|     | {System.err.println("Erreur");                         |                      |
|     | while ((ex != null))                                   |                      |
|     | {System.err.println("Statut : "+ ex.getSQLState());    |                      |
|     | System.err.println("Message : "+ ex.getMessage());     |                      |
|     | System.err.println("Code base : "+ ex.getErrorCode()); |                      |
|     | ex = ex.getNextException();}                           |                      |
|     | } } }                                                  |                      |

## Traitement des erreurs

Il est possible d'associer des traitements à chaque erreur répertoriée avant l'exécution du programme. On peut appeler des méthodes de la classe principale ou coder directement dans le bloc des exceptions.

Le code suivant (`Exceptions2.java`) insère un enregistrement dans la table Avion en gérant un certain nombre d'exceptions possibles. Le premier bloc des exceptions permet d'afficher un message personnalisé pour chaque type d'erreur préalablement répertorié (duplication d'identifiant primaire, mauvais nombre ou type de colonnes...). Si l'avion à insérer n'est pas rattaché à une compagnie existante (contrainte référentielle), exception `1452-Cannot add or update a child row: a foreign key constraint fails`. Le dernier bloc d'exceptions affiche une erreur qui n'a pas été prévue par le programmeur (erreur système ou de syntaxe dans l'instruction, par exemple).

| Code Java                                                                            | Commentaires                |
|--------------------------------------------------------------------------------------|-----------------------------|
|  try | Instructions du main.       |
| {Connection cx =                                                                     |                             |
| DriverManager.getConnection(...);                                                    |                             |
| String ordreSQL =                                                                    |                             |
| "INSERT INTO bdsoutou.Avion VALUES                                                   |                             |
| ('F-NOUV', 'A310', 5600, 'AF')";                                                     |                             |
| PreparedStatement étatPréparé =                                                      |                             |
| cx.prepareStatement(ordreSQL);                                                       |                             |
| System.out.println(étatPréparé.executeUpdate() +                                     |                             |
| " avion insere en base.");                                                           |                             |
| cx.close(); }                                                                        |                             |
| catch(SQLException ex){                                                              | Non unique.                 |
| if (ex.getErrorCode() == 1062)                                                       |                             |
| System.out.println("Avion déjà existant!");                                          |                             |
| else if (ex.getErrorCode() == 1044)                                                  | Mauvais nom de base.        |
| System.out.println("Nom de base inconnu!");                                          |                             |
| else if (ex.getErrorCode() == 1136)                                                  | Mauvais nombre de colonnes. |
| System.out.println("Trop ou pas assez de valeurs!");                                 |                             |
| else if (ex.getErrorCode() == 1146)                                                  | Mauvais nom de table.       |
| System.out.println("Nom de table inconnue!");                                        |                             |
| else if ((ex.getSQLState() == "01004") &&                                            | Mauvais type de colonnes    |
| (ex.getErrorCode() == 0))                                                            |                             |
| System.out.println("Valeur trop longue ou                                            |                             |
| valeur trop importante!");                                                           |                             |
| else if (ex.getErrorCode() == 1452)                                                  | Clé étrangère absente.      |
| System.out.println("Compagnie inconnue, a                                            |                             |
| inserer avec l'avion");                                                              |                             |
| else                                                                                 | Gestion des autres erreurs. |
| System.err.println("Erreur");                                                        |                             |
| while ((ex != null))                                                                 |                             |
| {System.err.println("Statut : " + ex.getSQLState());                                 |                             |
| System.err.println("Message : " + ex.getMessage());                                  |                             |
| System.err.println("Code Erreur base : " +                                           |                             |
| ex.getErrorCode());                                                                  |                             |
| ex = ex.getNextException();} }                                                       |                             |

L'objectif de ces exercices est de développer des méthodes de la classe Java `ExoJDBC` extraire et mettre à jour des données des tables du schéma *Parc informatique*.

## Exercice

### 8.1 Curseur statique

Écrire les méthodes :

- `ArrayList getSalles()` qui retourne sous la forme d'une liste les enregistrements table `Salle`.
- `main` qui se connecte à la base, appelle la méthode `getSalles` et affiche les résultats (ex donné ci-dessous) :

| nSalle | nomSalle | nbPoste | indIP      |
|--------|----------|---------|------------|
| <hr/>  |          |         |            |
| s01    | Salle 1  | 3       | 130.120.80 |
| s02    | Salle 2  | 2       | 130.120.80 |
| ...    |          |         |            |

Ajoutez une nouvelle salle dans la table `Salle` dans l'interface de commande, et lancez à nouveau le programme pour vérifier.

## Exercice

### 8.2 Curseur modifiable

Écrivez la méthode `void deleteSalle(int)` qui supprime de la table `Salle` l'enregistrement rang passé en paramètre. Vous utiliserez la méthode `deleteRow` appliquée à un curseur modifiable. Appelez dans le `main` cette méthode pour supprimer l'enregistrement de la table `Salle` que vous avez ajouté en test, dans l'exercice précédent. Si l'enregistrement est rattaché à un enregistrement qui ne forcez pas la contrainte référentielle, contentez-vous d'afficher le message d'erreur 1451 rencontré par MySQL, dans le bloc des exceptions.

## 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 deleteSalleSP(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.

# Utilisation avec PHP

Ce chapitre détaille les moyens de faire interagir un programme PHP 5 avec une base MySQL. PHP propose deux extensions (API) qui sont `mysql` et `mysqli`. La première convient aux bases MySQL de version antérieure à 4.1. Nous détaillerons donc l'extension `mysqli`, qui correspond toutefois à des bases plus anciennes (jusqu'à la version 3.22). Pour plus de détails, consultez une documentation en ligne mise à jour (<http://www.nexen.net/docs/php/tee/ref.mysql.php>).

## Configuration adoptée

---

Plusieurs configurations sont possibles en fonction de la version de PHP utilisée, de la version d'Apache et de celle de MySQL. Nous avons opté pour faire interagir un programme PHP avec une base MySQL 5 sous Apache 1.3. Je décris ici une procédure minimale sans explication. Vous trouverez sur le Web de nombreuses ressources à ce sujet.

### Logiciels

Récupérez et installez Apache ([www.apache.org](http://www.apache.org)). Lancez Apache.exe si il n'est pas automatiquement lancé après l'installation. Testez le service dans le navigateur, en fonction du nom du serveur que vous avez spécifié à l'installation (<http://camparols.dansmoncas>).

Installez PHP (<http://www.php.net/downloads.php>) en dézippant le fichier téléchargé dans le répertoire personnel (C:\PHP dans mon cas).

### Fichiers de configuration

Dans le fichier `httpd.conf` (situé dans C:\Program Files\Apache Group\Apache2\conf\httpd.conf dans mon cas), modifiez ou ajoutez les lignes suivantes (le « # » démarre un commentaire) :

```
modif pour MySQL et PHP ici 9999 par exemple
Port 9999
...
```

```
#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()`).

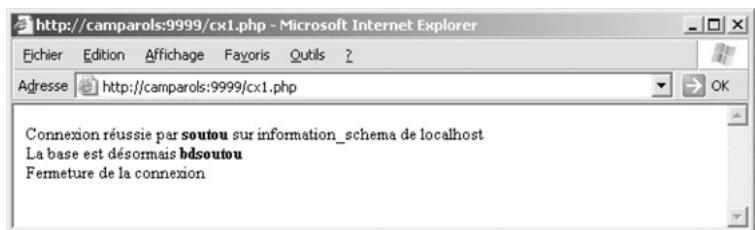
## Test d'Apache, de PHP et de MySQL

Il faut que le serveur MySQL soit démarré (vérifiez dans **Services** à partir du panneau de configuration). Écrivez le programme `cx1.php` suivant et disposez-le dans le répertoire contenant les sources PHP. Renseignez le nom d'utilisateur MySQL, le mot de passe et le nom de la base. Ici je lance une connexion à la base du dictionnaire des données puis je sélectionne par la suite la base `bdsoutou`.

```
<?php
if (($service=
 mysqli_connect('localhost','soutou','iut','information_schema'))
{print "Connexion réussie avec soutou sur information_schema
 de localhost";
if (($usebdsoutou = mysqli_select_db($service,'bdsoutou')) > 0)
{ print "
 La base est désormais bdsoutou";
else
{ print "
 Sélection impossible sur bdsoutou";
print "
 Fermeture de la connexion";
mysqli_close($service) ;
}
else
print "
 La connexion est un échec!";
?>
```

Tester ce programme dans le navigateur (`http://camparols:9999/cx1.php` dans ce cas). Vous devez obtenir un résultat analogue :

**Figure 9-1** Test d'une connexion



## API de PHP pour MySQL

Depuis PHP 5, le préfixe des fonctions de la dernière extension est désormais « `mysql` ».

Avec cette nouvelle API, de nouvelles fonctions apparaissent, elles concernent principalement les états préparés (*prepared statements*), la possibilité d'appeler des procédures catalogues, elles prennent en charge la programmation objet (classes) de PHP. Nous n'étudierons pas ce mode de développement en restant dans une programmation procédurale (des compléments seront mis en ligne).

La fonction `mysqli_connect` retourne un identifiant de connexion utilisé par la majorité des appels à la base. Les fonctions `mysqli_close`, `mysqli_select_db` et `mysqli_change_user` renvoient TRUE en cas de succès, FALSE en cas d'erreur. Une connexion se ferme implicitement en fin de programme même si elle n'a pas été clôturée explicitement.

Tableau 9-1 Fonctions de connexion et de déconnexion

Nom de la fonction	Paramètres
<code>ressource mysqli_connect(string serveur, string utilisateur, string motpasse [,string nomBase])</code>	Nom du serveur, utilisateur, mot de passe, nom de la base éventuellement. Retourne FALSE en cas d'erreur.
<code>boolean mysqli_close(ressource connexion)</code>	Ferme la connexion dont l'identifiant passe en paramètre.
<code>boolean mysqli_select_db(ressource connexion, string nomBase)</code>	Modifie la sélection de la base de la connexion dont l'identifiant passe en paramètre.
<code>boolean mysqli_change_user(ressource connexion, string utilisateur, string motpasse, string nomBase)</code>	Modifie l'utilisateur et la base de la connexion dont l'identifiant passe en paramètre.

## Interactions avec la base

La majorité des traitements SQL, lorsqu'ils incluent des paramètres, s'effectuent comme suit : connexion (*connect*), préparation de l'ordre (*parse*), association des paramètres à l'ordre SQL (*bind*), exécution dudit ordre (*execute*), lecture des lignes (pour les *SELECT, fetch*) et libération des ressources (*free* et *close*) après une éventuelle validation de la transaction courante (*commit* ou *rollback*).

### Préparation, exécution

La fonction `mysqli_prepare` prépare l'ordre SQL puis retourne un identifiant d'état qui peut être utilisé notamment par les fonctions `mysqli_stmt_bind_param` et `mysqli_stmt_execute`. La fonction `mysqli_prepare` retourne FALSE dans le cas d'une erreur, mais n'est pas valide ni sémantiquement ni syntaxiquement l'ordre SQL. Il faudra attendre pour cela sa propre exécution par `mysqli_stmt_execute`.

La fonction `mysqli_stmt_execute` exécute un ordre SQL préparé (renvoie TRUE en cas de succès, FALSE sinon). Le mode par défaut est *auto-commit*. Pour la programmation de transactions, désactiver ce mode (avec `mysqli_autocommit`) puis valider explicitement par `mysqli_commit`.

La fonction `mysqli_stmt_fetch` exécute pas à pas une requête préparée (retourne TRUE en cas de succès, FALSE sinon).

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

Nom de la fonction	Paramètres
ressource <b>mysqli_prepare</b> (ressource connexion, string ordreSQL)	Le premier paramètre désigne l'identifiant de connexion. Le second contient l'ordre SQL à lyser (SELECT, INSERT, UPDATE, DELETE, CREATE...).
boolean <b>mysqli_stmt_execute</b> (ressource ordreSQL)	Le paramètre désigne l'identifiant d'état à exécuter (renvoyé par <i>prepare</i> ).
boolean <b>mysqli_stmt_fetch</b> (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	Paramètres
boolean <b>mysqli_commit</b> (ressource connexion)	Valide la transaction de la connexion en paramètre.
boolean <b>mysqli_rollback</b> (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 moyen de PHP, les erreurs renvoyées par MySQL.

Tableau 9-4 Insertion d'un enregistrement

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

Si vous souhaitez connaître le nombre de lignes affectées par l'ordre SQL, utilisez « `mysqli_stmt_affected_rows($ordre)` » (voir la section *Métadonnées*).

## Constantes prédefinies

Les constantes suivantes permettent de positionner des indicateurs jouant le rôle de paramètres système (modes d'exécution) au sein d'instructions SQL. Nous verrons au long de nos exemples l'utilisation de certaines de ces constantes.

Tableau 9-5 Constantes prédefinies

Constante	Commentaires
<code>MYSQLI_ASSOC</code>	Utilisé par <code>mysqli_fetch_array</code> afin d'extraire un <i>associative array</i> comme résultat.
<code>MYSQLI_NUM</code>	Utilisé par <code>mysqli_fetch_array</code> afin d'extraire un <i>enumerated array</i> comme résultat.
<code>MYSQLI_BOTH</code>	Utilisé par <code>mysqli_fetch_array</code> afin d'extraire un <i>array</i> reconnaissant à la fois le mode associatif et le mode numérique en indices.

## Extractions

Les fonctions suivantes permettent d'extraire des données via un curseur que la documentation de PHP appelle *tableau*. On rappelle que MySQL retourne les noms de colonnes toujours en minuscules. Cette remarque intéressera les habitués des tableaux à accès associatif (exemple : `$tab[ 'prenom' ]`, prenom étant une colonne extraite d'une table).

Tableau 9-6 Fonctions d'extraction

Nom de la fonction	Paramètres
<code>ressource mysqli_query(ressource ordreSQL [,ressource connexion])</code>	Exécute la requête sur la base de données en cours. Retourne un identifiant de résultat ou <code>FALSE</code> en cas d'erreur.
<code>array mysqli_fetch_array(ressource idresultat [, int param] )</code>	Retourne un tableau qui contient la ligne du curseur suivante, ou <code>FALSE</code> en cas d'erreur ou en fin de curseur. Le tableau est accessible de manière associative ou numérique suivant le paramètre <i>param</i> qui peut être une combinaison de : <ul style="list-style-type: none"><li>• <code>MYSQLI_BOTH</code> (par défaut, identique à <code>MYSQLI_ASSOC + MYSQLI_NUM</code>).</li><li>• <code>MYSQLI_ASSOC</code> pour un tableau à accès associatif (comme <code>mysqli_fetch_assoc</code>).</li><li>• <code>MYSQLI_NUM</code> pour un tableau à accès numérique (comme <code>mysqli_fetch_row</code>).</li></ul>

Tableau 9-6 Fonctions d'extraction

Nom de la fonction	Paramètres
array <b>mysqli_fetch_assoc</b> (ressource <i>ordreSQL</i> )	Retourne la ligne du curseur suivante dans tableau associatif, ou FALSE en cas d'erreur ou en fin de curseur.
object <b>mysqli_fetch_object</b> (ressource <i>ordreSQL</i> )	Retourne la ligne du curseur suivante dans un PHP ou FALSE en cas d'erreur ou en fin de curseur.
array <b>mysqli_fetch_row</b> (ressource <i>ordreSQL</i> )	Retourne la ligne du curseur suivante dans tableau numérique, ou FALSE en cas d'erreur ou en fin de curseur.
boolean <b>mysqli_stmt_free_result</b> (ressource <i>ordreSQL</i> )	Libère les ressources associées aux curseurs créés après <b>mysqli_prepare</b> . Retourne TRUE en cas de succès, FALSE dans le cas inverse.

Illustrons à partir d'exemples certaines utilisations de quelques-unes de ces fonctions.

Le programme suivant (*select1.php*) utilise **mysqli\_fetch\_array** afin d'extraire les avions de la compagnie de code 'AF'. On suppose ici, et dans les programmes suivants, que la connexion à la base est réalisée et se nomme \$service. Le curseur obtenu est alors une ligne, il prend en compte les valeurs nulles éventuelles.

Tableau 9-7 Extraction à l'aide de **mysqli\_fetch\_array**

Code PHP	Commentaires
 \$requete = "SELECT immat,capacite,typeAvion FROM Avion WHERE compa = 'AF'" ;	Création de la requête.
if ( (\$resultat= <b>mysqli_query</b> (\$service, \$requete)) > 0 ) {\$ncols = <b>mysqli_num_fields</b> (\$resultat);	Chargement du curseur.
print "<H3>Avions de la compagnie 'AF'</H3>" ; print "<TABLE BORDER=1> ";	Obtention du nombre de colonnes.
while (\$ligne = <b>mysqli_fetch_array</b> (\$resultat)) {print "<TR> ";	Parcours des colonnes.
for ( \$i=0;\$i < \$ncols; \$i++) {print "<TD> \$ligne[\$i] </TD>" ;} print "</TR> " ; print "</TABLE> " ;	Affichage des colonnes.
<b>mysqli_free_result</b> (\$resultat); }	Libération des ressources.
else print " La requête est un échec!" ; <b>mysqli_close</b> (\$service);	Fermeture de la connexion.

- La fonction **mysqli\_num\_fields** renvoie le nombre de colonnes de la requête. Sa signature est détaillée à la section *Métadonnées*.

- La fonction `mysqli_num_rows($resultat)` aurait retourné 4 (nombre de lignes extraites).
- La fonction `mysqli_free_result` joue le même rôle que `mysqli_stmt_free_result`, mais s'applique aux instructions `SELECT`.

Vous devez obtenir un résultat analogue (en supposant que la compagnie 'AF' dispose d'au moins quatre avions dont un est affecté d'une capacité nulle) :

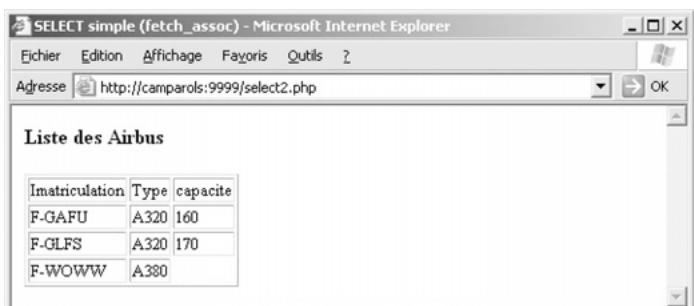
**Figure 9-2 Exemple avec `mysqli_fetch_array`**

F-GAFU	160	A320
F-GLFS	170	A320
F-WOWW		A380
F-WTSS	90	Concorde

Le programme suivant (`select2.php`) décrit l'utilisation de la fonction `mysqli_fetch_assoc` pour extraire tous les Airbus. Le tableau associatif obtenu est nommé `row`. L'accès à chaque cellule de ce tableau est réalisé à l'aide du nom des colonnes.

**Tableau 9-8 Extraction à l'aide de `mysqli_fetch_assoc`**

Code PHP	Commentaires
 \$requete = "SELECT immat,typeAvion,capacite FROM Avion WHERE typeAvion LIKE 'A%'" ;	Création de la requête.
if ( (\$resultat=mysqli_query(\$service, \$requete))>0 ) {print "<H3>Liste des Airbus</H3>" ; print "<table border=1>\n" ; print "<tr><td>Immatriculation</td> <td>Type</td><td>capacite</td>" ;	Exécution de la requête.
\$i=0 ; while ( (\$row = mysqli_fetch_assoc(\$resultat)) {print "<tr><td>".\$row["immat"]. "</td><td>".\$row["typeAvion"]. "</td><td>".\$row["capacite"].".</td></tr>" ; } print "</table>\n" ; mysqli_free_result(\$resultat) ; } else { print " La requête est un échec!" ; }	Parcours du curseur. Affichage des colonnes.
mysqli_close(\$service) ;	Libération des ressources. Fermeture de la connexion

**Figure 9-3** Exemple avec mysqli\_fetch\_assoc

## Instructions paramétrées

Les fonctions `mysqli_stmt_bind_param` et `mysqli_stmt_bind_result` permettent d'associer à des colonnes MySQL des variables PHP et inversement. Ces fonctions retournent TRUE en cas de succès, FALSE sinon.

**Tableau 9-9** Fonctions de passage de paramètres

Nom de la fonction	Paramètres
<code>boolean mysqli_stmt_bind_result(ressource ordreSQL, mixed &amp;variable1, [,mixed &amp;variable2...])</code>	Le premier paramètre est l'identifiant de l'ordre obtenu après <code>prepare</code> . Les paramètres suivants listent les variables PHP de réception.
<code>boolean mysqli_stmt_bind_param(ressource ordreSQL, string types, mixed &amp;variable1 [,mixed &amp;variable2...])</code>	Le premier paramètre est l'identifiant de l'ordre obtenu après <code>prepare</code> . Le deuxième paramètre décrit les types des variables à substituer aux colonnes. Les paramètres suivants listent les variables PHP en entrée.
<code>string types</code>	Description des types des variables (i pour numérique, d pour flottant, s pour chaîne de caractères et b pour <i>BLOB</i> ). Concaténer au moins deux de ces caractères qu'il existe de variables.

## Extractions

Le programme suivant (`select3.php`) utilise la fonction `mysqli_stmt_bind_result` afin d'extraire l'immatriculation et le type de tous les avions (au travers de variables PHP qui sont définies après exécution de la requête). Notez l'utilisation des fonctions `mysqli_`

`fetch` pour parcourir le résultat de la requête ligne après ligne, et `mysqli_stmt_close` pour fermer la requête préparée.

**Tableau 9-10** Extraction préparée avec la fonction mysqli\_stmt\_bind\_result

Web	Code PHP	Commentaires
	\$requete = "SELECT immat,typeAvion FROM bdsoutou.Avion";	Définition et exécution de la requête préparée.
	\$ordre = mysqli_prepare(\$service,\$requete);	
	if ( (\$res = mysqli_stmt_execute(\$ordre)) > 0)	
	{if ( (\$resbind =	Affectation des variables PHP.
	mysqli_stmt_bind_result(\$ordre,\$im,\$ty)) > 0 )	
	{print "<H4>Liste des avions</H4>";	
	print "<TABLE BORDER=1> ";	
	while (mysqli_stmt_fetch(\$ordre))	Parcours de la requête.
	{print "<TR> <TD> \$im</TD>" ;	
	print " <TD> \$ty</TD> </TR> " ; }	
	print "</TABLE> ";	
	} else print " La liaison est un échec!";	
	} else print " La requete est un échec!";	
	mysqli_stmt_close(\$ordre);	Fermeture de la requête et de la connexion.
	mysqli_close(\$service);	

### *Manipulations*

Le programme suivant (`insert2.php`) utilise la fonction `mysqli_stmt_bind_param`, faisant passer deux paramètres (variables PHP) lors de l'insertion d'une nouvelle compagnie. On retrouve la notion de *placeholders* (symbole « ? » désignant une valeur d'une colonne d'une table ou d'une vue) étudiée au chapitre 7. Notez le second paramètre de la fonction `mysqli_stmt_bind_param` (« ss » désigne deux types chaînes de caractères).

**Tableau 9-11** Insertion paramétrée avec la fonction `mysqli_stmt_bind_param`

Web	Code PHP	Commentaires
	mysqli_autocommit(\$service, FALSE); \$codeComp = "CAST"; \$nomComp = "Castanet Air"; \$insert2 = "INSERT INTO Compagnie VALUES(?,?)"; \$ordre = mysqli_prepare(\$service, \$insert2);	Affectation des variables PHP.
	if ((mysqli_stmt_bind_param(\$ordre, 'ss', \$codeComp, \$nomComp)) > 0) {if (( \$res = mysqli_stmt_execute(\$ordre)) > 0)	Définition de l'ordre paramétré.
	{print " Compagnie \$nomComp insérée"; mysqli_commit(\$service); mysqli_stmt_free_result(\$ordre); } else print " L'insertion est un échec!"; }	Association avec les variables PHP.
	else print " Problème au bind!"; mysqli_close(\$service);	Exécution de l'ordre.
		Validation.
		Libération des ressources.
		Fermeture de la connexion.

Pour toute extraction par `SELECT`, modification par `UPDATE` ou suppression par `DELETE`, le principe à adopter est le même.

## Gestion des séquences

La fonction `mysqli_insert_id($connexion)` retourne la valeur courante d'une séquence après avoir inséré un enregistrement dans une table contenant une colonne `auto_increment` (`INCREMENT`).

Le programme suivant (`insert3.php`) insère un affrètement (table décrite aux chapitres 7 et 8) à la date du jour pour la compagnie de code 'CAST' et l'avion immatriculé 'F-GRTC'. Il récupère la dernière valeur de la séquence après l'insertion.

Tableau 9-12 Insertion paramétrée avec la fonction `mysqli_insert_id`

Web	Code PHP	Commentaires
	<pre>mysqli_autocommit(\$service, FALSE); \$codeComp = "CAST"; \$immatric = "F-GRTC"; \$n         = 162; \$insert3 = "INSERT INTO bdsoutou.Affreter             (comp, immat, dateAff, nbPax) VALUES (?, ?, SYS- DATE(), ?)"; \$ordre   = mysqli_prepare(\$service, \$insert3); if ((mysqli_stmt_bind_param(\$ordre, 'ssi', \$codeComp,, \$immatric, \$n)) &gt; 0)     {if ((\$res = mysqli_stmt_execute(\$ordre)) &gt; 0)         {print "&lt;BR&gt;Affretement insérée, sequence : "             . mysqli_insert_id(\$service);         mysqli_commit(\$service);         mysqli_stmt_free_result(\$ordre); }      else print "&lt;BR&gt;L'insertion est un échec!";     } else print "&lt;BR&gt;Problème au bind!"; mysqli_close(\$service);</pre>	Affectation des variables PHP. Définition de l'ordre paramétré. Association avec les variables PHP. Exécution de l'ordre. Récupération de la séquence. Validation. Libération des ressources. Fermeture de la connexion.

## Traitement des erreurs

Les fonctions `mysqli_errno` et `mysqli_error` permettent de gérer les erreurs retournées par MySQL au niveau de la connexion. Les fonctions `mysqli_stmt_errno` et `mysqli_stmt_error` sont analogues au niveau d'une instruction préparée.

Le programme suivant (`erreur1.php`) utilise plusieurs de ces fonctions. Dans cet exemple, la suppression ne se déroule pas correctement du fait de l'existence d'enregistrements « `NULL` » dans la table `Avion`. Il est donc possible de dérouter le programme en fonction du code d'erreur MySQL renvoyé (comme pour les exceptions des procédures cataloguées).

Tableau 9-13 Fonctions pour la gestion des erreurs MySQL

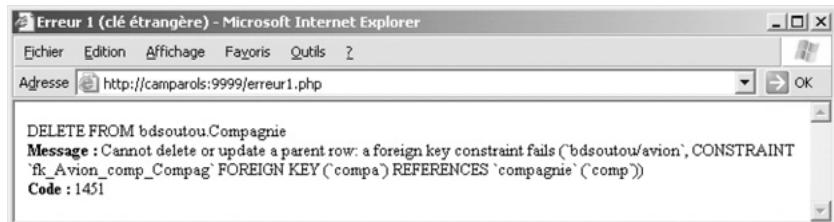
Nom de la fonction	Paramètres
<code>int mysqli_errno(ressource connexion)</code>	Retourne le code d'erreur du dernier appel à la base, sur la connexion désignée dans le paramètre (0 si aucune erreur n'est survenue lors du dernier échange).
<code>string mysqli_error(ressource connexion)</code>	Retourne le libellé de l'erreur lors du dernier échange (chaîne vide si aucune erreur).
<code>int mysqli_stmt_errno (ressource ordreSQL)</code>	<i>Idem</i> <code>mysqli_errno</code> à partir de l'identifiant d'état désigné dans le paramètre.
<code>string mysqli_stmt_error (ressource ordreSQL)</code>	<i>Idem</i> <code>mysqli_error</code> à partir de l'identifiant d'état désigné dans le paramètre.
<code>string mysqli_connect_error()</code>	Retourne le message d'erreur d'une mauvaise connexion.
<code>int mysqli_connect_errno()</code>	Retourne le code d'erreur d'une mauvaise connexion.

Tableau 9-14 Gestion d'erreurs

Web	Code PHP	Commentaires
	<pre>if ( (\$service = mysqli_connect('localhost', 'soutou',                                 'iut', 'bdsoutou')) &gt; 0)     {\$delete1 = "DELETE FROM bdsoutou.Compagnie";      \$ordre   = mysqli_prepare(\$service, \$delete1);      print \$delete1;      if ( (\$res = mysqli_stmt_execute(\$ordre)) &gt; 0)         { print "&lt;BR&gt;Suppression de Compagnie!";         }      else         {print "&lt;BR&gt;&lt;B&gt;Message : &lt;/B&gt;". mysqli_stmt_error(\$ordre);          print "&lt;BR&gt;&lt;B&gt;Code : &lt;/B&gt;". mysqli_stmt_errno(\$ordre); }      mysqli_stmt_free_result(\$ordre);      mysqli_close(\$service);     } else {print "L'utilisateur n'a pu se connecter &lt;BR&gt;";  print "&lt;B&gt;Message : &lt;/B&gt;". mysqli_connect_error(); }</pre>	Connexion.
		Préparation de l'ordre.
		Exécution.
		Affichage de l'erreur.
		Libération des ressources.
		Fermeture de la connexion.
		Erreur de connexion.

Le résultat est le suivant :

Figure 9-4 Exception SQL levée à l'aide de PHP



## Procédures cataloguées

Comme dans tout autre langage hôte, PHP permet d'invoquer des procédures cataloguées du serveur. Supposons que nous disposions de la procédure `augmenteCap` qui augmente la capacité (premier paramètre) des avions d'une compagnie donnée (deuxième paramètre).



```
CREATE PROCEDURE bdsoutou.augmenteCap(IN nbre TINYINT, IN compag CHAR)
BEGIN
 UPDATE Avion SET capacite = capacite + nbre WHERE compa = compag;
END;
```

### Paramètre en entrée

Le code suivant (`procedureCat.php`) appelle cette procédure afin d'augmenter de 50 la capacité des avions de la compagnie de code 'AF' en utilisant la fonction `mysqli_multi_query`. Notez l'utilisation des simples guillemets pour les paramètres en chaînes de caractères.



Pensez à donner l'autorisation à l'utilisateur appelant (ici `soutou`) d'exécuter la procédure que j'ai écrite sous root : `GRANT EXECUTE ON PROCEDURE bdsoutou.augmenteCap TO 'soutou'@'localhost'`).

Tableau 9-15 Appel d'une procédure cataloguée (paramètres d'entrée)

Code PHP	Commentaires
<pre>if ((\$service = mysqli_connect('localhost', 'soutou',                                'iut', 'bdsoutou')) &gt; 0)</pre>	Connexion.
<pre>{\$nb    = 50; \$comp = 'AF'; if (\$result = mysqli_multi_query(\$service,                                   "call bdsoutou.augmenteCap(\$nb,'\$comp')") &gt; 0)</pre>	Initialisation des variables PHP d'appel.
<pre>{ print "&lt;BR&gt;Procédure réalisée correctement.";</pre>	Appel de la procédure.
<pre>else { print "&lt;BR&gt;La procédure est un échec! ";         mysqli_error(\$service);     mysqli_close(\$service); } else print "&lt;BR&gt;La connexion est un échec!";</pre>	Fermeture de la connexion.

### Paramètre en sortie

Afin de travailler avec des paramètres en sortie, il est nécessaire d'utiliser des variables de session. Une fois ces variables de session initialisées au retour de l'appel du sous-programme, il faudra extraire chaque valeur à l'aide d'un `SELECT` dans une requête simple.

Le code suivant (`procedureCat2.php`) décrit l'appel de la procédure `leNomCompagnieEst` (décrise au chapitre 8, section *Procédures cataloguées*) ayant deux paramètres.

premier (en entrée) indique l'avion de la compagnie recherché, le second (en sortie) contient le nom de la compagnie.

Notez l'utilisation de :

- la fonction `mysqli_multi_query` pour appeler la procédure cataloguée ;
- la variable de session `@v_retour` pour récupérer le paramètre en sortie ;
- la fonction `mysqli_query` pour extraire la valeur de la variable de session (contenue à l'indice 0 du tableau résultat, car il n'y a ici qu'une valeur renournée dans le `SELECT`).

Tableau 9-16 Appel d'une procédure cataloguée (paramètre de sortie)

Web	Code PHP	Commentaires
	if ( (\$service = mysqli_connect('localhost', 'soutou', 'iut', 'bdsoutou')) > 0) {\$immat = 'F-GAFU'; if (\$result = mysqli_multi_query [\$service,"call bdsoutou.leNomCompagnieEst ('\$immat',@v_retour)") > 0) {if (\$result2 = mysqli_query(\$service, "SELECT @v_retour")) {\$ligne = mysqli_fetch_array(\$result2, MYSQLI_NUM); if (\$ligne[0] == null) print " Désolé, l'avion \$immat n'a pas de compagnie!"; else print " La compagnie de l'avion \$immat est :". \$ligne[0]; mysqli_free_result(\$result2); } else {print " Problème au retour du paramètre ". mysqli_error(\$service); } } else {print " La procédure est un échec! ". mysqli_error(\$service).\$result; } mysqli_close(\$service); } else print " La connexion est un échec!";	Connexion.
		Initialisation de la variable PHP d'appel.
		Appel de la procédure.
		Extraction de la variable de session.
		Affichage du résultat.
		Gestion des erreurs.
		Fermeture de la connexion.

## Métadonnées

Plusieurs fonctions permettent d'extraire des informations en provenance du dictionnaire de données (*meta data*) à partir d'une instruction SQL. Par exemple, `mysqli_stmt_result_metadata` retourne un identifiant de résultat permettant d'extraire des métadonnées à partir d'une requête préparée. La fonction `mysqli_fetch_field` retourne un objet qui contient les métadonnées des colonnes concernées par une requête.

Citons d'autres fonctions comme `mysqli_fetch_field_direct` et `mysqli_fetch_fields` qui sont similaires à `mysqli_fetch_field`. Une fois l'objet retourné par cette fonction, il faut extraire certains de ces champs à la demande.

Tableau 9-17 Fonctions pour les métadonnées

Nom de la fonction	Paramètres
<code>ressource mysqli_stmt_result_metadata (ressource ordreSQL)</code>	Le paramètre est l'identifiant d'état obtenu après <code>prepare</code> .
<code>object mysqli_fetch_field(ressource ordreSQL)</code>	Le paramètre est l'identifiant de résultat obtenu après <code>mysqli_query</code> , ou FALSE si aucune information n'est renvoyée pour l'ordre SQL concerné.
<code>int mysqli_num_fields(ressource ordreSQL)</code>	Retourne le nombre de colonnes concernées par l'ordre SQL. Le paramètre est l'identifiant de résultat obtenu après <code>mysqli_query</code> .

Tableau 9-18 Champs de l'objet retourné par `mysqli_fetch_field`

Nom du champ	Signification
<code>name</code>	Nom de la colonne.
<code>table</code>	Nom de la table à laquelle la colonne appartient (si elle n'a pas été calculée).
<code>def</code>	Valeur par défaut de la colonne.
<code>max_length</code>	Taille maximale de la colonne pour le jeu de résultats retourné par la requête.
<code>flags</code>	Entier représentant le <i>bit-flags</i> pour la colonne (codage des contraintes).
<code>type</code>	Code du type de données de la colonne.
<code>decimals</code>	Nombre de décimales de la colonne.

Tableau 9-19 Code du type de données

Type MySQL	Code équivalent
<code>DECIMAL</code>	0
<code>TINYINT</code>	1
<code>SMALLINT</code>	2
<code>INT</code>	3
<code>FLOAT</code>	4
<code>DOUBLE</code>	5
<code>TIMESTAMP</code>	7
<code>BIGINT</code>	8
<code>MEDIUMINT</code>	9
<code>DATE</code>	10
<code>TIME</code>	11
<code>DATETIME</code>	12
<code>YEAR</code>	13
<code>TEXT, TINYBLOB, TINYTEXT, BLOB, MEDIUMBLOB, MEDIUMTEXT, LONGBLOB, LONGTEXT</code>	252
<code>VARBINARY, VARCHAR</code>	253
<code>CHAR, BINARY, ENUM, SET</code>	254

## Fonction mysqli\_fetch\_field

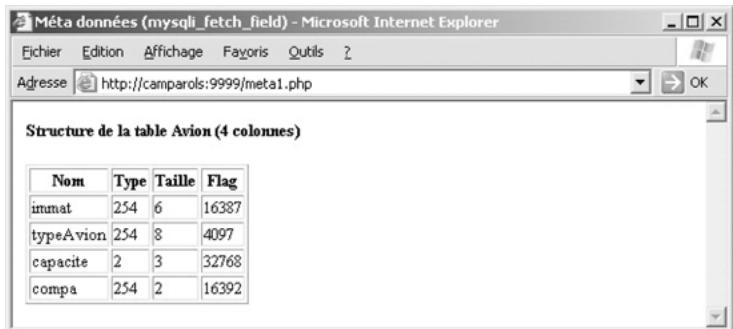
Le programme suivant (`meta1.php`) utilise la fonction `mysqli_fetch_field` afin d'extraire la structure complète (en termes de colonnes) d'une table.

Tableau 9-20 Extraction de la structure d'une table

Web	Code PHP	Commentaires
	<pre>if ( (\$res = mysqli_query       (\$service, "SELECT * FROM bdsoutou.Avion")) &gt; 0) { \$ncols = mysqli_num_fields(\$res);   print "&lt;H4&gt;Structure de la table Avion           (\$ncols colonnes)&lt;/H4&gt;";   print "&lt;table border=1&gt;";   print "&lt;tr&gt;&lt;th&gt;Nom&lt;/th&gt;&lt;th&gt;Type&lt;/th&gt;&lt;th&gt;Taille&lt;/th&gt;         &lt;th&gt;Flag&lt;/th&gt;&lt;/tr&gt;";   while (\$obj=mysqli_fetch_field(\$res))     print "&lt;tr&gt;&lt;td&gt;\$obj-&gt;name&lt;/td&gt;            &lt;td&gt;\$obj-&gt;type&lt;/td&gt;            &lt;td&gt;\$obj-&gt;max_length&lt;/td&gt;            &lt;td&gt;\$obj-&gt;flags&lt;/td&gt;&lt;/tr&gt;";   print "&lt;/table&gt;\n"; } else print "&lt;BR&gt;La requete est un échec!"; mysqli_close(\$service);</pre>	Requête. Extraction du nombre de colonnes. Affichage du nom, code du type, taille maximale et contraintes de chaque colonne extraite.

Le résultat est le suivant :

Figure 9-5 Extraction de la structure d'une table



Quelques explications :

- La taille vaut 2 pour la colonne `compa`, car seule la compagnie de code 'AF' est représentée dans mon jeu d'essai.

```
(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 colonne 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 256 /* 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 pointeur à l'attribut `result_metadata` de l'objet 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>"; ... }
```

L'objectif de ces exercices est de compléter des programmes PHP pour extraire et mettre à jour des données de certaines des tables du schéma *Parc informatique*.

## Exercice

### 9.1

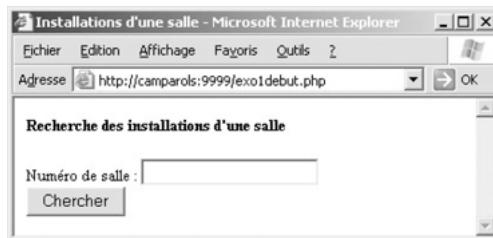
#### Extraction préparée

Écrire le programme `exo1suite.php` qui devra retrouver le nom du logiciel, le numéro de poste, date d'installation et la date d'achat du logiciel pour toutes les installations d'une salle donnée. Ce programme sera appelé à partir du programme `exo1debut.php` composant un formulaire de saisie.

```
<html> <head> <title>Installations d'une salle</title> </head>
<body>
<form action=". /ex01suite.php" method="POST"><p>
<H4>Recherche des installations d'une salle</H4><P>
 Numéro de salle : <input type="text" name="ns" maxlength="7"/>

 <input type="submit" value="Chercher"/>
</form>
</body> </html>
```

**Figure 9-6** Formulaire de saisie



Vous utiliserez :

- `$_POST[ 'ns' ]` pour récupérer la valeur du numéro de salle saisi dans le formulaire.
- `mysqli_prepare`, `mysqli_stmt_execute` et `mysqli_stmt_bind_result` pour préparer, exécuter et lier des variables PHP en sortie.
- Un affichage simple de type « Aucune installation dans la salle, si la salle ne contient aucun poste sur lequel un logiciel est installé » (exemple pour 's12')

Votre programme doit produire un résultat analogue à l'écran suivant :

**Figure 9-7** Extraction préparée

Nom Logiciel	Poste	Installation	Achat
Front Page	p12	2003-04-20 00:00:00	1997-06-03 00:00:00
I. I. S.	p11	2003-04-20 00:00:00	2002-04-12 00:00:00
SQL Server	p11	2003-04-20 00:00:00	1998-04-12 00:00:00

## Exercice

### 9.2 Appel d'un sous-programme

Utiliser de nouveau la procédure cataloguée `supprimeSalle(IN ns VARCHAR(7), OUT TINYINT)` décrite à l'exercice du chapitre 8. Cette procédure supprime une salle dont le numéro passé en premier paramètre et 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 le programme `exo2suite.php` qui, à partir d'une saisie du numéro de salle (programme `exo2debut.php` similaire à `exo1debut.php`), appelle le sous-programme `supprimeSalle`.

```
<html> <head> <title>Suppression d'une salle</title> </head>
<body>
<form action="./exo2suite.php" method="POST"><p>
 <H3>Salle à supprimer</H3><p>
 Numéro de salle : <input type="text" name="ns" maxlength="7"/>

 <input type="submit" value="Supprimer"/>
 <input type="reset" value="Reset"/>
</form>
</body> </html>
```

**Figure 9-8** Formulaire de saisie

Salle à supprimer

Numéro de salle :

## 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" />

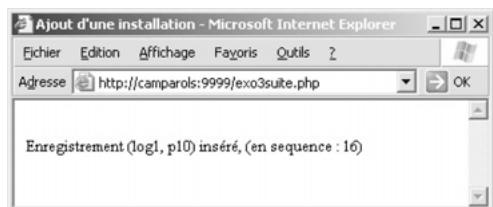
<input type="submit" value="Enregister"/>
<input type="reset" value="Reset"/>
</form>
</body> </html>
```

Figure 9-9 Saisie du formulaire

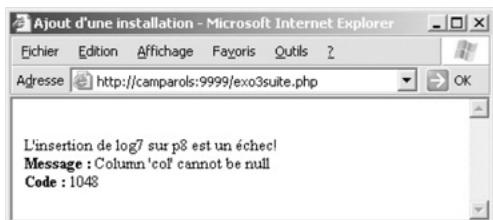


Pour tester une éventuelle erreur de compatibilité entre le type du poste et celui du logiciel (voir déclencheur de l'exercice en fin de chapitre 7), vous afficherez le message d'erreur (avec `mysql_stmt_error`) et le code d'erreur si l'insertion se passe mal. Tester une insertion correcte ('p10', 'log1')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

*Figure 9-10 Insertion correcte*



*Figure 9-11 Erreur après insertion*





## Outils graphiques

MySQL AB fournit gratuitement plusieurs outils graphiques afin d'administrer, de manipuler et d'interroger une base de données. Ce sont *MySQL Administrator*, *MySQL Query Browser*, et *MySQL Migration Toolkit*. L'utilisation de cet outil de migration sera abordée dans le cadre de cet ouvrage, nous ne l'étudierons donc pas.

Par ailleurs, il existe d'autres consoles graphiques d'administration que nous allons observer sommairement, à savoir le célèbre *phpMyAdmin* (interface Web écrite en PHP), *Toad for MySQL* (plus connu pour sa version Oracle), *Navicat* et *EMS SQL Manager*.

Ce chapitre survole les principales fonctionnalités de ces logiciels. N'y voyez pas ici un guide de référence.

### MySQL Administrator

---

*MySQL Administrator* est un outil d'administration (bases, tables, utilisateurs), de sauvegarde (*backup*), de restauration (*restore/recovery*) et de surveillance (*database monitoring*). Sur Windows, il se présente sous la forme d'un *Package Windows Installer* (extension *.msi*). L'installation ne pose aucun problème. Vous trouverez la documentation officielle sur le site <http://dev.mysql.com/doc/administrator/en/index.html>.

#### Connexion

La console se lance sous Windows, à partir de **Démarrer/Programmes/MySQL/MySQL Administrator**. Apparaît ensuite l'écran ci-après pour lequel, dans notre cas, il faut saisir le mode de passe de **root** sur le serveur local.

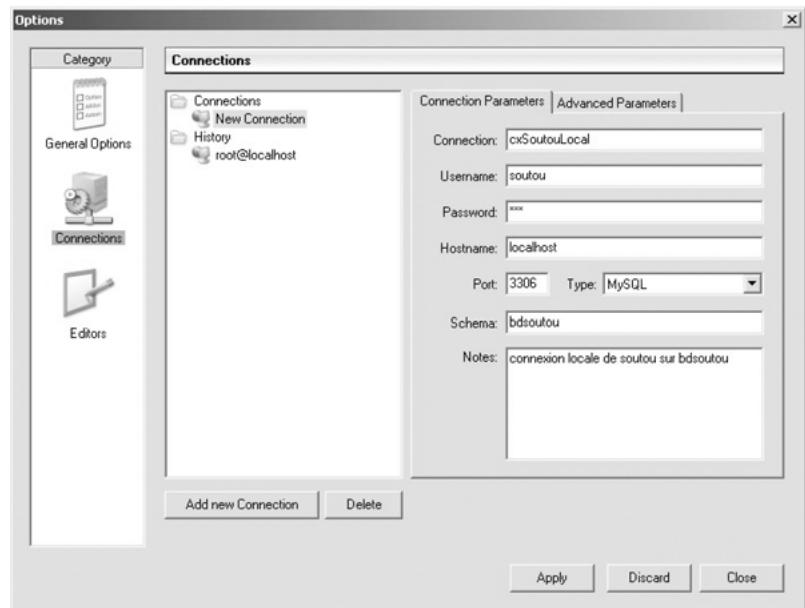
**Figure 10-1** Connexion à un serveur



## Connexion nommée

L'écran suivant décrit l'interface accessible (choix « ... » à côté de `Stored Connection`) pour prédefinir 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

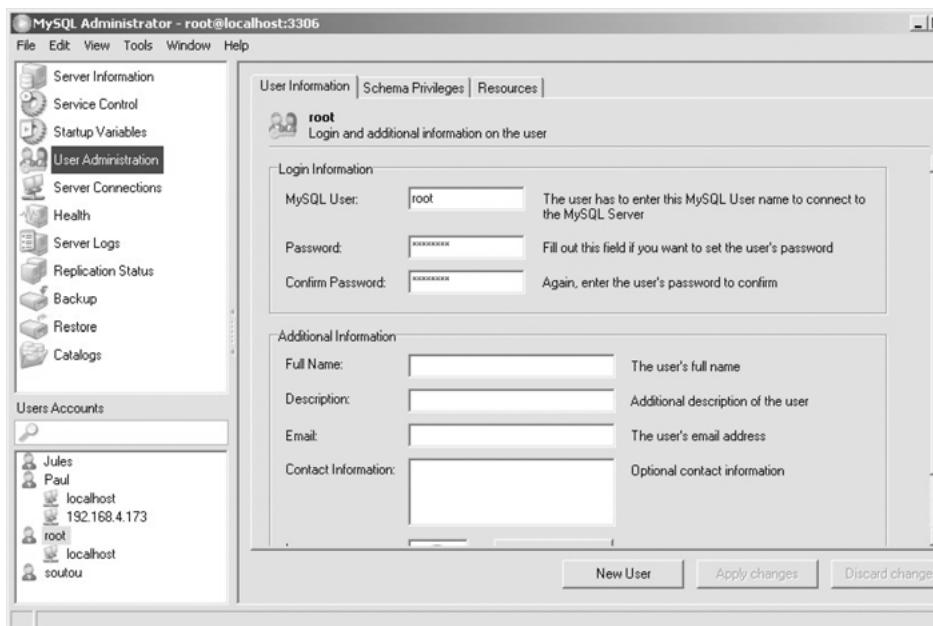


Une fois la connexion choisie, il faut s'identifier au niveau de la base de données cible.

## Liste des accès utilisateur

Lorsque la connexion est établie pour l'utilisateur choisi, il est possible de gérer une tout en respectant ses propres prérogatives. Ici, nous choisissons, sous `root` en local, lister les accès utilisateur. En sélectionnant un accès utilisateur (ici `root` sur `localhost`), il est possible de modifier ces caractéristiques et ces privilèges.

Figure 10-3 Liste des utilisateurs

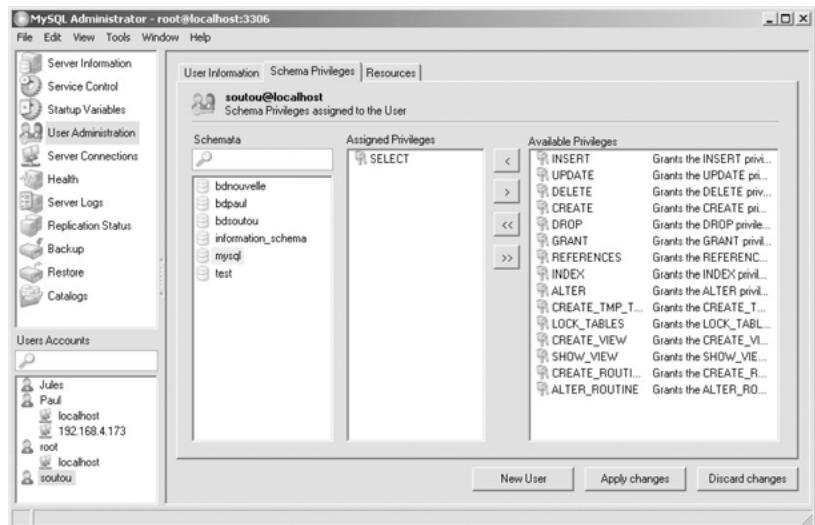


Quand vous ajoutez un nouvel utilisateur, pensez à autoriser sa connexion à partir de chaque machine cliente. Clic droit sur l'utilisateur dans la fenêtre principale Add Host To Which The User Can Connect. Ensuite, il faudra lui allouer des privilèges sur chaque base de données autorisée.

## Gestion des privilèges

En sélectionnant un utilisateur, l'onglet Schema Privileges permet, graphiquement d'affecter ou de révoquer tout privilège sur toute base. Dans l'écran suivant, `root` affecte l'accès utilisateur `soutou` en local le privilège SELECT sur la base `mysql`.

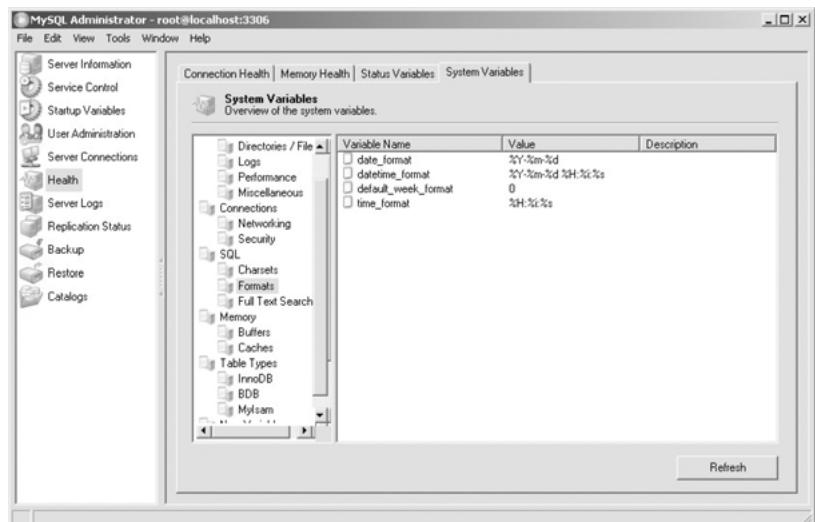
**Figure 10-4** Caractéristiques d'un utilisateur



## Caractéristiques système

Le choix **Health** de la fenêtre principale renseigne les occupations mémoire des process en cours sur le 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.

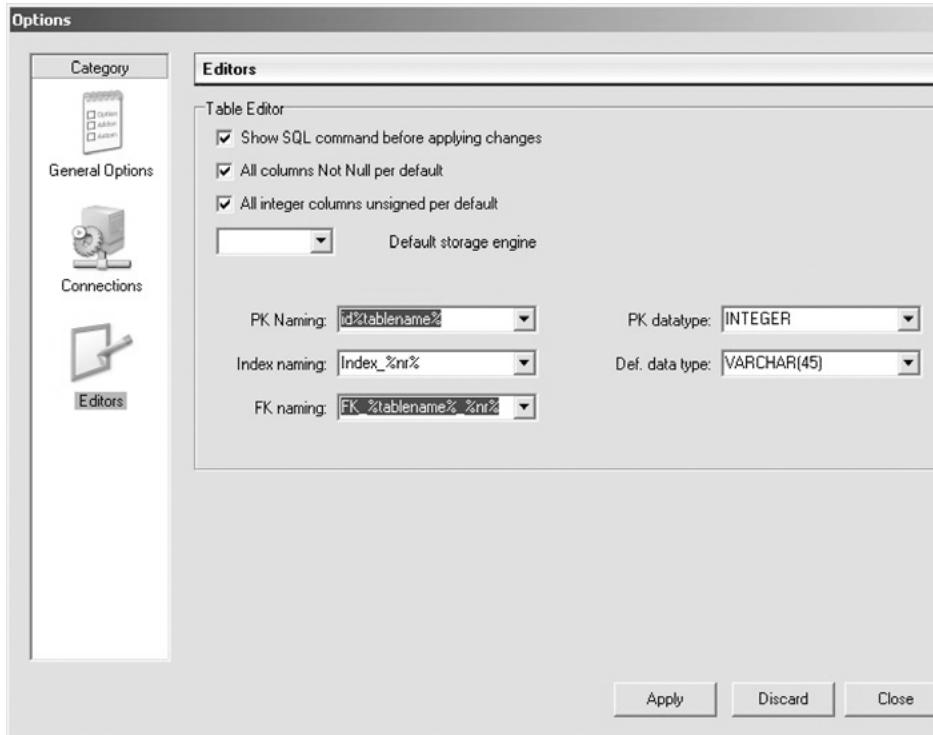
**Figure 10-5** Variables système



## Options scripts SQL

L'option Tools/Options (choix Editors) permet de personnaliser les scripts SQL qui seront automatiquement générés suite à des modifications structurelles d'une base (ajout d'une table, d'une contrainte, etc.). L'écran suivant présente les options par défaut. Par exemple, si une contrainte de clé primaire est ajoutée à une table Etudiant, la colonne nommée idEtudiant sera générée avec le type INTEGER et la valeur NOT NULL.

Figure 10-6 Options par défaut des scripts SQL



## Composition d'une base

Le choix Catalogs de la fenêtre principale donne des informations sur les tables, index, vues et procédures cataloguées d'une base. L'écran suivant présente la liste des tables de la base bdsoutou.

**Figure 10-7** Liste des tables d'une base

The screenshot shows the MySQL Administrator interface with the title bar "MySQL Administrator - root@localhost:3306". The left sidebar contains links like Server Information, Service Control, Startup Variables, User Administration, Server Connections, Health, Server Logs, Replication Status, Backup, Restore, and Catalogs. Under Catalogs, the "bdsoutou" schema is selected. The main pane displays the "Schema Tables" tab, showing a list of tables in the "bdsoutou" schema. The table data is as follows:

Table Name	Engine	Rows	Data length	Index length	Update time
Avion	InnoDB	4	16 kB	16 kB	
Compagnie	InnoDB	2	16 kB	0 kB	
Installeur	InnoDB	11	16 kB	32 kB	
Logiciel	InnoDB	8	16 kB	16 kB	
PCSeuls	InnoDB	4	16 kB	0 kB	
Poste	InnoDB	12	16 kB	48 kB	
Salle	InnoDB	6	16 kB	16 kB	
Segment	InnoDB	3	16 kB	0 kB	
Softs	InnoDB	8	16 kB	0 kB	
Types	InnoDB	6	16 kB	0 kB	

Below the table, there are buttons for "Details >>" and "Create Table", along with other maintenance buttons: "Edit Table", "Maintenance", and "Refresh".

## Composition d'une table

En double-cliquant sur le nom d'une table, on obtient le détail des colonnes (la composition des clés étrangères également), ainsi que les caractéristiques système relatives au mode de stockage. L'écran suivant nous révèle la structure de la table Poste située dans la base de données bdsoutou.

Figure 10-8 Détail des colonnes d'une table

The screenshot shows the MySQL Table Editor interface for the 'Poste' table in the 'bdsoutou' database. The 'Columns and Indices' tab is selected. The table structure is as follows:

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value	Comment
nPoste	VARCHAR(7)	✓		<input type="checkbox"/> BINARY		
nomPoste	VARCHAR(20)	✓		<input type="checkbox"/> BINARY		
indIP	VARCHAR(11)			<input type="checkbox"/> BINARY	NULL	
ad	VARCHAR(3)			<input type="checkbox"/> BINARY	NULL	
typePoste	VARCHAR(9)			<input type="checkbox"/> BINARY	NULL	
nSalle	VARCHAR(7)			<input type="checkbox"/> BINARY	NULL	
nbLog	TINYINT(2)			<input type="checkbox"/> UNSIGNED <input type="checkbox"/> ZEROFILL	0	

The 'Indices' tab is selected, showing three foreign key constraints:

- fk\_Poste\_indIP\_Segment
- fk\_Poste\_nSalle\_Salle
- fk\_Poste\_typePoste\_Types

The 'Foreign Keys' tab displays the settings for the first constraint:

Key Name:	fk_Poste_indIP_Segme	Ref. Table:	Segment
On Delete:	Restrict	Column:	indIP
On Update:	Restrict	Reference Column:	indIP

At the bottom are buttons for 'Apply Changes', 'Discard Changes', and 'Close'.

# Composition des index

L'onglet **Schema Indices** détaille les index présents dans une base. L'écran suivant montre les trois index de la table **Poste** située dans la base **bdsoutou**.

**Figure 10-9 Détail des index**

The screenshot shows the MySQL Administrator interface with the title bar "MySQL Administrator - root@localhost:3306". The left sidebar contains links like Server Information, Service Control, Startup Variables, User Administration, Server Connections, Health, Server Logs, Replication Status, Backup, Restore, and Catalogs. Below that is a "Schema" section listing databases: bdnouvelle, bdpaul, bdsoutou, information\_schema, mysql, and test. The main panel has tabs for Schema Tables, Schema Indices, Views, and Stored procedures. The Schema Indices tab is selected, showing the "bdsoutou All bdsoutou indices" section. A tree view lists indexes: PRIMARY (Installer, Logiciel), un\_installation (Installer, Logiciel), fk\_Logiciel\_typeLog\_Types (Logiciel), PRIMARY (Poste), fk\_Poste\_IndP\_Segment (Poste), fk\_Poste\_nSalle\_Salle (Poste), fk\_Poste\_typePoste\_Types (Poste), PRIMARY (Salle), fk\_Salle\_IndP\_Segment (Salle), PRIMARY (Segment), PRIMARY (Types). A table below provides detailed information about these 16 indices across 17 columns. The table includes columns for Index Name, Table Name, Type, Unique, and Not Null status.

Index Name	Table Name	Type	Unique	Not Null
PRIMARY	Installer	BTREE	UNIQUE	NOT NULL
PRIMARY	Logiciel	BTREE	UNIQUE	NOT NULL
un_installation	Installer	BTREE		
fk_Logiciel_typeLog_Types	Logiciel	BTREE		
PRIMARY	Logiciel	BTREE	UNIQUE	NOT NULL
fk_Poste_IndP_Segment	Poste	BTREE		
↳ indIP				
↳ fk_Poste_nSalle_Salle	Poste	BTREE		
↳ nSalle				
↳ fk_Poste_typePoste_Types	Poste	BTREE		
↳ typePoste				
PRIMARY	Poste	BTREE	UNIQUE	NOT NULL
fk_Salle_IndP_Segment	Salle	BTREE		
PRIMARY	Salle	BTREE	UNIQUE	NOT NULL
PRIMARY	Segment	BTREE	UNIQUE	NOT NULL
PRIMARY	Types	BTREE	UNIQUE	NOT NULL

## Modification d'un schéma

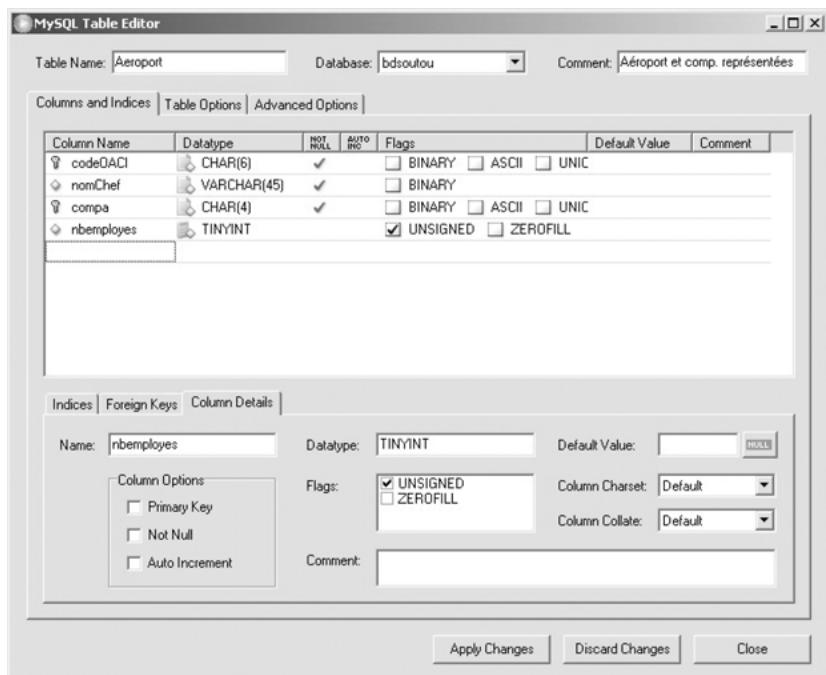
Étudions à présent la modification d'un schéma par le fait d'ajouter une table, puis une nouvelle contrainte à une table existante.

### Création d'une table

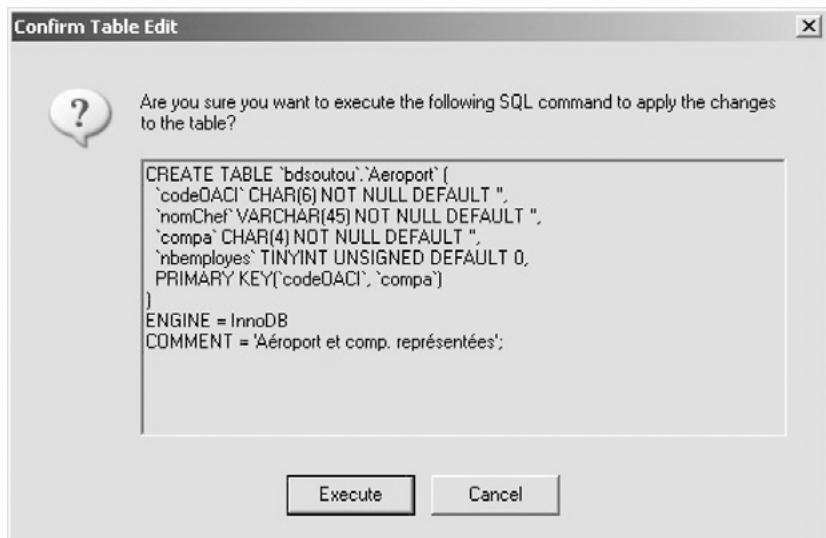
L'écran suivant illustre la création de la table **Aeroport** dont les colonnes **codeOACI** et **compa** composent la clé primaire.

À l'issue de cette création, après avoir cliqué sur **Apply Changes** 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 10-10** Création d'une table



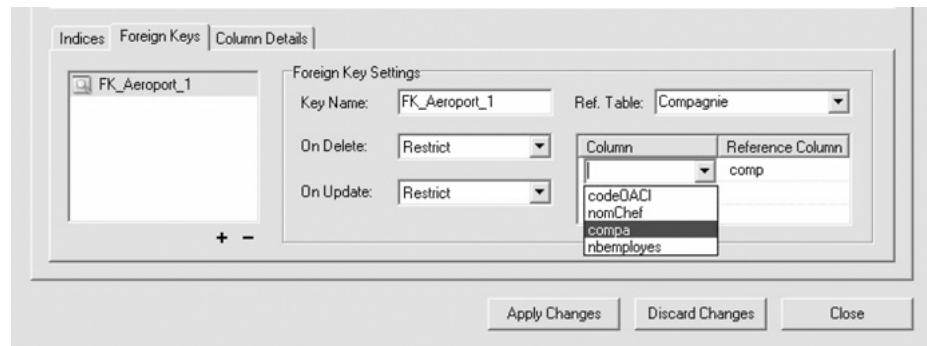
**Figure 10-11** Script SQL de création de table généré



## Ajout contrainte

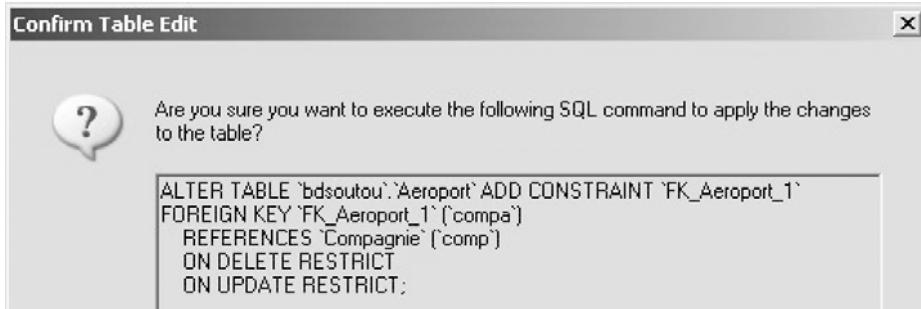
Insérons à présent la clé étrangère reliant la table Aeroport à la table Compagnie (« père »). Dans l'ordre, ajouter le nom de la contrainte « + », choisir la table cible puis colonne de la table « fils » (ici compa). Enfin, vous pouvez restreindre les actions en cascades.

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.

*MySQL Query Browser* est un outil graphique composé d'une interface pour créer, exécuter et optimiser des instructions SQL (LDD, LMD, LID et LCD). Bien que toutes les instructions générées par le biais de cet outil soient exécutables en ligne de commande (c'est une caractéristique pour vous qui avez ingurgité toute la syntaxe SQL...), la documentation affirme que certaines requêtes peuvent y être composées graphiquement de manière plus intuitive (ce n'est pas le cas pour le QBE d'Access).

*MySQL Query Browser* convient à des bases MySQL de version postérieure à 4.0. Sur Windows, il se présente sous la forme d'un *Package Windows Installer* (extension *.msi*). L'installation ne pose aucun problème. Vous trouverez la documentation officielle sur <http://dev.mysql.com/doc/query-browser/en/index.html>.

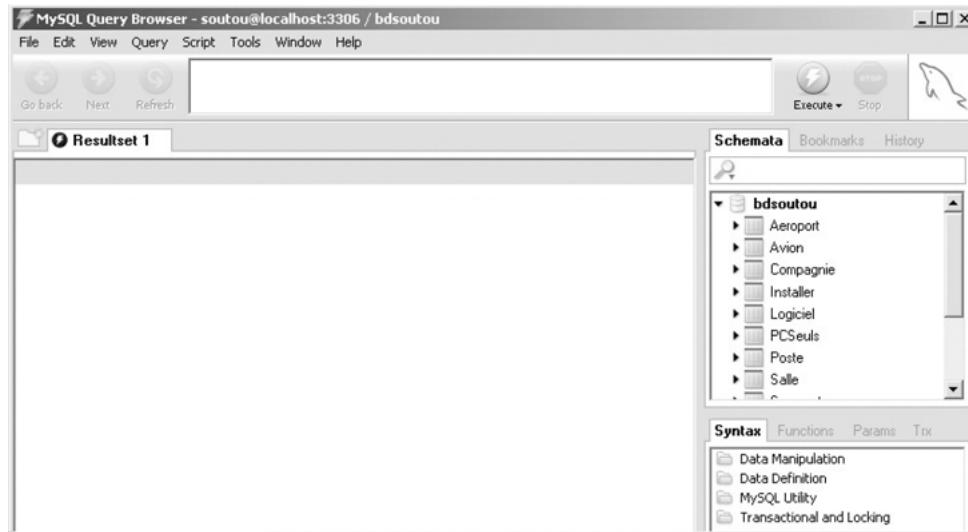
Une fois connecté avec la même interface que *MySQL Administrator* (dans laquelle vous choisissez l'utilisateur, le serveur, la base et saisissez le mot de passe) arrive la fenêtre principale.

## Fenêtre principale

L'interface inclut la palette d'outils suivante :

- *Query Toolbar* (zone de texte en haut au centre) pour créer et exécuter des instructions (requêtes et tout ordre de création ou de manipulation), et naviguer (avec *Next* et *Back*) dans l'historique de ces commandes.
- *Resultset* qui affiche le résultat d'une requête avec, en bas, un bandeau pour les onglets de contrôle et l'affichage d'éventuelles erreurs.
- *Script Editor* (onglets *Edit*, *Apply changes*, *Discard changes*, *First*, *Last*, *Search* qui se trouvent en bas de la fenêtre de résultats) vous donne le contrôle pour modifier et rechercher manuellement des données parmi les enregistrements extraits d'une requête.
- *Object Browser* vous permet de sélectionner les colonnes des tables des bases qui sont accessibles. Vous pouvez double-cliquer ou faire des glisser-déposer de colonnes (même de différentes tables pour composer des jointures) dans la zone de commandes. Les *bookmarks* et les historiques sont aussi gérés à ce niveau.
- *Inline Help* (zone de texte en bas à droite) vous donne un accès direct à l'aide de toutes les commandes SQL et fonctions. La syntaxe s'affiche dans le *Resultset*.

Figure 10-14 Fenêtre principale de MySQL Query Browser



## Extraction

L'écran suivant illustre une jointure composée à l'aide d'un glisser-déposer des colonnes concernées. Toute la requête ne se compose pas automatiquement (j'ai dû écrire manuellement le signe « = » du prédictat de jointure et la condition du deuxième prédictat).

Je ne suis pas un professionnel de cette interface qui a quand même, à mon sens, encore de progrès à faire pour faciliter vraiment l'écriture des requêtes (notamment pour les jointures avec sous-requêtes). Notez que c'est l'écriture relationnelle de la jointure qui est choisie par l'outil (ce qui confirme mes dires au chapitre 4 section *Jointure relationnelle*).

Figure 10-15 Jointure (requête multitable)

The screenshot shows the MySQL Query Browser interface. The query window displays the following SQL code:

```
SELECT A.immat, A.typeAvion, A.capacite, A.compa, C.nomComp
FROM Avion A, Compagnie C
WHERE A.capacite>200 AND A.compa= C.comp
```

The results are shown in a table titled "Resultset 1":

immat	typeAvion	capacite	compa	nomComp
F-GAFU	A320	210	AF	Air France
F-GLFS	A320	220	AF	Air France

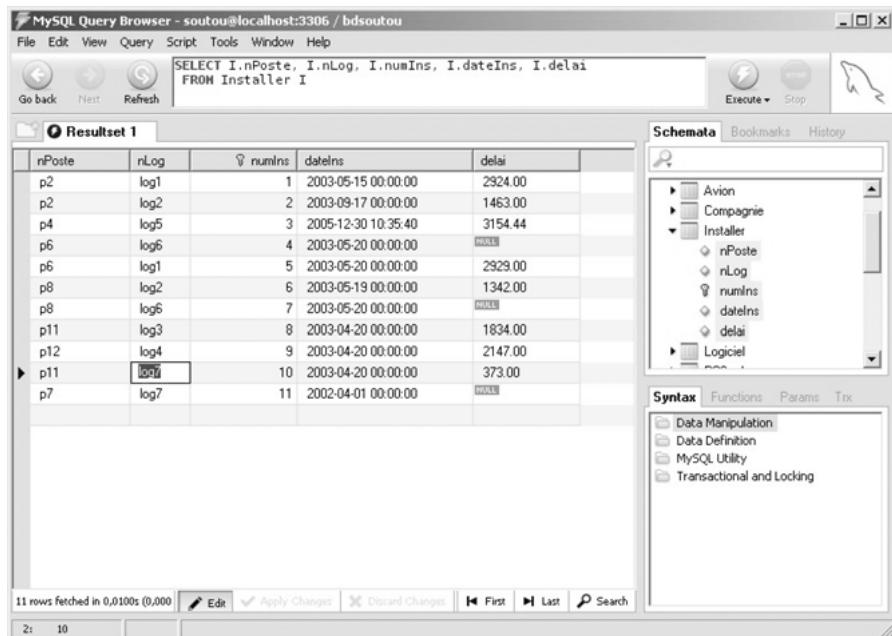
The right side of the interface includes a "Schema" tree view showing tables like Avion, Compagnie, and Installer, and a "Syntax" tab with various database manipulation options.

## Modification

L'écran suivant illustre une requête (monotable) dont le résultat permet de modifier colonne de la table en sélectionnant l'onglet Edit puis Apply changes.

On peut également ajouter un enregistrement (équivalent à INSERT) en se positionnant sur l'enregistrement vide en fin de résultat, puis faire un clic droit sur le choix Add Row, supprimer un enregistrement (équivalent à DELETE), le sélectionner, puis faire un clic droit sur le choix Delete Row(s). Toute modification devra bien sûr respecter les éventuelles contraintes d'unicité, de non nullité et d'intégrité référentielle.

Figure 10-16 Modification d'un enregistrement



## phpMyAdmin

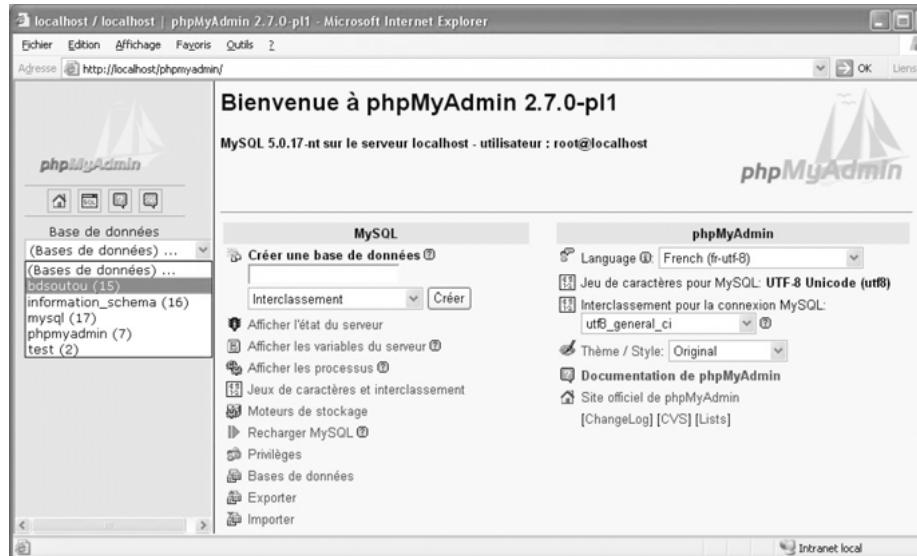
*phpMyAdmin* est un outil d'administration d'un ou de plusieurs serveurs MySQL. Les fonctionnalités principales de cet outil sont :

- de créer, modifier et supprimer des bases de données et des tables, de gérer les utilisateurs et leurs priviléges ;
- d'exécuter toute instruction SQL, même les requêtes par lot, de proposer *Query By Example* pour écrire des requêtes complexes ;
- d'importer des données provenant de fichiers texte ou d'exporter des données aux formats CSV, XML et Latex ;
- de créer des graphiques PDF du schéma de votre base de données ;
- de gérer `mysqli`, la dernière API PHP pour MySQL.

Plusieurs installations sont possibles, individuellement avec Apache et PHP, avec WAMP avec *EasyPHP* (mais la version actuelle ne reconnaît pas encore MySQL 5). Vous trouverez documentation officielle sur [http://www.phpmyadmin.net/pma\\_localized\\_docs/fr](http://www.phpmyadmin.net/pma_localized_docs/fr).

Une fois installé et configuré avec Apache arrive la fenêtre principale qui permet de sélectionner les principales fonctionnalités. Ici je sélectionne la base `bdsoutou`.

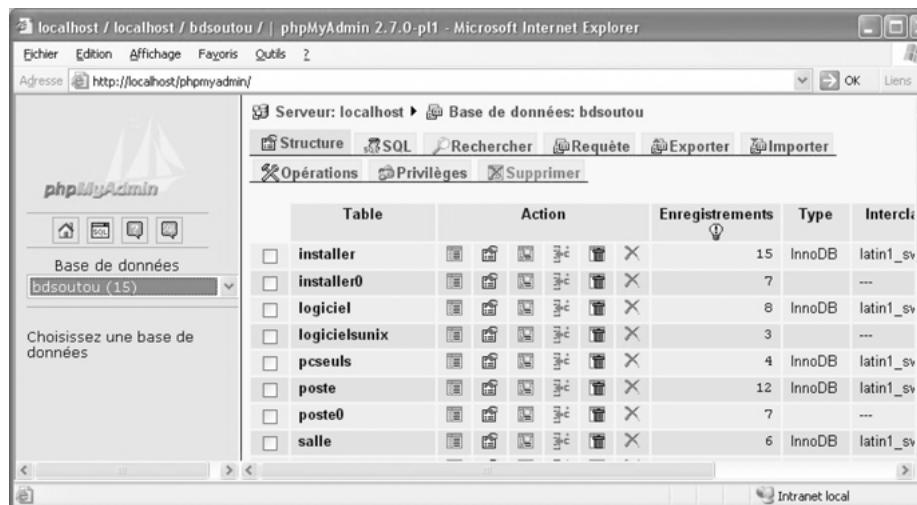
**Figure 10-17** phpMyAdmin



## **Composition de la base**

Les tables et vues (quinze en tout) apparaissent, sur lesquelles différentes actions sont possibles : les icônes dans la colonne Action (afficher, structurer, rechercher, insérer, vider et supprimer).

**Figure 10-18** Composition de la base



## Structure d'une table

En sélectionnant la structure d'une table (ou vue), il est possible d'ajouter, de modifier ou de supprimer une colonne ou une contrainte de clé ( primaire ou étrangère).

**Figure 10-19 Structure d'une table**

The screenshot shows the phpMyAdmin interface in Microsoft Internet Explorer. The left sidebar lists the database 'bdsoutou' (15 tables) and the table 'installer'. The main area displays the structure of the 'installer' table:

Champ	Type	Interclassement	Attributs	Null	Défaut	Extra
nPoste	varchar(7)	latin1_swedish_ci		Oui	NULL	
nLog	varchar(5)	latin1_swedish_ci		Oui	NULL	
numIns	int(5)			Non		auto_increment
dateIns	timestamp			Non	CURRENT_TIMESTAMP	
delaI	decimal(8,2)			Oui	NULL	

Below the table structure, there are sections for indexes and space usage:

Index:	Type	Cardinalité	Action	Champ	Type	Espace
PRIMARY	PRIMARY	15		numIns	Données	16 384 Octets
un_installation	UNIQUE	15		nPoste	Index	32 768 Octets
fk_Installer_nLog_Logiciel INDEX		15		nLog	Total	49 152 Octets

Buttons at the bottom include 'Ajouter 1 champ(s)', 'Exécuter', and 'Intranet local'.

## Administrer une table

L'onglet Opérations au niveau d'une table permet de modifier les caractéristiques d'une table à l'échelle de la *database*.

Figure 10-20 Administrer une table

localhost / localhost / bdssoutou / installer | phpMyAdmin 2.7.0-pl1 - Microsoft Internet Explorer

Base de données: bdssoutou (15)

Afficher Structure SQL Rechercher Insérer Exporter Importer Opérations

Ordonner la table par: nLog (à refaire après insertions/destructions) Croissant Exécuter

Changer le nom de la table pour: installer Exécuter

Déplacer la table vers (base.table): information\_schema . installer Exécuter

Copier la table vers (base.table): bdssoutou .

Maintenance de la table

Vérifier la table Défragmenter la table Recharger la table ("FLUSH")

Commentaires sur la table Exécuter

Intranet local

## Extractions

L'onglet SQL permet de saisir une instruction SQL (LDD, LMD, LID ou LCD). Ici j'ai fait un collage de la requête de division de l'exercice du chapitre 4. Malheureusement j'obtiens une erreur dans cette interface, alors qu'en ligne de commande deux lignes sont retournées (postes 6 et 8)

Figure 10-21 Requête SQL

localhost / localhost / bdssoutou | phpMyAdmin 2.7.0-pl1 - Microsoft Internet Explorer

Base de données: bdssoutou

Structure SQL Rechercher Requête Exporter Importer Opérations

Priviléges Supprimer

Exécuter une ou des requêtes sur la base bdssoutou:

```
SELECT nomPoste FROM Poste p WHERE NOT EXISTS
 (SELECT i2.nLog FROM Installer i2 WHERE i2.nPoste = 'p2' AND i2.nLog NOT IN
 (SELECT i1.nLog FROM Installer i1 WHERE i1.nPoste = p.nPoste))
AND NOT EXISTS
 (SELECT i1.nLog FROM Installer i1 WHERE i1.nPoste = p.nPoste AND i1.nLog NOT IN
 (SELECT i2.nLog FROM Installer i2 WHERE i2.nPoste = 'p2')) AND NOT (nPoste = 'p2');
```

Conserver cette requête dans les signets: divisionChap4

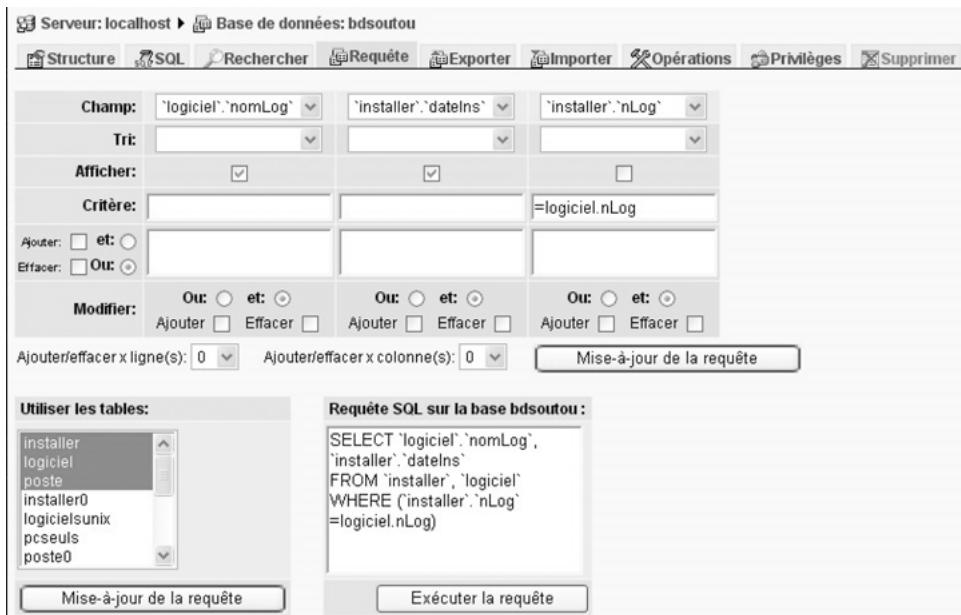
Signet visible pour les autres utilisateurs  Remplacer un signet existant du même nom

Réafficher la requête après exécution Exécuter

Intranet local

L'onglet **Requête** aide à la construction de requêtes SQL (de type *QBE*). Dans l'écran suivant, on compose une jointure entre les tables *installer* et *logiciel*. Notez que la clause de jointure doit être saisie explicitement. Ici on affiche le nom et la date d'installation des logiciels existant sur des postes de travail.

Figure 10-22 Assistant QBE



## Rechercher

L'onglet **Rechercher** permet de rechercher des valeurs dans une ou plusieurs tables (ou vues). Ici on recherche les tables qui contiennent un enregistrement dont une colonne contient le mot « Oracle ». Il est ensuite possible d'afficher ou d'effacer chacune des occurrences vérifiant cette condition.

**Figure 10-23** Recherche dans plusieurs tables

The screenshot shows the phpMyAdmin interface in Microsoft Internet Explorer. The left sidebar shows a database named 'bdsoutou (15)' containing tables like 'install', 'logiciel', 'poste', etc. The main area has tabs for Structure, SQL, Rechercher, Requête, Exporter, Importer, Opérations, Priviléges, and Supprimer. The 'Rechercher' tab is active. A search form is displayed with the placeholder 'Choisissez une base de données'. Below it, a search input field contains 'Mot(s) ou Valeur à rechercher (passé-partout: "%")' followed by 'Oracle'. A dropdown menu lists several table names: 'salleposte', 'salleprix', 'salleprixtotal', 'segment', 'softs', and 'types'. There are radio buttons for search types: 'au moins un mot', 'tous les mots', 'phrase exacte', and 'expression régulière'. A note says '\* Séparer les mots par un espace (" ")'. Buttons for 'Exécuter' and 'Intranet local' are at the bottom.

## Exporter

L'onglet Exporter permet de transférer des données (et des structures) d'une ou de plusieurs tables sous différents formats (voir l'écran ci-après).

**Figure 10-24** Exportation au format Word

The screenshot shows the phpMyAdmin interface in Microsoft Internet Explorer. The left sidebar shows the same database structure as Figure 10-23. The 'Export' tab is active. A large panel titled 'Schéma et données de la base' contains a 'Exporter' section with a dropdown menu showing table names ('install', 'logiciel', 'poste', etc.) and a 'Tout sélectionner / Tout désélectionner' section with various export formats (SQL, LaTeX, Microsoft Word 2000, etc.). To the right, there's an 'Options d'exportation Microsoft Word 2000' section with checkboxes for 'Structure' (checked), 'Données' (checked), and 'Remplacer NULL par NULL' (unchecked). Other options include 'Afficher les noms de champ en première ligne' and 'Transmettre' with a file name template '\_DB\_'. At the bottom, there are compression options ('aucune', 'zippé', 'gzippé') and a 'se souvenir du modèle' checkbox. The URL in the address bar is [http://localhost/phpmyadmin/db\\_import.php?lang=fr-utf-8&server=1&collation\\_connection=utf8\\_general\\_ci&db=bdsoutou&goto=db\\_detail](http://localhost/phpmyadmin/db_import.php?lang=fr-utf-8&server=1&collation_connection=utf8_general_ci&db=bdsoutou&goto=db_detail).

## Utilisateurs

L'onglet **Privilèges** autorise la gestion des accès utilisateur avec leur privilèges. En sélectionnant un utilisateur existant, il est aussi possible de modifier ses caractéristiques.

Figure 10-25 Gestion des utilisateurs

The screenshot shows the phpMyAdmin interface for managing MySQL users. The left sidebar lists databases, and the main area shows a table of users:

Utilisateur	Serveur	Mot de passe	Privilèges globaux	"Grant"
N'importe quel	%	--	USAGE	Non
Gerard	%	Oui	SELECT, EXECUTE	Non
root	localhost	Non	ALL PRIVILEGES	Oui

A message at the bottom right says: "Note: phpMyAdmin obtient la liste des privilèges directement à partir des tables MySQL. Le contenu de ces tables peut être modifié à tout moment par l'utilisateur." (Note: phpMyAdmin retrieves the list of privileges directly from the MySQL tables. The content of these tables can be modified at any time by the user.)



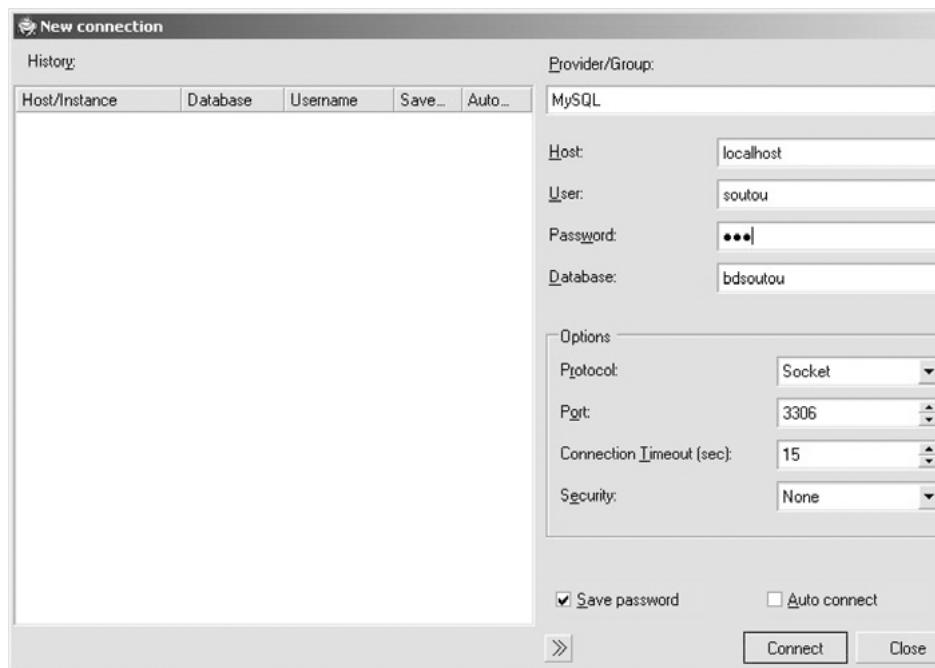
Le message en rouge, en bas de page, vous invite à modifier le mot de passe de root. J'ai eu des mauvaises surprises suite à plusieurs tentatives de modification. Méfiance...

## TOAD for MySQL

*Toad for MySQL* est un outil graphique d'administration et de développement qui est disponible gratuitement dans une version *Preview Release*. Toutefois ce statut de *freeware* semble temporaire, et le produit doit être retéléchargé tous les 60 jours ([http://www.toadsoft.com/toadmysql/toad\\_mysql.htm](http://www.toadsoft.com/toadmysql/toad_mysql.htm)).

Sous Windows, il se présente sous la forme d'une archive d'un installateur (fichier à extension **.msi**) dont l'exécution ne pose aucun problème. Quand la grenouille coasse, *TOAD* est pour vous. Au premier démarrage, des choix de mode d'affichage vous seront demandés. Puis, la suite, vous devrez vous connecter en donnant tous les paramètres nécessaires.

**Figure 10-26 Connexion via TOAD**



Une fois connecté arrive une interface à partir de laquelle, par le menu Tools et Views, les onglets se composent dans la fenêtre principale. L'affichage de ces onglets, qui sont constitués de boutons, listes déroulantes, choix, etc., est contrôlé par le menu Window.

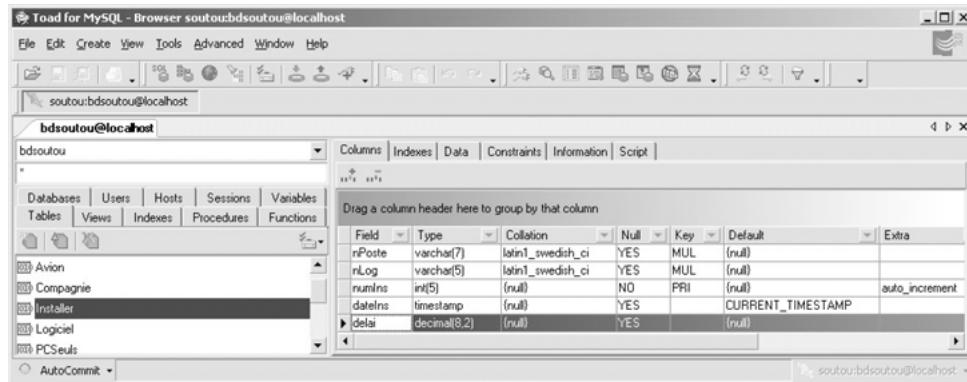
## Administration

Le choix Tools/Database Browser, en sélectionnant une base de données, renseigne les tables, index, vues, procédures cataloguées, privilèges des utilisateurs, etc. Tout objet relevant de la base se retrouve ici.

L'écran suivant présente les onglets disponibles pour la base bdsoutou. On s'intéresse à la composition de la table Installer. Toute modification de cette table au niveau de sa structure est possible (sous réserve de maintenir l'intégrité des données via d'éventuelles contraintes référentielles).

L'onglet Data extrait les lignes d'une table. L'onglet Indexes décrit les index présents sur la table. L'onglet Constraints liste les contraintes référentielles. L'onglet Information énumère les caractéristiques physiques de la table. L'onglet Script restitue le script SQL de création que vous pouvez sauvegarder sous la forme d'un fichier texte.

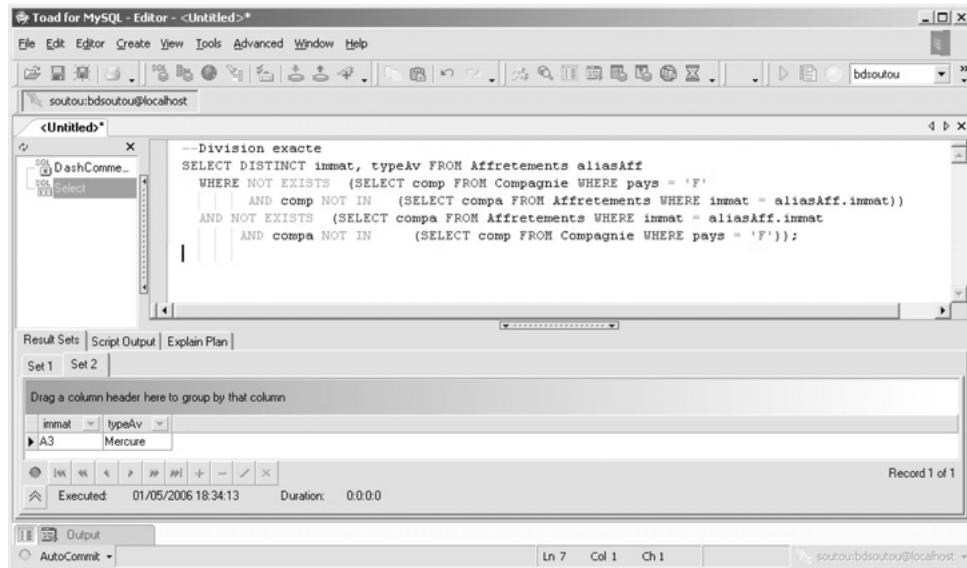
Figure 10-27 Composition d'une base



## Instructions en ligne

Le choix Tools/Editor de la fenêtre principale ouvre plusieurs fenêtres dont la plus importante est une zone de texte contenant une instruction SQL. L'écran suivant présente une requête qui réalise une division (voir chapitre 4). L'onglet Result Sets contient le(s) résultat(s) de l'extraction. L'onglet Explain Plan permet de visualiser le plan d'exécution.

Figure 10-28 Extraction de données



Toute autre instruction SQL (LMD, LDD et LCD) peut être exécutée dans ce mode. Le menu général Create génère automatiquement le début de la syntaxe SQL adéquate (création d'une table, vue, index ou procédure cataloguée).

## Développement

Le choix Tools/Database Browser, ensuite l'onglet Procedure en sélectionnant une procédure (clic droit puis Alter Procedure), permet de modifier puis de reconstruire (Editor/Execute SQL Statement) une procédure existante. Pour en créer une nouvelle, vous avez le choix entre le clic droit puis Create Procedure, ou passer par le menu général Create/Create/Procedure.

**Figure 10-29** Modification d'un sous-programme

```
CREATE PROCEDURE `leNomCompagnieEst` (IN p_inmat 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 bdsoutou.Comptagnie WHERE comp =
(SELECT compa FROM bdsoutou.Avion WHERE immat = p_inmat);
END;
IF flagNOTFOUND THEN
SET p_nomcomp := NULL;
END IF;
END
```

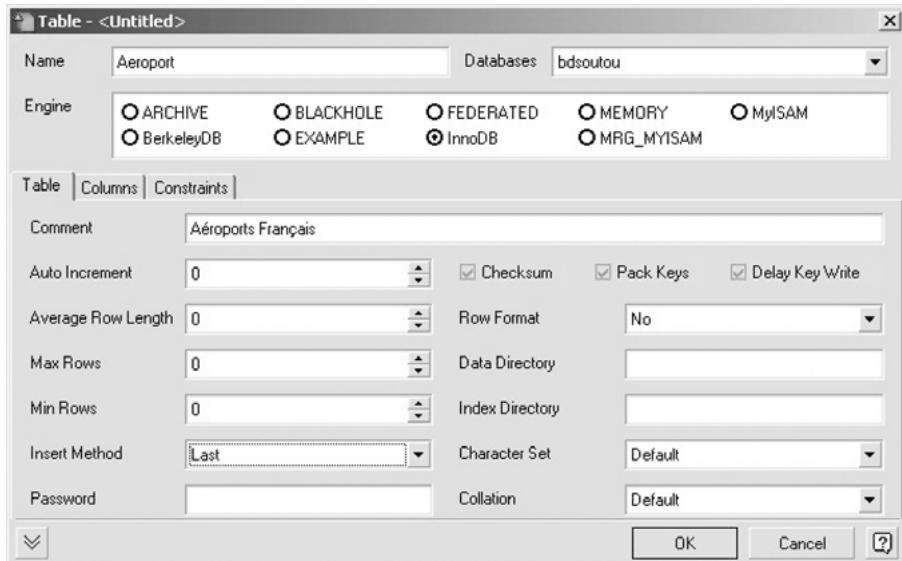
Result Sets Script Output Explain Plan

```
soutou:bdsoutou@localhost 01/05/2006 20:32:51 0:0:0
Command execution successful
CREATE PROCEDURE `leNomCompagnieEst` (IN p_inmat CHAR(6),OUT p_nomcomp VARCHAR(25))
BEGIN
DECLARE flagNOTFOUND BOOLEAN DEFAULT 0;
```

## Création d'objets

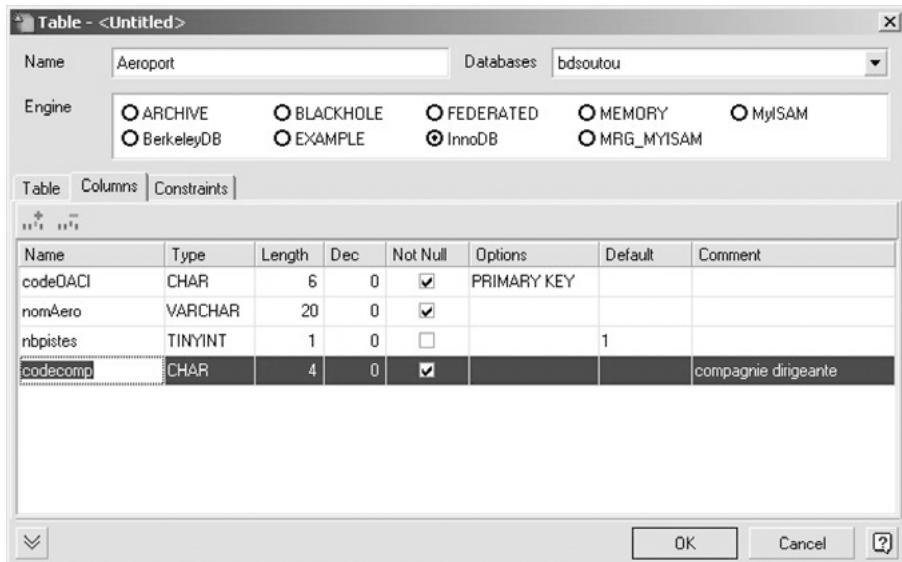
Le menu général Create permet de créer des tables, bases, vues, index et fonctions ou procédures cataloguées. En sélectionnant le choix Table, une fenêtre s'ouvre découvrant plusieurs onglets pour caractériser la table. Le premier onglet permet entre autres de définir le moteur de stockage, le nom, la base et d'autres informations que l'écran suivant illustre.

**Figure 10-30** Création d'une table (onglet Table)



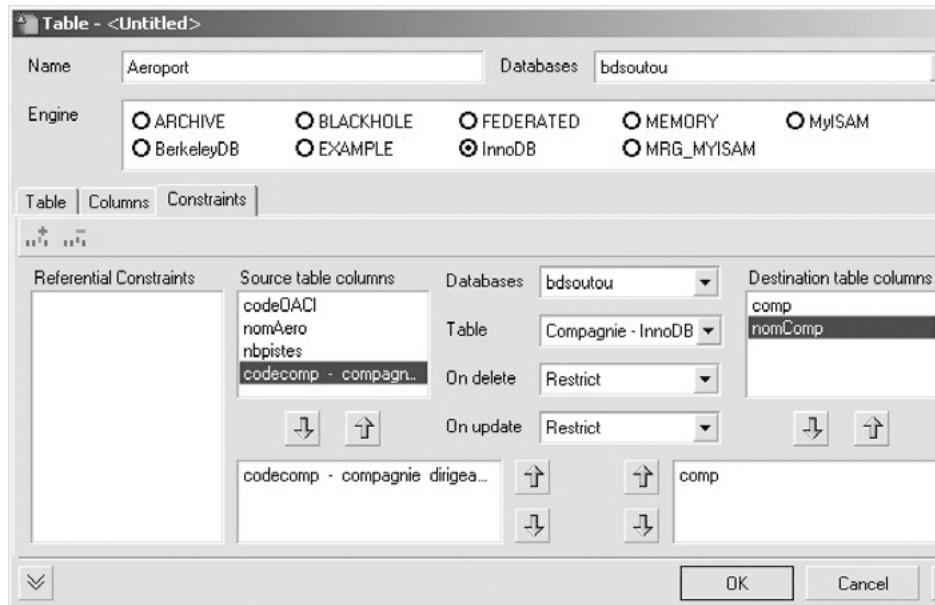
Le second onglet donne la possibilité de structurer la table (description de chaque colonne des contraintes en ligne).

**Figure 10-31** Création d'une table (onglet Columns)



Le dernier onglet permet de définir les éventuelles clés étrangères de la table. J'avoue avoir des problèmes pour en créer une qui soit implémentée dans la base... Vous essayerez vous-même.

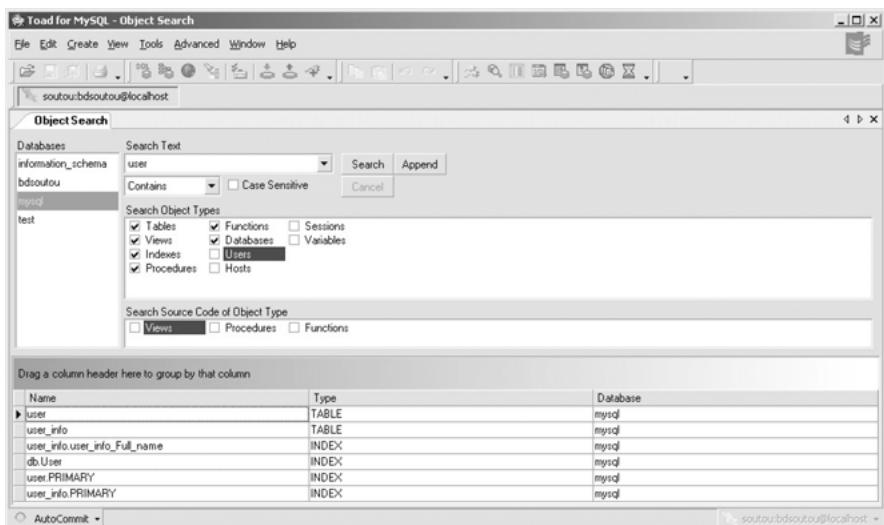
**Figure 10-32** Création d'une table (onglet Constraints)



## Recherche d'objets

Une fonctionnalité intéressante concerne la recherche d'objets (choix Tools/Object Search). Dans l'exemple suivant, on extrait tous les objets de la base de données mysql dont le nom contient la chaîne « user ». Notez aussi la possibilité de chercher un identifiant (par exemple le nom d'une table) au sein du corps de procédures, fonctions ou vues (les déclasseurs ne sont pas ici pris en compte visiblement).

Figure 10-33 Recherche d'objets



## Exportations

Le choix Tools/Export Wizard lance un assistant (un peu semblable à celui d'Access) sélectionnant une (ou plusieurs) table(s) ou une (ou plusieurs) base(s) de données afin d'exporter les données sous différents formats. Sont permis, notamment, les fichiers text (avec séparateurs entre colonnes), fichiers XML ou html, insertion INSERT préécrivées.

La partie de code suivante présente les deux premières lignes du fichier Installer.xml généré suite à l'exportation, dans ce format, de la table de même nom.

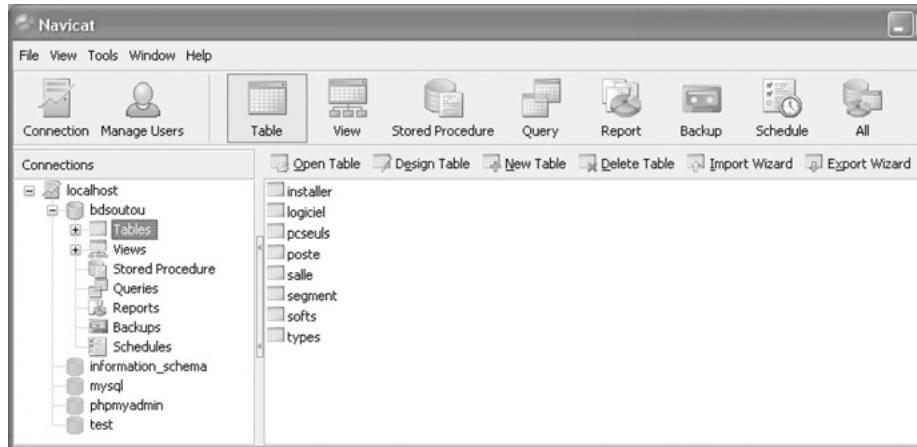
```
<Installer><NPOSTE>p2</NPOSTE><NLOG>log1</NLOG><NUMINS>1</NUMINS>
<DATEINS>15/05/2003 00:00:00</DATEINS><DELAI>2924,00</DELAI>
</Installer>
<Installer><NPOSTE>p2</NPOSTE><NLOG>log2</NLOG><NUMINS>2</NUMINS>
<DATEINS>17/09/2003 00:00:00</DATEINS><DELAI>1463,00</DELAI>
</Installer>
...
...
```

## Navicat

Navicat est un outil assez simple et intuitif. Il ravira ceux qui apprécient concevoir des requêtes graphiquement. Ici le QBE est bien supérieur à celui de *phpMyAdmin*. La documentation officielle se trouve à <http://support.navicat.com>.

Sous Windows, il se présente sous la forme d'un exécutable dont l'installation ne pose aucun problème. Au premier démarrage, vous devez vous connecter en donnant tous les paramètres nécessaires. La fenêtre principale s'affiche.

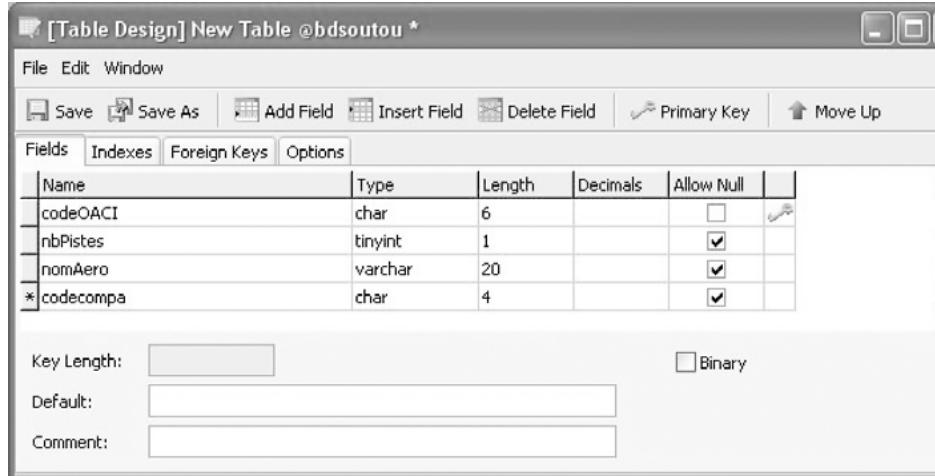
**Figure 10-34** Accueil de Navicat



## Création d'une table

Après avoir opté pour le choix **Design Table**, l'onglet **Fields** permet de définir facilement par listes déroulantes, les différentes colonnes de la nouvelle table.

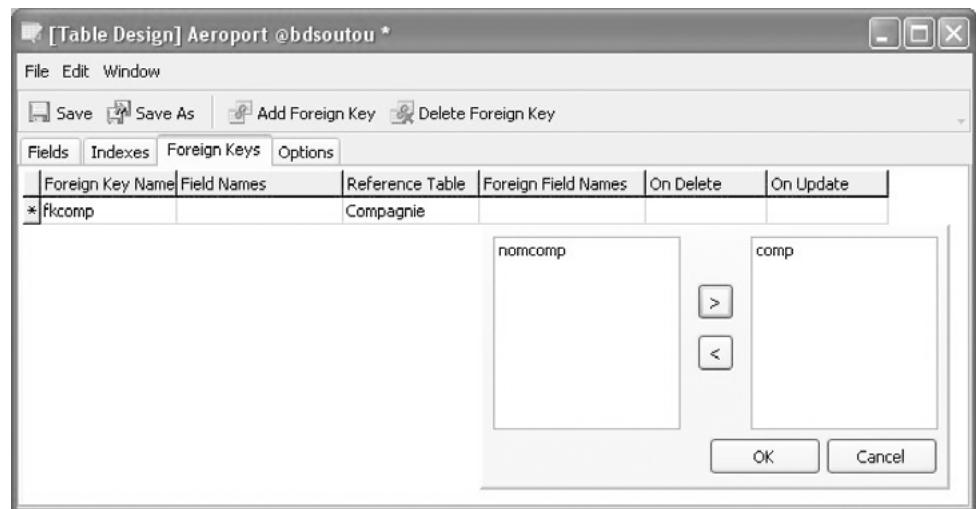
**Figure 10-35** Crédit des colonnes



## Définition d'une contrainte

L'onglet Foreign Key permet de créer aisément, par listes déroulantes, les clés étrangères (une fois que les colonnes sont créées).

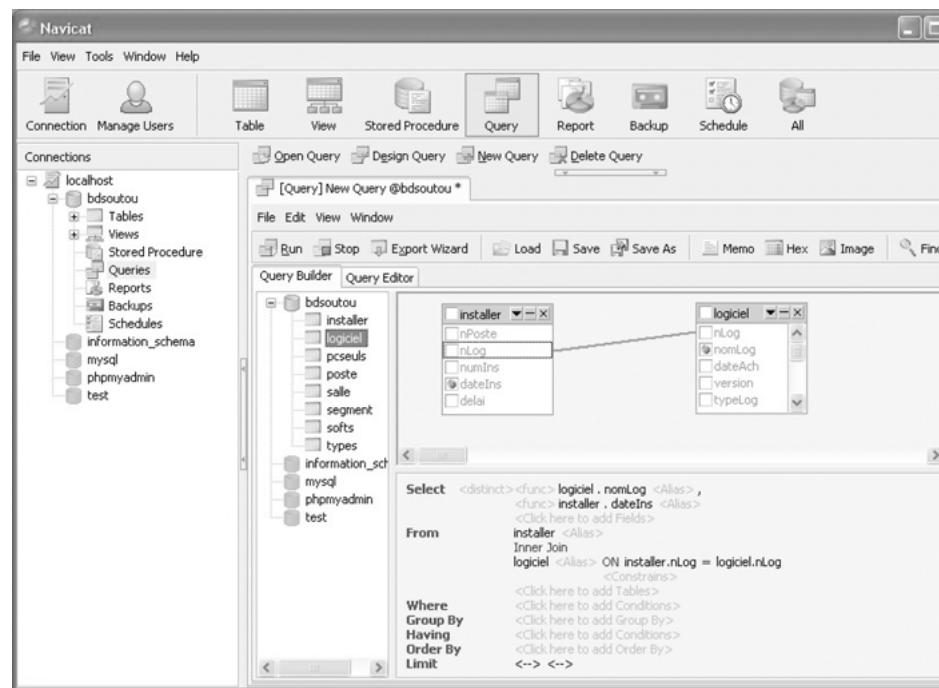
**Figure 10-36** Définition d'une clé étrangère



## Création d'une requête

Là, je dis « bravo ! ». Vous allez vous régaler à composer vos jointures en sélectionnant une colonne, puis en la glissant au niveau de la clé de la table associée. Vous verrez votre requête s'écrire au fur et à mesure que vous agirez sur le diagramme. Vous pourrez également ajouter un niveau de la requête en cliquant aux différents endroits indiqués.

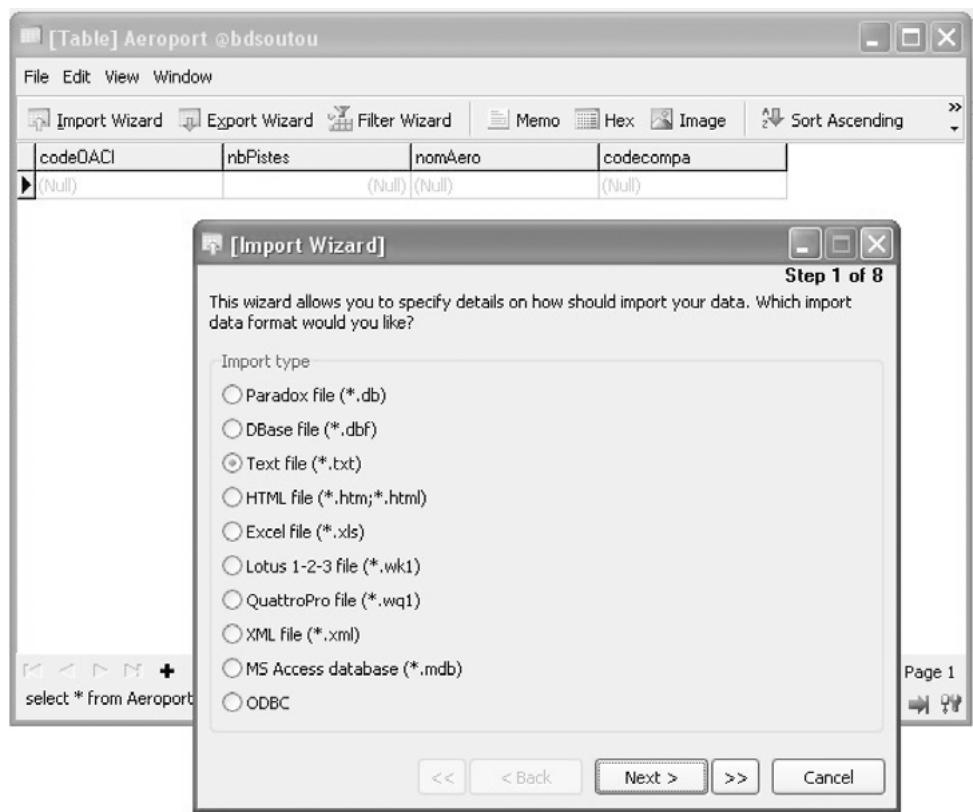
Figure 10-37 Crédit à la création d'une requête



## 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 différents formats différents.

Figure 10-38 Importation de données

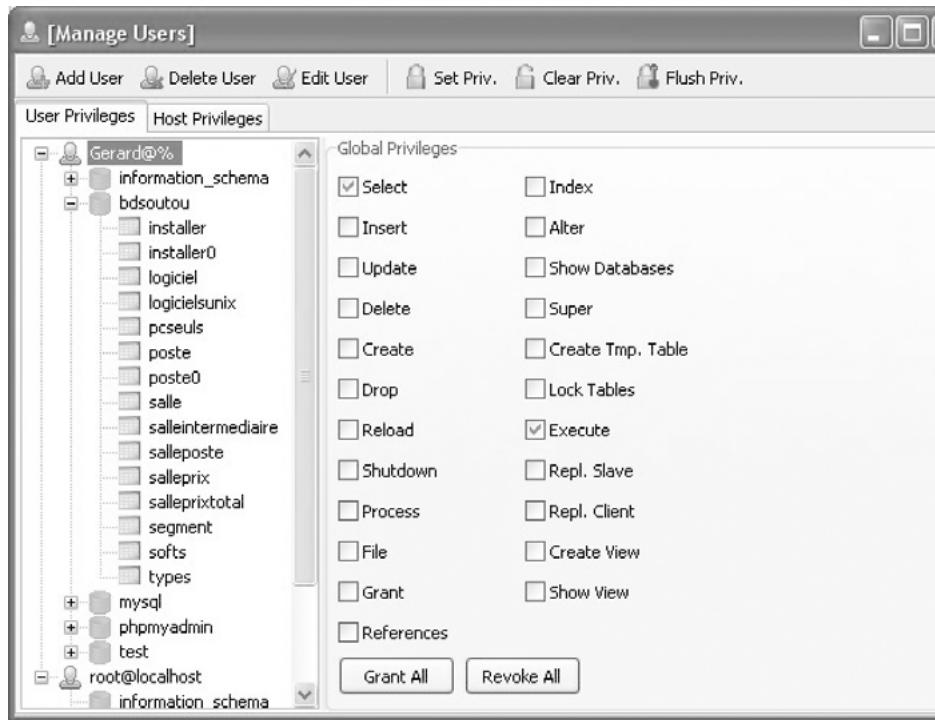


L'assistant d'exportation comporte cinq étapes et reconnaît une vingtaine de formats de données cibles.

## Gestion des utilisateurs

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

Figure 10-39 Gestion des accès et des priviléges



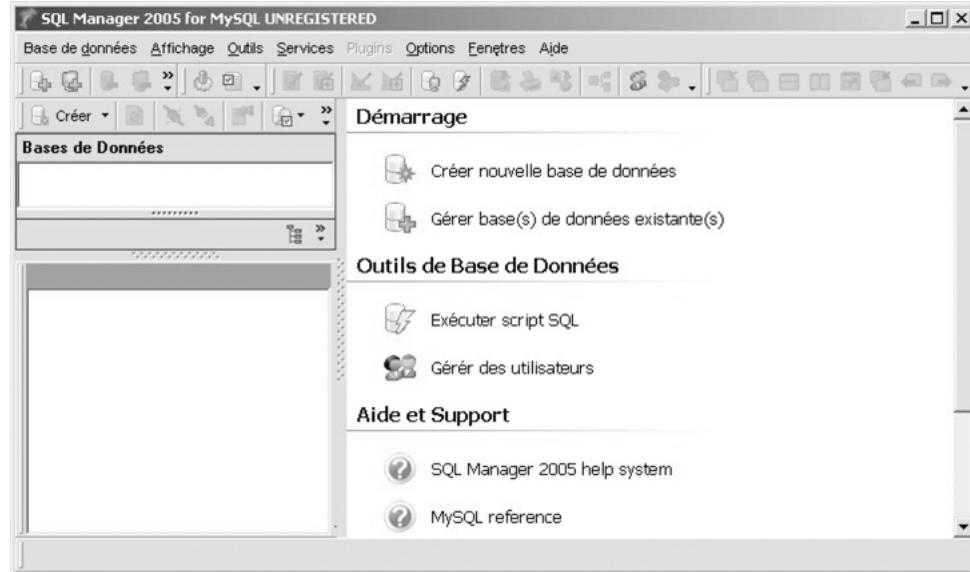
Enfin, il est aussi possible d'écrire des procédures cataloguées, de composer des rapports d'impression et de programmer des tâches d'administration.

## MySQL Manager

*MySQL Manager* (société EMS) est un outil payant, puissant, intuitif et possédant de nombreuses fonctionnalités. Il ravira ceux qui apprécient le QBE qui est ici aussi bien supérieur à celui de *phpMyAdmin*. La documentation officielle se trouve à <http://www.sqlmanager.net/fr/products/mysql/manager/documentation>.

Sous Windows, il se présente sous la forme d'une archive contenant un exécutable. L'installation ne pose aucun problème. Au premier démarrage, des choix de modes d'affichage vous seront demandés. Par la suite, vous devrez vous connecter en donnant tous les paramètres nécessaires. La fenêtre principale apparaît.

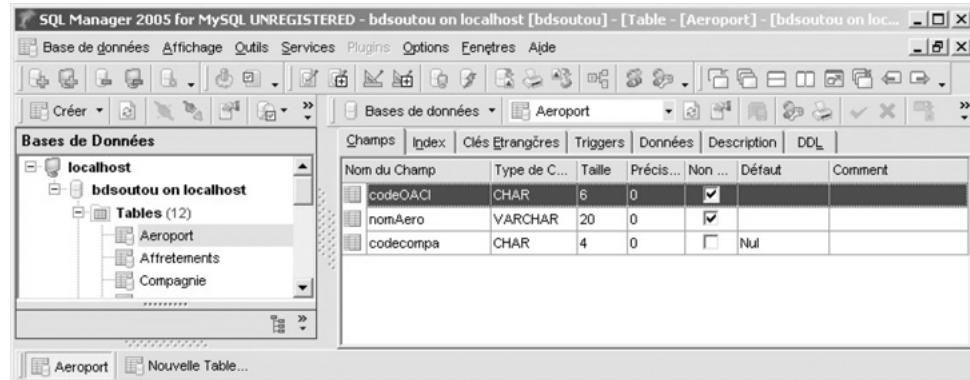
Figure 10-40 Accueil de SQL Manager



## Création d'une table

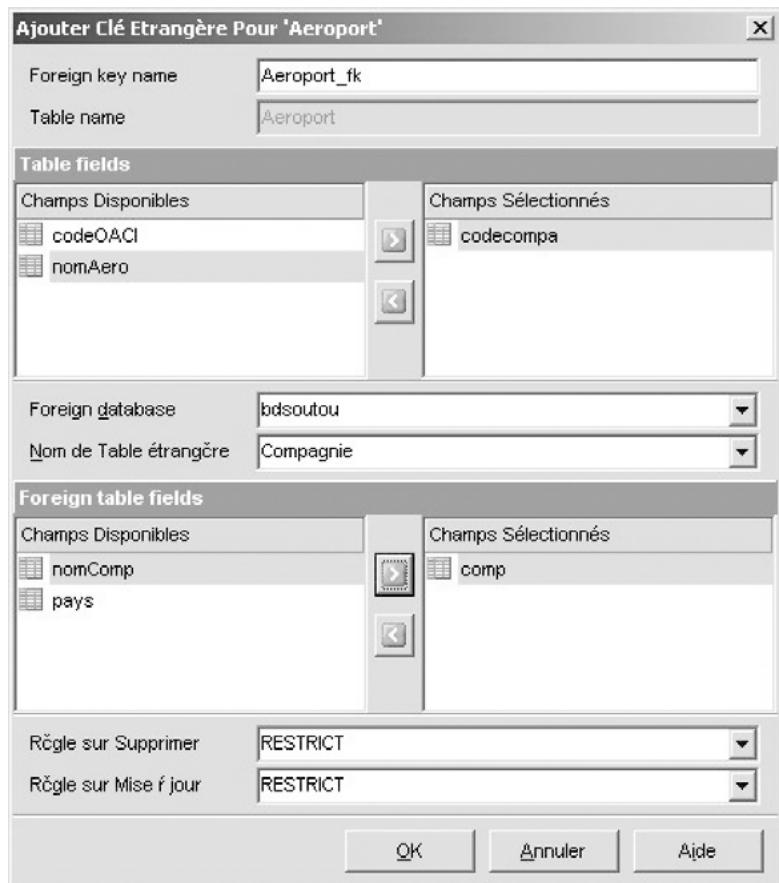
Plusieurs chemins existent pour lancer l'assistant de création d'une table, qui ressemble à ceux déjà étudiés pour les précédents outils. On retrouvera les différents onglets permettant de modifier les colonnes, index, clés étrangères et données. L'onglet DDL contient l'instruction SQL générée.

Figure 10-41 Crédit d'une table



Les clés étrangères se rajoutent après création de la structure totale de la table.

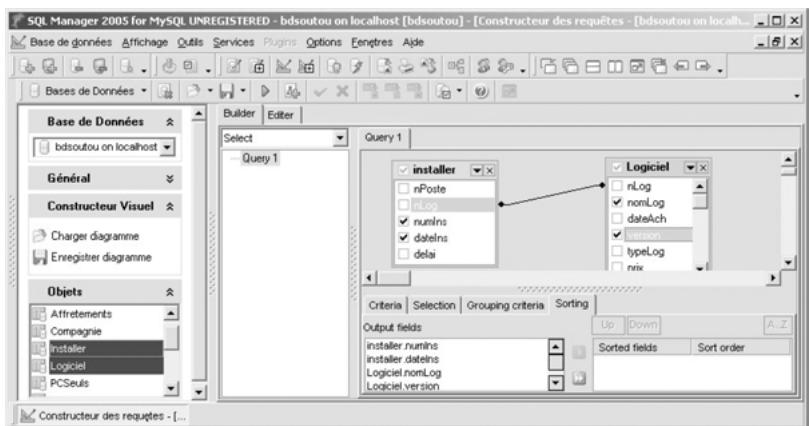
**Figure 10-42** Ajout d'une clé étrangère



## Création d'une requête

Comme pour *Navicat*, « bravo ! » également. Vous trouverez un outil efficace (plus encore que celui de *Navicat*) pour écrire vos requêtes. L'écran suivant illustre une jointure (générée comme pour *Navicat* sous la forme d'une écriture SQL2 avec `INNER JOIN`) affichant quatre champs.

Figure 10-43 Construction graphique d'une requête

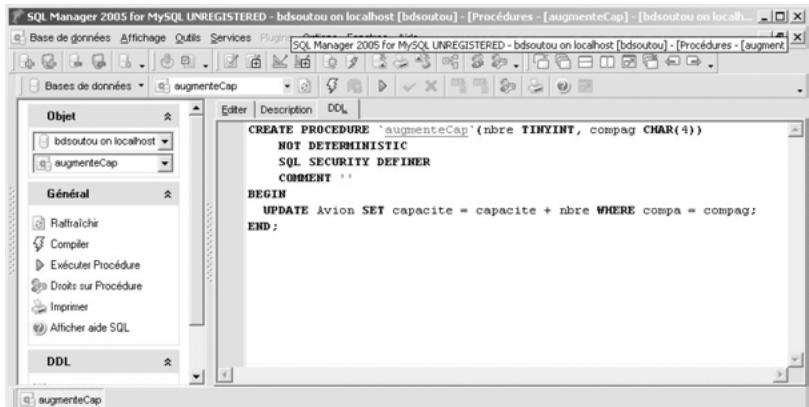


Le résultat peut s'exploiter de manière très efficace (en tant que grille ou sous forme imprimable).

## Procédures cataloguées

Un bon point pour cet outil pour pouvoir gérer les procédures en modification ou création, pour pouvoir les recompiler et affecter des priviléges d'exécution.

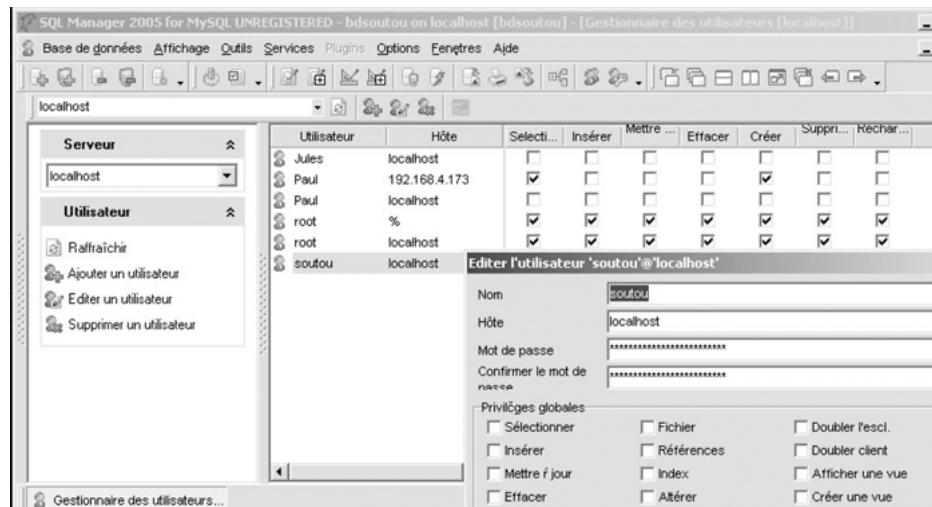
Figure 10-44 Procédure cataloguée



# Gestion des utilisateurs

L'assistant graphique est très clair :

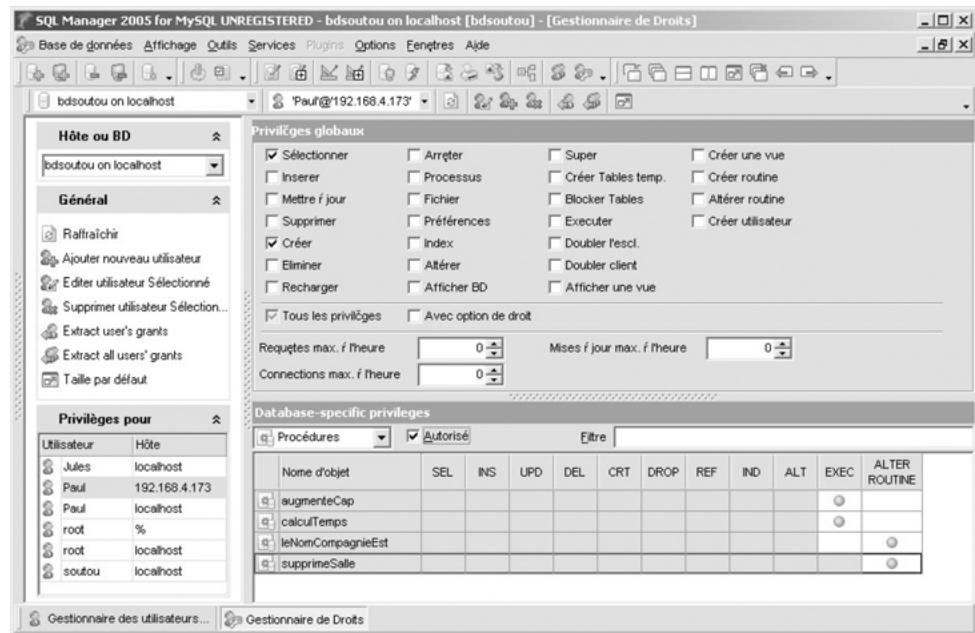
**Figure 10-45 Gestion des utilisateurs**



## Gestion des priviléges

L'onglet **Gestionnaire de droits** permet d'attribuer des priviléges à tous les niveaux. L'écran suivant illustre l'autorisation d'exécution de deux procédures, et l'autorisation de modification de deux autres à l'accès utilisateur Paul, à partir de la machine 192.168.4.173. Cet outil offre également d'autres fonctionnalités comme la gestion des variables système, assistants d'exportation, etc.

Figure 10-46 Priviléges au niveau routine



## Bilan

Les avantages des outils dédiés et professionnels (comme *Navicat*, *TOAD* et *EMS MySQL Manager*) résident dans le fait qu'ils sont fiables et permettent de garder une configuration de travail. De plus ils peuvent être facilement installés sur une autre machine que le serveur MySQL.

Pour les débutants, je leur conseille de commencer à travailler avec les deux outils de MySQL à savoir *MySQL Administrator* et *MySQL Query Browser* qui sont sûrs et simples d'utilisation. *PhpMyAdmin* sera préféré par les connaisseurs et plutôt réservé aux configurations de type application PHP hébergées, car il est le plus répandu. Néanmoins, les nombreuses configurations possibles entre les différentes versions des acteurs concernés (*Apache*, *PHP*, *MySQL*, *phpMyAdmin*) font que des mauvaises surprises peuvent facilement arriver...

# Bibliographie et webographie

## Magazines papiers et en ligne

---

*Programmez !* ([www.programmez.com](http://www.programmez.com))

*Database Journal* ([www.databasejournal.com](http://www.databasejournal.com))

## Livres

---

F. BROUARD, C. SOUTOU, *SQL*, Pearson Education, 2005.

E. DASPET, C.P. DE GEYER, *PHP 5 avancé*, Eyrolles, 2005.

P. DELMAL, *SQL2-SQL3, Applications à Oracle*, De Boeck Université, 2000.

M. KOFLER, *MySQL 5*, Eyrolles, 2005.

C. SOUTOU, *De UML à SQL*, Eyrolles, 2002.

## Sites Web

---

### Eyrolles

Éditeur : [www.editions-eyrolles.com](http://www.editions-eyrolles.com)

Compléments en ligne : sur la fiche de l'ouvrage sur [www.editions-eyrolles.com](http://www.editions-eyrolles.com) ou <http://icare.iut-blagnac.fr/soutou/perso/SQLpourMySQL/Complements.html>

### MySQL

<http://www.mysql.com/>

<http://mysql.developpez.com/>

<http://dev.mysql.com/tech-resources/faq.html>

## PHP

<http://fr.php.net/manual/fr/index.php>

<http://www.nexen.net/docs/>

## SQL et bases de données

<http://sqlpro.developpez.com/>

<http://fadace.developpez.com/>

## Outils

*MySQL Administrator, MySQL Query Browser :*

<http://www.mysql.com/products/tools/>

*phpMyAdmin :*

[http://www.phpmyadmin.net/home\\_page/index.php](http://www.phpmyadmin.net/home_page/index.php)

*Navicat :*

<http://www.navicat.com/>

*TOAD :*

[http://www.toadsoft.com/toadmysql/toad\\_mysql.htm](http://www.toadsoft.com/toadmysql/toad_mysql.htm)

*EMS SQL Manager :*

<http://www.sqlmanager.net/>

## Symboles

`$_POST` 344

`%` 154

`&` 100

`<<` 100

`>>` 100

`^` 100

`|` 100

`~` 100

### A

`ABS` 99

`absolute` 304

`Access` 294

`ACOS` 99

`ACTION_CONDITION` 273

`ACTION_ORIENTATION` 273

`ACTION_REFERENCE_NEW_ROW` 273

`ACTION_REFERENCE_NEW_TABLE` 273

`ACTION_REFERENCE_OLD_ROW` 273

`ACTION_REFERENCE_OLD_TABLE` 273

`ACTION_STATEMENT` 273

`ACTION_TIMING` 272

`ADD CONSTRAINT` 72

`ADD INDEX` 72

`ADDDATE` 101

`ADDTIME` 101

`AFTER` 265

`afterLast` 304

`ALGORITHM=MERGE` 176

`ALGORITHM=TEMPTABLE` 176

`ALGORITHM=UNDEFINED` 176

`alias`

  colonne 87

  en-tête 89

  table 87

`vue` 178

`WHERE` 95

`ALL` 134

`ALL PRIVILEGES` 166

`alpha` 4

`ALTER`

`COLUMN` 70

`DATABASE` 158

`FUNCTION` 243

`PROCEDURE` 243

`ROUTINE` 234

`VIEW` 188

`ALTER TABLE`

`ADD` 68

`ADD CONSTRAINT` 72

`ADD INDEX` 72

`ALTER COLUMN` 70

`CHANGE` 69

`DISABLE KEYS` 75

`DROP` 70

`DROP FOREIGN KEY` 74

`DROP PRIMARY KEY` 74

`ENABLE KEYS` 77

`MODIFY` 69

`Alter_priv` 161

`Alter_routine_priv` 162

`AND` 93

`ANY` 134

`Apache` 327

`API` 5

`AS SELECT` 90

`ASC` 89

`ASCII` 96

`association` 24

`ATAN` 99

`AUTO_INCREMENT` 44, 196

`autojointure` 125

`AVG` 110

`AVG_ROW_LENGTH` 196

## B

batch 13  
BEFORE 265  
beforeFirst 304  
begin 210  
beta 4  
BETWEEN 93  
BIGINT 27  
BIN 100  
BINARY 27  
BINARY() 21, 106  
BIT 27, 40  
BIT\_LENGTH 101  
BLOB 29  
block label 214  
BOOL 27  
BOOLEAN 27

## C

CallableStatement 318  
cancelRowUpdates 308  
CASCADE 34  
CASCADED 176  
CASE 218  
casse 20, 211  
CAST 106  
CATALOG\_NAME 195  
CEIL 99  
CHAR 26  
CHAR() 96  
CHARACTER SET 156  
CHARACTER\_OCTET\_LENGTH 197  
CHARACTER\_SET\_NAME 198  
CHECK 24, 273  
CHECK\_OPTION 193  
Class.forName 295  
CLASSPATH 292  
clé  
    candidate 3  
    étrangère 3  
     primaire 3  
client-serveur 209  
CLOSE 245

COLLATE 156  
COLLATION\_NAME 198  
colonne 2  
COLUMN\_COMMENT 198  
COLUMN\_DEFAULT 197  
COLUMN\_KEY 197  
COLUMN\_NAME 197, 203  
Column\_name 168  
Column\_priv 159, 168  
COLUMN\_PRIVILEGES 194, 203  
COLUMNS 194, 196  
COMMENT 20, 235  
commentaire 212  
    MySQL 21  
COMMIT 228  
comparaisons 106  
CONCAT 89, 96  
concaténation 89  
Connection 295  
CONSTRAINT\_NAME 199  
CONSTRAINT\_SCHEMA 198  
CONSTRAINT\_TYPE 199  
CONTAINS SQL 235  
CONTINUE 249  
contrainte 23  
    CHECK 24  
    FOREIGN KEY 24  
    in-line 23  
    out-of-line 23  
    PRIMARY KEY 23  
    référentielle 56  
    UNIQUE 23  
conventions 24  
conversions 105  
CONVERT 106  
COS 99  
COT 99  
COUNT 110  
CREATE 19  
    DATABASE 156  
    FUNCTION 235  
    INDEX 32  
    ROUTINE 234  
    SCHEMA 156

TABLE 19

TRIGGER 263

USER 153

VIEW 176

Create\_priv 161

Create\_routine\_priv 162

CREATE\_TIME 195

Create\_tmp\_table\_priv 163

Create\_user\_priv 161

Create\_view\_priv 161

CREATED 201, 273

createStatement 295

CROSS JOIN 141

CURDATE 101

CURRENT\_DATE 44, 101

CURRENT\_TIME 44

CURRENT\_TIMESTAMP 44, 101

CURRENT\_USER() 187

curseur 244

CURSOR 245

CURTIME 188

## D

data dictionary 190

Data Source Name 294

DATA\_LENGTH 196

DATA\_TYPE 197

database 6, 13, 155

DatabaseMetaData 314

DATE 28, 42

DATE() 101

DATE\_ADD 51, 101

DATE\_FORMAT 53, 101, 188

DATE\_SUB 101

DATEDIFF 51, 101

DATETIME 28, 42

DAY 101

DAY\_MINUTE 51

DAYNAME 101

DAYOFMONTH 101

DAYOFYEAR 101

Db 167

DBA 152

DEALLOCATE 279

DEC 28

DECIMAL 27

DECLARE 213

    CONDITION 260

déclencheur 262

DEFAULT 20, 38, 213

DEFAULT() 108

DEFAULT\_CHARACTER\_SET\_NAME 195

DEFAULT\_COLLATION\_NAME 195

DEFINER 201

definer 201

DEGREES 99

DELAYED 37

DELETE 54

Delete\_priv 160

deleteRow 308

delimiter 15, 20

DESC 89

DESCRIBE 29

DETERMINISTIC 235

dictionnaire des données 190

DISABLE KEYS 75

DISTINCT 87

DISTINCTROW 87

division 142

DO 279

DOUBLE 27

DOUBLE PRECISION 28

DriverManager 293

DROP 70

    FOREIGN KEY 74

    PRIMARY KEY 74

    TABLE 33

    TRIGGER 278

    USER 155

    VIEW 189

Drop\_priv 161

## E

ELSEIF 218

ENABLE KEYS 77

END 210

ENGINE 196  
ENUM 29, 40, 107  
equals 309  
équijointure 123  
ERROR  
    1045 154  
    1046 250  
    1048 39  
    1054 95  
    1062 39, 49, 285  
    1172 223, 252  
    1263 49  
    1265 40, 41  
    1288 179  
    1303 243  
    1326 245  
    1363 270  
    1369 181, 187  
    1394 182  
    1395 183  
    1422 275  
    1424 242  
    1442 277  
    1451 55, 61  
    1452 39, 49, 59, 285  
étiquette 214  
EVENT\_MANIPULATION 272  
EVENT\_OBJECT\_CATALOG 273  
EVENT\_OBJECT\_SCHEMA 272  
EVENT\_OBJECT\_TABLE 272  
exception 248  
    JDBC 322  
EXECUTE 239, 279  
execute 297, 316, 319  
Execute\_priv 162  
executeQuery 297, 319  
executeUpdate 297, 316, 319  
EXISTS 139  
EXIT 249  
exit 15  
EXP 100  
expression 88  
EXTRACT 53, 102

**F**  
FALSE 218  
FETCH 245  
FIELD 96  
FIELDS  
    ENCLOSED BY 63, 146  
    ESCAPED BY 63, 146  
    TERMINATED BY 63, 146  
FILE 146  
File\_priv 163  
first 304  
FIXED 28  
FLOAT 27  
FLOOR 100  
FLUSH PRIVILEGES 154  
fonction cataloguée 233  
FOR EACH ROW 265  
FOR UPDATE 247  
FOREIGN KEY 56  
FORMAT 108  
FROM 85  
FROM\_DAYS 53, 102  
FROM\_UNIXTIME 102  
FULL OUTER JOIN 131

**G**  
gamma 4  
GET\_FORMAT 53  
getClass 300  
getColumnCount 313  
getColumnName 313  
getColumns 314  
getColumnType 313  
getColumnName 313  
getConcurrency 308  
getConnection 297  
getDatabaseProductName 314  
getDatabaseProductVersion 314  
getErrorCode 322  
getFetchDirection 304  
getGeneratedKeys 312  
getMessage 322  
getMetaData 302

getName 300  
getNextException 322  
getObject 300  
getPrecision 313  
getResultSetConcurrency 308  
getResultSetType 308  
getSavepointId 321  
getSavepointName 321  
getScale 313  
getSchemaName 313  
getSQLState 322  
getTableName 313  
getTables 315  
getter methods 297  
getType 308  
getUpdateCount 297  
getUserName 315  
GOTO 222  
GRANT 164  
    OPTION 164, 165  
Grant\_priv 161  
GRANTEE 202  
Grantor 168, 169  
GREATEST 108  
GROUP BY 109  
GROUP\_CONCAT 110

## H

HANDLER 249  
handler 248  
HAVING 109  
help 13  
HEX 101  
host 7, 13  
HOUR 102  
html 13, 145

## I

identificateur 212  
IDENTIFIED BY 153  
IF 217  
    EXISTS 34, 158, 189  
    NOT EXISTS 19, 156

IFNULL 109  
IGNORE 47, 54, 63  
IN 94, 134  
index 30  
    B-tree 32  
    FULLTEXT 32  
    SPATIAL 32  
    UNIQUE 32  
Index\_priv 161  
inéquijointure 127  
INFORMATION\_SCHEMA 190  
INNER JOIN 124  
InnoDB 20  
INOUT 237  
INSERT 37  
INSERT() 96  
Insert\_priv 160  
insertRow 308  
instead of 179  
INSTR 96  
INT 28  
INTEGER 27  
intégrité référentielle 56  
INTO OUTFILE 146  
invoker 201  
IS NULL 94, 214  
IS\_GRANTABLE 202  
IS\_NULLABLE 197

isAfterLast 304  
isBeforeFirst 304  
isFirst 304  
isLast 304  
isNullable 313  
ITERATE 221

## J

JCreator 293  
JDBC 289  
JOIN 124  
jointure 121  
    équi join 123  
    externe 128  
    inner join 123

mixte 136  
naturelle 140  
outer join 128  
procédurale 132  
relationnelle 122  
self join 125  
SQL2 122

## K

key preserved 184  
KEY\_COLUMN\_USAGE 194, 199

## L

LANGUAGE SQL 234  
last 304  
LAST\_ALTERED 201  
LAST\_DAY 102  
LAST\_INSERT\_ID() 44  
LEAST 109  
LEAVE 221  
LEFT 97  
LENGTH 97  
LIKE 94  
LIMIT 48, 54, 90  
LINES 63, 146  
LMD 37  
LN 100  
LOAD DATA INFILE 62  
LOB (Large Object Binary) 2  
LOCAL 176  
localhost 154  
LOCALTIME 102  
LOCALTIMESTAMP 102  
LOCATE 97  
Lock\_tables\_priv 163  
LOG 100  
LONGBLOB 29  
LONGTEXT 27  
LOOP 221  
LOW\_PRIORITY 37, 47, 54  
LOWER 97  
lower\_case\_table\_names 21

LPAD 97  
LTRIM 99

## M

MAKEDATE 102  
MAKETIME 102  
MAX 110  
max\_connections 162  
MAX\_CONNECTIONS\_PER\_HOUR 165  
MAX\_QUERIES\_PER\_HOUR 165  
max\_questions 162  
max\_updates 162  
MAX\_UPDATES\_PER\_HOUR 165  
MAX\_USER\_CONNECTIONS 165  
max\_user\_connections 162  
MEDIUMBLOB 29  
MEDIUMMINT 27  
MEDIUMTEXT 27  
MEMORY 20  
metadata 190  
MICROSECOND 102  
MIN 110  
MINUTE 102  
MOD 100  
MODIFIES SQL DATA 235  
MODIFY 69  
MONTH 102  
MONTHNAME 102  
moveToCurrentRow 308  
moveToInsertRow 308  
mysqli\_prepare 330  
mutating tables 277  
my.ini 15, 21, 157  
MyISAM 20  
MySQL  
    sous-programme 233  
mysql 10  
MySQL AB 3  
MySQL Administrator 349  
MySQL Manager 379  
MySQL Query Browser 359  
mysql.columns\_priv 168  
mysql.db 159, 167

mysql.host 173  
mysql.procs\_priv 168  
mysql.tables\_priv 168  
mysql.user 153, 159  
mysql\_conf.h 343  
mysqli 327  
MYSQLI\_ASSOC 332  
MYSQLI\_BOTH 332  
mysqli\_change\_user 330  
mysqli\_close 330  
mysqli\_commit 331  
mysqli\_connect 330  
mysqli\_errno 338  
mysqli\_error 338  
mysqli\_fetch\_array 332, 333  
mysqli\_fetch\_assoc 333, 334  
mysqli\_fetch\_field 342  
mysqli\_fetch\_object 333  
mysqli\_fetch\_row 333  
mysqli\_free\_result 334  
mysqli\_insert\_id 337  
mysqli\_multi\_query 339  
MYSQLI\_NUM 332  
mysqli\_num\_fields 333, 341  
mysqli\_num\_rows 334  
mysqli\_query 332  
mysqli\_rollback 331  
mysqli\_select\_db 330  
mysqli\_stmt\_bind\_param 335  
mysqli\_stmt\_bind\_result 335  
mysqli\_stmt\_close 336  
mysqli\_stmt\_errno 338  
mysqli\_stmt\_error 338  
mysqli\_stmt\_execute 330  
mysqli\_stmt\_fetch 330, 336  
mysqli\_stmt\_free\_result 333  
mysqli\_stmt\_result\_metadata 343

## N

NATURAL JOIN 140  
Navicat 374  
NEW 267  
next 302

NO SQL 235  
NOT 92  
DETERMINISTIC 235  
EXISTS 139  
FOUND 249  
IN 134  
NULL 20, 23  
NOW 28, 101, 102  
NULL 20, 38  
NULLIF 109  
NUMERIC 28  
NUMERIC\_PRECISION 198  
NUMERIC\_SCALE 198

**I**

OCT 101  
OCTET\_LENGTH 101  
ODBC 290, 294  
OLD 265  
ON DELETE  
  CASCADE 60  
  SET NULL 60  
ON UPDATE  
  CASCADE 60  
  SET NULL 60  
OPEN 245  
OR 93  
  REPLACE 176  
ORDER BY 48, 55, 89  
ORDINAL\_POSITION 197, 199  
OUTER JOIN 129

## P

paquetage 244  
password 13  
PASSWORD() 154  
PERIOD\_DIFF 102  
PHP 327  
phpMyAdmin 362  
PI() 99  
placeholder 279, 336  
POSITION\_IN\_UNIQUE\_CONSTRAINT 199

POW 100  
PREPARE 279  
prepareCall 295, 318  
prepared statement 278  
prepareStatement 295, 316  
previous 304  
privilege 158  
PRIVILEGE\_TYPE 202  
Proc\_priv 169  
procédure cataloguée 233  
Process\_priv 163  
procs\_priv 160, 168  
production 4  
produit cartésien 119, 141  
prompt 13, 15

## Q

QUICK 55  
quit 15

## R

RADIANS 100  
RAISE 250  
RAND 100  
READS SQL DATA 235  
REAL 28  
récursivité 241  
REFERENCED\_COLUMN\_NAME 200  
REFERENCED\_TABLE\_NAME 200  
REFERENCED\_TABLE\_SCHEMA 200  
References\_priv 163  
registerOutParameter 319  
relative 304  
releaseSavepoint 321  
RENAME 67  
    TO 67  
    USER 154  
REPEAT 220  
Repl\_client\_priv 163  
Repl\_slave\_priv 163  
REPLACE 54, 62, 97  
requête 83

RESIGNAL 250  
RESTRICT 34  
ResultSet 301  
ResultSetMetaData 313  
RETURNS 235  
REVERSE 98  
REVOKE 170  
    ALL PRIVILEGES 171  
ROLLBACK 228  
    TO SAVEPOINT 230  
root 153  
ROUND 100  
ROUTINE\_COMMENT 201  
ROUTINE\_DEFINITION 201  
ROUTINE\_NAME 201  
Routine\_name 169  
ROUTINE\_SCHEMA 201  
ROUTINE\_TYPE 201  
Routine\_type 169  
ROUTINES 200  
row 2  
row trigger 265  
RPAD 98  
RTRIM 98, 99

## S

Savepoint 321  
savepoint  
    JDBC 321  
    MYSQL 230  
schéma 6  
SCHEMA\_PRIVILEGES 194, 202  
SCHEMATA 194  
SEC\_TO\_TIME 52, 53, 102  
SECOND 102  
SECURITY\_TYPE 201  
SELECT 84  
    fonctions 95  
SELECT... INTO 223  
Select\_priv 160  
SEQUEL 1  
séquence 44, 196  
    JDBC 311

SERIAL	28	SQL2	1
SET	29, 47, 107	SQL3	1
SET AUTOCOMMIT	228	SQLEXCEPTION	250
SET FOREIGN_KEY_CHECKS	75, 77	SQLException	322
setAutoCommit	295	SQLSTATE	249
setFetchDirection	304	SQLWARNING	249
setMaxRows	297	SQRT	100
setNull	316	Statement	297
setSavepoint	321	STATISTICS	194
setter methods	296	STDDEV	110
SHOW	190	stored procedures	233
	COLUMNS 204	stored routines	233
	CREATE DATABASE 204	STR_TO_DATE	53, 102
	CREATE TABLE 204	SUBDATE	102
	CREATE VIEW 189	SUBSTR	98
	DATABASES 204	SUBTIME	102
	ENGINES 204	SUM	110
	ERRORS 204, 239	SUPER	263, 278
	GRANTS 204	Super_priv	163
	GRANTS FOR 165	supportsSavepoints	315
	INDEX 204	supportsTransactions	315
	PRIVILEGES 204	SYSDATE	28, 88, 102
	TABLE STATUS 204		
	TABLES 204		
	TRIGGERS 204		
Show_db_priv	161	T	
Show_view_priv	161	table	2, 19
Shutdown_priv	163	dominante	128
SIGN	100	fils	56
silent	13	key preserved	184
SIN	100	père	56
SINH	100	subordonnée	128
skip-column-names	13	TEMPORARY	19
SLEEP()	232	TABLE_COLLATION	196
SMALLINT	27	TABLE_COMMENT	196
SOUNDEX	98	TABLE_CONSTRAINTS	194, 198
source	15	TABLE_NAME	193
sous-interrogation	133	Table_name	168
	dans le FROM	Table_priv	168
	137	TABLE_PRIVILEGES	194, 202
	synchronisée	TABLE_ROWS	196
sous-programme	210	TABLE_SCHEMA	193, 202
SQL dynamique	278	TABLE_TYPE	195
SQL SECURITY	235	TABLES	195
SQL_PATH	195	tables_priv	159, 168

TAN 100  
tee 13, 15  
TEMPORARY 19  
TEXT 27  
TIME 28, 42, 52, 53, 102  
TIME\_FORMAT 53  
TIME\_TO\_SEC 53, 103  
TIMEDIFF 103  
TIMESTAMP 28, 103  
TIMESTAMPADD 103  
TIMESTAMPDIFF 103  
TINYBLOB 29  
TINYINT 27  
TINYTEXT 27  
TO\_DAYS 103  
TOAD 368  
transaction 227  
TRIGGER\_CATALOG 273  
TRIGGER\_NAME 272  
TRIGGER\_SCHEMA 272  
TRIGGERS 272  
TRIM 99  
TRUE 218  
TRUNCATE 55, 100

## U

UNDO 249  
UNHEX 101  
UNION 116  
    ALL 116  
UNIX\_TIMESTAMP 53, 103  
UNSIGNED 28  
UNTIL 220  
UPDATE 47, 54  
Update\_priv 160  
updater methods 297  
updateRow 308  
UPPER 99  
USAGE 164  
USE 157  
use 15  
user 6, 13, 152  
    variables 215

USER\_PRIVILEGES 194, 201  
user-defined function 277  
USING 141  
UTC\_DATE 103  
UTC\_TIME 44, 103  
UTC\_TIMESTAMP 103

## V

VALUES 54  
VARBINARY 27  
VARCHAR 26  
variable  
    session 215  
VARIANCE 110  
verbose 13  
version 13  
VERSION() 12  
vertical 13  
VIEW\_DEFINITION 191  
VIEWS 194  
vue 175  
vue monutable 177

## W

wasNull 319  
WEEKDAY 103  
WEEKOFYEAR 103  
WHERE 48, 55  
WHILE 220  
WITH CHECK OPTION 176

## X

xml 13, 145

## Y

YEAR 28

## Z

ZEROFILL 28