

Base de données SQL

FLORENT FERCOQ

Sommaire

Introduction et prise en main

Le modèle relationnel, conception et création de tables

Pratique du SQL avec MySQL

Tables transactionnelles InnoDB

SQL procédural

Connexions, droits d'accès, sécurité

Introduction à l'administration

Le schéma d'une base de données

Exemple développé

Encyclopédie des langues et villes pays du monde :

CITY (ID, Name, #CountryCode, District, Population)

COUNTRY (Code, Name, Continent, Region,
SurfaceArea, IndepYear, Population, ...)

COUNTRYLANGUAGE(#CountryCode, Language,
IsOfficial, Percentage)

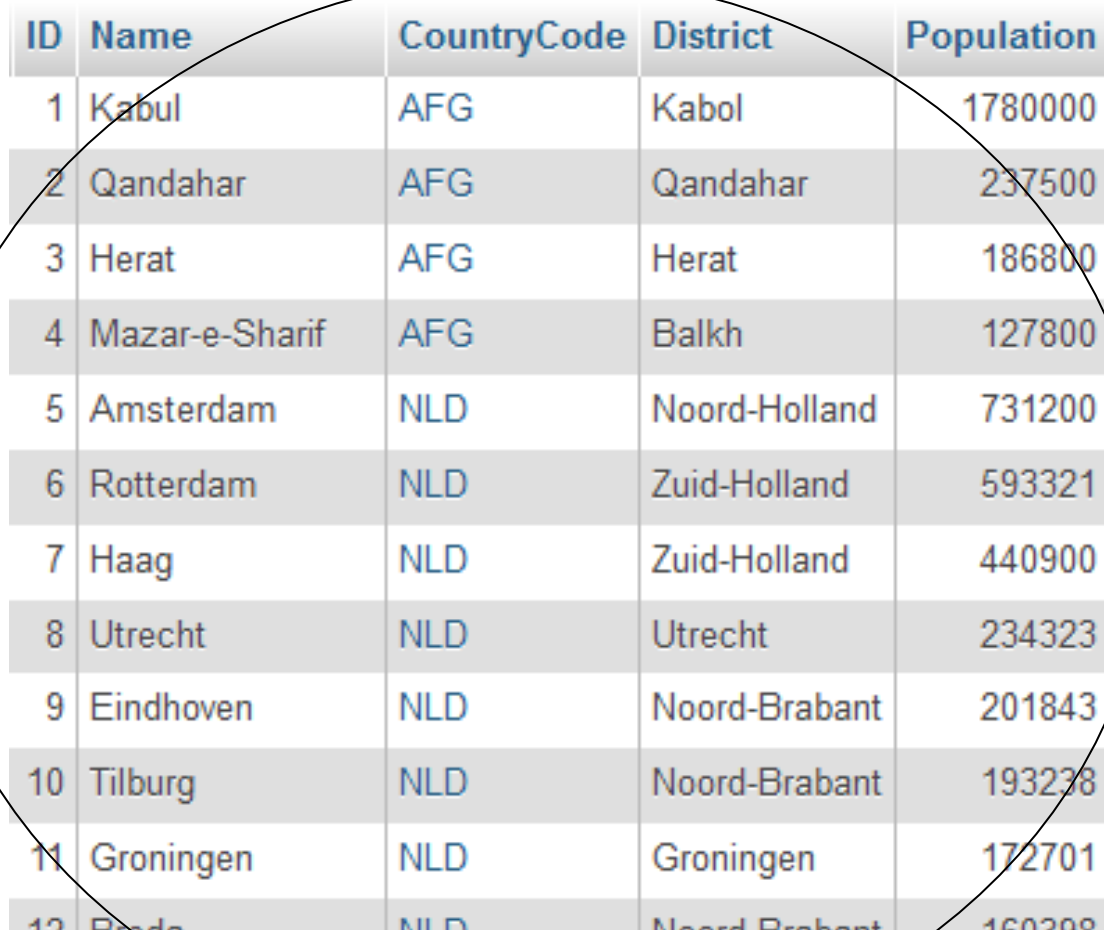
Les bases de données : vocabulaire

Une TABLE est un ensemble de données organisées sous forme d'un tableau où les colonnes correspondent à des catégories d'information et les lignes à des enregistrements

Les CHAMPS correspondent aux attributs (= *colonne*)

Les ENREGISTREMENTS correspondent aux données saisies (= *ligne*)

Les bases de données : les tables

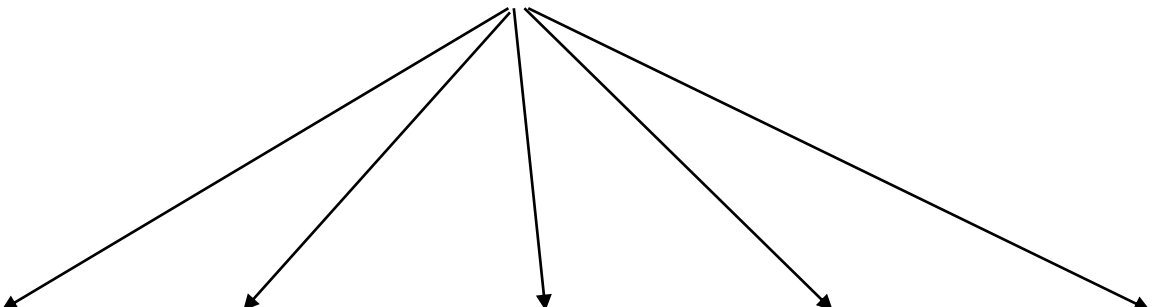


ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
7	Haag	NLD	Zuid-Holland	440900
8	Utrecht	NLD	Utrecht	234323
9	Eindhoven	NLD	Noord-Brabant	201843
10	Tilburg	NLD	Noord-Brabant	193238
11	Groningen	NLD	Groningen	172701
12	Eindhoven	NLD	Noord-Brabant	160200

Table CITY

Les bases de données : les champs

Champs de la table CITY



ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	M...	AFG	D...	107000

Les bases de données : les enregistrements

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
7	Haag	NLD	Zuid-Holland	440900
8	Utrecht	NLD	Utrecht	234323
9	Eindhoven	NLD	Noord-Brabant	201843
10	Tilburg	NLD	Noord-Brabant	193238
11	Groningen	NLD	Groningen	172701
12	Brda	NLD	Noord-Brabant	160200

Un enregistrement de la table CITY

Un marché riche

Différents systèmes de gestion de bases de données relationnelles :

➤ Des payants :

- Excel
- Access
- Oracle
- SQL Server
- OpenOffice
- LibreOffice

➤ Des « gratuits » :

- MySQL
- PostgreSQL
- MariaDB
- SQLite

Le langage de requête SQL

SQL : Structured Query Language – Langage de Requête Structurée

Trois types d'opérations :

Projection : projette les champs sélectionnés

Restriction : projette les enregistrements demandés

Tri : ordonne les enregistrements demandés selon un critère

Le langage de requête SQL : la projection

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
7	Haag	NLD	Zuid-Holland	440900
8	Utrecht	NLD	Utrecht	234323
9	Eindhoven	NLD	Noord-Brabant	201843
10	Tilburg	NLD	Noord-Brabant	193238
11	Groningen	NLD	Groningen	172701
12	Breda	NLD	Noord-Brabant	160209

Projection :

- Name
- Population

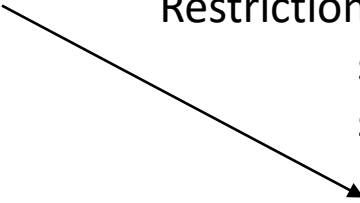
Name	Population
Kabul	1780000
Qandahar	237500
Herat	186800
Mazar-e-Sharif	127800
Amsterdam	731200
Rotterdam	593321
Haag	440900
Utrecht	234323
Eindhoven	201843
Tilburg	193238

Le langage de requête SQL : la restriction

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
7	Haag	NLD	Zuid-Holland	440900
8	Utrecht	NLD	Utrecht	234323
9	Eindhoven	NLD	Noord-Brabant	201843
10	Tilburg	NLD	Noord-Brabant	193238
11	Groningen	NLD	Groningen	172701
12	Brada	NLD	Noord-Brabant	160200

Restriction :

seulement les villes
suisses



Name	CountryCode	District	Population
Zürich	CHE	Zürich	336800
Geneve	CHE	Geneve	173500
Basel	CHE	Basel-Stadt	166700
Bern	CHE	Bern	122700
Lausanne	CHE	Vaud	114500

Le langage de requête SQL : le tri

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
7	Haag	NLD	Zuid-Holland	440900
8	Utrecht	NLD	Utrecht	234323
9	Eindhoven	NLD	Noord-Brabant	201843
10	Tilburg	NLD	Noord-Brabant	193238
11	Groningen	NLD	Groningen	172701
12	Brda	NLD	Noord-Brabant	160208

Tri par
Name

Name ▲	CountryCode	District	Population
A Coruña (La Coruña)	ESP	Galicia	243402
Aachen	DEU	Nordrhein-Westfalen	243825
Aalborg	DNK	Nordjylland	161161
Aba	NGA	Imo & Abia	298900
Abadan	IRN	Khuzestan	206073
Abaetetuba	BRA	Pará	111258
Abakan	RUS	Hakassia	169200
Abbotsford	CAN	British Colombia	105403
Abeokuta	NGA	Ogun	427400
Aberdeen	GBR	Scotland	213070
Abha	SAU	Asir	112300
Abidjan	CIV	Abidjan	2500000
Abiko	JPN	Chiba	126670
Abilene	USA	Texas	115930

Le langage de requête SQL : la jointure

Liste des pays où le français est parlé

- Projection du nom des villes
- Restriction sur la langue française

Table COUNTRYLANGUAGE

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
ABW	English	F	9.5
ABW	Papiamentu	F	76.7
ABW	Spanish	F	7.4
AFG	Balochi	F	0.9
AFG	Dari	T	32.1
AFG	Pashto	T	52.4
AFG	Turkmenian	F	1.9
AFG	Uzbek	F	8.8
AGO	Ambo	F	2.4
AGO	Chokwe	F	4.2
AGO	Kongo	F	13.2
AGO	Luchazi	F	2.4
AGO	Lumbe-nganguela	F	5.4

Table COUNTRY

Code	Name	Continent	Region	SurfaceArea	IndepYear	Popul
ABW	Aruba	North America	Caribbean	193.00	NULL	10
AFG	Afghanistan	Asia	Southern and Central Asia	652090.00	1919	2272
AGO	Angola	Africa	Central Africa	1246700.00	1975	1287
AIA	Anguilla	North America	Caribbean	96.00	NULL	
ALB	Albania	Europe	Southern Europe	28748.00	1912	340
AND	Andorra	Europe	Southern Europe	468.00	1278	7
ANT	Netherlands Antilles	North America	Caribbean	800.00	NULL	21
ARE	United Arab Emirates	Asia	Middle East	83600.00	1971	244
ARG	Argentina	South	South	2780400.00	1816	3703

Nom	Type
<u>CountryCode</u>	char(3)
<u>Language</u>	char(30)
IsOfficial	enum('T', 'F')
Percentage	float(4,1)



Nom	Type
<u>Code</u>	char(3)
Name	char(52)
Continent	enum('Asia', 'Europe', 'North America', 'Africa')
Region	char(26)
SurfaceArea	float(10,2)
IndepYear	smallint(6)
Population	int(11)
LifeExpectancy	float(3,1)
GNP	float(10,2)
GNPOld	float(10,2)
LocalName	char(45)
GovernmentForm	char(45)
HeadOfState	char(60)
Capital	int(11)
Code2	char(2)

- Le lien se fait grâce au code du pays

➤ Jointure : lien entre les tables

Name	Language
Andorra	French
Burundi	French
Belgium	French
Canada	French
Switzerland	French
France	French
Guadeloupe	French
Haiti	French
Italy	French
Lebanon	French
Luxembourg	French
Monaco	French
Madagascar	French
Martinique	French
Mauritius	French
Mayotte	French
New Caledonia	French
French Polynesia	French

MySQL - Introduction

MySQL

MySQL est un système de gestion de bases de données relationnelles (SGBDR) open-source

Historiquement, MySQL a été créé par une société suédoise, MySQL AB

- Développement commencé en 1994
- Première version en mai 1995

Quelques dates clés dans la vie de MySQL

- 2008 : Achat de MySQL AB par Sun Microsystems
- Janvier 2010 : Achat de Sun Microsystems par Oracle

Il existe deux versions de MySQL

- La version open-source *MySQL Community Server*
- La version propriétaire *MySQL Server* qui partage la même base que la version open-source avec un support et des plugins en plus
 - Disponible en 3 éditions : Standard, Enterprise, Cluster Carrier Grade

Une version très légère et dénommée *MySQL Classic* existe et est destinée aux fabricants de systèmes embarqués

MySQL

La version 5.6 publiée en février 2013 a été riche en nouvelles fonctionnalités avec, en particulier :

- Procédures stockées
- Curseurs
- Triggers

La dernière version stable de MySQL est la version 8.2, datée d'octobre 2023

En raison du rachat de MySQL par Oracle, un *fork* a été créé par la communauté sous le nom de MariaDB

- La compatibilité entre MySQL et MariaDB est importante
- Plus de détails sur <https://mariadb.com/kb/en/mariadb/mariadb-vs-mysql-compatibility/>

La documentation

La documentation MySQL est très riche et claire

<https://dev.mysql.com/doc/refman/5.7/en/>

<https://dev.mysql.com/doc/refman/8.0/en/>

<https://dev.mysql.com/doc/refman/8.2/en/>

Ne pas hésiter à s'y attarder pour apprendre ou découvrir de nouvelles choses

L'installation

Installation

L'installation de MySQL est différente selon le système d'exploitation utilisé

Les binaires peuvent être téléchargés sur le site de MySQL

<https://dev.mysql.com/downloads/mysql/>

Installation - Windows

Sous Windows, le mieux est de passer par l'installateur mis à disposition

- Il installe le produit et ajoute un service Windows pour la base de données

Pour installer le serveur MySQL

- <https://dev.mysql.com/downloads/mysql/>
 - Prérequis : Visual Studio 2019 x64 Redistribuable :
https://aka.ms/vs/17/release/vc_redist.x64.exe
1. Lancer l'installateur
 2. Accepter la licence
 3. Choisir l'installation *Typical*
 4. Lancer l'installation
 5. Configurer avec *MySQL Configurator*

Pour installer l'outil MySQL Workbench

- <https://dev.mysql.com/downloads/workbench/>

Installation - Windows

Après l'installation, un assistant permet de configurer MySQL

- Standalone MySQL Server
- Choisir le type de machine (pour l'optimisation des ressources)
 - Development Machine
- Configurer l'accès au réseau
- Donner un mot de passe pour l'utilisateur root (administrateur général)
- Configurer le service Windows
- Lancer la configuration

Installation - Mac

Sous Mac OS, l'installation s'effectue via l'archive DMG disponible sur le site

Documentation d'installation :

- <https://dev.mysql.com/doc/refman/8.2/en/macos-installation.html>

Double-cliquer sur cette archive pour la monter et faire apparaitre le fichier PKG

Double-cliquer sur le fichier PKG pour procéder à l'installation en suivant les instructions de l'assistant

A la fin de l'installation, un mot de passe administrateur (root) sera présenté

- Ne pas l'oublier et le changer dès la première connexion

MySQL sera installé dans `/usr/local/mysql`

Pour lancer le serveur

- `sudo ./bin/mysqld_safe` puis CTRL-Z puis `bg`

Installation - Linux

Pour installer MySQL sous Linux, le mieux est d'utiliser les dépôts officiels

L'installation varie selon la distribution utilisée

- S'orienter vers la documentation officielle
- <https://dev.mysql.com/doc/refman/8.2/en/linux-installation.html>

Tester l'installation

mysql est l'outil qui est utilisé de base pour se connecter à la base de données via un terminal

- On lui préférera généralement l'utilisation de *MySQL Workbench*
- Ne pas oublier d'ajouter, sous Windows, `C:\Program Files\MySQL\MySQL Server 8.2\bin` dans le PATH afin que Windows retrouve la commande `mysql`

`mysql -u root -p` puis saisir le mot de passe root configuré lors de l'installation

Une connexion à la base est :

`exit` pour quitter

```
C:\Users\florent>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.19-log MySQL Community Server (GPL)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> exit
Bye
```

Les outils

Les outils

L'outil mysql permet de faire des requêtes sur le serveur

mysql propose de nombreuses options

```
mysql --help
```

Pour se connecter à une base

```
mysql -u root -p
```

-u pour spécifier l'utilisateur

-p pour demander le mot de passe

Les outils

On peut utiliser -e pour spécifier une commande à exécuter

```
mysql -u root -p -e "source fichier.sql"
```

D'autres options sont utilisables

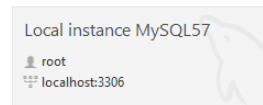
- -h pour spécifier la machine hôte
- -P pour spécifier le port d'écoute de la machine hôte
- -D pour spécifier le nom de la base à utiliser

Les outils

MySQL Workbench est un outil graphique complet permettant de concevoir et de manipuler une base MySQL

Choisir une connexion MySQL déjà configurée ou en créer une nouvelle

MySQL Connections ⊕ ⓘ



Saisir le mot de passe de l'utilisateur concerné si demandé

Une interface complète s'ouvre permettant de manipuler la base de données

Plus convivial que le mode en ligne de commande mais cet outil n'est pas toujours disponible

Les bases de données

Base de données

Une base de données, en informatique, est un ensemble de données gérées par un système informatique

Ces informations sont structurées et organisées afin que l'information voulue soit disponible rapidement, et ce même dans le cas d'un gros volume de données

Les instructions envoyées à la base de données constituent des requêtes

Une base de données est composée :

- D'un système dont le but est de gérer la donnée
- D'un langage permettant d'agir avec cette base de données

Base de données

MySQL est un Système de Gestion de Base de Données (SGBD) qui fonctionne sur le principe du client / serveur

Il s'agit même d'un SGBDR (Système de Gestion de Base de Données Relationnel) car il utilise des relations

Un client (programme) quel qu'il soit pourra se connecter au serveur MySQL afin de récupérer de l'information ou d'en ajouter

La création d'une base

Pour travailler, il est nécessaire de disposer d'une base de données

Création d'une base MySQL

- `CREATE DATABASE nom_base;`
- **Exemple :** `CREATE DATABASE comptabilite;`

Suppression d'une base

- `DROP DATABASE nom_base;`
- **Exemple :** `DROP DATABASE comptabilite;`

Affichage des bases de données du serveur

- `SHOW DATABASES;`

Utiliser une base

Quand une base existe, il faut indiquer qu'on souhaite l'utiliser dans le cadre de la connexion en cours

- `USE nom_base;`
- Exemple : `USE comptabilite;`

Il est possible de ne pas saisir cette requête en indiquant le nom de la base comme paramètre de la commande mysql

- `mysql -u root -p comptabilite;`

INFORMATION_SCHEMA

Dans MySQL une base nommée `INFORMATION_SCHEMA` est disponible

Cette base contient des informations sur la base de données et ses composantes

On y retrouve entre autres :

- Les tables et leur structure
- Les vues et leur structure
- Les utilisateurs
- Les droits d'accès

Les jeux de caractères

Le jeu de caractère définit l'encodage dans lequel les données sont stockées dans la base

Les paramètres en cours sont accessibles via la commande

```
SHOW VARIABLES LIKE "%character%";
```

La liste des jeux de caractères supportés est accessible par la commande

```
SHOW CHARACTER SET;
```

Les jeux de caractères

La commande `SET NAMES 'jeu_caracteres'` permet d'indiquer, au cours d'une session, dans quel jeu de caractères le client enverra les requêtes et dans quel jeu de caractères le serveur devra répondre

Lors de la création d'une base, il est possible de spécifier le jeu de caractères utilisés

```
CREATE DATABASE nom_base CHARACTER SET 'jeu_caracteres';
```

Exemple :

```
CREATE DATABASE comptabilite CHARACTER SET 'utf8';
```


L'interclassement

L'interclassement définit :

- L'ordre des caractères
- Les caractères équivalents

Il est utile lors de la recherche de données textuelles ou lors du tri des résultats

Pour consulter tous les interclassements disponibles pour un jeu de caractères

```
SHOW COLLATION WHERE Charset = 'utf8';
```

Le suffixe est important :

- `cs` : *case sensitive* - sensible à la casse
- `ci` : *case insensitive* - insensible à la casse

L'interclassement

Pour consulter les réglages en cours dans la base

```
SHOW VARIABLES LIKE "%collation%";
```

MySQL permet de spécifier un interclassement spécifique pour une base ou une table

Le langage SQL

SQL (*Structured Query Language*) est le langage normalisé qui permet de communiquer avec la base de données

- Créé en 1974
- Normalisé en 1986

MySQL utilise le SQL

Quelques règles de bases en SQL :

- Chaque instruction se termine par ;
- Tant qu'un point-virgule n'est pas saisi la requête ne sera pas exécutée
- Les lignes qui commencent par -- (tiret-tiret-espace) sont considérées comme des commentaires
- Les chaînes de caractères sont entourées par des simples cotes '
- Exemple : 'MySQL'
- Pour saisir ' dans une chaîne, on utilise deux fois la simple cote ''
- MySQL propose ses propres règles pour les caractères spéciaux en utilisant le caractère d'échappement \
- Exemple : \' pour ', \n pour un retour à la ligne, \t pour une tabulation, \\ pour un antislash

Les tables

L'information est contenue dans des structures représentées sous la forme de tables

Une table porte un nom et elle est constituée d'un en-tête et d'un corps

L'en-tête est formé par plusieurs attributs (noms des colonnes)

id	nom	prenom	age	telephone
----	-----	--------	-----	-----------

Le corps est constitué des différents enregistrements qu'on appelle lignes ou tuples

id	nom	prenom	age	telephone
1	Foucalt	Patrick	53	0123456789
2	Dubois	Blandine	30	0645678965
3	Malou	Manuel	44	0299999999

Les types

Lors de la création des tables, il convient de spécifier les types des différents attributs

MySQL propose de nombreux types selon l'information qui sera représentée

- Types numériques
- Types alphanumériques
- Types temporels

Certains types sont spécifiques à MySQL, d'autres sont communs avec les SGBD standards

Les types - Numériques

Les nombres entiers

Type	Taille (octets)	Valeur minimale (signé / non-signé)	Valeur maximale (signé / non-signé)
TINYINT	1	-128 / 0	127 / 255
SMALLINT	2	-32768 / 0	32767 / 65535
MEDIUMINT	3	-8388608 / 0	8388607 / 16777215
INT	4	-2147483648 / 0	2147483647 / 4294967295
BIGINT	8	-9223372036854775808 / 0	9223372036854775807 / 18446744073709551615

Pour spécifier un type non-signé, on ajoute UNSIGNED au type

- Exemple : INT UNSIGNED

Pour spécifier le nombre de chiffres minimum à l'affichage, il est possible de spécifier ce nombre entre parenthèses

- Accompagné de ZEROFILL pour ajouter des zéros à gauche
- Exemple : INT(5) ZEROFILL

En cas de dépassement, MySQL prendra la valeur la plus proche supportée

Les types - Numériques

Les nombres décimaux exacts

On trouve deux types sous MySQL, qui sont les mêmes :

- NUMERIC
- DECIMAL

Ces types prennent deux paramètres :

- Le nombre de chiffres significatifs stockés
- Le nombre de chiffres après la virgule

Exemple : `DECIMAL(8, 5)` permet de stocker 8 chiffres maximum au total dont 5 seront après la virgule

En cas de dépassement, MySQL prendra la valeur la plus proche supportée

Les types - Numériques

Les nombres décimaux flottants

On trouve deux types sous MySQL, qui sont les mêmes :

- `FLOAT` (4 octets)
- `DOUBLE` (8 octets)

Il est possible de spécifier des détails sur la précision dans les paramètres

Attention ! Il s'agit de types flottants

- Une valeur exacte n'est pas forcément stockée, ce qui peut poser des problèmes lors des comparaisons ou de l'affichage

Les types - Alphanumériques

Les chaînes de caractères

Deux types :

- VARCHAR
- CHAR

Ils prennent un paramètre :

- Pour CHAR, la longueur maximale entre 0 et 255
- Pour VARCHAR, la taille maximale en octets de 0 à 65535

Lors du stockage

- des espaces sont ajoutés à gauche du CHAR pour venir le compléter et arriver à sa taille
- la taille d'un VARCHAR changera selon le texte stocké

Les types - Alphanumériques

Le texte

Le type `TEXT` et ses dérivés permettent de stocker du texte de grande longueur

Type	Longueur maximale	Occupation stockage
<code>TINYTEXT</code>	2^8 octets	longueur + 1 octet
<code>TEXT</code>	2^{16} octets	longueur + 2 octets
<code>MEDIUMTEXT</code>	2^{24} octets	longueur + 3 octets
<code>LONGTEXT</code>	2^{32} octets	longueur + 4 octets

Les types - Temporels

Le type `DATE` permet de stocker une date

- Représentée au format 'YYYY-MM-DD'
- Date minimale : '1001-01-01'
- Date maximale : '9999-12-31'
- Plusieurs saisies sont acceptées
 - 'YYYY-MM-DD'
 - 'YYMMDD'
 - 'YYYY/MM/DD'
 - N'importe quelle ponctuation peut être utilisée pour séparer les composantes
 - YYMMDD
- Dans le cas d'une saisie d'année à 2 chiffres
 - Entre 0 et 69, la date sera de 2000 à 2069
 - Entre 70 et 99, la date sera de 1970 à 1999

Les types - Temporels

Le type `DATETIME` permet de stocker une date et une heure

- Représentée au format `'YYYY-MM-DD HH:MM:SS'`
- Valeur minimale : `'1001-01-01 00:00:00'`
- Valeur maximale : `'9999-12-31 23:59:59'`
- Plusieurs saisies sont acceptées
 - `'YYYY-MM-DD HH:MM:SS'`
 - `'YYYY/MM/DD HH*MM*SS'`
 - N'importe quelle ponctuation peut être utilisée pour séparer les composantes
 - `YYMMDDHHMMSS`

Les types - Temporels

Le type `TIME` permet de stocker une heure ou une durée

- Permet de stocker un nombre de jours ou une valeur négative
- Valeur minimale : `'-838:59:59'`
- Valeur maximale : `'838:59:59'`
- Plusieurs saisies sont acceptées
 - `'HH:MM:SS'`
 - `'HHH:MM:SS'`
 - `'MM:SS'`
 - `'J HH:MM:SS'`
 - `HHMMSS`

Les types - Temporels

Le type `YEAR` permet de stocker une année sur un octet

- Valeur minimale : 1901
- Valeur maximale : 2155
- Plusieurs saisies sont acceptées
 - `YYYY`
 - `YY` (mêmes règles que pour `DATE`)

Le type `TIMESTAMP` permet de stocker le nombre de secondes écoulées depuis le 01/01/1970

- Stockage sur 4 octets
- Valeur maximale : le 19/01/2038 03:14:07
- En réalité, la valeur est stockée au format numérique `AAAAMMJJHHMMSS`

Les types - Temporels

Dans le cas de la saisie d'une valeur qui dépasse la capacité maximale du type ou d'une saisie incorrecte, une valeur par défaut est stockée à la place

Type	Par défaut
DATE	'0000-00-00'
DATETIME	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000
TIMESTAMP	0000000000000000

NULL / NOT NULL

Pour chaque colonne, on pourra indiquer si on accepte qu'aucune valeur ne soit enregistrée (`NULL`)

Par défaut, `NULL` est autorisé

Si on ne souhaite pas l'accepter pour une colonne, il faudra indiquer que la colonne est `NOT NULL`

Clé primaire

La clé primaire constitue une contrainte d'intégrité gérée par MySQL

Il s'agit d'une contrainte d'unicité, composée d'une ou plusieurs colonnes

La clé primaire permet d'identifier de manière unique une ligne dans la table

Généralement, on définit une colonne comme clé primaire dans une table

Lorsqu'une clé primaire est de type entière, il est possible de laisser le soin à MySQL d'incrémenter automatiquement la valeur de cette clé pour chaque enregistrement grâce au mot-clé `AUTO_INCREMENT`

Les moteurs

MySQL propose plusieurs moteurs qui sont chargés de gérer le fonctionnement des tables

Ces moteurs se traduisent par un stockage différent des fichiers de la table sur le disque

InnoDB

- Moteur par défaut depuis MySQL 5.5.5
- Gestion des clé étrangères
- Gestion des transactions
- Meilleure gestion du stockage pour obtenir de meilleures performances

MyISAM

- Moteur par défaut avant MySQL 5.5.5
- Pas de transactions
- Pas de clés étrangères

Les moteurs

MEMORY

- Moteur qui gère les données en mémoire
- Vulnérable aux crashes
- Utile pour des tables temporaires

MERGE

- Permet de combiner des tables MyISAM de structures identiques et de les considérer comme une seule

D'autres types de moteurs sont proposés par MySQL

- CSV : Stockage au format CSV sur le disque
- ARCHIVE : Stockage sans indexation de gros volumes de données
- BLACKHOLE : Moteur qui accepte de la donnée mais qui ne la stocke pas - Utile par exemple pour faire du filtrage comme serveur intermédiaire
- FEDERATED : Permet d'utiliser une base MySQL distante
- EXAMPLE : Ne fait rien. Sert d'exemple pour développer un nouveau moteur

Créer une table

Pour créer une table, ou une relation, on utilise la commande `CREATE TABLE` en spécifiant les différents attributs de cette dernière

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nom_table (  
    nom_attribut TYPE_ATTRIBUT [OPTIONS]  
    ...  
)
```

Le mot-clé `TEMPORARY` permet de créer une table dont la durée de vie sera égale au temps de connexion au serveur

L'option `IF NOT EXISTS` permet de créer la table uniquement si elle n'existe pas

Créer une table

Exemple :

```
CREATE TABLE client (  
    nom VARCHAR(50),  
    prenom VARCHAR(50),  
    code_postal INTEGER,  
    ville VARCHAR(100)  
);
```

Créer une table

Pour un attribut

- l'expression `PRIMARY KEY` permet de spécifier qu'il s'agit de la clé primaire de la table
- l'expression `AUTO_INCREMENT` permet d'utiliser l'auto-incrémentation pour une colonne
- l'expression `NOT NULL` permet d'indiquer que la valeur `NULL` n'est pas acceptée comme valeur de la colonne
- l'expression `UNIQUE` permet d'indiquer que la valeur doit être unique parmi tous les enregistrements
- l'expression `DEFAULT` permet de spécifier une valeur par défaut

Le moteur de stockage utilisé pour la table peut être indiqué en terminant le `CREATE TABLE` par `ENGINE=nom_moteur`

Créer une table

Exemple :

```
CREATE TABLE client (  
    id INTEGER AUTO_INCREMENT PRIMARY KEY,  
    nom VARCHAR(50) NOT NULL,  
    prenom VARCHAR(50) NOT NULL,  
    code_postal INTEGER DEFAULT 75001,  
    ville VARCHAR(100) DEFAULT 'Paris'  
  
    ) ENGINE=INNODB;
```

Créer une table

Pour créer une clé primaire composite (clé primaire composée de plusieurs attributs), il convient de la définir à la fin du `CREATE TABLE`

Le même principe d'applique pour `UNIQUE`

```
CREATE TABLE client(  
    nom VARCHAR(50) NOT NULL,  
    prenom VARCHAR(50) NOT NULL,  
    code_postal INTEGER DEFAULT 75001,  
    ville VARCHAR(100) DEFAULT 'Paris',  
    PRIMARY KEY(nom,prenom),  
    UNIQUE(code_postal, ville)  
);
```


Les clés étrangères

Les clés étrangères permettent de garantir l'intégrité d'une table

Table client

id	nom	prenom	age	telephone
1	Foucault	Patrick	53	0123456789
2	Dubois	Blandine	30	0645678965
3	Malou	Manuel	44	0299999999

Table facture

id	client	montant
1	2	125,52
2	2	856.32
3	1	4126.79

Avec les clés étrangères

- Il ne sera pas possible d'insérer une facture pour un client qui n'existe pas
- Il ne sera pas possible de supprimer un client s'il possède toujours des factures

Les clés étrangères

Pour créer une clé étrangère lors de la création d'une table, il faut ajouter cette contrainte à la fin du `CREATE TABLE`

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nom_table (  
    nom_attribut TYPE_ATTRIBUT [OPTIONS]  
    .../  
    CONSTRAINT nom_contrainte FOREIGN KEY(colonne_cle_etrangere)  
    REFERENCES table_reference(colonne_reference)  
)
```

Exemple

```
CREATE TABLE facture (  
    ...  
    CONSTRAINT fk_client_id_facture FOREIGN KEY(client)  
    REFERENCES client(id)  
);
```

Supprimer une table

Pour supprimer une table, on utilise la commande `DROP TABLE`

```
DROP TABLE nom_table;
```

Exemple : `DROP TABLE client;`

Modifier une table

La commande `ALTER TABLE` permet de modifier la structure d'une table

- ajouter ou supprimer un attribut
- ajouter ou supprimer une clé primaire
- ajouter ou supprimer une contrainte d'unicité
- changer la valeur par défaut d'une colonne
- modifier les paramètres d'une colonne
- changer le nom de la table

Ajout d'un attribut

- `ALTER TABLE nom_table ADD definition_attribut;`
- **Exemple :** `ALTER TABLE client ADD rue VARCHAR(100) NOT NULL;`
- L'attribut est ajouté à la fin
- Pour le mettre au début, terminer la commande par `FIRST`
- Pour le mettre à une place spécifique, terminer la commande par `AFTER nom_attribut_precedent`

Modifier une table

Supprimer un attribut

- `ALTER TABLE nom_table DROP nom_attribut;`
- **Exemple :** `ALTER TABLE client DROP rue;`

Créer une clé primaire

- `ALTER TABLE nom_table ADD PRIMARY KEY(attribut);`
- **Exemple :** `ALTER TABLE client ADD PRIMARY KEY(nom);`

Supprimer une clé primaire

- `ALTER TABLE nom_table DROP PRIMARY KEY;`
- **Exemple :** `ALTER TABLE client DROP PRIMARY KEY;`

Ajout d'une contrainte d'unicité

- `ALTER TABLE nom_table ADD UNIQUE [nom_contrainte](attributs);`
- **Exemple :** `ALTER TABLE client ADD UNIQUE(nom, prenom);`

Modifier une table

Changer la valeur par défaut

- `ALTER TABLE nom_table ALTER nom_attribut SET DEFAULT valeur;`
- `ALTER TABLE nom_table ALTER nom_attribut DROP DEFAULT;`
- **Exemple :** `ALTER TABLE client ALTER prenom SET DEFAULT 'INDEFINI';`

Modifier la définition d'un attribut sans le renommer :

- `ALTER TABLE nom_table MODIFY nom_attribut nouvelle_definition;`
- **Exemple :** `ALTER TABLE client MODIFY code_postal VARCHAR(5);`

Modifier la définition d'un attribut en le renommant :

- `ALTER TABLE nom_table CHANGE nom_attribut nouveau_nom nouvelle_definition;`
- **Exemple :** `ALTER TABLE client CHANGE ville town VARCHAR(100) DEFAULT 'Paris';`

Modifier une table

Renommer la table

- `ALTER TABLE nom_table RENAME nouveau_nom;`
- **Exemple :** `ALTER TABLE client RENAME customer;`

Ajouter une clé étrangère

- `ALTER TABLE nom_table ADD CONSTRAINT nom_contrainte FOREIGN KEY (colonne_cle_etrangere) REFERENCES table_reference(colonne_reference);`
- **Exemple :** `ALTER TABLE facture ADD CONSTRAINT fk_client_id_facture FOREIGN KEY(client) REFERENCES client(id);`

Supprimer une clé étrangère

- `ALTER TABLE nom_table DROP FOREIGN KEY nom_contrainte;`
- **Exemple :** `ALTER TABLE facture DROP FOREIGN KEY fk_client_id_facture;`

SHOW et DESC

La requête `SHOW TABLES;` permet d'afficher les tables existantes

La commande `DESC nom_table;` permet d'afficher la structure d'une table

Exemple : `DESC client;`

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI		auto_increment
nom	varchar(50)	NO			
prenom	varchar(50)	NO			
code_postal	int(11)	YES		75001	
ville	varchar(100)	YES		Paris	

Insertion de données

Courte introduction au SELECT

Pour obtenir le contenu d'une table, on utilise l'instruction SELECT

```
SELECT * FROM nom_table;
```

Exemple : `SELECT * FROM client;`

Insérer des données

L'insertion de données passe par l'utilisation de la commande INSERT

```
INSERT INTO nom_table  
VALUES(valeur_col_1, valeur_col_2, valeur_col_3,...);
```

Exemple :

```
INSERT INTO client VALUES (1, 'Foucault', 'Patrick', 76000, 'Rouen');
```

Dans le cas d'une clé primaire en AUTO_INCREMENT, on peut lui mettre NULL comme valeur et elle sera attribuée automatiquement

```
INSERT INTO client VALUES (NULL, 'Foucault', 'Patrick', 76000, 'Rouen');
```

Insérer des données

Si on ne souhaite pas renseigner certaines colonnes, on peut utiliser `INSERT INTO` en précisant les colonnes remplies

Exemple : `INSERT INTO client (nom, prenom) VALUES ('Duclos', 'Franck');`

Les attributs non renseignés se verront attribuer leur valeur par défaut ou `NULL`

La clé primaire en `AUTO_INCREMENT` se verra attribuer la prochaine valeur

Insérer des données

Pour insérer plusieurs enregistrements à la fois, il faut séparer les lignes par une virgule

Exemple :

```
INSERT INTO client (nom, prenom)
VALUES ('Duclos', 'Franck'),
('Balu', 'Jacqueline'),
('Draveil', 'Théo');
```

Les sélections simples

L'instruction SELECT

L'instruction `SELECT` permet de sélectionner des enregistrements dans une table

Elle comprend généralement 3 parties :

- La clause `SELECT` (obligatoire) pour préciser les colonnes qui constitueront le résultat
- La clause `FROM` (obligatoire) pour indiquer les tables où récupérer les valeurs
- La clause `WHERE` (facultative) pour spécifier des conditions permettant de définir si une ligne doit faire partie du résultat

```
SELECT  liste_attributs  
FROM    nom_table  
WHERE   conditions;
```

L'instruction SELECT

L'instruction SELECT peut comporter d'autres clauses

```
SELECT liste_colonnes  
FROM liste_tables  
[WHERE jointures AND criteres_selections]  
[GROUP BY attributs_regroupement]  
[HAVING criteres_restrictions];
```

Exemple :

```
SELECT nom, prenom  
FROM client  
WHERE code_postal = 75001;
```


La projection

La projection permet d'indiquer les colonnes que l'on souhaite récupérer

Pour sélectionner toutes les colonnes

- `SELECT *`
- Exemple : `SELECT * FROM client;`

Pour ne sélectionner que quelques colonnes

- `SELECT liste_colonnes`
- Exemple : `SELECT nom, prenom FROM client;`

La projection

Si on ne souhaite récupérer que les enregistrements distincts, on ajouter le mot clé `DISTINCT`

- `SELECT DISTINCT liste_colonnes`
- **Exemple :** `SELECT DISTINCT nom, prenom FROM client;`

Dans un `SELECT`, on peut renommer les colonnes du résultat

- `SELECT nom 'Nom', prenom 'Prénom' FROM client;`

La restriction

Pour restreindre les résultats, on utilise la clause `WHERE` en ajoutant des conditions que les enregistrements devront respecter

```
SELECT *  
FROM client  
WHERE code_postal = 75001;
```

Les opérateurs de comparaison

Pour écrire ces restrictions, on dispose de plusieurs opérateurs de comparaison

Opérateur	Signification
=	Egal
<	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal
<> ou !=	Différent
<=>	Egal (valable pour NULL)

Les opérateurs de comparaison

Exemple :

```
SELECT nom, prenom  
FROM client  
WHERE age > 60;
```

Pour NULL, les opérateurs = et != ne fonctionneront pas. Pour cela, on utilise <=> ou IS (éventuellement accompagné de NOT)

```
SELECT nom, prenom  
FROM client  
WHERE age IS NULL;
```

```
SELECT nom, prenom  
FROM client  
WHERE age IS NOT NULL;
```

Les opérateurs logiques

Les opérateurs logiques permettent de combiner les critères de sélection

AND (ou &&) pour le ET

```
SELECT nom, prenom  
FROM client  
WHERE age < 60 AND age > 40;
```

OR (ou ||) pour le OU

```
SELECT nom, prenom  
FROM client  
WHERE code_postal = 75001 OR code_postal = 75002;
```

Les opérateurs logiques

NOT (ou !) pour le NON

```
SELECT nom, prenom  
FROM client  
WHERE age > 30 AND NOT code_postal=75001;
```

XOR pour le OU exclusif

```
SELECT nom, prenom  
FROM client  
WHERE age > 18 XOR code_postal = 75001;
```

Récupération des clients qui sont soit âgés de plus de 18 ans, soit habitant le code postal 75001, mais pas les deux

Le texte

Le `LIKE` permet de faire des recherches dans des champs de type texte

On peut utiliser des jokers

- `_` remplace n'importe quel caractère
- `%` remplace n'importe quelle chaîne de caractères (y compris vide)

Utiliser le caractère d'échappement `\` pour rechercher `%` ou `_`

Exemples :

```
SELECT nom, prenom FROM client WHERE nom LIKE 'Du%';
```

```
SELECT nom, prenom FROM client WHERE nom LIKE '%o%';
```

```
SELECT nom, prenom FROM client WHERE nom NOT LIKE '_o%';
```

Il est possible d'utiliser `LIKE` pour des nombres avec MySQL

Les intervalles

L'opérateur `BETWEEN` permet de spécifier un intervalle lors des sélections

```
SELECT * FROM client WHERE age BETWEEN 18 AND 60;
```

`BETWEEN` fonctionne aussi pour les dates

```
SELECT * FROM facture WHERE date_facture BETWEEN  
'2017-01-01' AND '2017-12-31';
```

Associer `NOT` avec `BETWEEN` permet de faire des recherches hors intervalle

Les ensembles

L'opérateur `IN` permet de spécifier des ensembles de valeurs lors des recherches

Il évite l'utilisation de nombreux opérateurs logiques `OR`

```
SELECT * FROM client WHERE age IN(20,25,30,35);
```

```
SELECT * FROM client WHERE ville NOT IN ('Paris','Lyon','Rennes');
```

Le tri

La clause `ORDER BY` permet de spécifier un tri pour le résultat

`ORDER BY attribut1, attribut2, ...`

Exemple : `SELECT nom, prenom FROM client ORDER BY age;`

Le tri est effectué de façon croissante

Pour spécifier un tri décroissant, on ajoute `DESC` au nom de l'attribut lors du tri

Exemple : `SELECT nom, prenom FROM client ORDER BY age DESC, nom;`

Limiter le résultat

MySQL offre la possibilité de limiter le nombre de lignes retournées par un `SELECT` en ajoutant `LIMIT` à la fin de la requête

```
LIMIT nombre_lignes
```

```
SELECT * FROM client LIMIT 20;
```

Il est possible aussi de commencer la récupération du résultat à partir d'une certaine ligne (décalage)

```
LIMIT nombre_lignes OFFSET ligne_debut
```

```
SELECT * FROM client LIMIT 20 OFFSET 20
```

Modification et suppression

La suppression

La suppression d'enregistrements s'effectue grâce à la commande DELETE

```
DELETE FROM nom_table WHERE conditions;
```

Les conditions permettent de spécifier les enregistrements qui devront être supprimés

ATTENTION ! Ne pas spécifier de conditions videra la table de tous ses enregistrements

- `DELETE FROM client;`

Exemple : `DELETE FROM client WHERE code_postal=75001;`

Pour écrire les conditions, on peut utiliser tous les opérateurs utilisés dans la clause WHERE du SELECT

La modification

La modification des enregistrements passe par la commande UPDATE

```
UPDATE nom_table  
SET colonne_1=valeur_1, colonne_2=valeur_2,...  
WHERE conditions;
```

Exemple :

```
UPDATE client  
SET code_postal=76100, ville='Rouen'  
WHERE nom='Foucault' AND prenom='Patrick';
```

Attention ! Ne pas spécifier de conditions viendra modifier tous les enregistrements

Les clés primaires sont très utiles pour venir modifier un enregistrement en particulier

La modification

Parfois, MySQL pourra refuser la modification si la clause `WHERE` du `UPDATE` ne se base pas sur une clé primaire

- Il s'agit du mode *safe-updates*

Pour désactiver ce mode

```
SET SQL_SAFE_UPDATES = 0;
```


Sélections multi-tables

Les jointures

Les jointures permettent de joindre plusieurs tables

Table facture

id	client	montant
1	2	125,52
2	2	856.32
3	1	4126.79

Table client

id	nom	prenom	age	telephone
1	Foucault	Patrick	53	0123456789
2	Dubois	Blandine	30	0645678965
3	Malou	Manuel	44	0299999999



Après une jointure

id	client	montant	id	nom	prenom	age	telephone
1	2	125,52	2	Dubois	Blandine	30	0645678965
2	2	856.32	2	Dubois	Blandine	30	0645678965
3	1	4126.79	1	Foucault	Patrick	53	0123456789

La jointure interne

La jointure interne s'effectue par la clause `INNER JOIN` et exige qu'il y ait des données de part et d'autre de la jointure

On utilise la clause `ON` pour spécifier les conditions de la jointure

```
SELECT *  
FROM table_1  
INNER JOIN table_2  
ON table_1.colonne = table_2.colonne;
```

Exemple :

```
SELECT client.nom, client.prenom, facture.montant  
FROM client  
INNER JOIN facture  
ON client.id = facture.client  
WHERE client.age > 18;
```

Les clients qui n'ont pas de facture ne seront pas récupérés

La jointure externe

La jointure externe peut se faire par la gauche ou par la droite

La jointure externe permet de sélectionner les lignes qui n'ont pas forcément de correspondance

Par la gauche

- On utilise la clause `LEFT JOIN`
- On indique qu'on veut toutes les lignes de la table de gauche, même sans correspondance dans la table de droite

Par la droite

- On utilise la clause `RIGHT JOIN`
- On indique qu'on veut toutes les lignes de la table de droite, même sans correspondance dans la table de gauche

La jointure externe

Par la gauche

```
SELECT *  
FROM table_1  
LEFT JOIN table_2  
ON table_1.colonne_1 = table_2.colonne_1  
AND table_1.colonne_2 = table_2.colonne_2;
```

Exemple :

```
SELECT client.nom, client.prenom, facture.montant  
FROM client  
LEFT JOIN facture  
ON client.id = facture.client  
WHERE client.age > 18;
```

Les clients qui n'ont pas de facture seront récupérés avec un enregistrement *facture* à NULL

Pour une jointure par la droite, on remplace LEFT JOIN par RIGHT JOIN

La jointure naturelle

La jointure naturelle permet de faire une jointure interne sans spécifier les colonnes utilisées pour la jointure

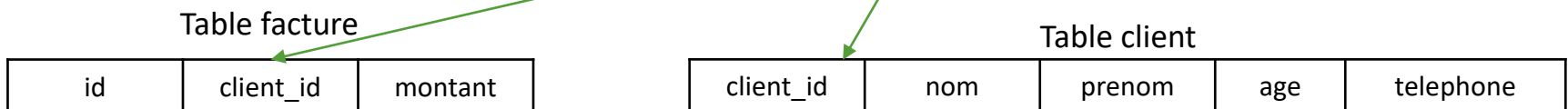
MySQL fera la jointure automatiquement selon les colonnes qui ont le même nom dans les deux tables

```
SELECT *  
FROM table_1  
NATURAL JOIN table_2;
```

Exemple:

```
SELECT client.nom, client.prenom, facture.montant  
FROM client  
NATURAL JOIN facture  
WHERE client.age > 18;
```

La jointure est faite automatiquement selon `client_id` (nom identique dans les deux tables)



La jointure sans JOIN

Il est possible d'écrire des jointures internes sans utiliser `INNER JOIN`

```
SELECT *  
FROM table_1, table_2  
WHERE table_1.colonne_1 = table_2.colonne_1  
AND table_1.colonne_2 = table_2.colonne_2;
```

Exemple :

```
SELECT client.nom, client.prenom, facture.montant  
FROM client, facture  
WHERE client.id = facture.client  
AND client.age > 18;
```

Cette syntaxe est déconseillée car elle mélange les jointures avec les conditions de recherche

Les requêtes imbriquées

Les requêtes imbriquées permettent d'utiliser le résultat d'une requête (un `SELECT`) dans une autre requête (autre `SELECT`)

Exemple dans les conditions:

```
SELECT *  
FROM facture  
WHERE client_id IN  
(SELECT id FROM client WHERE age > 25);
```


Les requêtes imbriquées

Le résultat d'une requête `SELECT` peut être utilisé dans la clause `FROM` d'un autre `SELECT`

Exemple :

```
SELECT *  
FROM (SELECT id, nom, prenom FROM client WHERE age > 18) AS client_majeur  
WHERE client_majeur.nom LIKE 'A%';
```

ANY, ALL et SOME

Dans le contexte des sous-requêtes, SQL propose les opérateurs ANY, ALL et SOME

ANY

- Synonyme de SOME
- Permet de spécifier la condition "Au moins une des valeurs"
- =ANY et équivalent à IN
- Exemple

```
SELECT *  
FROM client  
WHERE age > ANY  
  (SELECT age FROM client WHERE prenom='Patrick');
```

ALL

- Permet de spécifier la condition "Toutes les valeurs"
- <>ALL est équivalent à NOT IN
- Exemple

```
SELECT *  
FROM client  
WHERE age < ALL  
  (SELECT age FROM client WHERE prenom='Patrick');
```

Les unions

L'union permet d'additionner les résultats de deux `SELECT`

- Les deux `SELECT` doivent avoir le même nombre de colonnes dans leur résultat
- Les types des colonnes peuvent être différents

```
SELECT ...
```

```
UNION
```

```
SELECT ...
```

Exemple :

```
SELECT client.nom, client.prenom FROM client WHERE  
code_postal = 75001
```

```
UNION
```

```
SELECT fournisseur.nom, fournisseur.prenom FROM fournisseur  
WHERE code_postal = 75001;
```

Les unions

Lorsqu'on fait une union, les doublons sont retirés

Pour conserver les doublons, on dispose de `UNION ALL`

Exemple :

```
SELECT client.nom, client.prenom FROM client WHERE  
code_postal = 75001  
UNION ALL  
SELECT fournisseur.nom, fournisseur.prenom FROM fournisseur  
WHERE code_postal = 75001;
```

Les fonctions

Les opérations simples

Dans la clause `SELECT`, il est possible d'utiliser les opérateurs mathématiques standards

Opérateur	Signification
+	Addition
-	Soustraction
*	Multiplication
/	Division
DIV	Division entière
% ou MOD	Modulo

```
SELECT montant*2, 'EUR' FROM facture;
```

La première colonne du résultat sera le montant de la facture multiplié par 2

La deuxième colonne du résultat sera toujours la chaîne de caractères 'EUR'

Les fonctions

Une fonction permet d'effectuer une suite précise d'opérations et retourne le résultat de cette dernière

Une fonction porte un nom qui permet de l'identifier

Une fonction peut accepter des paramètres qui seront les données d'entrée de la fonction

- Ce sont les arguments de la fonction

On appelle une fonction lorsqu'on la sollicite en lui communiquant les informations dont elle a besoin pour fonctionner

Exemples :

```
SELECT PI ( ) ;
```

```
SELECT MAX (age) FROM client ;
```

On peut utiliser les fonctions n'importe où dans une requête, par exemple dans la clause `SELECT` ou la clause `WHERE`

Fonctions générales

`VERSION()`

- Pour obtenir la version en cours de MySQL
- `SELECT VERSION();`

`LAST_INSERT_ID()`

- Pour obtenir le dernier id utilisé lors d'une auto-incrémentation
- Peut être utilisée dans un `INSERT` qui suit un autre `INSERT`
- `INSERT INTO facture(client, montant)
VALUES(LAST_INSERT_ID(), 85.25);`

`FOUND_ROWS()`

- Pour obtenir le nombre de lignes retournées par la dernière requête exécutée
- `SELECT FOUND_ROWS();`

Quelques fonctions

Certaines fonctions permettent de faire des calculs

Nom	Description	Exemple
CEIL(<i>n</i>)	Arrondi supérieur	CEIL(15.28) retourne 16
FLOOR(<i>n</i>)	Arrondi inférieur	FLOOR(15.28) retourne 15
ROUND(<i>n</i> , <i>nb_decimales</i>) ROUND(<i>n</i>)	Arrondi à <i>nb_decimales</i> décimales, ou sans décimales ni non renseigné	ROUND(15.28, 1) retourne 15.3 ROUND(15.28) retourne 15
TRUNCATE(<i>n</i> , <i>nb_dec</i>)	Enlève les décimales en trop	TRUNCATE(15.28, 1) retourne 15.2
POWER(<i>n</i> , <i>e</i>)	<i>n</i> à la puissance <i>e</i>	POW(2, 4) retourne 16
SQRT(<i>n</i>)	Racine carrée	SQRT(16) retourne 4
RAND()	Nombre aléatoire entre 0 et 1	
SIGN(<i>n</i>)	Signe d'un nombre (-1 si négatif, 0 si 0, 1 si positif)	SIGN(-4) retourne -1
MOD(<i>n</i> , <i>div</i>)	Reste de la division entière de <i>n</i> par <i>div</i>	MOD(8, 3) retourne 2
ABS(<i>n</i>)	Valeur absolue	ABS(-45) retourne 45

Quelques fonctions

Certaines fonctions permettent de traiter des chaînes de caractères

Nom	Description	Exemple
<code>BIT_LENGTH(s)</code>	Longueur d'une chaîne en bits	<code>BIT_LENGTH('aé')</code> retourne 24 (8 + 16)
<code>CHAR_LENGTH(s)</code>	Longueur d'une chaîne en caractères	<code>CHAR_LENGTH('aé')</code> retourne 2
<code>LENGTH(s)</code>	Longueur d'une chaîne en octets	<code>CHAR_LENGTH('aé')</code> retourne 3
<code>STRCMP(s1, s2)</code>	Compare deux chaînes (0 si identique, -1 si <i>s1</i> avant <i>s2</i> , 1 si <i>s1</i> après <i>s2</i>)	<code>STRCMP('car', 'pic')</code> retourne -1
<code>REPEAT(s, n)</code>	Répète <i>n</i> fois la chaîne <i>s</i>	<code>REPEAT('ah', 4)</code> retourne 'ahahahah'
<code>LPAD(s, long, c)</code>	Donne à la chaîne <i>s</i> la longueur <i>long</i> donnée (raccourcie si trop longue, ajout du caractère <i>c</i> à gauche sinon pour compléter)	<code>LPAD('abcdef', 3, '%')</code> retourne 'abc' <code>LPAD('ahah', 6, '%')</code> retourne '%%ahah'
<code>RPAD(s, long, c)</code>	Comme <code>LPAD</code> mais ajout à droite	<code>LPAD('ahah', 6, '%')</code> retourne 'ahah%%'
<code>TRIM(s)</code>	Supprime les espaces situés au début et à la fin de la chaîne <i>s</i> (voir la documentation pour toutes les possibilités)	<code>TRIM(' abc ')</code> retourne 'abc'
<code>SUBSTRING(c, pos, long)</code>	Extraction d'une sous-chaîne Jusqu'à la fin si <i>long</i> non spécifié	<code>SUBSTRING('abcdefgh', 5)</code> retourne 'efgh' <code>SUBSTRING('abcdefgh', 5, 2)</code> retourne 'ef')

Quelques fonctions

Autres fonctions pour les chaînes de caractères

Nom	Description	Exemple
<code>LOWER(s)</code>	Mise en minuscule	<code>LOWER('Maison')</code> retourne 'maison'
<code>UPPER(s)</code>	Mise en majuscule	<code>UPPER('Maison')</code> retourne 'MAISON'
<code>LEFT(s, n)</code>	n premiers caractères de s en partant de la gauche	<code>LEFT('maison', 3)</code> retourne 'mai'
<code>RIGHT(s, n)</code>	n premiers caractères de s en partant de la droite	<code>RIGHT('maison', 3)</code> retourne 'son'
<code>REVERSE(s)</code>	Inverse une chaîne	<code>REVERSE('maison')</code> retourne 'nosiam'
<code>REPLACE(s, old, new)</code>	Remplace des caractères dans s	<code>REPLACE('toto', 'o', 'i')</code> retourne 'titi'

<https://dev.mysql.com/doc/refman/5.7/en/string-functions.html>

Les fonctions

MySQL propose un grand nombre de fonctions

La documentation officielle permet d'avoir des détails sur chacune d'elles accompagnés de nombreux exemples

<https://dev.mysql.com/doc/refman/5.7/en/functions.html>

Les agrégats

Fonctions d'agrégation

MySQL met à disposition plusieurs fonctions qui s'avèrent très utiles

- COUNT pour compter
- MAX pour obtenir un maximum
- MIN pour obtenir un minimum
- AVG pour obtenir une moyenne
- SUM pour calculer une somme

Ces fonctions peuvent être utilisées telles quelles dans une requête

Obtenir le nombre de clients :

- `SELECT COUNT(*) FROM client;`

Obtenir le nombre de clients dont l'âge n'est pas NULL :

- `SELECT COUNT(age) FROM client;`

Obtenir l'âge du client le plus âgé :

- `SELECT MAX(age) FROM client;`

La clause GROUP BY

La clause `GROUP BY` permet de spécifier un regroupement sur lequel sera appliqué une fonction d'agrégation

Dans un `SELECT`, la clause `GROUP BY` se trouve après le `WHERE`

```
SELECT colonnes  
FROM nom_table  
WHERE conditions  
GROUP BY nom_colonne_1, nom_colonne_2,...;
```

Exemples :

- Obtenir l'âge du client le plus âgé selon le prénom

```
SELECT prenom, MAX(age) FROM client GROUP BY prenom;
```

- Obtenir le nombre de clients selon l'âge

```
SELECT age, COUNT(*) FROM client GROUP BY age;
```

La clause GROUP BY

Dans un `SELECT` qui utilise une clause `GROUP BY`, on ne pourra utiliser que :

- Une fonction d'agrégation
- La ou les colonnes utilisées dans la clause `GROUP BY`

MySQL permettra d'utiliser d'autres colonnes dans le `SELECT` mais le résultat obtenu dans ces colonnes n'est pas prévisible

- `SELECT nom, MAX(age) FROM client;` **ne retournera pas** le nom et l'âge du client le plus âgé !

Le tri des données suite à une requête utilisant `GROUP BY` est toujours possible

```
SELECT prenom, MAX(age) as max_age
FROM client
GROUP BY prenom
ORDER BY max_age;
```


La clause GROUP BY

La requête dans laquelle on utilise un GROUP BY peut comporter des jointures

Exemple pour obtenir le montant total des factures pour chaque client

```
SELECT client.nom, client.prenom, SUM(facture.montant)
FROM client
INNER JOIN facture ON client.id = facture.id_client
GROUP BY client.id;
```

Il est bien évidemment possible de spécifier plusieurs critères de regroupement dans la clause GROUP BY

Exemple pour obtenir l'âge du client le plus âgé par couple prénom/ville

```
SELECT prenom, ville, MAX(age)
FROM client
GROUP BY prenom, ville
```

Fonction GROUP_CONCAT

La fonction `GROUP_CONCAT` est une fonction d'agrégation permettant de concaténer des chaînes de caractères

Exemple pour obtenir une chaîne de caractères contenant la liste des clients pour chaque ville

```
SELECT ville, GROUP_CONCAT(nom)
FROM client
GROUP BY ville;
```

Syntaxe

```
GROUP_CONCAT (
    [DISTINCT] col1 [, col2, ...]
    [ORDER BY col [ASC | DESC]]
    [SEPARATOR sep]
)
```

`DISTINCT` permet de supprimer les doublons
`ORDER BY` permet de faire un tri
`SEPARATOR` permet de spécifier le séparateur

Les super-agrégats

Lorsqu'on ajoute `WITH ROLLUP` à la fin d'un `GROUP BY`, cela nous permet d'obtenir des super-agrégats

Des valeurs par rupture sont ajoutées à nos résultats et un total final est affiché

Exemple

```
SELECT COALESCE(client.age, 'Total') as age,  
       COALESCE(client.ville, '') as ville,  
       SUM(montant) as total  
FROM client  
INNER JOIN facture ON client.id = facture.id_client  
GROUP BY client.age, client.ville WITH ROLLUP;
```

Ici, on obtient le total des factures selon l'âge et la ville des clients

- Un cumul par âge est également ajouté
- Un cumul total est ajouté à la fin du résultat
- La fonction `COALESCE` permet d'éviter l'affichage de `NULL`

age	ville	total
12	Paris	13.92
12		13.92
14	Paris	2.21
14		2.21
41	Paris	4.25
41		4.25
55	Rouen	4.63
55		4.63
89	Paris	6.21
89		6.21
Total		31.22

Les vues

Les vues

Une vue est composée :

- D'un nom
- D'une requête

Une fois qu'une vue est créée, on peut l'interroger comme on le ferait avec une table

Une vue est donc une "table virtuelle" composée des éléments définis dans la requête qui lui est associée

Les vues peuvent permettre, entre autres :

- D'accéder directement à une information sans devoir refaire à chaque fois une requête SQL complexe
- De limiter l'accès à certaines informations grâce à la gestion des droits

Créer une vue

Pour créer une vue, la syntaxe est

```
CREATE [OR REPLACE] VIEW nom_vue  
AS requete;
```

Exemple :

```
CREATE VIEW client_facture  
AS SELECT client.nom, client.prenom, facture.montant  
FROM client  
INNER JOIN facture ON client.id = facture.id  
ORDER BY facture.montant;
```

Ensuite, on peut requêter la vue comme on le fait pour une table

```
SELECT * FROM client_facture;
```

Attention ! Modifier la structure des tables qui sont utilisées dans des vues peut rendre ces vues inopérantes (par exemple en supprimant une colonne utilisée dans une vue)

Créer une vue

Les colonnes de la vue peuvent être renommées

- Cela est utile si deux colonnes de la vue ont par défaut le même nom

Pour cela, on donne le nom de chaque colonne après le nom de la vue dans le CREATE

Exemple :

```
CREATE OR REPLACE VIEW client_facture(nom_client,  
prenom_client, montant_facture)  
AS SELECT client.nom, client.prenom, facture.montant  
FROM client  
INNER JOIN facture ON client.id = facture.id  
ORDER BY facture.montant;
```

Lors de la création d'une vue, on ne peut pas utiliser un FROM se basant sur un SELECT

Supprimer une vue

Pour supprimer une vue, on utilise la syntaxe

```
DROP VIEW [IF EXISTS] nom_vue;
```

Exemple :

```
DROP VIEW client_facture;
```


Modifier une vue

Pour modifier une vue, on dispose de deux syntaxes

```
CREATE OR REPLACE VIEW nom_vue AS requete;
```

```
ALTER VIEW nom_vue AS requete;
```

Exemple :

```
ALTER VIEW client_facture  
AS SELECT client.nom, client.prenom, facture.montant  
FROM client  
INNER JOIN facture ON client.id = facture.id  
ORDER BY facture.montant DESC;
```

Les transactions

Les transactions

Les transactions permettent d'assurer l'intégrité d'une base de données

Elles permettent de regrouper plusieurs requêtes dans un même bloc

Le bloc dans son intégralité devra être exécuté pour que la transaction soit valide

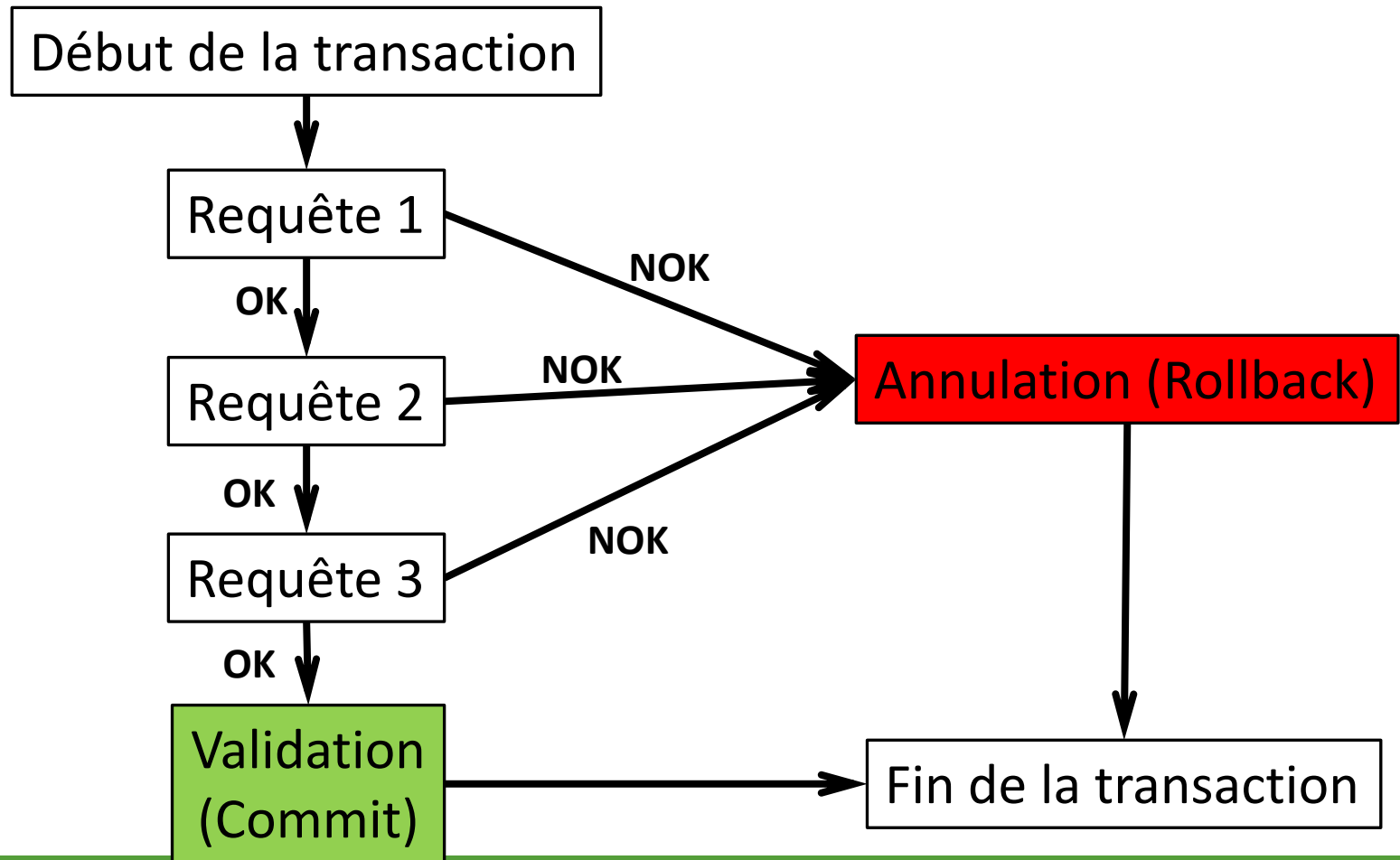
Elles sont très utiles dans le cas d'un traitement comprenant plusieurs requêtes

- On ne souhaite appliquer les changements que si toutes les requêtes se sont déroulées sans erreur

Sous MySQL, on ne peut pas utiliser les transactions sur les tables gérées par le moteur MyISAM

Les tables gérées par le moteur InnoDB supportent les transactions

Les transactions



Les transactions

Quand on valide une transaction, il s'agit d'un *commit*

- On *commite* les changements

Quand on annule une transaction, on effectue un *rollback*

Par défaut, MySQL est en mode *autocommit*

- Chaque requête est validée automatiquement à la fin de son exécution

```
SHOW VARIABLES WHERE variable_name='autocommit';
```

Pour désactiver le mode *autocommit* pour la connexion en cours :

```
SET autocommit=0;
```

Les commandes

Pour valider les changements, on utilise la commande `COMMIT`

```
COMMIT;
```

Rien ne sera commité tant que la commande `COMMIT` n'a pas été lancée

- Couper la connexion à MySQL en cours de transaction invalidera la transaction

Pour faire un rollback, on utilise la commande `ROLLBACK`

```
ROLLBACK;
```

Exemple :

```
1) INSERT INTO client(nom, prenom) VALUES ('Dupont',  
    'Jacques'), ('Keita', 'Marcel');  
2) COMMIT;  
3) SELECT * FROM client;  
4) UPDATE client SET code_postal=76000 WHERE id=8;  
5) SELECT * FROM client;  
6) ROLLBACK;  
7) SELECT * FROM client;
```

Les transactions

Pendant une transaction

- Les changements sont visibles dans la connexion qui effectue la transaction
- Les changements ne sont pas visibles dans une connexion autre que celle qui effectue la transaction

Lorsque le mode auto-commit est activé, il est possible de démarrer quand même une transaction par la commande

```
START TRANSACTION;
```

Attention ! Toutes les commandes qui créent, modifient ou suppriment des objets dans la base valident implicitement les transactions

Pendant une transaction, des verrous sont posés sur les lignes en cours de modification

- Une autre session qui souhaite modifier une de ces lignes sera mise en attente tant que la transaction de la session initiale n'est pas commitée

Les savepoints

Il n'est pas possible de démarrer une transaction au sein d'une transaction

Il est cependant possible de créer des jalons (*savepoints*) au sein d'une transaction

- Il sera ainsi possible d'annuler les requêtes effectuées depuis un savepoint particulier

Créer un savepoint

- `SAVEPOINT nom_savepoint;`

Revenir à l'état d'un savepoint

- `ROLLBACK TO SAVEPOINT nom_savepoint;`

Supprimer un jalon

- `RELEASE SAVEPOINT nom_savepoint;`

Les savepoints

Exemple :

1. `INSERT INTO client(nom, prenom) VALUES ('Durand', 'Marie');`
2. `SAVEPOINT svp;`
3. `INSERT INTO client(nom, prenom) VALUES ('Duroc', 'Paul');`
4. `ROLLBACK TO SAVEPOINT svp;`
5. `COMMIT;`
6. `SELECT * FROM client;`

Les requêtes préparées

Les requêtes préparées

Les requêtes préparées sont très utilisées dans le développement d'applications

Elles permettent

- D'empêcher l'injection SQL
- De gagner en performances

L'idée d'une requête préparée est de créer une requête générale qui pourra être utilisée avec des valeurs différentes

Exemple :

- Je récupère régulièrement un client grâce à son nom et son prénom
- J'ai toujours besoin d'une requête du type

```
SELECT * FROM client WHERE nom=valeur_nom AND prenom=valeur_prenom;
```

Les requêtes préparées

Dans une requête préparée, on remplace les éléments changeant par un signe ?

Exemple

- `SELECT * FROM client WHERE nom=? AND prenom=?;`

Attention ! Il n'est pas possible d'utiliser le ? dans la clause `SELECT` ou la clause `FROM`

Dans MySQL, la déclaration d'une requête préparée passe par la commande `PREPARE` et il convient de donner un nom à la requête préparée pour pouvoir l'utiliser ensuite

```
PREPARE nom_requete
FROM requete;
```

Exemple

- `PREPARE select_client
FROM 'SELECT * FROM client WHERE nom=? AND prenom=?';`

Les variables

Pour exécuter une requête préparée, il faut fournir à MySQL les valeurs voulues pour chaque paramètre (les ?) de la requête

Pour cela, il convient d'utiliser les variables

```
SET @nom_variable = valeur;
```

Exemple : `SET @nom = 'Foucault';`

Une variable est disponible uniquement dans la connexion en cours

Ce sont des variables qui seront utilisées pour l'exécution d'une requête préparée

Il est bien évidemment possible d'utiliser les variables directement dans une requête `SELECT`

- `SELECT * FROM client WHERE nom=@nom;`

L'exécution

Pour exécuter une requête préparée, on utilise la commande `EXECUTE` accompagnée du nom de la requête préparée et des paramètres éventuels

```
EXECUTE nom_requete  
[USING @param1, @param2, ...];
```

Exemple complet :

1. `PREPARE clients_cp
FROM 'SELECT * FROM client WHERE code_postal=?';`
2. `SET @code_postal=75001;`
3. `EXECUTE clients_cp USING @code_postal;`
4. `SET @code_postal=76000;`
5. `EXECUTE clients_cp USING @code_postal;`

SQL procédural

Les procédures stockées

Les procédures stockées sont une suite d'instructions SQL stockées dans la base

Comme une requête préparée, une procédure stockée porte un nom et on utilise ce nom pour exécuter la procédure

Cependant, contrairement aux requêtes préparées qui n'existent que dans la session de l'utilisateur, les procédures stockées sont enregistrées dans la base et restent donc disponibles

Les procédures stockées existent depuis la version 5 de MySQL

Création et suppression

Pour créer une procédure d'une requête, on utilise la syntaxe

```
CREATE PROCEDURE nom_procedure([parametres])  
instructions_procedure
```

Exemple

```
CREATE PROCEDURE sel_clients_majeurs()  
SELECT * FROM client WHERE age >= 18;
```

Pour supprimer une procédure

```
DROP PROCEDURE nom_procedure;
```

Exemple

```
DROP PROCEDURE sel_clients_majeurs;
```

Exécution

L'exécution d'une procédure passe par l'utilisation de la commande
CALL

```
CALL nom_procedure( [parametres] );
```

Exemple

```
CALL sel_clients_majeurs();
```

Les blocs d'instructions

Pour mettre plusieurs instructions dans une procédure stockée, on doit déclarer un bloc qui commence par `BEGIN` et se termine par `END`

```
DELIMITER //
CREATE PROCEDURE nom_procedure([parametres])
BEGIN
  instructions_procedure
END//
```

Il est nécessaire de modifier le délimiteur avec `DELIMITER` afin de pouvoir utiliser `;` dans les instructions

Les délimiteurs les plus utilisés sont

- `//`
- `|`

Les blocs d'instructions

Exemple :

```
DELIMITER //  
CREATE PROCEDURE sel_clients_majeurs()  
BEGIN  
SELECT * FROM client WHERE age >= 18;  
END //
```

Les paramètres

Les paramètres d'une procédure stockée permettent de modifier son comportement selon des valeurs qui lui sont transmises

Un paramètre est composé

- D'un sens
 - IN : Paramètre entrant. Il s'agit une valeur transmise à la procédure stockée
 - OUT : Paramètre sortant. Il s'agit d'une valeur qui sera donnée par la procédure
 - INOUT : Paramètre à la fois entrant et sortant
- D'un nom
- D'un type SQL

Exemple :

```
DELIMITER //  
CREATE PROCEDURE sel_clients_sup_age(IN age_min INT)  
BEGIN  
    SELECT * FROM client WHERE age >= age_min;  
END //
```

```
CALL sel_clients_sup_age(30) //
```

Les paramètres

Les paramètres `OUT` sont utilisés pour récupérer de l'information depuis une procédure stockée

Exemple

1.

```
DELIMITER //  
CREATE PROCEDURE clients_age_range(OUT age_min INTEGER, OUT  
age_max INTEGER)  
BEGIN  
    SELECT MIN(AGE), MAX(age) INTO age_min, age_max FROM  
client;  
END //
```
2.

```
CALL clients_age_range(@age_minimal, @age_maximal) //
```
3.

```
SELECT @age_minimal, @age_maximal //
```

La structure

Une procédure stockée peut comporter des blocs `BEGIN...END;` imbriqués

Au début d'un bloc, il est possible de déclarer des variables locales au bloc

- `DECLARE nom_variable type_variable [DEFAULT valeur_defaut];`

Ainsi, un bloc `BEGIN` a toujours cette structure

```
BEGIN

-- Déclarations de variables

-- Instructions

END;
```

La structure IF

Les traitements conditionnels permettent de faire d'exécuter des instructions selon certaines conditions

```
IF condition_1 THEN
    instructions_1
ELSEIF condition_2 THEN
    instructions_2
ELSE
    instructions_3
END IF;
```

Les conditions s'expriment de la même façon que dans les requêtes SQL, avec les mêmes opérateurs (=, <, >, <=, >=, LIKE, IS NULL,...)

Selon la condition qui est respectée, le bloc d'instruction exécuté changera

Les instructions du ELSE sont exécutées si aucune condition n'est respectée

La structure IF

Exemple :

```
1. DELIMITER //
```

```
2. CREATE PROCEDURE majeur_mineur(IN id_client INTEGER)
  BEGIN
    DECLARE age_client INT DEFAULT 0;

    SELECT age INTO age_client FROM client
                                     WHERE id=id_client;

    IF age_client < 18 THEN SELECT 'Mineur';
    ELSE SELECT 'Majeur';
    END IF;
  END //
```

```
3. CALL majeur_mineur(3) //
```

La structure CASE

La structure CASE permet de faire un traitement différent selon la valeur d'une variable. Les WHEN définissent les valeurs possibles.

```
CASE variable
    WHEN possibilite_1 THEN instructions_1
    [WHEN possibilite_2 THEN instructions_2]
    [ELSE instructions]
END CASE;
```

Exemple :

```
CREATE PROCEDURE dpt(IN dpt_num INTEGER)
BEGIN
    DECLARE dpt VARCHAR(20);
    CASE dpt_num
        WHEN 75 THEN SELECT 'Paris' INTO dpt;
        WHEN 95 THEN SELECT 'Val d\'Oise' INTO dpt;
        WHEN 22 THEN SELECT 'Côtes d\'Armor' INTO dpt;
        ELSE SELECT 'Inconnu' INTO dpt;
    END CASE;
    SELECT dpt;
END //
```

La structure CASE

La partie ELSE est importante

- Sans celle-ci, si aucune correspondance n'est trouvée, une erreur est lancée

Autre notation :

```
CREATE PROCEDURE region(IN dpt_num INTEGER)
BEGIN
    DECLARE region VARCHAR(20);
    CASE
        WHEN dpt_num IN(22,35,56,29) THEN
            SELECT 'Bretagne' INTO region;
        WHEN dpt_num IN(14,27,50,61,76) THEN
            SELECT 'Normandie' INTO region;
        ...
        ELSE SELECT 'Inconnu' INTO region;
    END CASE;

    SELECT region;
END //
```

La boucle WHILE

La boucle `WHILE` permet de répéter un bloc d'instructions tant qu'une condition est vraie

```
WHILE condition DO
    instructions;
    .....
END WHILE;
```

La condition est une combinaison d'expressions sous la forme classique SQL

```
BEGIN
    DECLARE x INTEGER DEFAULT 150;

    WHILE x <= 210 DO
        INSERT INTO numero(valeur) VALUES (x) ;
        SELECT x+1 INTO x;
    END WHILE;
END//
```

La boucle UNTIL

La boucle `UNTIL` permet de répéter un bloc d'instructions jusqu'à ce qu'une condition devienne vraie

```
REPEAT
    instructions;
    .....
UNTIL condition END REPEAT;
```

La condition est une combinaison d'expressions sous la forme classique SQL

```
BEGIN
    DECLARE x INTEGER DEFAULT 400;

    REPEAT
        INSERT INTO numero(valeur) VALUES (x) ;
        SELECT x+1 INTO x;
    UNTIL x >= 500 END REPEAT;

END//
```

LEAVE et ITERATE

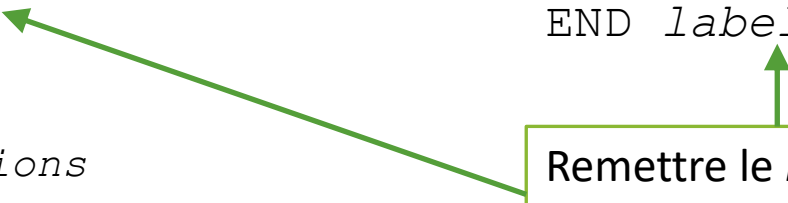
Les boucles et les blocs d'instructions peuvent être nommés avec les *labels*

```
label: WHILE condition DO  
    instructions  
END WHILE label;
```

```
label: REPEAT  
    instructions  
UNTIL condition END REPEAT;
```

```
label: BEGIN  
    instructions  
END label;
```

Remettre le *label* à la fin est facultatif



L'instruction `LEAVE label;` permet de quitter immédiatement la boucle ou le bloc nommé `label`

L'instruction `ITERATE label;` permet d'arrêter immédiatement l'itération en cours de la boucle `label` dans une boucle et de passer à la suivante

La boucle LOOP

Le LOOP est la structure itérative de base

Elle ne peut être arrêtée qu'avec un LEAVE

```
BEGIN
    DECLARE x INTEGER DEFAULT 0;

    boucle: LOOP
        SELECT x+1 INTO x;
        IF x > 10 THEN LEAVE boucle; END IF;
        INSERT INTO numero(valeur) VALUES (x) ;

    END LOOP;
END;
```

Les erreurs

Dans une procédure stockée, il peut arriver qu'une instruction particulière engendre une erreur SQL (contrainte de clé étrangère, mise à NULL d'une colonne NOT NULL,...)

Il est possible de capturer ces erreurs et de les traiter plutôt que de laisser la procédure se traiter en erreur

Il faut pour cela déclarer un gestionnaire d'erreur (*handler*)

```
DECLARE { EXIT | CONTINUE }  
HANDLER FOR { numero_erreur | { SQLSTATE identifiant_erreur } | condition }  
instructions
```

Un gestionnaire d'erreur se déclare après la déclaration des variables locales et avant les instructions

Le travail du gestionnaire sera de prendre en charge l'erreur spécifiée si elle a lieu

Les erreurs

```
DECLARE { EXIT | CONTINUE }  
HANDLER FOR { numero_erreur | { SQLSTATE identifiant_erreur } | condition }  
instructions
```

EXIT et CONTINUE indiquent quoi faire après l'exécution du *handler*

- EXIT : arrêter la procédure
- CONTINUE : continuer la procédure

Pour identifier une erreur, on dispose de 3 possibilités

- Erreur MySQL
- Identifiant de l'état SQL (SQLSTATE)
- Une condition

```
mysql> SELECT * FROM table_inexistante;  
ERROR 1146 (42S02): Table 'comptabilite.table_inexistante' doesn't exist
```



Erreur MySQL (numérique)

SQLSTATE (5 caractères)

Les erreurs

La liste des états SQL et des erreurs est documentée

<https://dev.mysql.com/doc/refman/5.7/en/error-messages-server.html>

Les erreurs les plus courantes :

Code MySQL	SQLSTATE	Description
1048	23000	La colonne ne peut pas être NULL
1169	23000	Violation de contrainte d'unicité (UNIQUE)
1216	23000	Violation de clé étrangère : insertion ou modification impossible (table avec la clé étrangère)
1217	23000	Violation de clé étrangère: suppression ou modification impossible (table avec la référence de la clé étrangère)
1172	42000	Plusieurs lignes de résultats alors qu'on ne peut en avoir qu'une seule
1242	21000	La sous-requête retourne plusieurs lignes de résultats alors qu'on ne peut en avoir qu'une seule

Les erreurs

Exemple de gestion d'erreur

```
DELIMITER //
CREATE PROCEDURE add_client(IN c_nom VARCHAR(50), IN c_prenom VARCHAR(50))
BEGIN
    DECLARE EXIT HANDLER FOR SQLSTATE '23000'
        SELECT 'Erreur ! Arrêt de la procédure.';

    INSERT INTO client(nom, prenom) VALUES(c_nom, c_prenom);
END //
```

L'état SQL '23000' correspond à une erreur de contrainte

Si la colonne `prenom` est en NOT NULL dans la table `client`, l'instruction `CALL add_client('Durand', NULL);` retournera

```
+-----+
| Erreur ! Arrêt de la procédure. |
+-----+
| Erreur ! Arrêt de la procédure. |
+-----+
1 row in set (0.00 sec)
```

Les erreurs

Autre exemple avec l'utilisation de l'erreur MySQL

```
DELIMITER //
CREATE PROCEDURE add_client(IN c_nom VARCHAR(50), IN c_prenom VARCHAR(50))
BEGIN
    DECLARE EXIT HANDLER FOR 1048
    BEGIN
        SELECT 'Une colonne est NULL';
        SELECT 'Arrêt de la procédure';
    END;

    INSERT INTO client(nom, prenom) VALUES(c_nom, c_prenom);
END //
```

L'erreur MySQL 1048 correspond à une erreur d'ajout de NULL dans une colonne NOT NULL

Les erreurs

La troisième possibilité pour déclarer un gestionnaire d'erreur est d'utiliser une `CONDITION`

```
DECLARE { EXIT | CONTINUE }  
HANDLER FOR condition  
instructions
```

MySQL propose 3 conditions prédéfinies

- `SQLWARNING` : Avertissements et notes - Etats SQL commençant par '01'
- `NOT FOUND` : Utilisé pour les curseurs - Etats SQL commençant par '02'
- `SQLEXCEPTION` : Les erreurs - Autres états SQL

Utilisation :

```
DECLARE EXIT HANDLER FOR SQLEXCEPTION  
BEGIN  
    SELECT 'Une erreur est survenue...';  
    SELECT 'Arrêt prématuré de la procédure';  
END;
```

Les erreurs

Il est possible de venir définir ses propres conditions

```
DECLARE nom_condition CONDITION FOR { SQLSTATE  
etat_SQL | erreur_MySQL };
```

Exemple :

```
DECLARE constraint_violation CONDITION FOR SQLSTATE '23000';  
  
DECLARE EXIT HANDLER FOR constraint_violation  
BEGIN  
    SELECT 'Constraint violation';  
    SELECT 'Stop';  
END;
```

Les erreurs

Il est tout à fait possible de

- Déclarer plusieurs gestionnaires pour la même erreur
- Utiliser un gestionnaire pour plusieurs erreurs

Exemple :

```
DECLARE fk_violation CONDITION FOR 1452;
DECLARE unique_violation CONDITION FOR 1062;

DECLARE EXIT HANDLER FOR fk_violation
BEGIN
    SELECT 'Foreign key violation';
END;

DECLARE EXIT HANDLER FOR unique_violation
BEGIN
    SELECT 'Unique violation';
END;

DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING
BEGIN
    SELECT 'Error';
END;
```

Les curseurs

Au sein d'une procédure stockée, les curseurs permettent de parcourir les résultats d'une requête `SELECT`

Afin d'être utilisé, un curseur doit être déclaré puis ouvert

A la fin de son utilisation, un curseur doit être fermé

Déclaration

```
DECLARE nom_curseur CURSOR FOR requete_select;
```

Ouverture

```
OPEN nom_curseur;
```

Fermeture

```
CLOSE nom_curseur;
```


Les curseurs

Une fois ouvert, un curseur est positionné sur le premier résultat du `SELECT`

Il devient alors possible de récupérer ce résultat et d'avancer le curseur au résultat suivant grâce à la commande

```
FETCH nom_curseur INTO variable_1, variable_2, ...;
```

Il n'est pas possible de mettre à jour **directement** la ligne courante d'un curseur

Enfin, le parcours d'un résultat avec un curseur doit obligatoirement se faire ligne par ligne. Il n'est pas possible de sauter des lignes ou de revenir en arrière

Les curseurs

Exemple :

```
DELIMITER //
CREATE PROCEDURE parcours_clients()
BEGIN
    DECLARE vnom VARCHAR(50);
    DECLARE vage INTEGER;

    DECLARE c_clients CURSOR
        FOR SELECT nom, age FROM client ORDER BY nom;

    OPEN c_clients;

    FETCH c_clients INTO vnom, vage;

    SELECT CONCAT(vnom, ' : ', vage) AS 'Client 1';

    FETCH c_clients INTO vnom, vage;

    SELECT CONCAT(vnom, ' : ', vage) AS 'Client 2';

    CLOSE c_clients;
END//
```

Les curseurs

Pour parcourir l'ensemble d'un résultat, on utilise la boucle `LOOP` dans laquelle on ajoute une instruction `LEAVE` en utilisant un gestionnaire d'erreur adapté

```
DELIMITER //
CREATE PROCEDURE parcours_clients_all()
BEGIN
    DECLARE vnom VARCHAR(50);
    DECLARE vage INTEGER;
    DECLARE fin_cursor BOOLEAN DEFAULT FALSE;
    DECLARE c_clients CURSOR FOR SELECT nom, age FROM client ORDER BY nom;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET fin_cursor = TRUE;

    OPEN c_clients;

    boucle_clients: LOOP
        FETCH c_clients INTO vnom, vage;

        IF fin_cursor THEN LEAVE boucle_clients; END IF;

        SELECT CONCAT(vnom, ' : ', vage) AS 'Client';
    END LOOP;

    CLOSE c_clients;
END//
```

Les fonctions stockées

Une fonction stockée est une suite d'instructions qui retourne un résultat unique

Contrairement à une procédure stockée, une fonction stockée peut être appelée directement depuis un `SELECT`

Création d'une fonction

```
CREATE FUNCTION nom_fonction(parametre_1,parametre_2,...)  
RETURNS type_retour  
instructions
```

L'instruction `RETURN` est utilisée dans la fonction pour retourner le résultat au sein de la fonction

Les paramètres d'une fonction sont toujours de type `IN`

Pour supprimer une fonction, on utilise `DROP FUNCTION`

```
DROP FUNCTION nom_fonction;
```

Les fonctions stockées

Exemple de fonction stockée

```
DELIMITER //
CREATE FUNCTION categorie_client(p_age INTEGER)
RETURNS VARCHAR(10)
BEGIN
    DECLARE categorie varchar(10);

    IF p_age < 18 THEN
        SET categorie = 'JUNIOR';
    ELSEIF p_age < 60 THEN
        SET categorie = 'ACTIF';
    ELSE
        SET categorie = 'SENIOR';
    END IF;

    RETURN categorie;
END//
```

Utilisation

```
SELECT nom, categorie_client(age) FROM client;
```

Les triggers

Les triggers

Un trigger, appelé aussi déclencheur, est lié à une table et permet d'effectuer des opérations lorsqu'une ou plusieurs lignes sont insérées, modifiées ou supprimées

Ainsi, un trigger peut être déclenché par :

- INSERT
- UPDATE
- DELETE

Par ailleurs, un trigger peut être déclenché avant ou après l'instruction liée à ce trigger

Un trigger pourra manipuler toute les tables à l'exception de celle qui le concerne ou encore modifier la ligne insérée, supprimée ou modifiée

- Un trigger ne pourra donc pas manipuler la table qui l'a déclenché

Depuis la version 5.7 plusieurs triggers peuvent être liés au même évènement

Les triggers

Les triggers peuvent être utiles pour :

- S'assurer de la validité d'une donnée avant insertion ou mise à jour
- Effectuer un historique des opérations effectuées sur une table
- Effectuer des traitements en cascade

Un trigger est composé d'instructions, comme c'est le cas avec les procédures stockées

L'écriture d'un trigger respectera les mêmes règles que celles des procédures stockées

Les triggers

Création d'un trigger

```
CREATE TRIGGER nom_trigger  
    moment_trigger evenement_trigger  
    ON nom_table FOR EACH ROW  
    instructions_trigger
```

- *nom_trigger* indique quelle est la table liée à ce trigger
- *moment_trigger* indique si le trigger doit être exécuté avant ou après l'instruction de déclenchement (valeurs possibles : BEFORE et AFTER)
- *evenement_trigger* indique quel évènement sur la table doit enclencher le trigger (valeurs possibles : INSERT, UPDATE et DELETE)
- Comme le suggère la partie FOR EACH ROW, les instructions du trigger seront exécutées pour chaque ligne insérée, mise à jour ou effacée

OLD et NEW

Dans les instructions d'un trigger on dispose de deux mots-clés très utiles

- OLD
- NEW

OLD contient les valeurs de la ligne en cours de traitement avant qu'elle ne soit modifiée par l'évènement déclencheur

- Lecture possible mais modification impossible
- Existe pour un trigger UPDATE et DELETE

NEW contient les valeurs de la ligne en cours de traitement après qu'elle ait été modifiée par l'évènement déclencheur

- Lecture et modification possibles
- Existe pour un trigger UPDATE et INSERT

Supprimer un trigger

Pour supprimer un trigger

```
DROP TRIGGER nom_trigger;
```

On ne peut pas modifier un trigger

- Il faut le supprimer pour le recréer

Supprimer une table supprime aussi les triggers qui lui sont associés

Les triggers

Exemple de trigger pour l'enregistrement de modifications dans une table

```
DELIMITER //
CREATE TRIGGER upt_client
AFTER UPDATE ON client FOR EACH ROW
BEGIN
    INSERT INTO client_archive VALUES(OLD.id, OLD.nom, OLD.prenom, OLD.code_postal, OLD.ville, now());
END;
//
```

Exemple de trigger qui force le code postal à 99999 si, lors de l'insertion, le code postal d'un client est inférieur à 1000

```
DELIMITER //
CREATE TRIGGER ins_client BEFORE INSERT ON client FOR EACH ROW
BEGIN
    IF NEW.code_postal < 1000 THEN
        SET NEW.code_postal = 99999;
    END IF;
END //
```

Les erreurs dans les triggers

Dans le cas d'une erreur au sein d'un trigger `BEFORE`, la requête qui a déclenché le trigger ne sera pas exécutée, de même qu'un éventuel trigger `AFTER`

Dans le cas d'une erreur au sein d'un trigger `AFTER`, la requête qui a déclenché le trigger échouera

La finalité globale de l'opération dépendra du moteur utilisé pour le stockage des tables

- Tables transactionnelles (InnoDB) : un `ROLLBACK` sera effectué
- Tables non-transactionnelles (MyISAM) : Les modifications sont enregistrées jusqu'au moment de l'erreur

La sécurité

Les utilisateurs

Pour se connecter à une base MySQL il faut s'authentifier

L'authentification se caractérise par

- Le nom du serveur
- L'identifiant
- Le mot de passe

MySQL fournit alors un mécanisme de droits afin de déterminer ce que chaque utilisateur est autorisé à faire

Un utilisateur possède des privilèges

MySQL dispose donc de deux couches de droits :

- A la connexion (identifiant / mot de passe)
- A l'utilisation (privilèges)

Les utilisateurs

Dans la base mysql, on retrouve les tables qui contiennent les informations sur les utilisateurs et leurs droits

- `user`
- `db`
- `tables_priv`
- `columns_priv`
- `procs_priv`

Il est tout à fait possible de manipuler ces tables pour gérer les droits des utilisateurs

Néanmoins, MySQL met à disposition des commandes dédiées à la gestion des utilisateurs et des privilèges

Les utilisateurs

Création d'un utilisateur

- `CREATE USER 'identifiant'@'hote' [IDENTIFIED BY 'mdp'];`
- La valeur *hote* indique d'où l'ordinateur va se connecter (IP, nom de machine, localhost) - L'utilisation de `%` est possible
- Exemple :

```
CREATE USER 'florent'@'localhost' IDENTIFIED BY 'secret';
```

Suppression d'un utilisateur

- `DROP USER 'identifiant'@'hote';`
- Exemple :

```
DROP USER 'florent'@'localhost';
```

Les utilisateurs

Renommer un utilisateur

- `RENAME USER 'identifiant'@'hote'`
`TO 'nouvel_identifiant'@'nouveau_hote';`

Modification d'un mot de passe pour un utilisateur

- `SET PASSWORD FOR 'identifiant'@'hote' = 'mot_de_passe';`
- **Exemple :**
`SET PASSWORD FOR 'florent'@'localhost' = 'secret';`

Les privilèges

Un utilisateur nouvellement créé ne dispose d'aucun privilège

Pour accorder ou retirer des privilèges à un utilisateur, on utilise les commandes `GRANT` et `REVOKE`

MySQL dispose de plusieurs privilèges

- `SELECT`, `INSERT`, `UPDATE`, `DELETE` pour les requêtes classiques

Privilège	Explication
<code>CREATE TABLE</code>	Création de tables
<code>CREATE TEMPORARY TABLE</code>	Création de tables temporaires
<code>CREATE VIEW</code>	Création de vues
<code>ALTER</code>	Modification de tables
<code>DROP</code>	Suppression de tables, de bases et de vues

Les privilèges

Privilège	Explication
CREATE ROUTINE	Création de procédures et de fonctions stockées
ALTER ROUTINE	Modification de procédures et de fonctions stockées
EXECUTE	Exécution de procédures et de fonctions stockées
TRIGGER	Création et suppression de triggers
CREATE USER	Gestion des utilisateurs

MySQL dispose d'un grand nombre de privilèges attribuables, chacun ayant ses particularités ou fonctionnant avec un autre

La documentation se révèle indispensable pour gérer finement les droits

<https://dev.mysql.com/doc/refman/5.7/en/privileges-provided.html>

Les privilèges

Quand un privilège est attribué, il faut spécifier à quel élément il est attribué

Niveau	Explication
<code>*.*</code>	Privilège global. Appliqué à toutes les bases et objets. Table <code>mysql.user</code>
<code>*</code>	Identique à <code>*.*</code> si aucune base n'a été sélectionnée auparavant avec <code>USE</code> . Sinon, le privilège s'appliquera à la base en cours. Table <code>mysql.db</code>
<code>nom_base.*</code>	Privilège de base de données. S'applique à toute la base. Table <code>mysql.db</code>
<code>nom_base.nom_table</code>	Privilège appliqué sur une table. Table <code>mysql.tables_priv</code>
<code>nom_table</code>	Privilège appliqué sur une table avec la base préalablement sélectionnée avec <code>USE</code> . Table <code>mysql.tables_priv</code>
<code>nom_base.nom_routine</code>	Privilège appliqué à une procédure ou fonction stockée. Table <code>mysql.procs_priv</code>

Les privilèges

Ajout de privilège

```
GRANT privilege[,privilege,...] ON [type_objet] niveau_privilege  
TO utilisateur [IDENTIFIED BY 'mot_de_passe']
```

Exemple

- GRANT SELECT, INSERT ON comptabilite.client TO 'florent'@'localhost';
- GRANT SELECT ON comptabilite.* TO 'florent'@'localhost';

Il est possible de préciser les colonnes sur lesquelles s'applique un privilège

Exemple

- GRANT UPDATE(nom,prenom) ON comptabilite.client TO 'florent'@'localhost';

Les privilèges

Il est possible d'afficher les privilèges d'un utilisateur

```
SHOW GRANTS FOR 'florent'@'localhost';
```

Pour supprimer les droits, la commande REVOKE fonctionne sur le même modèle que GRANT

- `REVOKE SELECT, INSERT ON comptabilite.client FROM 'florent'@'localhost';`
- `REVOKE SELECT ON comptabilite.* FROM 'florent'@'localhost';`

Les privilèges

MySQL propose des privilèges spéciaux

ALL PRIVILEGES

- Identique à ALL
- Attribue tous les droits (sauf GRANT OPTION) à un utilisateur pour un niveau spécifié

USAGE

- Aucun privilège
- Utile pour utiliser la commande GRANT sans spécifier de nouveaux droits
- Exemple pour modifier un mot de passe :

```
GRANT USAGE ON *.* TO 'florent'@'localhost' IDENTIFIED BY 'azerty';
```

GRANT OPTION

- Autorisation pour attribuer des privilèges
- Un utilisateur ne pourra attribuer que les privilèges qu'il possède déjà
- Il est possible de l'utiliser :
 - En tant que privilège classique
 - A la fin d'un GRANT avec la clause WITH GRANT OPTION (sauf pour ALL PRIVILEGES)

Les privilèges

Lors d'une requête, MySQL vérifiera que l'utilisateur a les droits pour cette requête

Il consultera les tables

- `user`
- `db`
- `tables_priv`
- `columns_priv`

Si un droit positif est spécifié sur une de ces tables, alors la requête est exécutée

Les privilèges

Pour les vues, procédures, fonctions et triggers, ce sont les privilèges de l'auteur de ces objets qui sont vérifiés lors de l'exécution, et non ceux de l'utilisateur qui utilise ces objets

Ainsi, un utilisateur peut exécuter par exemple une procédure agissant sur des tables sur lesquelles il n'a aucun privilège

- Il doit simplement avoir le droit `EXECUTE`

Il est possible de modifier le contexte d'exécution (par défaut `DEFINER`) en ajoutant `SQL SECURITY DEFINER` ou `SQL SECURITY INVOKER` lors de la création

- `CREATE SQL SECURITY INVOKER PROCEDURE...`
- `CREATE SQL SECURITY INVOKER VIEW nom_vue AS...`

Si un utilisateur possède le privilège `SUPER`, il est en mesure de modifier l'utilisateur qui définit un objet (le définisseur), ce qui influera sur l'utilisation de l'objet

- `CREATE DEFINER = CURRENT_USER() PROCEDURE...`
- `CREATE DEFINER = 'utilisateur'@'hote' PROCEDURE...`

L'import / export

L'import / export

L'export d'informations dans un fichier peut se faire en utilisant un simple
`SELECT avec INTO OUTFILE`

```
SELECT *  
INTO OUTFILE 'C:/ProgramData/MySQL/MySQL Server 5.7/Uploads/export.csv'  
FIELDS TERMINATED BY ';'   
ENCLOSED BY '"'   
LINES TERMINATED BY '\r\n'  
FROM client;
```

Le fichier ne doit pas exister sur le disque

Attention ! le paramètre `secure-file-priv` de la base de données peut limiter le répertoire d'export possible

- `SHOW VARIABLES LIKE "secure_file_priv";`

L'import / export

Pour l'import des données dans la table avec un fichier CSV, l'instruction `LOAD DATA INFILE` est utilisée

```
LOAD DATA [LOCAL] INFILE 'nom_fichier'
  INTO TABLE nom_table
  [FIELDS
    [TERMINATED BY 'car_fin_colonne']
    [[OPTIONALLY] ENCLOSED BY 'car_entoure']
    [ESCAPED BY 'car_echappement']
  ]
  [LINES
    [STARTING BY 'car_debut_ligne']
    [TERMINATED BY 'car_fin_ligne']
  ]
  [IGNORE number LINE]
  [(nom_col_1,nom_col_2,...)]
```

L'import / export

Pour un export plus complet, MySQL propose l'outil `mysqldump`

```
mysqldump [options] db_name [tbl_name ...]
```

```
mysqldump [options] --databases db_name ...
```

```
mysqldump [options] --all-databases
```

La sortie peut être redirigée vers un fichier

- `> sortie.sql`
- `-r sortie.sql`

L'option `--no-data` permet de n'exporter que la structure

Exemple

```
mysqldump -u root -p comptabilite > C:/Users/florent/export.sql
```

L'import / export

Pour importer un fichier issu de mysqldump, l'outil mysql est sollicité

```
mysql -u root -p comptabilite < C:/Users/florent/export.sql
```

```
mysql -u root -p comptabilite -e "source fichier.sql"
```

Il est aussi possible de se connecter à la base de données et ensuite utiliser la commande

```
SOURCE export.sql
```

L'installation de MySQL

La configuration

La configuration de MySQL se fait par l'intermédiaire d'un fichier

- `.ini` sous Windows
- `.cnf` sous Linux

Le programme d'installation de MySQL crée un fichier par défaut (pour la version 5.7) dans `C:\ProgramData\MySQL\MySQL Server 5.7`

Ce fichier de configuration est spécifié dans les propriétés du service Windows MySQL

Le fichier de configuration est rangé par sections définies par [...]

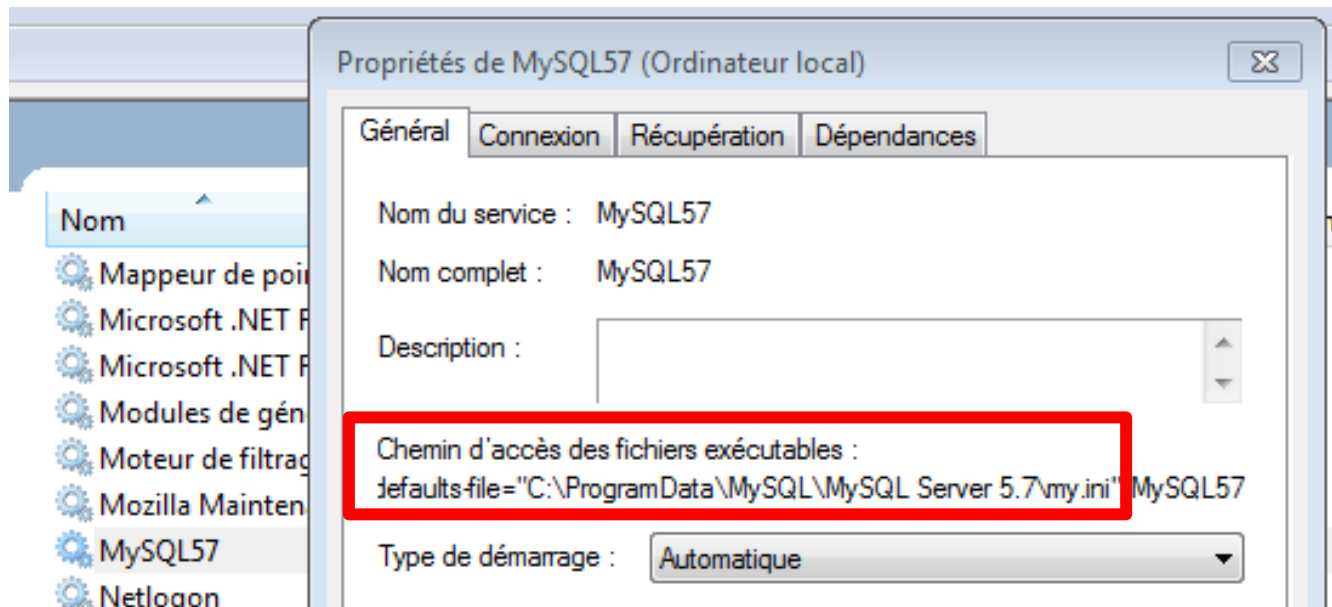
Les paramètres sont définis de la forme

- `parametre = valeur` (pour les paramètres à plusieurs valeurs)
- `option` (pour les paramètres binaires)

Le fichier est commenté avec des `#`

La configuration

Le fichier de configuration est spécifié comme paramètre de lancement dans le service Windows



La configuration

Pour changer un paramètre

- Modifier le fichier `my.ini` ou `my.cnf`

Pour certains paramètres, des commandes peuvent être utilisées

- Commande `SET SESSION <paramètre> = <valeur>` (pour les paramètres de niveau `SESSION`)
- Commande `SET GLOBAL <paramètre> = <valeur>`

Les changements des paramètres via `SET` sont perdus après

- La fermeture de la session
- Le redémarrage du serveur MySQL

La configuration

Pour consulter les paramètres

- Consulter le fichier de paramétrage
- Commande `SHOW GLOBAL VARIABLES;`
- MySQL 5.6 : `INFORMATION_SCHEMA.GLOBAL_VARIABLES`
- MySQL 5.7 : `performance_schema.global_variables`
- Commande `SELECT @@global.nom_parametre`

On retrouve les mêmes techniques pour les paramètres de session

La commande `status` permet de connaître l'état du serveur

La configuration

Dans MySQL, il y a un répertoire par base de données

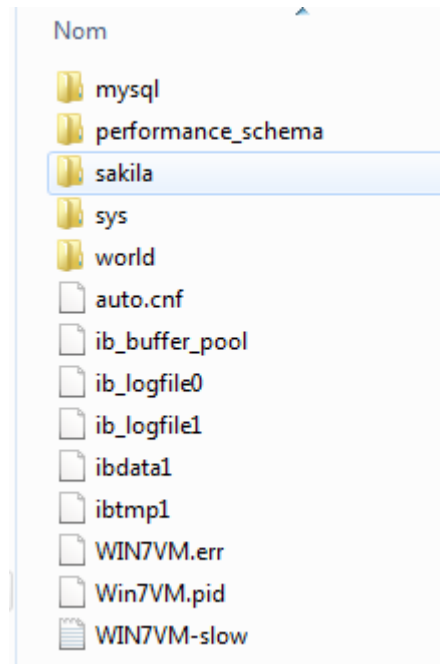
Chaque répertoire porte le nom de la base de données

C'est le paramètre `datadir` du fichier de configuration qui spécifie l'emplacement des répertoires des bases (ou schémas)

```
# Path to the database root
datadir=C:/ProgramData/MySQL/MySQL Server 5.7\Data
```

3 bases par défaut

- `mysql` = base système
- `information_schema` = dictionnaire de données (virtuelle)
- `performance_schema` (depuis la version 5.5)



Fin
