

MySQL, prise en main

Dr-Ing Pierre-Emmanuel Gros
Responsable Pôle Innovation Neuresys

Pierre-emmanuel.gros@neuresys.fr



MySQL, prise en main

- Introduction et rappels
- Modèle relationnel, conception et création d'une base
- Pratique du SQL avec MySQL
- Tables transactionnelles InnoDB
- SQL procédural
- Connexions, droits d'accès, sécurité
- Introduction à l'administration

MYSQL, un « gros » produit

- MySQL est un produit qui vient d'une volonté d'être simple
- Depuis la 5.x , il est de plus en plus complexe Mais les fonctionnalités ajoutées sont rarement utilisées
- Le smiley représente les parties simples
- L'Alien marque les parties complexes
- Certaines fonctionnalités doivent être comprises et d'autres simplement notées

Introduction et rappels

- **Introduction et rappels**
 - Versions, panorama des fonctionnalités et des outils.
 - Ressources et documentation.
 - Procédure d'installation.
 - Paramétrages de post-installation et premiers tests.
 - L'outil client ligne de commandes mysql.
 - L'outil graphique MySQL Query Browser.
- Pratique du SQL avec MySQL
- Modèle relationnel, conception et création d'une base
- Tables transactionnelles InnoDB
- SQL procédural
- Connexions, droits d'accès, sécurité
- Introduction à l'administration



Introduction et rappels

Le serveur MySQL, les versions, la documentation.

- Contenu
 - Visualisation de « l'écosystème » de MySQL
- Objectif pratiques
 - Savoir où trouver la documentation de MySQL
 - Savoir quel est la dernière version du Serveur



Le Serveur MySQL

- Premier « vrai » moteur Open Source GPL.
- Créé avec une volonté d'être simple.
- MySQL est un serveur de bases de données relationnelles SQL développé dans un souci de performances élevées en lecture.
- Il est multithread et multi-utilisateurs.
- MySQL fonctionne sur de nombreux systèmes d'exploitation différents, incluant Unix (AIX, IBM i-5, BSDi, FreeBSD, HP-UX, **Linux**, Mac OS X, NetWare, NetBSD, OpenBSD, OS/2 Warp, SGI IRIX, **Solaris**, SunOS, SCO OpenServer, SCO UnixWare, Tru64 Unix, **Windows** (2000, XP, Vista et 7)).



MySQL

1. UTILISATION

MySQL est un choix populaire de base de données pour une utilisation dans les applications web, et est une composante centrale de la couche LAMP (un acronyme pour "Linux, Apache, MySQL, Perl / PHP / Python").

MySQL est utilisé dans certains des sites les plus visités sur Internet, y compris Flickr, Nokia.com, YouTube, Wikipedia, Mais aussi SAP, de nombreux acteurs du monde financier (C.A...)

2. PLATEFORMES ET INTERFACES

MySQL est écrit en C/C++ et s'utilise sur la plupart des plateformes Unix et Windows. Le moteur est accessible via de nombreux langages tel que C/C++/Java/.NET... à travers les normes ODBC/JDBC/.Net Connector

3. OUTILS D'ADMINISTRATION

1. OFFICIEL

L'outil MySQL Workbench est un environnement libre intégré développé par Oracle, qui permet aux utilisateurs d'administrer graphiquement et visuellement les bases de données MySQL, les structures des bases et de faire des dessins de conception

L'outil « MySQL » est l'environnement en mode texte permettant d'adresser le serveur MySQL. Enfin, les sauvegardes de la base se font via l'outil « MySQL Dump »

2. NON OFFICIEL

Toad for MySQL - un outil de développement et d'administration de MySQL de Quest Software

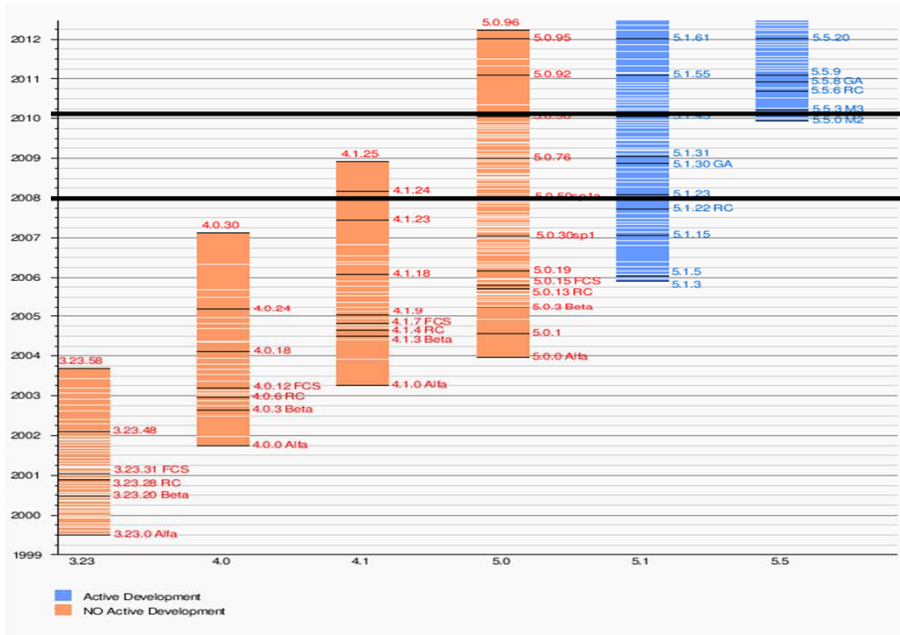
Navicat - une série d'applications graphiques de gestion de bases de données, développées pour Windows, Macintosh et Linux.

4. LE CLOUD

Virtual Machine Image : les utilisateurs peuvent télécharger une image de leur propre avec MySQL ou utiliser une image prête à l'emploi avec une installation optimisée de MySQL

Deux services notables de cloud computing MySQL sont le 'Amazon Relational Database', Xeround, qui fonctionnent sur EC2, Rackspace et Heroku.

Les différentes versions de MySQL



Les versions de MySQL

Le développement original de MySQL par Michael Widenius et David Axmark à partir de 1994

Première version interne le 23 mai 1995

Version de Windows a été libéré le 8 Janvier 1998 pour Windows 95 et NT

Version 3.19: la fin de 1996, à partir www.tcx.se

Version 3.20: Janvier 1997

Version 3.21: version de production 1998, à partir www.mysql.com

Version 3.22: alpha, bêta à partir de 1998

Version 3.23: beta de Juin 2000, 22 Janvier version de production 2001

Version 4.0: bêta à partir de Août 2002, la production de presse Mars 2003

Version 4.01: beta de Août 2003, Jyoti adopte MySQL pour une base de suivi

Version 4.1: bêta à partir de Juin 2004, (R-arbres et B-arbres, les sous-requêtes, les requêtes préparées)

Version 5.0: bêta à partir de Mars 2005, (curseurs, procédures stockées, triggers, vues, les transactions XA)

Sun Microsystems a acquis MySQL AB, le 26 Février 2008.

Version 5.1: 27 version de production Novembre 2008 (événement à l'horaire, le partitionnement, plug-in API, ligne basé sur la réplication, des tables de log du serveur)

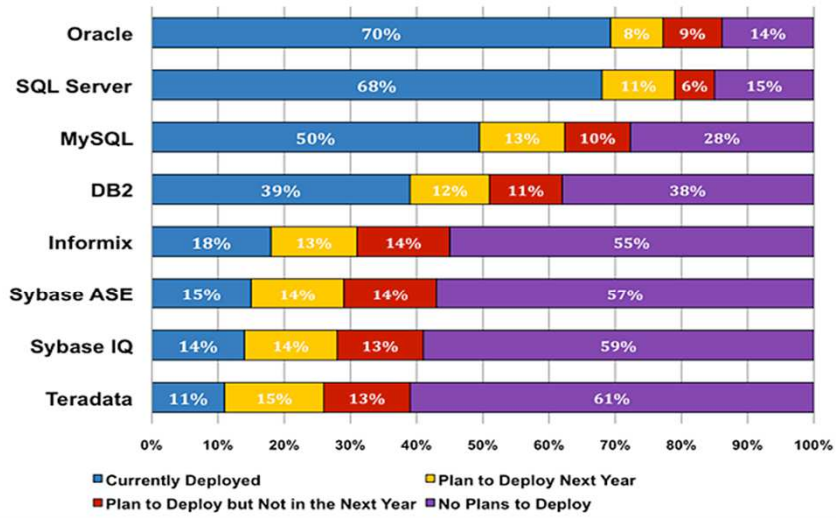
La version 5.1 contient 20 bogues connus résultat . MySQL 5.1 et 6.0 ont montré des performances médiocres lorsque utilisés pour le data warehouse .En partie à cause de son incapacité à utiliser les multiples cœurs de processeurs pour le traitement d'une requête unique .

Oracle a acquis Sun Microsystems, le 27 Janvier 2010.

MySQL Server 5.5 est actuellement généralement disponible (à partir de Décembre 2010). Améliorations et caractéristiques comprennent:

- Le moteur de stockage InnoDB par défaut, qui prend en charge les transactions et les contraintes d'intégrité référentielles.
- Amélioration de InnoDB I / O sous-système
- Amélioration du support SMP
- Réplication Semis-ynchronous.

Installation et déploiement

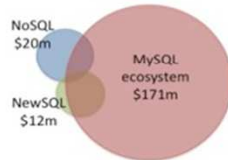


Gartner Study Shows Strong Growth in the DBMS Market - 2008

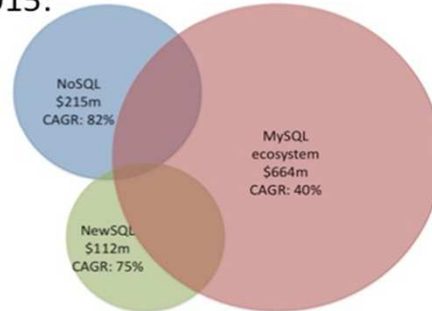
Futur de MySQL

MySQL, NoSQL, NewSQL revenue Research

2011:



2015:



451 Research,
http://blogs.the451group.com/information_management/tag/nosql/ - 2012



NoSQL vs MySQL

1. NOSQL

NoSQL est un système de gestion de base de données conçu spécifiquement pour gérer le stockage et la récupération de grandes quantités de données sans relations définies (les données Big).

Mais, les données stockées dans une base de données NoSQL peuvent être structurées. Les systèmes de NoSQL:

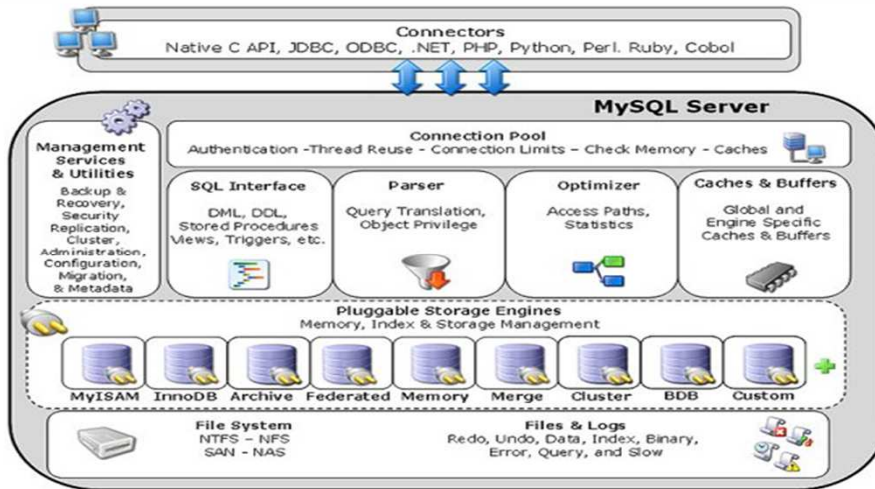
- N'utilisent pas SQL comme langage de requête,
- Peuvent garantir une cohérence ACID éventuelle
- Conçus pour la répartition de charge et la tolérance aux pannes

2. POINTS DE RÉFLEXION ENTRE MYSQL ET LES NOSQL

Les raisons d'hésitation entre MySQL et les bases NoSQL sont:

- **L'élasticité**: Il s'agit de la fonctionnalité de pouvoir ajouter/retirer des « colonnes » pour des données bien choisies
- La répartition de charge, distribution de charge: Cette fonctionnalité est la capacité de pouvoir mettre des données sur plusieurs machines ... tolérantes aux pannes
- **Le typage** (l'absence) est capable de capturer des données beaucoup plus hétérogènes que les bases telles que MySQL
- **La facilité d'utilisation** des bases NoSQL rappelle celle de MySQL lors de ses débuts

MySQL, en interne



Les fonctionnalités de MySQL

- ANSI SQL 99, Stored procedures, Triggers, Cursors, Vue modifiable, support Unicode
- Commit en deux temps avec le moteur **InnoDB engine**
- Moteur de stockage indépendant (MyISAM pour la lecture, InnoDB pour le transactionnel et l'intégrité)
- **Transactions et point de sauvegarde avec le moteur InnoDB**
- **Support SSL**
- **Cache de requêtes et sous SELECT.**
- Support de la replication
- *Index Full text avec le moteur MyISAM engine*
- *MySQL embarqué*
- Support de l'ACID avec InnoDB
- **Partition de table via "Pruning" des partitions dans l'optimiseur**



Introduction et rappels

Documentation

- Contenu:
 - Quelques références sur MySQL, SQL et la modélisation de données

La documentation

- MySQL Manuel de référence:
 - MySQL : <http://dev.mysql.com/doc/refman/5.5/en/>
 - InnoDB : <http://dev.mysql.com/doc/innodb/1.1/en/index.html>
 - Article Technique : <http://dev.mysql.com/tech-resources/articles/index.html>
- Ouvrage de Référence MySQL:
 - O'Reilly MySQL Cookbook (2nd Edition)
 - O'Reilly MySQL Stored Procedure Programming
 - O'Reilly Understanding MySQL Internals
 - O'Reilly High Performance MySQL

La documentation

- Ouvrage SQL:
 - O'Reilly, Learning SQL
 - O'Reilly, The Art of SQL
- Ouvrage Modélisation de données
 - O'Reilly The Data Model Resource Book, Vol. 1: A Library of Universal Data Models for All Enterprises
 - O'Reilly The Data Model Resource Book, Vol. 2: A Library of Data Models for Specific Industries
 - O'Reilly The Data Model Resource Book, Vol. 3: Universal Patterns for Data Modeling (Volume 3)



Introduction à l'administration

Procédure d'installation

- **Contenu:**
 - Voir une installation de MySQL sous Linux
 - Voir quelques paramètres serveurs
- **Objectifs:**
 - Savoir configurer dans les grandes lignes un serveur MySQL

Survol de l'installation de MySQL.

- L'Installation sous Windows est trivial
- Sous Unix:
 - Soit via RPM
 - Soit via les sources
- Installation sous Windows du WorkBench (création des utilisateurs, tables ...)

Préconfiguration MySQL

```
shell> groupadd mysql  
shell> useradd -g mysql mysql
```

Compilation/installation MySQL

```
shell> tar zxvf mysql-VERSION.tar.gz
shell> cd mysql-VERSION
shell> ./configure --prefix=/usr/local/mysql
shell> make
shell> make install
```

Postinstallation MySQL

```
shell> cd /usr/local/mysql
shell> chown -R mysql .
shell> chgrp -R mysql .
shell> bin/mysql_install_db --user=mysql
shell> chown -R root .
shell> chown -R mysql var
# Mise en place d'un fichier de configuration
shell> cp support-files/my-medium.cnf
/etc/my.cnf
shell> bin/mysqld_safe --user=mysql &
# mise en place du daemon
shell> cp support-files/mysql.server
/etc/init.d/mysql.server
```



Introduction à l'administration

Paramétrages de post-installation et premiers tests.

- **Contenu:**
 - Voir quelques paramètres serveurs
- **Objectifs:**
 - Savoir configurer dans les grandes lignes un serveur MySQL

Paramètres serveur essentiels .

- Au démarrage, MySQL cherche le fichier `my.cnf` ou `my.ini`
- Sous Unix par ordre:
 - `/etc/my.cnf`
 - `datadir/my.cnf`
 - `~/my.cnf`
- Sous Windows:
 - `%SystemRoot%/my.ini`
 - `C:\my.cnf`
- Il est possible de modifier le chargement via `--defaults-file=/path/to/file`

Paramètres serveur essentiels

- [mysqld]
- default-character-set=latin1 port=3306
socket=/var/lib/mysql/english.sock
- basedir="C:/Program Files/MySQL/MySQL Server 5.5/«
- datadir="C:/ProgramData/MySQL/MySQL Server 5.5/Data/"
- default-storage-engine=INNODB

Paramètres serveur essentiels

- Paramètres génériques:
 - `query_cache_size=0`
 - `table_cache=256`
 - `tmp_table_size=35M`

- Paramètres InnoDB:
 - `innodb_buffer_pool_size=107M`
 - `innodb_log_file_size=54M`



Introduction à l'administration

outil client MySQL.

- **Contenu:**
 - Voir les principaux clients MySQL
- **Objectifs:**
 - Pouvoir débiter avec MySQL

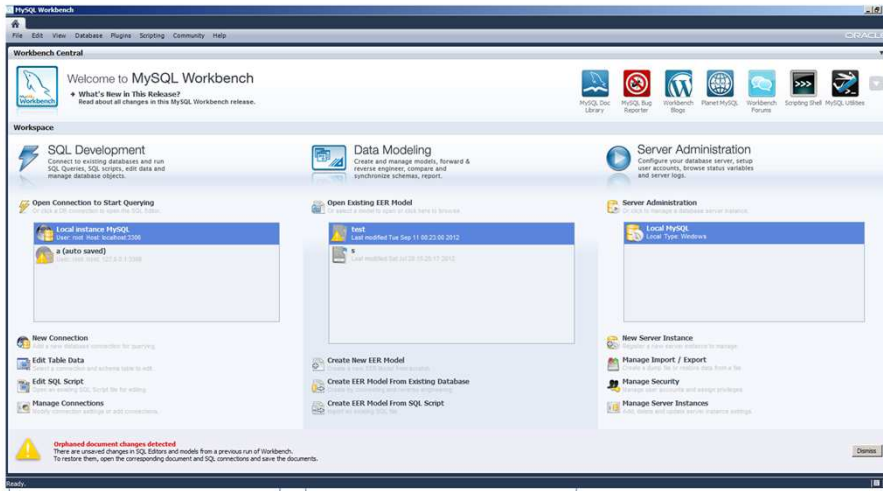
Les clients

- Client gratuit:
 - mysql
 - MySQL Query Browser
 - Squirrel
 - PHPMysqlAdmin (Web)
- Client payant
 - Navicat Premium
 - EMS for MySQL

Le client MySQL

- C'est le client en ligne de commande
- Utilisation classique :
 - `mysql --host=<localhost > --user=<login> --password=<password> <base de donnée>`

Le client MySQL Query Browser

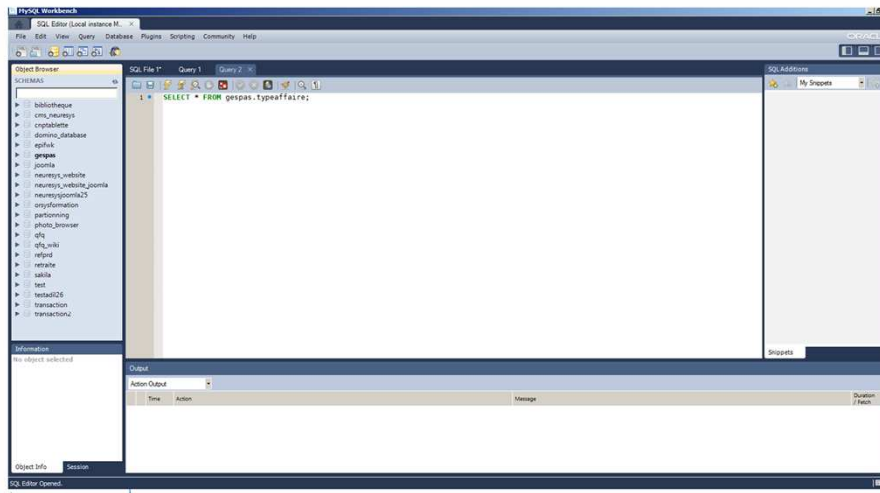


Espace SQL

Espace Modelisation

Espace Administration

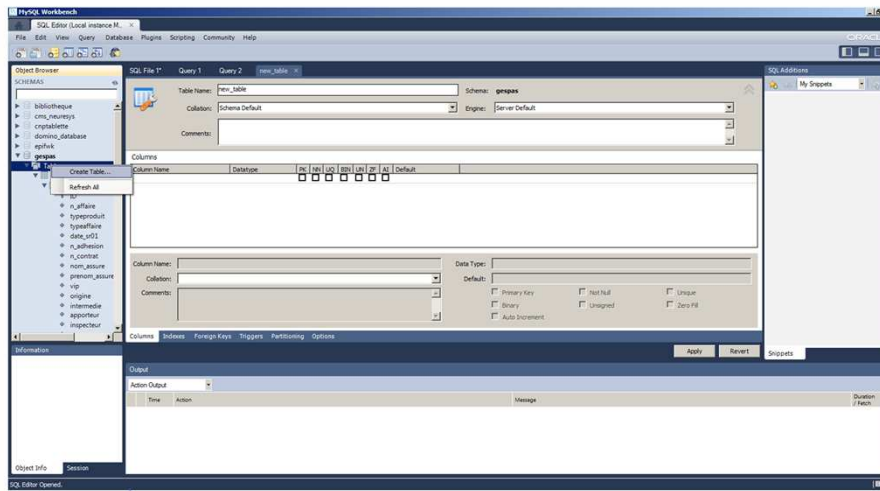
Le client MySQL Query Browser



Espace BD

Espace SQL

Le client MySQL Query Browser



Création d'une table

Pratique du SQL avec MySQL

- Introduction et rappels
- **Pratique du SQL avec MySQL**
 - Sélections simples, comparaisons, tris.
 - Sélections multitables, différents types de jointures.
 - Requêtes imbriquées, préparées.
 - Modifications et gestion des vues.
- Modèle relationnel, conception et création d'une base
- Tables transactionnelles InnoDB
- SQL procédural
- Connexions, droits d'accès, sécurité
- Introduction à l'administration



Pratique du SQL avec MySQL

Les tables, et les données

- Contenu
 - Créer une base de données
 - Créer une table
 - Charger des données dans vos tables
 - Récupérer des données à partir des tables de différentes façons
- Objectifs
 - Se rappeler les bases du langage SQL



Les tables, et les données

- Création d'une base de données:
 - `CREATE DATABASE <Nom de base >`
- Utilisation de la base :
 - `USE <Nom de base >`
- Connexion à la base:
 - `mysql -h hote -u utilisateur -p <Nom de base >`
Ex (`mysql -h localhost -u root -p <Nom de base>`)



Créer et sélectionner une base de données

1. CRÉATION D'UNE BASE DE DONNÉES:

Soit la base de données est créée par l'administrateur, soit il est possible de la créer via:

```
CREATE DATABASE bibliotheque
```

Attention sous UNIX, le nom de la base de données est sensible à la casse (bibliothèque est différent de Bibliothèque)

2. SÉLECTION DE LA BASE

Même si la base de données est créée, il faut signaler que nous voulons l'utiliser:

```
USE bibliotheque
```

```
> Database changed
```

Afin d'utiliser cette base de données, soit il faut se connecter à mysql et invoquer la commande **USE** soit on signale à l'outil mysql que l'on souhaite l'utiliser directement via:

```
shell> mysql -h hote -u utilisateur -p bibliotheque
```

```
Enter password: *****
```

Attention : le « -p » n'est pas le paramètre du password !

Les tables, et les données

- Création d'une première table:
 - **CREATE TABLE** livres
(
 livre_id INT, # l'identifiant d'un livre
 titre VARCHAR(50), -- le titre d'un livre
 auteurs VARCHAR(50) / les auteurs du livre */*
);
- Visualisation de la table:
 - **DESCRIBE** livres
- Il est possible de rajouter une colonne via:
 - **ALTER TABLE** livres **ADD COLUMN** description **VARCHAR(50);**
 - ALTER TABLE <nom de table> ADD/MODIFY/DROP ...

Création d'une table

1. CRÉATION D'UNE TABLE

La base de données est vide au démarrage tel que le montre la commande :

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

La partie vraiment difficile est la modélisation des données tel que celle d'un livre. Ici le livre est modélisé par un identifiant, un titre et un auteur.

Un identifiant peut être un entier codé dans MySQL avec le mot **INT**

Le titre ainsi que les auteurs peuvent être vus comme des chaînes de caractères de par exemple 50 caractères., ce qui signifie en langage MySQL: **VARCHAR(50)**

La création d'une table peut se faire via la commande **CREATE TABLE**, le nom de la table, et enfin une liste de colonne- type de colonne.

2. RETROUVER LES TABLES DANS UNE BASE

La commande **SHOW TABLES** permet donc de retrouver les tables:

```
mysql> SHOW TABLES;
+-----+
| Tables in bibliotques |
+-----+
| livres                |
+-----+
```

La commande **DESCRIBE livres** permet de retrouver la définition de la table:

```
mysql> describe livres;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| livre_id   | int(11)       | YES  |     | NULL    |       |
| titre      | varchar(50)   | YES  |     | NULL    |       |
| auteurs    | varchar(50)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

Les tables, et les données

- Ajouter des données:
 - `INSERT INTO LIVRES VALUES (1, 'Mon Livre', 'Moi')`
 - `INSERT INTO LIVRES(livre_id, titre , auteurs) VALUES (2,'Le livre Sans Nom', Anonyme)`
 - `INSERT INTO LIVRES(livre_id,titre) VALUES (3,'Le livre de la Mort')`

```
mysql> select * from livres;
```

```
+-----+-----+-----+
| livre_id | titre                | auteurs |
+-----+-----+-----+
|          1 | Mon Livre            | Moi     |
|          2 | Le livre Sans Nom   | Anonyme |
|          3 | Le livre de la Mort | NULL    |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

Ssssssss
Sdqsdqd
Qsdqdqs
Wfcxwxcxw
Qsdqdqdqs
Azedzedzaeda

Les tables, et les données

- Sélectionner toutes les données
 - `Select * from <nom de table>`
- Sélectionner des lignes particulières:
 - `Select * from <nom de table> where condition`
- Sélectionner des colonnes particulières
 - `Select <nom de colonne> from <nom de table> where condition`

```
Select * from livres where titre='Mon Livre'
```


Les tables, et les données

- **Trier les enregistrements**
 - `Select <nom de colonne> from <table_name> ORDER BY nom de colonne`
- **Limitation des enregistrements:**
 - `Select <nom de colonne> from <table_name> ORDER BY nom de colonne LIMIT X`
- **La valeur maximale d'une colonne**
 - `SELECT MAX(nom de colonne) from <table name>`
- **Plus complexe faire des alias:**
 - `SELECT <nom de colonne> from <table_name> AS T`

Des exemples

1. CREATION DE LA TABLE DES CAPITAINES

```
create table capitaine  
(nom VARCHAR(50),age INT);
```

```
insert into capitaine values ('rackam',30),('bonny',45),('read',32),('morgan',23);
```

2. SELECTION SIMPLE

```
mysql> select * from capitaine;
```

```
+-----+-----+  
| nom   | age |  
+-----+-----+  
| rackam | 30 |  
| bonny  | 45 |  
| read   | 32 |  
| morgan | 23 |  
+-----+-----+  
4 rows in set (0.00 sec)
```

3. SELECTION DES NOMS

```
mysql> select nom from capitaine;
```

```
+-----+  
| nom   |  
+-----+  
| rackam |  
| bonny  |  
| read   |  
| morgan |  
+-----+  
4 rows in set (0.00 sec)
```

Des exemples

1. SELECTION AVEC UNE CONDITION

```
mysql> select nom from capitaine where age>30;
+-----+
| nom   |
+-----+
| bonny |
| read  |
+-----+
2 rows in set (0.00 sec)
```

2. SELECTION AVEC UNE CONDITION COMPLEXE

```
mysql> select nom from capitaine where age>30 AND age<40;
+-----+
| nom   |
+-----+
| read  |
+-----+
1 row in set (0.00 sec)
```

3. SELECTION AVEC ORDRE ET LIMIT

```
mysql> select nom,age from capitaine order by age LIMIT 2;
+-----+-----+
| nom   | age |
+-----+-----+
| morgan | 23  |
| rackam | 30  |
+-----+-----+
2 rows in set (0.00 sec)
```

Les tables, et les données

- Suppression d'une ligne:
 - **DELETE FROM** <TABLE> **WHERE** <CONDITIONS>
- Mise a jour d'une ligne:
 - **UPDATE** <TABLE> **SET** COL1 = Nouvelle Valeur, ... COLX = Nouvelle Valeur **WHERE** <CONDITION>
- Exemple:

```
mysql> delete from livres01 where livre_id=1;
Query OK, 1 row affected (0.00 sec)
mysql> update livres01 set titre='le cimetiere du
diable' where livre_id=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Des exemples

1. SUPPRESSION DU CAPITAINE BONNY

```
mysql> delete from capitaine where nom='bonny';  
Query OK, 1 row affected (0.03 sec)
```

```
mysql> select * from capitaine;
```

```
+-----+-----+  
| nom   | age |  
+-----+-----+  
| rackam | 30 |  
| read  | 32 |  
| morgan | 23 |  
+-----+-----+  
3 rows in set (0.00 sec)
```

2. MISE A JOUR DU NOM DU CAPITAINE REED

```
mysql> update capitaine set nom='Read' where nom='read';  
Query OK, 1 row affected (0.00 sec)  
Rows matched: 1 Changed: 1 Warnings: 0
```

```
mysql> select * from capitaine;
```

```
+-----+-----+  
| nom   | age |  
+-----+-----+  
| rackam | 30 |  
| Read  | 32 |  
| morgan | 23 |  
+-----+-----+  
3 rows in set (0.00 sec)
```

Mémento SQL, Obtention des données

- `SELECT <liste des noms de colonnes> FROM <liste des noms de tables>`
- `SELECT * FROM tab1` (toutes les colonnes)
- `SELECT col1,col3 FROM tab1` (une partie des colonnes)
- `SELECT DISTINCT col1 FROM tab1` (élimine les doublons)
- `SELECT nom AS "nom personne" FROM tab1` (renommage des colonnes)

Mémento SQL, les restrictions

- `SELECT * FROM tab1 WHERE <condition(s)>`
- `SELECT * FROM tab1 WHERE ville IN ('Brest ','Rennes ','Paris ')`
- `SELECT * FROM tab1 WHERE age NOT BETWEEN 15 AND 20`
- `SELECT * FROM tab1 WHERE travail IS NULL`
- `SELECT * FROM tab1 WHERE travail IS NOT NULL`

Mémento SQL, Tri et présentation

- `SELECT * FROM tab1 ORDER BY col1` (tri ascendant par défaut)
- `SELECT * FROM tab1 ORDER BY col5,col7` (tri par col5 puis tri par col7)
- `SELECT * FROM tab1 ORDER BY age ASC, sexe DESC` (tri ascendant ou descendant)

Mémento SQL, Expression des jointures

- `SELECT * FROM tab1,tab2` (jointure sans qualification = produit cartésien)
- `SELECT * FROM tab1,tab2 WHERE tab1.col1=tab2.col2` (jointure avec égalité = équijointure)
- `SELECT * FROM tab1 t1,tab2 t2,tab3 t3 WHERE t1.col1=t2.col2 AND t2.col2=t3.col3`
(jointures en cascades)

Memento SQL, Jointures

- La jointure simple : c'est la jointure avec le '=' ou 'INNER JOIN'. Il s'agit d'une intersection simple
- Il est possible de préciser LEFT ou RIGHT JOIN afin de préserver une des tables de la jointures
- Le FULL JOIN s'émule avec une UNION de LEFT et RIGHT JOIN

Mémento SQL, Les fonctions statistiques

- AVG (moyenne)
- COUNT (nombre d'éléments)
- MAX (maximum)
- MIN (minimum)
- SUM (somme)
- `SELECT COUNT(*)FROM tab1`
- `SELECT SUM(col1) FROM tab2`

Mémento SQL, Sous-requêtes SQL

- `SELECT * FROM tab1 WHERE prix > (SELECT MIN(prix) FROM tab2)`
- `SELECT * FROM tab1 WHERE nom NOT IN (SELECT nom FROM tab2)`
- `SELECT * FROM tab1 WHERE prix > ALL (SELECT prix FROM tab2)`
(sup. à ttes les valeurs)
- `SELECT * FROM tab1 WHERE prix > ANY (SELECT prix FROM tab2)`
(sup. à au moins 1)



Pratique du SQL avec MySQL

Les types de données

- Contenu
 - Les types de données de MySQL sont de type entier, chaîne de caractères, texte
- Objectif
 - Il est impossible de connaître par cœur tous les types de données !, par contre il faut savoir qu'ils existent pour les retrouver.



Les types de données: Entier

- **TINYINT [UNSIGNED]**
Occupe 1 octet. Ce type peut stocker des nombres entiers de -128 à 127 si il ne porte pas l'attribut UNSIGNED, dans le cas contraire il peut stocker des entiers de 0 à 255.
- **SMALLINT [UNSIGNED]**
Occupe 2 octets. Ce type de données peut stocker des nombres entiers de -32 768 à 32 767 si il ne porte pas l'attribut UNSIGNED, dans le cas contraire il peut stocker des entiers de 0 à 65 535.
- **MEDIUMINT [UNSIGNED]**
Occupe 3 octets. Ce type de données peut stocker des nombres entiers de -8 388 608 à 8 388 607 si il ne porte pas l'attribut UNSIGNED, dans le cas contraire il peut stocker des entiers de 0 à 16 777 215.
- **INT [UNSIGNED]**
Occupe 4 octets. Ce type de données peut stocker des nombres entiers de -2 147 483 648 à 2 147 483 647 si il ne porte pas l'attribut UNSIGNED, dans le cas contraire il peut stocker des entiers de 0 à 4 294 967 295.
- **INTEGER [UNSIGNED]**
Même chose que le type INT.
- **BIGINT [UNSIGNED]**
Occupe 8 octets. Ce type de données stocke les nombres entiers allant de -9 223 372 036 854 775 808 à 9 223 372 036 854 775 807 sans l'attribut UNSIGNED, et de 0 à 18 446 744 073 709 551 615.

Les types de données: Flottant

- **FLOAT** (précision simple de 0 à 24 et précision double de 25 à 53) [UNSIGNED]
Occupe 4 octets si la précision est inférieure à 24 ou 8 au delà.
Stocke un nombre de type flottant.
- **LOAT[[M,D]]** [UNSIGNED]
Occupe 4 octets. M est le nombre de chiffres et D est le nombre de décimales.
Ce type de données permet de stocker des nombres flottants à précision simple. Va de $-1.175494351E-38$ à $3.402823466E+38$. Si UNSIGNED est activé, les nombres négatifs sont retirés mais ne permettent pas d'avoir des nombres positifs plus grands.
- **DOUBLE PRECISION[[M,D]]**
Occupe 8 octets. Même chose que le type DOUBLE
- **DOUBLE [[M,D]]**
Occupe 8 octets. Stocke des nombres flottants à double précision de $-1.7976931348623157E+308$ à $-2.2250738585072014E-308$, 0, et de $2.2250738585072014E-308$ à $1.7976931348623157E+308$.
Si UNSIGNED est activé, les nombres négatifs sont retirés mais ne permettent pas d'avoir des nombres positifs plus grands.
- **REAL[[M,D]]**
Occupe 8 octets. Même chose que le type DOUBLE
- **DECIMAL[[M,D]]**
Occupe $M+2$ octets si $D > 0$, $M+1$ octets si $D = 0$
Contient des nombres flottants stockés comme des chaînes de caractères.
- **NUMERIC [[M,D]]**
Même chose que le type DECIMAL

Les types de données: Les dates

- **DATE**
Occupe 3 octets. Stocke une date au format 'AAAA-MM-JJ' allant de '1000-01-01' à '9999-12-31'
- **DATETIME**
Occupe 8 octets. Stocke une date et une heure au format 'AAAA-MM-JJ HH:MM:SS' allant de '1000-01-01 00:00:00' à '9999-12-31 23:59:59'
- **TIMESTAMP**
Occupe 4 octets. Stocke une date sous forme numérique allant de '1970-01-01 00:00:00' à l'année 2037. L'affichage dépend des valeurs de M : AAAAMMJJHHMMSS, AAMMJJHHMMSS, AAAAMMJJ, ou AAMMJJ pour M égal respectivement à 14, 12, 8, et 6
- **TIME**
Occupe 3 octets. Stocke l'heure au format 'HH:MM:SS', allant de '-838:59:59' à '838:59:59'
- **YEAR**
Occupe 1 octet. Année à 2 ou 4 chiffres allant de 1901 à 2155 (4 chiffres) et de 1970-2069 (2 chiffres).

Les types de données: Les caractères

- **CHAR (M)**

Occupe M octets. Stocke des caractères. Si vous stockez un caractère et que M vaut 255, la donnée prendra 255 octets. Autant donc employer ce type de données pour des mots de longueur identique.

- **VARCHAR (M)**

Occupe L+1 octets (ou L représente la longueur de la chaîne).
Ce type de données stocke des chaînes de 255 caractères maximum.

Les types de données: Les caractères

- **CHAR (M)**
Occupe M octets. Stocke des caractères. Si vous stockez un caractère et que M vaut 255, la donnée prendra 255 octets. Autant donc employer ce type de données pour des mots de longueur identique.
- **VARCHAR (M)**
Occupe L+1 octets (ou L représente la longueur de la chaîne).
Ce type de données stocke des chaînes de 255 caractères maximum.

Les types de données: Les BLOB

- **TINYBLOB** (L représente la longueur de la chaîne)
Occupe L+1 octets., Stocke des chaînes de 255 caractères maximum. Ce champ est sensible à la casse.
- **TINYTEXT**
Occupe L+1 octets. Stocke des chaînes de 255 caractères maximum. Ce champ est insensible à la casse.
- **BLOB**
Occupe L+1 octets. Stocke des Chaînes de 65535 caractères maximum. Ce champ est sensible à la casse.
- **TEXT**
Occupe L+2 octets. Stocke des chaînes de 65535 caractères maximum. Ce champ est insensible à la casse.
- **MEDIUMBLOB**
Occupe L+3 octets. Stocke des chaînes de 16777215 caractères maximum.
- **MEDIUMTEXT**
Occupe L+3 octets. Chaîne de 16 777 215 caractères maximum. Ce champ est insensible à la casse.
- **LOBLOB**
Occupe L+4 octets. Stocke des chaînes de 4 294 967 295 caractères maximum. Ce champ est sensible à la casse.
- **LONGTEXT**
Occupe L+4 octets. Stocke des chaînes de 4 294 967 295 caractères maximum.

Les types de données: Les enums

- `ENUM('valeur_possible1','valeur_possible2','valeur_possible3',...)`
Occupe 1 ou 2 octets (la place occupée est fonction du nombre de solutions possibles : 65 535 valeurs maximum).
- `SET('valeur_possible1','valeur_possible2',...)`
Occupe 1, 2, 3, 4 ou 8 octets, selon de nombre de solutions possibles (de 0 à 64 valeurs maximum)



Pratique du SQL avec MySQL

Création de table avancé

- Contenu
 - Création de table temporaire
 - Création de table par copie
 - Création de valeur par défaut
- Objectif
 - Le langage SQL de MySQL est **très** riche, après avoir fait des créations de tables simple il sera envisagé de faire des créations complexes



Création de table avancée

- Création d'une table par définition:
 - `CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [(create_definition,...)]`
- Création d'un table par copie:
 - `CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name [() LIKE old_tbl_name ()];`

Création de table avancée

- TEMPORARY: la création de la table se limite à la durée de la connexion.
 - Plusieurs connexions peuvent avoir la même table.
 - Utilisé pour la mise en place de « Cache »
- IF NOT EXISTS: Autorise la création de la table si une définition n'existe pas.
 - Utilisé dans la mise en place de script de création de base
- LIKE : Création par copie:

```
Create table livre2 LIKE livres  
mysql> describe livre2;
```

Field	Type	Null	Key	Default	Extra
livre_id	int(11)	NO	PRI	NULL	auto_increment
titre	varchar(50)	YES		NULL	
auteurs	int(11)	YES	MUL	NULL	
description	text	YES		NULL	

4 rows in set (0.00 sec) 3 rows in set (0.00 sec)

Création de table avancée

- CREATE TABLE <Nom de TABLE>

```
(col_name type [NOT NULL | NULL] [DEFAULT default_value]  
[AUTO_INCREMENT] [[PRIMARY] KEY] [COMMENT 'string']..  
)
```

- NOT NULL/NULL: Autorise ou pas la création d'une valeur vide
- DEFAULT: Lors de l'insertion d'une donnée, c'est une valeur par défaut
- PRIMARY KEY: Ce champ sera unique dans la table
- COMMENT: C'est un commentaire

Création de table avancée

- Exemple:

```
CREATE TABLE emprunteurs
(
  id_emprunteur INT PRIMARY KEY,
  Nom VARCHAR(50) NOT NULL,
  Prenom VARCHAR(50) NOT NULL,
  Age INT NULL,
  Date_emprunt TIMESTAMP NOT NULL DEFAULT NOW()
);
```



Pratique du SQL avec MySQL

Gestion des vues.

- Contenu:
 - Définition des vues
 - Modification d'une vue
- Objectif:
 - Comment modéliser un aspect métier via une vue
 - Créer des vues.

Les vues

Les vues sont des tables virtuelles issues de l'assemblage d'autres tables en fonction de critères. Techniquement les vues sont créées à l'aide d'une requête *SELECT*.

Elles ne stockent pas les données qu'elles contiennent mais conservent juste la requête permettant de les créer.

Les vues

- La requête *SELECT* qui génère la vue référence une ou plusieurs tables. La vue peut donc être, par exemple:
 - une jointure entre différentes tables
 - l'agrégation ou l'extraction de certaines colonnes d'une table
 - créée à partir d'une autre vue.

Les vues

- Contrôler l'intégrité en restreignant l'accès aux données pour améliorer la confidentialité.
 - Partitionnement vertical et/ou horizontal pour cacher des champs aux utilisateurs, ce qui permet de personnaliser l'affichage des informations suivant le type d'utilisateur.
- Masquer la complexité du schéma.
 - Indépendance logique des données, utile pour donner aux utilisateurs l'accès à un ensemble de relations représentées sous la forme d'une table. Les données de la vue sont alors des champs de différentes tables regroupées, ou des résultats d'opérations sur ces champs.
- Modifier automatiquement des données sélectionnées (*sum()*, *avg()*, *max()*,...).
 - Manipuler des valeurs calculées à partir d'autres valeurs du schéma.
- Conserver la structure d'une table si elle doit être modifiée.
 - Le schéma peut ainsi être modifié sans qu'il ne soit nécessaire de changer les requêtes du côté applicatif.

Les vues

- Colonne calculée
 - Une table ne devrait pas avoir de colonne calculée. Les vues servent à cela.
- Compatibilité
 - Lors d'une mise à jour du schéma, il est possible via des vues de « masquer » ces modifications.

Création d'une vue

- `CREATE [ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}]VIEW [Nom de la vue AS[SELECT requete]`
- Requête: Il s'agit de la requête permettant de "remplir" la vue
- 3 algorithmes pour créer une vue MERGE, TEMPTABLE, UNDEFINED

Création d'une vue

- La requête associée ne doit pas contenir de sous select
- Les tables temporaires ne peuvent participer à une vue
- Une vue ne peut être associée à des triggers

Exemple

- Il s'agit de la création d'une vue des produits plus chers que la moyenne:

```
CREATE VIEW vwProducts
AS
SELECT productCode,      productName,      buyPrice FROM products
WHERE
buyPrice > (
    SELECT AVG (buyPrice)      FROM  products)ORDER BY buyPrice DESC
)
```

Comment ça marche ?

- En interne, les systèmes se font souvent par réécriture de requêtes

- Définition :

```
CREATE VIEW vwProducts AS SELECT productCode,      productName,  
buyPrice FROM products WHERE buyPrice > ( SELECT AVG (buyPrice)  
FROM products)ORDER BY buyPrice DESC )
```

- Requête:

```
Select productCode from vwProducts
```

- En interne:

```
Select TMP.productCode from  
(SELECT productCode, productName,      buyPrice FROM products AS TMP  
WHERE  
buyPrice > (  
    SELECT AVG (buyPrice)      FROM products)ORDER BY buyPrice DESC  
)
```

MERGE/TEMPTABLE/UNDEFINED

- Il s'agit des stratégies de MySQL pour utiliser une requête au regard de la requête de la vue:
- MERGE: Uniquement applicable si une ligne de la vue correspondent à une ligne dans la table sous jacente. Ces types de vues sont « updatable » (Il s'agit d'une vue par réécriture).
- TEMPTABLE: MySQL va construire une table temporaire avec les données des tables sous jacentes. Ces types de vue ne sont pas updatable. (Dans ce cas, une table est créée avec CREATE TABLE XXX AS ...)
- UNDEFINES: MySQL choisie la stratégie.

Mise à jour des vues

En général pas possible sauf :

- La requête qui génère la vue doit permettre à MySQL de retrouver la trace de l'enregistrement à modifier dans la ou les tables sous-jacentes ainsi que celle de toutes les valeurs de chaque colonne. La requête *SELECT* créant la vue ne doit donc pas contenir de clause *DISTINCT*, *GROUP BY*, *HAVING...* et autres fonctions d'agrégation.
- En pratique , une vue peut rarement être mise à jour

Mise à jour des vues

Les contraintes sont donc :

- La vue ne doit contenir qu'une table
- La requête ne doit pas contenir de GROUP BY, de HAVING, d'agrégats, de colonnes calculées..
- La vue ne doit pas contenir de vues qui ne soient pas updatable.

Modification d'une vue

- Destruction d'une vue:
 - `DROP VIEW [IF EXISTS] [nom de la vue]`
- Modification d'une vue:
 - `ALTER [ALGORITHM = {MERGE | TEMPTABLE | UNDEFINED}] VIEW [nom de la vue] AS [SELECT statement]`

Vue matérialisée

- Il s'agit d'une table se comportant comme une vue
- Lorsque les données changent dans les tables sous jacentes, alors la table « vue » doit changer
- N'existe pas en MySQL

Travaux sur les vues matérialisées

- PostgreSQL Materialized Views, Jonathan Gardner,
http://tech.jonathangardner.net/wiki/PostgreSQL/Materialized_Views
- Materialized Views that Really Work by Dan Chak,
<http://www.pgcon.org/2008/schedule/events/69.en.html>
- Flexview, <http://code.google.com/p/flexviews/>

Algorithme de base

1. créer une « vraie » vue sur les données sous jacentes
2. créer une table effectuant la liaison entre la « vraie » vue et la vue matérialisée
3. créer la vue matérialisée
4. gérer le « rafraichissement » de la vue matérialisée.

Algorithme de base : outil

- Il est utile de pouvoir créer des procédures stockées permettant de gérer cet algorithme.
- Ces procédures en général doivent pouvoir exécuter du SQL.

Execute, les vues et MySQL

- Algorithme de la photo:

```
CREATE TABLE matviews (  
  mv_name VARCHAR NOT NULL PRIMARY KEY - nom de la vue materialisé  
  , v_name VARCHAR NOT NULL - nom de la vue  
  , last_refresh TIMESTAMP WITH TIME ZONE -date du dernier "refresh"  
);
```

Création de la vue materialisé via une procedure stocké:

```
CREATE PROCEDURE createView(in nom_de_vue VARCHAR(50))  
Creation d'une table "mat_"nom_de_vue via la commande  
Insertion du nom de la vue dans la table matviews  
Gestion des log
```



Pratique du SQL avec MySQL

Les requêtes préparés

- Contenu:
 - Syntaxe d'une requête préparés
- Objectif:
 - Voir la syntaxe
 - Nous reverrons les requêtes préparées dans la partie programmation

Execute

- PREPARE: prépare une requête à être exécutée
- EXECUTE: exécute la requête « préparée »
- DEALLOCATE PREPARE: libère la requête

- Commande :
 - `PREPARE stmt_name FROM preparable_stmt;`
 - `EXECUTE stmt_name [USING @var_name [, @var_name] ...];`
 - `DEALLOCATE PREPARE stmt_name;`

Execute

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+-----+
| hypotenuse |
+-----+
|      10      |
+-----+
mysql> DEALLOCATE PREPARE stmt2;
```

Modèle relationnel, conception et création d'une base

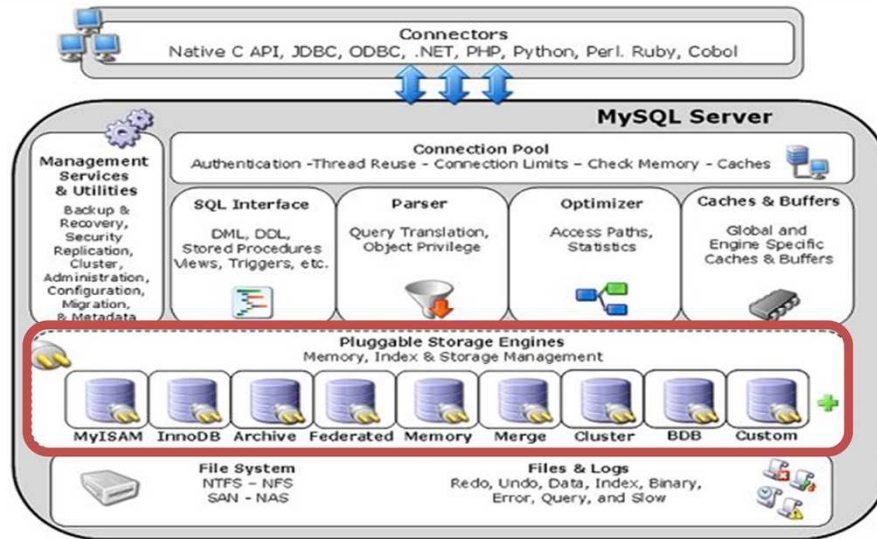
- Introduction et rappels
- Pratique du SQL avec MySQL
- **Modèle relationnel, conception et création d'une base**
 - Types de tables (MYISAM, MEMORY, MERGE, InnoDB...).
 - Eléments de conception d'un schéma de base de données.
 - Contraintes d'intégrité.
 - Fonctions intégrées de MySQL.
 - Création de bases et de tables.
 - La base INFORMATION_SCHEMA.
 - Jeux de caractères, internationalisation.
- Tables transactionnelles InnoDB
- SQL procédural
- Connexions, droits d'accès, sécurité
- Introduction à l'administration



Type de table dans MySQL

- Contenu
 - Descriptif succinct des types de tables dans MySQL
- Objectif
 - Connaître les fonctionnalités de chacune des types de tables.

Types de tables dans MySQL



Pluggable Storage Engine

- C'est une fonctionnalité majeure de MySQL permettant de supporter plusieurs moteurs de stockage différents
- Les principaux :
 - ISAM
 - MyISAM
 - InnoDB *
 - BerkeleyDB
 - MERGE
 - HEAP

Pluggable Storage Engine

- ISAM
 - Ancien moteur de MySQL supprimé depuis la 5.X
 - Limité à 4Gb
 - Remplacé par MyISAM
- MyISAM
 - Rapide mais sans transaction
 - Limité à 64 clefs par table
- InnoDB
 - Moteur transactionnel complet
 - Index passé sur les BTree

Plugable Storage Engine

- BDB:
 - Wrapper autour du moteur de Berkeley
 - Moteur Transactionnel
- Merge:
 - Moteur capable d'agréger des tables MyISAM
- Heap:
 - Moteur entièrement en mémoire
 - Ne supporte pas ni les BLOB, ni les « auto increment »
 - C'est le moteur le plus rapide Mais ne supporte pas les coupures de courant



Modèle relationnel, conception et création d'une base

Les contraintes d'intégrité

- Contenu:
 - Voir une contrainte d'intégrité
 - Voir les primary key, foreign key
 - Voir les NULL
- Objectifs:
 - Se rappeler les bases de la modélisation relationnel
 - Comprendre ce qu'implique la pose d'une contrainte ou pas

Les contraintes d'intégrité

- PRIMARY KEY: Identifiant unique non nul d'une ligne
- UNIQUE : Identifiant unique d'une ligne
- FOREIGN KEY: les valeurs d'une colonne d'une table T existent dans une autre colonne d'une table T2. C'est la notion de relation.
- NULL: la colonne peut contenir des valeurs nulles
- NOT NULL: la colonne doit contenir une valeur non nulle

PRIMARY/UNIQUE KEY

- Les clefs « unique » identifient une ligne d'une table.
 - Elles identifient les lignes existantes
 - Et les lignes à venir
- Les clefs primaires sont des clefs uniques n'autorisant pas la valeur NULL
- Il n'existe qu'une clef primaire par table
- Les clefs uniques simples peuvent être multiples et contenir des valeurs NULL.

PRIMARY/UNIQUE KEY

- Sur une table existante
 - ALTER TABLE <TABLE identifier> ADD [CONSTRAINT <CONSTRAINT identifier>] PRIMARY KEY (<COLUMN expression> {, <COLUMN expression>}...)
 - ALTER TABLE <TABLE identifier> ADD [CONSTRAINT <CONSTRAINT identifier>] UNIQUE (<COLUMN expression> {, <COLUMN expression>}...)
- A la création d'une table
 - CREATE TABLE TABLE_NAME (id_col INT PRIMARY KEY, col2 CHARACTER VARYING(20), ... key_col SMALLINT UNIQUE, ...)



Les clefs uniques

1. LES CLEFS UNIQUES

Dans une base de données relationnelle, une clé unique peut identifier de manière unique chaque ligne de données d'une table.

Elle comprend une ou plusieurs colonnes, et implique que deux lignes d'une table ne peuvent pas être identiques.

Une clé primaire est une clef unique ayant une contrainte NOT NULL. Autrement dit, les valeurs des colonnes d'une clé primaire ne peuvent contenir la valeur NULL.

Selon sa conception, une table de base de données peut avoir plusieurs indices clés uniques, mais au plus une clé primaire.

Une clé unique doit identifier de façon unique toutes les lignes possibles qui existent dans une table et non pas seulement les lignes qui existent actuellement]. Des exemples de clés uniques sont le numéro de sécurité sociale (associé à une personne en particulier ou un numéro ISBN (associé à un livre spécifique).

Les clefs uniques sont les références des clefs étrangères (Foreign keys).

2. DEFINITION D'UNE CLEF UNIQUE

1. SUR UNE TABLE EXISTANTE

```
ALTER TABLE <TABLE identifieur> ADD [ CONSTRAINT <CONSTRAINT identifieur> ] PRIMARY KEY ( <COLUMN expression> {, <COLUMN expression>}... )
```

```
ALTER TABLE <TABLE identifieur> ADD [ CONSTRAINT <CONSTRAINT identifieur> ] UNIQUE ( <COLUMN expression> {, <COLUMN expression>}... )
```

2. A LA CRÉATION D'UNE TABLE

```
CREATE TABLE TABLE_NAME ( id_col INT PRIMARY KEY, col2 CHARACTER VARYING(20), ... key_col SMALLINT UNIQUE, ... )
```

PRIMARY KEY/AUTO INCREMENT

- Il s'agit de la capacité de signaler à MySQL qu'une clef primaire est :
 - Un entier
 - Chaque ligne obtient automatiquement une valeur donnée par MySQL
- Syntaxe
 - `CREATE TABLE <table name> (nom colonne type entier PRIMARY KEY AUTO_INCREMENT)`

Exemples

```
CREATE TABLE `livres` (  
  livre_id int(11) NOT NULL AUTO_INCREMENT,  
  titre varchar(50) DEFAULT NULL,  
  auteurs varchar(50) DEFAULT NULL,  
  description text,  
  PRIMARY KEY (livre_id)  
)
```

Exemples

```
insert into livres(titre,auteurs,description) values('un  
titre','moi','mon livre')
```

```
Select * from livres
```

livre_id	titre	auteurs	description
1	un titre	moi	mon livre



AUTO_INCREMENT

Si vous spécifiez une colonne AUTO_INCREMENT dans une table, la table InnoDB va ajouter dans le dictionnaire de données un compteur spécial appelé le compteur auto-incrément, qui est utilisé pour assigner les nouvelles valeurs de la colonne. Le compteur est stocké uniquement en mémoire, et non pas sur le disque.

La requête exécutée par MySQL est :

```
SELECT MAX(ai_col) FROM T
```

La valeur lue par la commande est incrémentée d'une unité, et assignée à la colonne auto-incrément et au compteur de table. Si la table est vide, la valeur de 1 est assignée.

Après l'initialisation du compteur d'auto-incrémentation, si un utilisateur insère une ligne qui définit explicitement la valeur de la colonne, et que cette valeur est plus grande que la valeur courante du compteur, le compteur prend alors cette valeur. Si l'utilisateur ne spécifie pas de valeur, MySQL incrémente le compteur d'une unité, et assigne une nouvelle valeur à cette colonne.

Le comportement du mécanisme auto-increment n'est pas défini si vous assignez une valeur négative à la colonne, ou si cette dernière dépasse la capacité de la colonne.

FOREIGN KEY

- Il s'agit d'une contrainte référentielle entre deux tables.
- Il s'agit du champ d'une table qui a une référence sur une clef unique d'une autre table

- Il s'agit de l'idée d'associer deux tables, par exemple une table de clients et une table de commandes.
 - Chaque commande est faite par un client
 - Chaque client ne fait pas forcément une commande
- Il y a donc une référence des commandes vers les clients.

FOREIGN KEY

- Syntaxe:
 - CREATE TABLE <nom de table> (id INTEGER PRIMARY KEY, col2 VARCHAR(20), col3 INTEGER, ... FOREIGN KEY(col3) REFERENCES <nom de table> (key_col))
 - ALTER TABLE <nom de table> ADD FOREIGN KEY (<nom de colonne> REFERENCES <nom de table>
- Exemple:

```
CREATE TABLE auteurs (  
  auteur_id int(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,  
  prenom varchar(50) DEFAULT NULL,  
  nom varchar(50) DEFAULT NULL,  
  country varchar(50) DEFAULT NULL  
)
```

FOREIGN KEY

auteurs	livres
auteur_id: int	livre_id: int
premiere: varchar(50)	titre: varchar(50)
nom: varchar(50)	auteurs: varchar(50)
country: varchar(50)	description: text
PRIMARY	PRIMARY

- Volonté de mettre en relation nos livres avec nos auteurs:
 - **ALTER TABLE livres CHANGE auteurs auteurs INT**
 - **ALTER TABLE livres ADD CONSTRAINT `relation_auteurs` FOREIGN KEY (`auteurs`) REFERENCES `auteurs`(`auteur_id`);**

FOREIGN KEY

- Maintenant la table livres est « protégée » si les auteurs n'existent pas.

```
mysql> insert into
livres(titre, auteurs, description) values ('un
titre', 1, 'livre');

ERROR 1452 (23000): Cannot add or update a child
row: a foreign key constraint fails
(`bibliotheque`.`livres`, CONSTRAINT
`livres_ibfk_1` FOREIGN KEY (`auteurs`) REFERENCES
`auteurs` (`auteur_id`))
```

Exemple

Nous insérons l'auteur , et MySQL accepte maintenant notre insertion.

```
mysql> insert into auteurs(prenom,nom,country)
values('moi','toi','france');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select * from auteurs;
+-----+-----+-----+-----+
| auteur_id | prenom | nom  | country |
+-----+-----+-----+-----+
|          1 | moi   | toi  | france  |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> insert into livres
(titre,auteurs,description) values ('mon
titre',1,'livre');
Query OK, 1 row affected (0.00 sec)
```



Modèle relationnel, conception et création d'une base

Notion de base de données MySQL

- Contenu
 - Description des objets théoriques des bases de données
- Objectifs
 - Toucher du doigt la modélisation E/R
 - Eviter les pièges grossiers d'une base MySQL.

Notion de base de données MySQL

- L'idée générale est d'avoir des « trucs » pour créer un schéma « résistant » au temps
- Quelques objets théoriques aident à modéliser ce schéma (en fait à éviter les erreurs)
- Le but est de séparer la modélisation des données de l'utilisation métier

Objet théorique

- **1FN - première forme normale:**
 - tout attribut contient une valeur atomique
 - tous les attributs sont non répétitifs
 - tous les attributs sont constants dans le temps.

Le non-respect des deux premières conditions de la 1FN rend la recherche parmi les données plus lente parce qu'il faut analyser le contenu des attributs. La troisième condition quant à elle évite qu'on doive régulièrement mettre à jour les données.

1FN en pratique

Au lieu de :

Produit	Fournisseur
Téléviseur	VIDEO SA, HITEK LTD

Faire:

Produit	Fournisseur
Téléviseur	VIDEO SA
Téléviseur	HITEK LTD

1FN en pratique: une valeur atomique

- *L'attribut NOM est composé de 2 attributs atomiques.*
- Au lieu de :

CLIENT_ID	NOM
1	Gérard Dupont
2	Léon Durand

- Faire:

CLIENT_ID	NOM	PRENOM
1	Dupont	Gérard
2	Durand	Léon

1FN pratique: les attributs sont non répétitifs

- L'attribut *FOURNISSEURS* est une liste.

- Au lieu de

PRODUIT_ID	DESCRIPTION	FOURNISSEURS
1	Téléviseur	Sony, Sharp, LG

- Faire

Produit_id	Fournisseur_id
1	Sony
1	Sharp
1	LG

PRODUIT_ID	DESCRIPTION
1	Téléviseur

1FN en pratique: Constance des attributs

- *L'attribut AGE n'est pas constant dans le temps.*
- *Au lieu de :*

NOM	PRENOM	AGE
BOB	Eponge	35

- *Faire:*

NOM	PRENOM	DATE DE NAISSANCE
BOB	Eponge	1977

Notion de base de données MySQL: Objet théorique

- 2FN - deuxième forme normale
 - tous les attributs non-clés sont totalement dépendants fonctionnellement de la totalité de la clé primaire.

Le non-respect de la 2FN entraîne une redondance des données qui encombre alors inutilement la mémoire et l'espace disque.

2FN en pratique

- Au lieu de :

Produit	Fournisseur	Adresse fournisseur
téléviseur	VIDEO SA	13 rue du cherche-midi
écran plat	VIDEO SA	13 rue du cherche-midi
téléviseur	HITEK LTD	25 Bond Street

2FN en pratique

- Faire:

Produit	Fournisseur
téléviseur	VIDEO SA
écran plat	VIDEO SA
téléviseur	HITEK LTD

Fournisseur	Adresse
VIDEO SA	13 rue du cherche-midi
HITEK LTD	25 Bond Street

2FN en pratique

- L'attribut *DESCRIPTION_ARTICLE* ne dépend que d'une partie de la clef primaire.
- Au lieu de :

COMMANDE_ID	ARTICLE_ID*	DESCRIPTION ARTICLE
1	15	TV haute définition avec amplificateur dolby 5.1

- Faire

COMMANDE_ID	ARTICLE_ID*
1	15

ARTICLE_ID*	DESCRIPTION ARTICLE
15	TV haute définition avec amplificateur dolby 5.1

2FN en pratique

- L'attribut *DESCRIPTION_ARTICLE* ne dépend que d'une partie de la clef primaire.
- Au lieu de :

COMMANDE_ID	ARTICLE_ID*	DESCRIPTION ARTICLE
1	15	TV haute définition avec amplificateur dolby 5.1

- Faire

COMMANDE_ID	ARTICLE_ID*
1	15

ARTICLE_ID*	DESCRIPTION ARTICLE
15	TV haute définition avec amplificateur dolby 5.1

Objet théorique

- 3FN - troisième forme normale
 - tout attribut n'appartenant pas à une clé ne dépend pas d'un attribut non clé

Le non-respect de la 3FN peut également entraîner une redondance des données.

3FN en pratique

- L'attribut *NOM_CLIENT* dépend de *CLIENT_ID*.
- Au lieu de:

COMMANDE_ID	CLIENT_ID	NOM_CLIENT
1	1	Durand

- Faire :

COMMANDE_ID	CLIENT_ID	CLIENT_ID	NOM_CLIENT
1	1	1	Durand

Objet théorique

- **FNBC - forme normale de Boyce – Codd**
 - Si une entité ou une relation en troisième forme normale a une clé composée, aucune des propriétés élémentaires de cette clé ne doit être en dépendance fonctionnelle d'une autre propriété.

Le non-respect de la 2FN, 3FN et la FNBC entraîne de la redondance. Une même information étant répétée un nombre considérable de fois.

FNBC en Pratique

- Si Durand arrête d'enseigner les Mathématiques, on supprime la ligne et l'on perd la relation Matière-Salle.

ENSEIGNANT_ID	MATIERE_ID	SALLE_ID
DURAND	MATHS	3A
DUPONT	ANGLAIS	6A

FNBC en Pratique

- *Si Durand arrête d'enseigner les Mathématiques, on supprime la ligne et l'on perd la relation Matière-Salle.*

ENSEIGNANT_ID	MATIERE_ID	SALLE_ID
DURAND	MATHS	3A
DUPONT	ANGLAIS	6A

Objet théorique

- **FNDC - forme normale domaine clef**
 - Une relation est en FNDC si et seulement si toutes les contraintes sont la conséquence logique des contraintes de domaines et des contraintes de clefs qui s'appliquent à la relation.

FNDC En pratique

- On remarque que le type VL (véhicule léger) ou PL (poids lourd) est déterminé par la valeur du PTAC.

- Au lieu de :

CONSTRUCTEUR	MODELE	TYPE	PTAC
Renault	Estafette	VL	2500
Iveco	Eurostar 440	PL	19000
Berliet	GDM 1934	PL	15000
Volkswagen 2 900	combi	VL	2 900

- Faire :

CONSTRUCTEUR	MODELE	PTAC
Renault	Estafette	2500
Iveco	Eurostar 440	19000
Berliet	GDM 1934	15000
Volkswagen 2 900	combi	2 900

Implémentation d'une liste

- Il est souvent utile de pouvoir modéliser une liste d'objets
 - Ex : une liste d'employés
- 4 types d'implémentation:
 - Sac d'objet: les éléments n'ont pas d'ordre précis
 - Liste chaînée : les éléments se suivent (insertion simplifiée)
 - Liste doublement chaînée : les éléments se suivent et se précèdent
 - Implémentation « tableau »

SAC d'objet

- Il s'agit d'une simple table avec une clef pour identifier la liste

ID LISTE	EMPLOYE ID
1	1
1	2
1	3
2	4

EMPLOYEE_ID	NOM
1	X
2	XX
3	YY
4	ZZ

Liste chaînée

- Il s'agit d'une simple table avec une clef pour identifier la liste

ID NOEUD	NEXT_ID	EMPLOYEE ID
1	2	1
2	3	2
3	NULL	3
4	NULL	4

EMPLOYEE_ID	NOM
1	X
2	XX
3	YY
4	ZZ

Liste doublement chaînée

- Il s'agit d'une simple table avec une clef pour identifier la liste

ID NOEUD	NEXT_ID	PREVIOUS_ID	EMPLOYEE ID
1	2	NULL	1
2	3	1	2
3	NULL	2	3
4	NULL	NULL	4

EMPLOYEE_ID	NOM
1	X
2	XX
3	YY
4	ZZ

Liste Tableau

- Il s'agit d'une simple table avec une clef pour identifier la liste

ID LISTE	ORDER	EMPLOYE ID
1	1	1
1	2	2
1	3	3
2	1	4

EMPLOYE_ID	NOM
1	X
2	XX
3	YY
4	ZZ

Implémentation d'un arbre

- Il est souvent utile de pouvoir modéliser un arbre d'objets
 - Ex : la généalogie parents enfants
- 2 types d'implémentation « classique » :
 - Mode Parent Enfant
 - Mode Enfant Parent

Arbre Parent Enfant

- Il s'agit d'une simple table avec une clef pour identifier la liste

ID ARBRE	EMPLOYEE_ID	CHILD_ID
1	1	1
1	2	NULL
1	3	NULL
2	4	NULL

PARENT_ID	CHILD_ID
1	2
1	3

EMPLOYEE_ID	NOM
1	X
2	XX
3	YY
4	ZZ

Arbre Enfant Parent

- Il s'agit d'une simple table avec une clef pour identifier la liste

ID TREE	ID NOEUD	EMPLOYEE_ID	ID PARENT
1	1	1	NULL
1	2	2	1
1	3	3	1
2	4	4	NULL

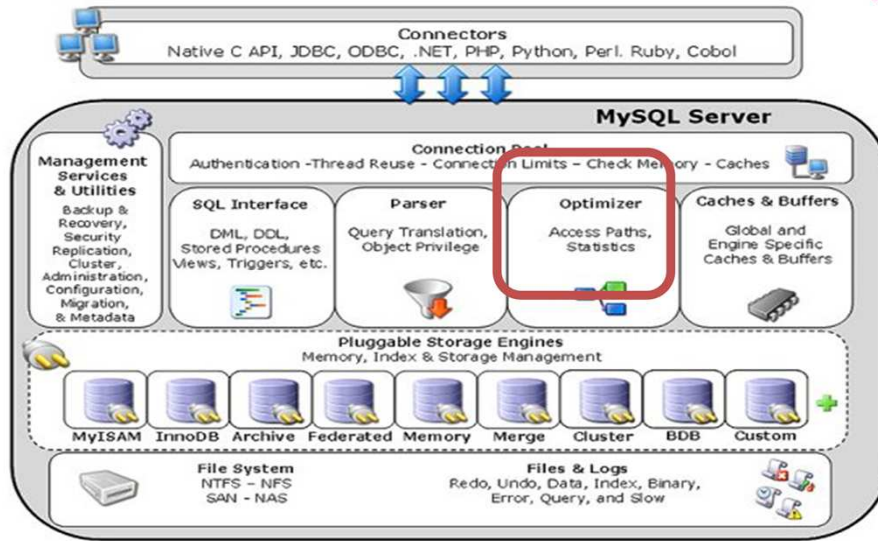
EMPLOYEE_ID	NOM
1	X
2	XX
3	YY
4	ZZ



BTree

- Contenu
 - Une démonstration de l'algorithme des Btree
- Objectifs
 - Comprendre comment marche les index d'une base de données
 - Comment marche un optimiseur de requête
 - Voir un algorithme d'indexation

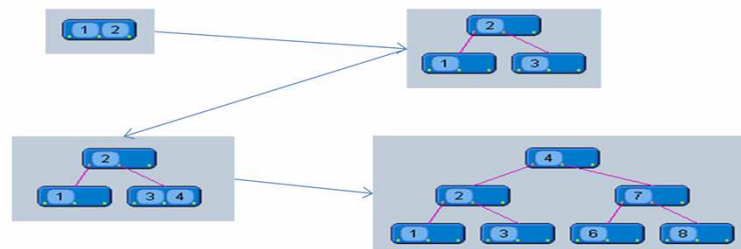
BTree dans MySQL



L'Objet BTree

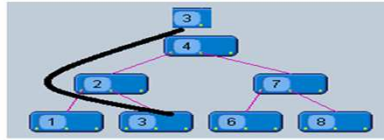
- Arbre de recherche équilibré
- Capable d'avoir plus d'une clef par nœud (limite la taille de l'arbre)
- Modélise les index d'une base de données

L'objet Btree : Insertion



- Ce qu'il faut retenir:
 - Un Btree est un arbre de recherche binaire
 - Plusieurs clefs peuvent être contenues dans une feuille

Btree : Recherche



- L'arbre de recherche (ici un Btree) est très fort pour chercher un élément



BTREE/HASH

Tout d'abord, selon le moteur de stockage utilisé, vous pouvez tout simplement pas avoir le choix (par exemple InnoDB est exclusivement en utilisant BTREE pour son indice).

En outre, BTREE est le type d'index par défaut pour la plupart des moteurs de stockage.

Maintenant ... Il ya des cas, lors de l'utilisation d'autres types d'index peut entraîner une amélioration des performances. Il existe (cas relativement rare) quand un index HASH peut aider. Notez que quand un index HASH est créé, un indice BTREE est également produit. C'est en partie dû au fait que les index hash ne peuvent résoudre les prédicats d'égalité. (une condition telle que WHERE Prix > 12,0 pouvait pas être manipulée par un index hash).

En bref: Continuer à utiliser BTREE, que ce soit implicitement (si BTREE est la valeur par défaut pour le stockage d'occasion), ou explicitement. Renseignez-vous sur les autres types d'index afin que vous sachiez à leur sujet serait le besoin s'en fait sentir.

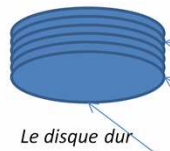
Index HASH sont plus génériques (non limités à un type particulier d'application ou de données), et on peut généralement suivre sa compréhension intuitive de hachages pour obtenir un indice quant au moment où ceux-ci peuvent surperformer.

En Pratique

PK	Cru	Qualité	DEVRE	MILLESIME
1	Beaujolais	Excellent	11	1987
2	Chenas	Mediocre	12	1990
3	Julienas	Bonne	12	1990
5	Beaujolais	Mediocre	10	1985
6	Julienas	Bonne	12	1987
7	Chenas	Mediocre	11	1989
14	Chenas	Bonne	10	1994
15	Julienas	Mediocre	11	1995
18	Beaujolais	Bonne	10	1990
25	Julienas	Bonne	12	1988

La table des vins est organisée en fonction des PK

En Pratique



PK	Cru	Qualité	DEVRE	MILLESIM E
1	Beaujolais	Excellent	11	1987
2	Chenas	Mediocre	12	1990
3	Julienas	Bonne	12	1990
5	Beaujolais	Mediocre	10	1985
6	Julienas	Bonne	12	1987
7	Chenas	Mediocre	11	1989
14	Chenas	Bonne	10	1994
15	Julienas	Mediocre	11	1995
18	Beaujolais	Bonne	10	1990
25	Julienas	Bonne	12	1988

Les PK sont indexées via un Btree+ (organisées par clef croissante) et les index "secondaires" sont organisés en Btree

En Pratique

Index BTREE sur les cru

Beaujolais	1,5,18
Chenas	2,7,14
Julienas	3,6,15,25

Index BTREE sur degre

10	5,14,18
11	1,7,15
12	2,3,6,25

PK	Cru	Qualité	DEVRE	MILLESIM E
1	Beaujolais	Excellent	11	1987
2	Chenas	Mediocre	12	1990
3	Julienas	Bonne	12	1990
5	Beaujolais	Mediocre	10	1985
6	Julienas	Bonne	12	1987
7	Chenas	Mediocre	11	1989
14	Chenas	Bonne	10	1994
15	Julienas	Mediocre	11	1995
18	Beaujolais	Bonne	10	1990
25	Julienas	Bonne	12	1988

Il y a 2 index secondaires, les "cru" et les "degre" organisés en Btree

En Pratique

Index BTREE sur les cru

Beaujolais	1,5,18
Chenas	14,2,7
Julienas	3,6,15,25

Index BTREE sur degree

10	5,14,18
11	1,7,15
12	2,3,6,25

PK	Cru	Qualité	DEVRE	MILLESIM E
1	Beaujolais	Excellent	11	1987
2	Chenas	Mediocre	12	1990
3	Julienas	Bonne	12	1990
5	Beaujolais	Mediocre	10	1985
6	Julienas	Bonne	12	1987
7	Chenas	Mediocre	11	1989
14	Chenas	Bonne	10	1994
15	Julienas	Mediocre	11	1995
18	Beaujolais	Bonne	10	1990
25	Julienas	Bonne	12	1988

Select * from table where cru='chenas' AND millesime>1986 AND degree=12

En pratique

- `Select * from table where cru='chenas' AND millesime>1986 AND degre=12`
- Utilisation des index cru et chenas
- En interne:
- C =LIRE index CRU entée CHENAS
- D =LIRE index degre entée 12
- L=union (C,D)
- Puis `Select * from L where millesime>1986`

Btree & Btree+

Index BTREE sur les cru

Beaujolais	1,5,18
Chenas	14,2,7
Julienas	3,6,15,25

Index BTREE+ sur les cru

Beaujolais	1,5,18
Chenas	2,7,14
Julienas	3,6,15,25

Index BTREE sur degre

10	5,14,18
11	1,7,15
12	2,3,6,25

Index BTREE+ sur degre

10	5,14,18
11	1,7,15
12	2,3,6,25

Fusion

Chenas	14,2,7
12	2,3,6,25

Fusion (btree+)

Chenas	2,7,14
12	2,3,6,25

En pratique

	BTREE	BTREE+	Hash Index
Recherche	Orange	Orange	Green
Recherche Multiple	Green	Green	Orange
Efficacité vis-à-vis du disque	Orange	Green	Green
Efficacité vis-à-vis de l'opérateur '='	Green	Green	Green
Efficacité vis-à-vis de l'opérateur '<' ou '>'	Orange	Green	Orange



Les Index

- Contenu
 - Comment accélérer un schéma de base de données via la mise en place d'Index
- Objectifs
 - Créer un Index
 - Comprendre comment placer un Index.

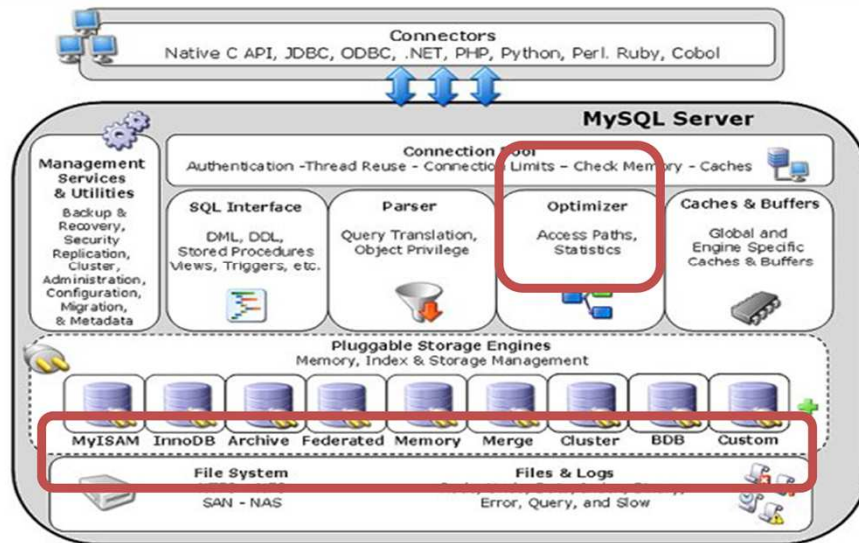
Les index MySQL : placements, efficacité

Indexer, pourquoi ?

L'indexation peut avoir plusieurs buts :

- Accéder à ses données plus rapidement, les index sont en effet l'outil le plus puissant pour **accélérer les temps d'exécution de vos requêtes** jusqu'à parfois plusieurs centaines de % !
- Définir le degré d'unicité d'une colonne donnée : chaque champ doit-il être unique ? les doublons sont-ils autorisés ?

Les index MySQL : la théorie



Les index MySQL

- Primary Key : MySQL définit automatiquement un index sur chaque clé primaire ; déclarer une colonne comme clé primaire a pour effet d'y interdire les Null et les doublons. (BTree+)
- Unique : index interdisant les doublons, et mettant ainsi en œuvre une contrainte d'unicité. (BTree+)
- Key ou Index : index simple, autorisant Null et doublons. (BTree)

```
ALTER TABLE <Nom de table> ADD INDEX (Col)
```

Les index MySQL : le test

```
CREATE TABLE `index_table` (`champ` int)
CREATE TABLE `index_table2` (`champ` int)
CREATE INDEX `champ_index` ON `index_table2`
(`champ`);
Procedure generaterandom()
BEGIN
DECLARE rand_variable INT DEFAULT 0;
DECLARE x INT;
SET x = 1;
REPEAT
  select RAND(100) into rand_variable;
  insert into index_table(champ) values(rand_variable);
  set x=x+1;
UNTIL x > 1000000
END REPEAT;
END
```

Les index MySQL : le test

- Select count(*) where champ>50;
 - Index_table: 70ms
 - Index_table2:0.09ms
- Select * from ... order by champ;
 - Index_table:1.81 s
 - Index_table2:1.20s
- Select count(*) from index_table2 AS T1 JOIN index_table2 AS T2 ON T1.champ=T2.champ;
 - Index_table : ????
 - Index_table2:512s

Les index MySQL : comment faire ?

- **Quels types de champ indexer ?**
 - Plus l'index est court, mieux c'est
 - Soyez radins !
- **Où placer ses index ?**
 - Les champs concernés par une clause **WHERE**, **ORDER BY**, **GROUP BY**, **MIN()**, **MAX()**, ainsi que les champs qui permettent de **relier des tables entre elles**,
 - Les index composés (ou multiples) et la règle du leftmost prefixing
- **Mesurer l'efficacité des index avec EXPLAIN**
- Ne pas se reposer sur les Index !



Modèle relationnel, conception et création d'une base

La couche applicative

- Contenu
 - Comment se connecter en JDBC à MySQL
- Objectifs
 - Créer une connexion JDBC
 - Exécuter une requête
 - Optimiser la couche applicative

Éléments de base

- Il faut un driver (JDBC, ODBC ou PHP)
- Une connexion est une relation à la base de données qu'il faut ouvrir... et fermer.
- Un « statement » modélise une requête
- Un prepared statement modélise une requête dans le cache de requête.
- Un resultSet est un résultat de requête.
- Un cache de second niveau est un cache applicatif
- Une connexion pool est une boîte de connexion non fermée

Éléments techniques

- Un cache de second niveau : ehCache
- Une connexion pool : apache common dbcp

Faire du JDBC simple

```
public static Connection initConnection() throws SQLException, ClassNotFoundException
{
    Class.forName("com.mysql.jdbc.Driver");
    return DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "");
}
```

```
public static Long findByPrimaryKeyWithoutQueryPlan(Connection c, Long pk) throws
SQLException {
    ResultSet resultSet=null;
    Statement statement=null;
    try{Long result=0L;
        statement=c.createStatement();
        resultSet=statement.executeQuery("select A from B where C=1");
        while (resultSet.next())
        {
            result=resultSet.getLong(1);
        }
        return result;
    }finally
    {
        if (resultSet!=null) resultSet.close();
        if (statement!=null) statement.close();
    }
}
```

Faire une connexion Pool

```
public static DataSource initConnectionPool()
{
    BasicDataSource bs = new BasicDataSource();
    bs.setDriverClassName("com.mysql.jdbc.Driver");
    bs.setUsername("root");
    bs.setUrl("jdbc:mysql://localhost:3306/test" );
    bs.setLogAbandoned(true);
    bs.setTestOnBorrow(true);
    bs.setTestOnReturn(true);
    bs.setTestWhileIdle(true);
    bs.setMaxActive(30);
    bs.setValidationQuery("select 1");
    bs.setMaxIdle(5);
    bs.setInitialSize(3);
    bs.setPoolPreparedStatements(true);
    return bs;
}
```

Faire une requête préparée

```
public static Long findByPrimaryKeyQueryPlan(Connection c, Long pk) throws SQLException {
    ResultSet resultSet=null;
    PreparedStatement statement=null;
    try{Long result=0L;
        statement=c.prepareStatement("select A from B where C=?");
        statement.setLong(1, pk);
        resultSet=statement.executeQuery();
        while (resultSet.next())
        {
            result=resultSet.getLong(1);
        }
        return result;
    }finally
    {
        if (resultSet!=null) resultSet.close();
        if (statement!=null) statement.close();
    }
}
```

Faire un cache de second niveau

```
public static Long findByPrimaryKeyThroughCache(Connection c, Long pk) throws
SQLException {
    Cache cache = CacheManager.getInstance().getCache("cacheName");
    Long result=null;
    Element el=cache.get(pk);
    if (el!=null)
    {
        result= (Long)el.getValue();
    }else
    {
        result=findByPrimaryKeyQueryPlan(c,pk);
        cache.put(new Element(pk,result));
    }
    return result;
}
```


Performance 1000 connexions/10000 requêtes

Sélectionner 1 ligne avec 1 clef primaire

Performance	Sans Pool, Sans requête préparée, sans cache	Avec Pool, Sans requête préparée, sans cache	Avec Pool, Avec requête préparée, sans cache	Avec Pool, Avec requête préparée, Avec cache
Connexion	6.721 ms /Connexion	0.611 ms /Connexion	0.601 ms /Connexion	0.621 ms /Connexion
Requête	0.5363 ms /requêtes	0.489 ms /requêtes	0.3483 ms /requêtes	0.011 ms /requêtes

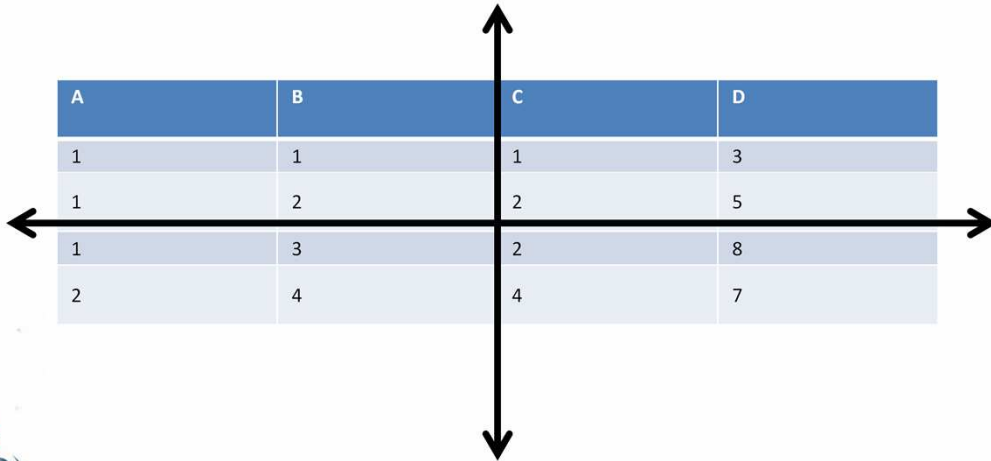


Le partitionning

- Contenu
 - Comment accélérer un schéma de base de données via le partitionning
- Objectifs
 - Monter une partition horizontale
 - Monter une partition verticale

Partition horizontal et vertical

- L'idée général est de couper les tables horizontalement et verticalement pour réduire la taille des tables.



A	B	C	D
1	1	1	3
1	2	2	5
1	3	2	8
2	4	4	7

Partition verticale

A	B
1	1
1	2
1	3
2	4



C	D
1	3
2	5
2	8
4	7


Partition verticale

- Implémentation d'une table X partitionné en p1, p2
- Créer une table p1 et p2
- Créer une vue p1 jointe p2

```
CREATE TABLE `P1` ( `ID` INT UNSIGNED PRIMARY KEY  
AUTO_INCREMENT, `DATA` INTEGER)  
CREATE TABLE `P2` ( `ID` INT UNSIGNED UNIQUE NOT NULL,  
`DATA` BLOB)  
  
CREATE VIEW TABLE_COMPLETE AS  
SELECT * FROM P1 p INNER JOIN P2 ph ON p.ID = ph.ID;
```

Partition horizontale

A	B	C	D
1	1	1	3
1	2	2	5
1	3	2	8
2	4	4	7



Partition horizontale

- Partitionnement par intervalle : partition de ligne en fonction d'une valeur
- Partition par liste: partition selon les valeurs d'une liste
- Partition par haschage
- Partition par clé

Test

```
CREATE TABLE transac
(
  id INT UNSIGNED NOT NULL,
  montant INT UNSIGNED NOT NULL,
  jour DATE NOT NULL,
  codePays ENUM('FR', 'BE', 'UK', 'US', 'CA', 'JP') NOT NULL
)

DELIMITER //
CREATE PROCEDURE remplir_transaction(nbTransacs INT)
BEGIN
  DECLARE i INT DEFAULT 1; DECLARE nbAlea DOUBLE;
  DECLARE _jour DATE; DECLARE _montant INT UNSIGNED;
  DECLARE _codePays TINYINT UNSIGNED;

  WHILE i <= nbTransacs DO
    SET nbAlea = RAND();
    SET _jour = ADDDATE('1996-01-01', FLOOR(nbAlea * 4015));
    SET _montant = FLOOR(1 + (nbAlea * 9999));
    SET nbAlea = RAND();
    SET _codePays = FLOOR(1 + (nbAlea * 6));
    INSERT INTO transac (id, montant, jour, codePays) VALUES (i, _montant,
    _jour, _codePays); SET i = i + 1;
  END WHILE;
END //
DELIMITER ;
```


Partition par intervalle

- L'idée est de partitionner la table en fonction d'un champ valeur.
- L'optimisation réside dans le fait qu'une requête va « attaquer » un nombre restreint de partitions.
- En pratique, les tables dont les requêtes sont basées sur des dates et des entiers « ordonnées » sont de bonnes candidates.

Partition par intervalle

```
CREATE TABLE transac_part
(
  id INT UNSIGNED NOT NULL,
  montant INT UNSIGNED NOT NULL,
  jour DATE NOT NULL,
  codePays ENUM('FR', 'BE', 'UK', 'US', 'CA', 'JP') NOT NULL
) PARTITION BY RANGE(YEAR(jour))
(
  PARTITION p1 VALUES LESS THAN(1997),
  PARTITION p2 VALUES LESS THAN(1998),
  PARTITION p3 VALUES LESS THAN(1999),
  PARTITION p4 VALUES LESS THAN(2000),
  PARTITION p5 VALUES LESS THAN(2001),
  PARTITION p6 VALUES LESS THAN(2002),
  PARTITION p7 VALUES LESS THAN(2003),
  PARTITION p8 VALUES LESS THAN(2004),
  PARTITION p9 VALUES LESS THAN(2005),
  PARTITION p10 VALUES LESS THAN(2006),
  PARTITION p11 VALUES LESS THAN MAXVALUE
);

CALL remplir_transaction(130948);
INSERT INTO transac_part SELECT * FROM transac;
```

Partition par Liste de valeurs

- L'idée est de partitionner la table en fonction d'une liste de valeurs.
- L'optimisation réside dans le fait qu'une requête va « attaquer » une partition en fonction d'une valeur .
- Très utile par exemple pour sélectionner des données issues d'environnements géographiques.

Partition par Liste de valeurs

```
CREATE TABLE transac_part
(
  id INT UNSIGNED NOT NULL,
  montant INT UNSIGNED NOT NULL,
  jour DATE NOT NULL,
  codePays TINYINT UNSIGNED NOT NULL
) PARTITION BY LIST(codePays)
(
  PARTITION pEurope VALUES IN (1, 2, 3),
  PARTITION pAmeriqueNord VALUES IN (4, 5),
  PARTITION pAsie VALUES IN (6)
);
INSERT INTO transac_part SELECT * FROM transac;
```

id	code
1	BE
2	FR
3	UK
4	US
5	CA
6	JP

Partition par Hash et Clef

- L'idée est de partitionner la table en fonction des résultats du hash d'une valeur. (pratique pour les valeurs non entières).
- Difficile à mettre en place en pratique (quelle fonction de hash choisir en fonction des requêtes?).
- Le partitionnement par clef est un partitionnement par Hash. L'idée est d'avoir un Hash Index sur la clef et de répartir les données de façon homogène

Partition par Hash (erreur)

```
CREATE TABLE transac_part
(
  id INT UNSIGNED NOT NULL,
  montant INT UNSIGNED NOT NULL,
  jour DATE NOT NULL,
  codePays ENUM('FR', 'BE', 'UK', 'US', 'CA', 'JP') NOT NULL
) PARTITION BY HASH(YEAR(jour)) PARTITIONS 11;
```

Partition par Clef

```
CREATE TABLE transac_part
(
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
  montant INT UNSIGNED NOT NULL,
  jour DATE NOT NULL,
  codePays ENUM('FR', 'BE', 'UK', 'US', 'CA', 'JP') NOT NULL
) PARTITION BY KEY() PARTITIONS 5;
```

Le partitionnement

- Gain de performance en lecture pouvant aller à X 10
- Potentialité de perte de performance si les partitions sont mauvaises
- En insertion et suppression, il est possible d'obtenir plus de performance.



Modèle relationnel, conception et création d'une base

Utilisation de la base information_schema.

- Contenu
 - Présentation de la base Information_schema
- Objectif
 - Voir les informations résidant dans ce schéma

INFORMATION_SCHEMA

- La commande SHOW n'est pas standard
- En général, les bases contiennent un schéma de méta données
Information Schema est celui de MySQL
- Les "métadonnées" sont des informations sur les données, tels que le nom des bases de données, des tables, le type de données des colonnes ou les droits d'accès. On appelle aussi ces données le "dictionnaire de données" ou le "catalogue système".

Exemple

```
SELECT table_name, table_type, engine
-> FROM information_schema.tables
-> WHERE table_schema = 'db5'
-> ORDER BY table_name DESC;
```

table_name	table_type	engine
v56	VIEW	NULL
v3	VIEW	NULL
v2	VIEW	NULL

Information Schema

SCHEMATA	
CATALOG_NAME	VARCHAR(512)
SCHEMA_NAME	VARCHAR(64)
DEFAULT_CHARACTER_SET_NAME	VARCHAR(32)
DEFAULT_COLLATION_NAME	VARCHAR(32)
SQL_PATH	VARCHAR(512)

TABLES	
TABLE_CATALOG	VARCHAR(512)
TABLE_SCHEMA	VARCHAR(64)
TABLE_NAME	VARCHAR(64)
TABLE_TYPE	VARCHAR(64)
ENGINE	VARCHAR(64)
VERSION	BIGINT(21)
ROW_FORMAT	VARCHAR(10)
TABLE_ROWS	BIGINT(21)
AVG_ROW_LENGTH	BIGINT(21)
DATA_LENGTH	BIGINT(21)
MAX_DATA_LENGTH	BIGINT(21)
INDEX_LENGTH	BIGINT(21)
DATA_FREE	BIGINT(21)
AUTO_INCREMENT	BIGINT(21)
CREATE_TIME	DATETIME
UPDATE_TIME	DATETIME
CHECK_TIME	DATETIME
TABLE_COLLATION	VARCHAR(32)
CHECKSUM	BIGINT(21)
CREATE_OPTIONS	VARCHAR(255)
TABLE_COMMENT	VARCHAR(2048)

ENGINES	
ENGINE	VARCHAR(64)
SUPPORT	VARCHAR(8)
COMMENT	VARCHAR(80)
TRANSACTIONS	VARCHAR(3)
XA	VARCHAR(3)
SAVEPOINTS	VARCHAR(3)

TABLE_CONSTRAINTS	
CONSTRAINT_CATALOG	VARCHAR(512)
CONSTRAINT_SCHEMA	VARCHAR(64)
CONSTRAINT_NAME	VARCHAR(64)
TABLE_SCHEMA	VARCHAR(64)
TABLE_NAME	VARCHAR(64)
CONSTRAINT_TYPE	VARCHAR(64)

Extrait de la base Information Schema

Fonctionnalités avancées

- Introduction et rappels
- Extension procédurale de SQL
- Objets de MySQL
- Fonctionnalités avancées
 - Index fulltext et recherches fulltext.
 - Plans d'exécution, commande EXPLAIN.
 - Requêtes préparées.
 - Les transactions.
 - Niveaux d'isolation.
 - Verrouillage des tables.
- Connexions, droits d'accès, sécurité
- Introduction à l'administration



Modèle relationnel, conception et création d'une base

Index fulltext et recherches fulltext.

- Contenu :
 - Comprendre la notion de Fulltext
- Objectifs :
 - Positionner un index Fulltext
 - Effectuer une recherche FullText

Fonctionnalités avancées: fulltext

```
mysql> select * from ville limit 5;
```

departement	INSEE	commune
00	00000	Etranger
01	00000	Commune inconnue
01	01001	L'Abergement-Clémenciat
01	01002	L'Abergement-de-Varey
01	01004	Ambérieu-en-Bugey

5 rows in set (0.03 sec)

La table « ville » contient toute les villes de France (DOM-TOM comprises).

Recherche Full-Text

- Recherche dans les chaînes de caractères:
 - `SELECT * FROM ville WHERE commune LIKE 'Saint%'` (70 ms)

```
mysql> SELECT * FROM ville WHERE commune LIKE
'Saint%';
+-----+-----+-----+
| departement | INSEE | commune |
+-----+-----+-----+
| 01          | 01331 | Saint-Alban |
| 01          | 01332 | Saint-André-de-Bâgé |
| 01          | 01333 | Saint-André-de-Corcy |
| 01          | 01334 | Saint-André-dHuriat |
| 01          | 01335 | Saint-André-le-Bouchoux |
+-----+-----+-----+
```


Recherche Full-Text: Index Full Text

- Disponible sur le moteur MyISAM uniquement:

```
ALTER TABLE `bibliotheque`.`ville` ENGINE = MyISAM
```

```
ADD FULLTEXT INDEX `FT_COMMUNE` (`commune` ASC);
```

FULL-TEXT: fonction MATCH

MATCH (col1,col2,...) AGAINST (expr [IN BOOLEAN MODE | WITH QUERY EXPANSION])

```
mysql> SELECT * FROM ville WHERE MATCH(commune) AGAINST ('Saint')
-> ;
```

departement	INSEE	commune
51	51513	Saint-Remy-en-Bouzemont-Saint-Genest-et-Isson
42	42239	Saint-Jean-Saint-Maurice-sur-Loire
43	43187	Saint-Geneyss-près-Saint-Paulien
79	79276	Saint-Martin-de-Saint-Maixent
38	38427	Saint-Michel-de-Saint-Geoirs

La fonction Match

- Renvoie la pertinence de la recherche
- MySQL utilise un filtre très simple pour séparer le texte en mots. Un "mot" est n'importe quelle chaîne de caractères constituée de lettres, chiffres, "" et '_'.
• Un mot trop court est ignoré.
- Les mots appartenant à la liste des « mots bannis » sont supprimés
- Les mots trop « communs » sont bannis

La fonction Match en Pratique

```
SELECT commune,MATCH(commune) AGAINST ('mon paris') as score FROM ville
WHERE MATCH(commune) AGAINST ('mon paris')
"Mon" est un mot "bannis"
```

```
+-----+-----+
| commune                | score                |
+-----+-----+
| Paris                   | 7.293986797332764   |
| Paris 20e Arrondissement | 7.211991786956787   |
| ...                     |                      |
| Paris 10e Arrondissement | 7.211991786956787   |
| Paris 9e Arrondissement  | 7.211991786956787   |
| Paris-lHopital          | 7.211991786956787   |
| ...                     |                      |
| Le Touquet-Paris-Plage  | 7.131819725036621   |
+-----+-----+
23 rows in set (0.00 sec)
```

FullText : la vraie vie

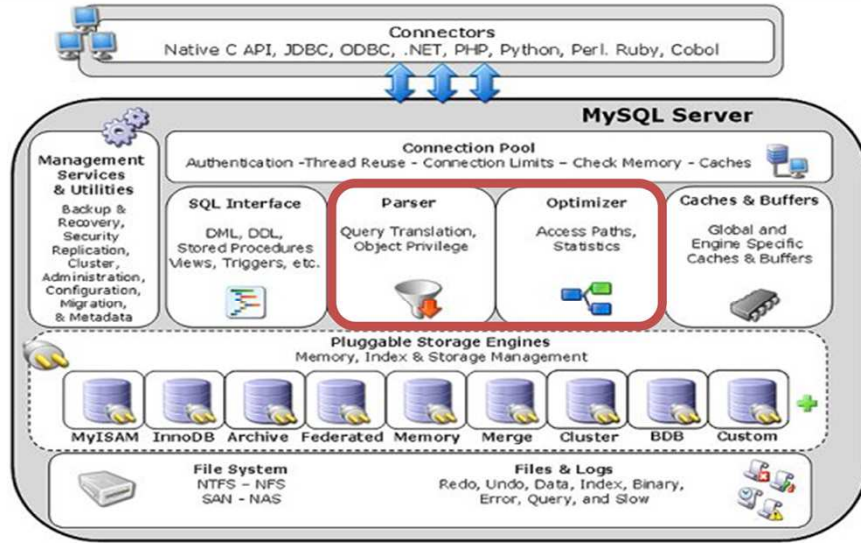
- Marche correctement et simplement pour des bases de taille « moyenne » MAIS:
- Obligation d'utiliser des tables MyISAM/InnoDB non partitionné
 - Pas de clusteur
 - Pas d'optimisation sur les partition
- Devient «lent » à partir de 1gb de données
- Base Wikipedia (15g de texte):
 - Création d'un index sur le titre: 273 s
 - Création d'un index sur le titre et le corps: +6h
 - Requetage 5 * plus lent qu'avec des outils tiers tel que Sphinx



La commande Explain

- Contenu:
 - Qu'est ce qu'un plan de calcul?
 - Qu'est ce que la commande Explain
- Objectifs:
 - Connaître la commande Explain
 - Savoir exploiter les résultats de la commande

Les index MySQL : la théorie



La commande EXPLAIN

- Préfixer la commande avec le mot EXPLAIN
 - MySQL n'exécute pas la requête, mais l'analyse
 - EXPLAIN permet de connaître le « plan de calcul » de MySQL
 - EXPLAIN renvoie une table explicitant les opérations.
- Le but de cette commande est de permettre l'optimisation en:
 - Modifiant/créant des index
 - Modifiant les requêtes
 - Modifiant les données
- Le but est de permettre une exécution plus rapide et moins coûteuse

La commande EXPLAIN

```
create table invites
(
invite_ID serial,
surnom VARCHAR(50) NOT NULL,
salle_ID BIGINT,
arrive TINYINT
);
EXPLAIN select * from invites
```

Id	Select_type	Rows
1	SIMPLE	9802

La commande EXPLAIN

```
explain select * from invites where SALLE_ID IS NOT NULL OR SALLE_ID IS NULL
```

- select * from invites where **SALLE_ID IS NOT NULL OR SALLE_ID IS NULL**
- plusieurs plans de calcul différents:
 - (Select * from invites where SALLE_ID IS NOT NULL) OR (select * from invites where SALLE_ID IS NULL)
 - Select * from invites
 - Select * from invites where OR (salle_ID IS not NULL, salle_ID IS NULL)

Id	Select_type	Rows	Extra
1	SIMPLE	9802	Using where

La commande EXPLAIN

- id : identifiant de SELECT, le numéro séquentiel de cette commande SELECT dans la requête.
- select_type:
 - SIMPLE:
 - PRIMARY
 - Second et autres UNION SELECTs.
- table : La table à laquelle la ligne fait référence.
- possible_keys : La colonne possible_keys indique quels index MySQL va pouvoir utiliser pour trouver les lignes dans cette table. Notez que cette colonne est totalement dépendante de l'ordre des tables.

La commande EXPLAIN

- key : la colonne key indique l'index que MySQL va décider d'utiliser. Si la clé vaut NULL, aucun index n'a été choisi.
- rows : la colonne rows indique le nombre de ligne que MySQL estime devoir examiner pour exécuter la requête.

```
mysql> explain select * from invites where SALLE_ID>123;  
Select_type: SIMPLE  
Possible_keys,keys:NULL  
Rows:9680
```

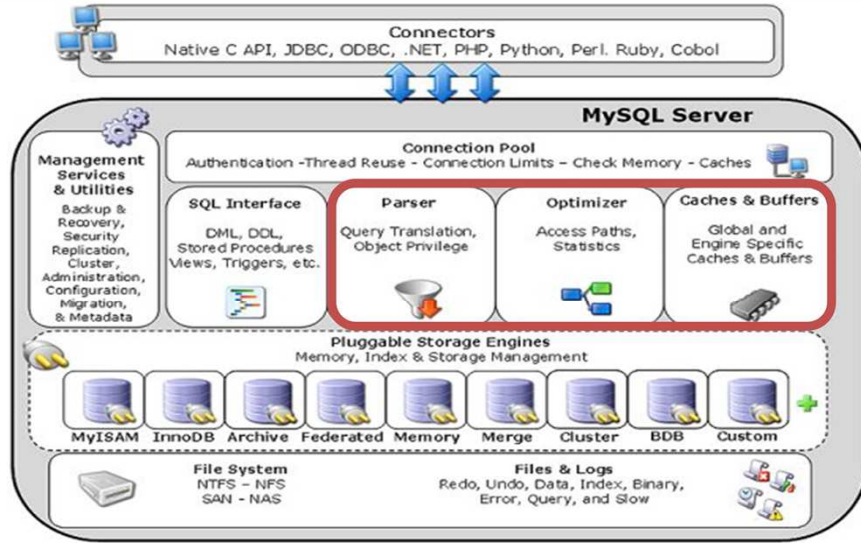
Interpretation

- Pas d'index trouvé :
 - MySQL effectue un full scan
 - Un full scan est lent!
 - Peut-être un index doit être mis?
- ALTER TABLE invites ADD INDEX invites_IDX(SALLE_ID)
- EXPLAIN:
 - Select_type: SIMPLE
 - Table : invites
 - Possible_keys:invites_IDX
 - Key:NULL ??
 - Rows:9640

Interprétation

- ALTER TABLE invites ADD INDEX arrive_IDX(arrive)
- explain select * from invites where SALLE_ID>123 and arrive=1
 - Possible_keys:invite_IDX,arrive_IDX
 - Key:invite_IDX
 - Rows : 5014
- Deux index sont possibles mais MySQL décide de pas utiliser arrive_IDX
 - Le « spread » de la variable est faible
 - Dépendant des requêtes précédentes

Les index MySQL : la théorie



Les performances

- La commande Explain exprime surtout le nombre de lignes que doit regarder MySQL pour un select
 - Ce n'est pas une mesure de la performance réelle
- En cas de problèmes de performance :
 - Regarder l'influence de la mise en place d'Index via la commande Explain
 - Faire la chasse aux conditions « fantômes »
 - Chercher un spécialiste!

Optimisation pour le disque

- Le facteur limitant : l'accès aux disques
 - L'accès aux disques dépend plus ou moins du fait que les Index sont en mémoire
- Pour une petite table, la table Index est souvent en mémoire -> 1 accès disque
- Pour une grosse table, la table Index n'est pas en mémoire et le nombre d'accès disque est fourni par :
 - $\log(\text{row_count}) / \log(\text{index_block_length} / 3 \times 2 / (\text{index_length} + \text{data_pointer_length})) + 1$. app $\log(N)$
 - Pour 500000 ligne de MEDIUM INT -> 4 accès disques

Optimisation disque

- La taille du cache des Index est donc critique
 - Pour 500000 lignes -> entre 3 et 10 Mo de taille mémoire
- Pas assez d'Index ralentit les requêtes car provoquant des full scan
- Trop d'Index encombre le cache d'Index

Pratique du SQL avec MySQL

- Introduction et rappels
- Pratique du SQL avec MySQL
- Modèle relationnel, conception et création d'une base
- **Tables transactionnelles InnoDB**
 - Notion de transaction, niveaux d'isolation.
 - Structure physique des tables.
 - Programmation des transactions (START TRANSACTION, COMMIT, ROLLBACK).
- SQL procédural
- Connexions, droits d'accès, sécurité
- Introduction à l'administration



Tables transactionnelles InnoDB

Les Transactions

- Contenu
 - Présenter les Transactions
 - Comprendre ce que veut dire ACID
- Objectifs
 - Comprendre les commandes « Commit » et « Rollback »

Transactions et MySQL

- **Qu'est ce qu'une Transaction**
- **ACID**
 - Atomicity: La requête est exécutée dans une transaction soit complètement soit pas du tout !
 - Consistency: Les règles d'intégrité peuvent être rompues uniquement pendant la transaction
 - Isolation: Les données utilisées pendant une transaction ne peuvent pas être utilisées dans une autre transaction sans que la première soit terminée
 - Durability: Une transaction terminée n'est plus réversible.

InnoDB Transaction

- Les Transactions sont comprises entre les mots clefs *BEGIN* et *COMMIT*.
- Un Exemple:

```
mysql> CREATE TABLE t (f INT) TYPE=InnoDB;
mysql> BEGIN; Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO t(f) VALUES (1); Query OK, 1 row
affected (0.01 sec)
mysql> SELECT * FROM t;
+-----+
| f |
+-----+
| 1 |
+-----+ 1 row in set (0.02 sec)
mysql> ROLLBACK; Query OK, 0 rows affected (0.01 sec)
mysql> SELECT * FROM t; Empty set (0.00 sec)
```

- La présence du mot **ROLLBACK** supprime la transaction.

COMMIT/ROLLBACK

- BEGIN commence une transaction
- COMMIT valide la transaction
- ROLLBACK annule la transaction

- Il peut y avoir plusieurs requêtes entre le BEGIN (début de transaction) et un COMMIT ou un ROLLBACK.

Lecture consistante

- Connection 1:

```
mysql> BEGIN;
Query OK,
INSERT INTO t (f) VALUES (1);
SELECT * FROM t;
+-----+
| f     |
+-----+
| 1     |
```

- Connection 2:

```
SELECT * FROM t;
Empty set (0.02 sec)
```


Lecture consistante

- Connexion 1:

```
mysql> COMMIT;  
Query OK, 0 rows affected (0.00 sec)
```

- Connexion 2:

```
mysql> SELECT * FROM t;  
+-----+  
|   f   |  
+-----+  
|   1   |  
+-----+  
1 row in set (0.00 sec)
```

Commits Explicite

- Il est possible de dire a MySQL que toutes les requêtes seront dans des transactions:

```
SET AUTOCOMMIT=0;
```

- Connexion 1:

```
mysql> INSERT INTO t (f) VALUES (2);
```

- Connexion 2:

```
mysql> SELECT * FROM t;
+-----+
| f |
+-----+
| 1 |
+-----+ 1 row in set (0.00 sec)
```

- Connexion 1

```
mysql> COMMIT;
Query OK, 1 rows affected (0.00 sec)
```

Commits Explicite

- pour savoir si vous êtes (1) en mode autocommit ou non (0) :

```
SELECT @@autocommit ;
```

- pour enlever l'autocommit (deux possibilités) :

```
SET @@autocommit = 0 ;  
SET @@autocommit = off ;
```

- pour le remettre :

```
SET @@autocommit = 1 ;  
SET @@autocommit = ON ;
```



Tables transactionnelles InnoDB

Différents types de transaction

- Contenu
 - Read lock for Updating
 - Read lock for Sharing
 - Niveau d'Isolation

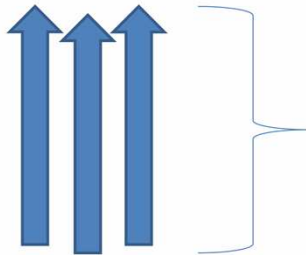
- Objectifs
 - Approcher la notion d'isolation et de lock
 - Apprendre à utiliser ces notions

Les Locks

Locks sur la table

Table X	
Colonne A	Colonne B
A	B
Aa	Bb
Aaa	Bbb
Aaaa	Bbbb

Locks sur
les valeurs



Locks sur les
transactions : « Niveau
d'isolation »

Read locks for Updating (1/3)

- La lecture consistante n'est pas forcément un comportement voulu. En particulier, lors de la manipulation d'agrégats.
- Autrement dit, il est intéressant de dire à d'autres transactions que nous allons modifier une donnée

Read locks for Updating (2/3)

- Connexion 1:

```
mysql> SELECT MAX(f) FROM t; → 3  
mysql> INSERT INTO t(f) VALUES (4);
```

- Connexion 2:

```
mysql> SELECT MAX(f) FROM t; -> 3  
mysql> INSERT INTO t(f) VALUES(4);  
mysql> COMMIT;
```

- Connexion 1:

```
mysql> COMMIT;  
mysql> SELECT * FROM t; -> 1 2 3 4 4
```

- Le But d'un Read Lock for Updating est de pouvoir non pas avoir 1 2 3 4 4 mais 1 2 3 4 5

Read locks for Updating (3/3)

- Connexion 1:

```
mysql> BEGIN; Query OK
mysql> SELECT MAX(f) FROM t FOR UPDATE; -> 3
mysql> INSERT INTO t(f) VALUES (4); Query OK, 1 row affected
(0.00 sec)
```

- Connexion 2:

```
mysql> SELECT MAX(f) FROM t FOR UPDATE;
MySQL se met en attente du fait du Lock for Update !!
```

- Connexion 1:

```
mysql> COMMIT;
Débloque la connexion 2
```

- Connexion 2:

```
mysql> SELECT MAX(f) FROM t FOR UPDATE-> 4
```


Read locks for sharing (1/2)

- Un autre de type lock est le Read Lock for Sharing. Il est en effet possible qu'une transaction doit pouvoir lire la dernière version d'une donnée alors qu'elle est modifiée dans une autre transaction. Ce cas est géré via un "LOCK IN SHARE MODE" qui bloque toutes les mises à jour d'une donnée en train d'être lue ou modifiée dans une autre transaction.

Read locks for sharing (2/2)

- Connexion 1:

```
mysql> BEGIN; Query OK, 0 rows affected (0.00 sec)
mysql> SELECT MAX(f) FROM t LOCK IN SHARE MODE -> 5
```

- Connexion 2:

```
mysql> UPDATE t SET f = 55 WHERE f=5;
```

- **Du fait du Lock in Share, nous sommes en attente ici !**

- Connexion 1:

```
mysql> COMMIT;
```

- Connexion 2

```
mysql> UPDATE t SET f = 55 WHERE f=5; Query OK, 1 row
affected (43.30 sec) Rows matched: 1 Changed: 1
Warnings: 0
```

Niveau d'Isolation

- READ UNCOMMITTED: La transaction n'est pas isolée (lecture de données non "commité"). Autrement appelée "dirty read".
- READ COMMITTED: interdit la lecture sale
- REPEATABLE READ: **Il s'agit du mécanisme par défaut.** Les lectures sont cohérentes d'un début de transaction à la fin. On parle de lecture fantôme.
- SERIALIZABLE: fait comme si les transactions étaient validées les unes après les autres et non en même temps

Niveau d'Isolation

- Pour connaître votre niveau d'isolation :

```
SELECT @@tx_isolation ;
```

- Pour le modifier, vous pouvez soit utiliser les syntaxes standards de SET :

```
SET tx_isolation = 'READ-COMMITTED' ; -- avec tiret et apostrophes  
SET tx_isolation = 1 ;  
SET @@tx_isolation = 1 ;  
SET GLOBAL tx_isolation = 'READ-COMMITTED' ;  
SET GLOBAL tx_isolation = 1 ;
```

- ou encore utilisez la syntaxe SET TRANSACTION ISOLATION LEVEL :

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;
```

READ UNCOMMITTED

- READ UNCOMMITTED "dirty read" : Les commandes SELECT non-verrouillantes sont effectuées sans rechercher de versions plus récentes de la ligne. Elles sont donc non-cohérentes, sous ce niveau d'isolation. Sinon, ce niveau fonctionne comme le niveau READ COMMITTED.

READ UNCOMMITTED

Connexion 1

```
set tx_isolation='read-uncommitted';

create table isolation_test (id int primary
key auto_increment, name varchar(8))
engine=InnoDB;

insert into isolation_test (name) values
('a'),('b');

select id,name from isolation_test;
+----+-----+
| id | name |
+----+-----+
| 1  | a    |
| 2  | b    |
+----+-----+
```

```
select id,name from isolation_test;
+----+-----+
| id | name |
+----+-----+
| 1  | a    |
| 2  | b    |
| 3  | a    |
| 4  | b    |
+----+-----+
4 rows in set (0.00 sec)
```

Connexion 2

```
set autocommit=0;
insert into isolation_test (name) values
('a'),('b');
```

- Les lignes 3,4 ne sont pas
« committer » dans la connexion 2
mais déjà visibles dans la connexion
1

READ COMMITTED

- Proche du niveau d'isolation d'Oracle. Toutes les commandes SELECT ... FOR UPDATE et SELECT ... LOCK IN SHARE MODE ne verrouillent que les lignes d'index, et non pas les trous entre elles, ce qui permet l'insertion de nouvelles lignes, adjacentes aux lignes verrouillées.
- UPDATE et DELETE qui utilisent un seul index avec une condition de recherche unique ne verrouille que la ligne d'index trouvée, par le trou qui la précède. Mais, pour un intervalle de lignes traitées par UPDATE et DELETE, InnoDB doit mêler des verrous de prochaines clés ou des verrous de trous et bloquer les insertions des autres utilisateurs dans les espaces couverts par l'intervalle. Ceci est nécessaire car les "lignes fantômes" doivent être bloquées pour la réplication MySQL et la restauration.

READ COMMITTED

Connexion 1

```
mysql> set tx_isolation='read-committed';
Query OK, 0 rows affected (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from isolation_test;
Empty set (0.00 sec)
```

Connexion 2

```
mysql> set tx_isolation='read-committed';
Query OK, 0 rows affected (0.00 sec)

mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from isolation_test;
Empty set (0.00 sec)
```

```
mysql> select * from isolation_test;
+----+-----+
| id | name |
+----+-----+
| 13 | c    |
| 14 | d    |
+----+-----+
```

- Les lignes 13,14 sont « committer » dans la connexion 2 sont déjà visibles dans la transaction de la connexion 1

REPEATABLE READ

Connexion 1

```
mysql> set tx_isolation='repeatable-read';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> begin;
mysql> select * from isolation_test ;
+----+-----+
| id | name |
+----+-----+
| 1  | a    |
| 2  | b    |
+----+-----+
```

```
mysql> select * from isolation_test ;
+----+-----+
| id | name |
+----+-----+
| 1  | a    |
| 2  | b    |
+----+-----+
```

Connexion 2

```
begin;
delete from isolation_test where id=1;
mysql> commit;
mysql> select * from isolation_test;
+----+-----+
| id | name |
+----+-----+
| 2  | b    |
| 3  | a    |
+----+-----+
```

- La ligne 1 est un fantôme.

Serializable

Connexion 1

```
mysql> set tx_isolation='serializable';  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> begin;
```

```
mysql> select * from isolation_test ;
```

Connexion 2

```
mysql> begin;  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> delete from isolation_test;  
Query OK, 2 rows affected (0.00 sec)
```

- La connexion 1 est bloquée

Table Locking

- Le verrouillage des tables bloque l'accès des tables aux autres connexions.
- En général, ce mécanisme est **inutile** et dangereux
- Le verrouillage des tables n'est nécessaire en général que:
 - Si on souhaite faire de nombreuses modifications en un temps record
 - Si on utilise des tables ne supportant pas les transactions.

Table Locking

- Syntax:
 - LOCK TABLES tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE} [, tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}]
...
 - UNLOCK TABLES

LOCK TABLE FOR READ

Connexion 1

```
LOCK TABLES isolation_test READ;
```

```
UNLOCK TABLES
```

Connexion 2

```
mysql> select * from isolation_test;
+----+-----+
| id | name |
+----+-----+
| 13 | c    |
| 14 | d    |
+----+-----+
2 rows in set (0.00 sec)

mysql> insert into isolation_test(name)
values('e');
```

```
Query OK, 1 row affected (1 min 16.02 sec)
```

- La connexion 2 est bloquée lors de l'insertion

LOCK TABLE FOR READ

Connexion 1

```
LOCK TABLES isolation_test WRITE;
```

```
UNLOCK TABLES
```

Connexion 2

```
mysql> select * from isolation_test;
```

```
+----+-----+  
| id | name |  
+----+-----+  
| 13 | c    |  
| 14 | d    |  
| 15 | e    |  
+----+-----+  
3 rows in set (10.84 sec)
```

- La connexion 2 est bloquée lors de la lecture

Extension procédurale de SQL

- Introduction et rappels
- Pratique du SQL avec MySQL
- Modèle relationnel, conception et création d'une base
- Tables transactionnelles InnoDB
- **SQL procédural**
 - Procédures stockées et fonctions.
 - Définition des procédures. Déclencheurs (Triggers).
 - Gestion des erreurs.
- Connexions, droits d'accès, sécurité
- Introduction à l'administration



SQL procédural

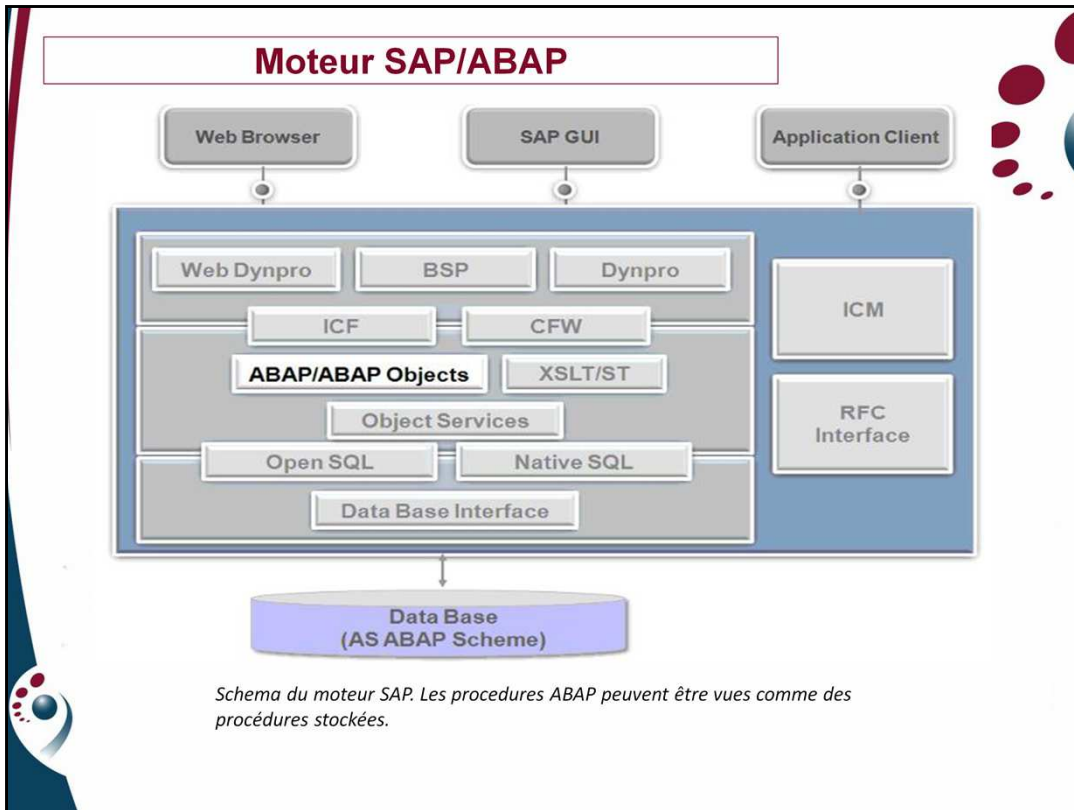
Procédures et fonctions stockées

- Contenu
 - Visualisation globale des procédures et fonctions stockées
- Objectifs
 - Ces fonctionnalités sont rarement utilisées en pratique sauf si:
 - La sécurité de la base est cruciale
 - Intégrer des multiples applications avec la base
 - La vitesse est primordiale.

Attention que dans ce cas, les procédures doivent être écrites par un expert.

Procédures stockées et les fonctions

- Les procédures stockées et les fonctions sont de nouvelles fonctionnalités de MySQL depuis la version 5.0. Une procédure stockées est un jeu de commandes SQL qui réside sur le serveur.
- Les procédures stockées peuvent fournir un gain de performances, car moins d'informations sont échangées entre le serveur et le client.
- Quelques situations où les procédures stockées sont utiles :
 - Lorsque plusieurs applications clientes sont écrites dans différents langages
 - Lorsque la sécurité est prioritaire.



Procédure Simple

```
mysql> delimiter |
mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
-> BEGIN
-> SELECT COUNT(*) INTO param1 FROM t;
-> END
->|
Query OK, 0 rows affected (0.00 sec)
mysql> CALL simpleproc(@a)|
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT @a
| +-----+
| @a |
+-----+
| 3 |
```

Fonction Simple

```
mysql> delimiter |
mysql> CREATE FUNCTION bonjour (s CHAR(20)) RETURNS
CHAR(50)
-> RETURN CONCAT('Bonjour, ',s,'!');
-> |
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT bonjour('le monde')|
+-----+
| bonjour('le monde') |
+-----+
| Bonjour, le monde!
```

BEGIN/END

- En pratique, le corps d'une procédure commence par un BEGIN et se termine par un END.

- [begin_label:] BEGIN statement(s) END [end_label]

```
mysql> delimiter |
mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
-> BEGIN
-> SELECT COUNT(*) INTO param1 FROM t;
-> END
-> |
```

CREATE PROCEDURE / FUNCTION

- **CREATE PROCEDURE** sp_name ([paramètre [...]]) [caractéristique ...] routine
- **CREATE FUNCTION** sp_name ([paramètre [...]]) [RETURNS type] [caractéristique ...] routine
- paramètre : [IN | OUT | INOUT] nom
- type : un type MySQL
- caractéristique : **LANGUAGE SQL** | [NOT] DETERMINISTIC | SQL SECURITY {DEFINER | INVOKER} | COMMENT string
- routine : Commande(s) SQL valide(s)

PARAMETRES

- 3 modes:
 - IN : mode par défaut. La procédure stockée/fonction ne modifie pas le paramètre
 - OUT : la procédure stockée ne lit pas le paramètre, mais le modifie
 - INOUT: la procédure stockée lit et modifie le paramètre
- Exemple:

```
DELIMITER //
CREATE PROCEDURE GetOfficeByCountry(IN countryName
VARCHAR(255))
BEGIN
SELECT city, phone FROM offices WHERE country =
countryName;
END //
DELIMITER ;
```

EXAMPLE

```
DELIMITER $$  
CREATE PROCEDURE CountOrderByStatus(  
    IN orderStatus VARCHAR(25),  
    OUT total INT)  
BEGIN  
    SELECT count(orderNumber)  
    INTO total  
    FROM orders  
    WHERE status = orderStatus;  
END$$  
DELIMITER ;  
  
CALL CountOrderByStatus('Shipped',@total);
```


Les variables

- **DECLARE : déclarer une variable locale:**
 - `DECLARE var_name[,...] type [DEFAULT value]`
- **Commande d'affectation de variables SET**
 - `SET variable = expression [,...]`
- **Syntaxe de SELECT ... INTO**
 - `SELECT column[,...] INTO variable[,...] table_expression`

Exemple

```
DELIMITER $$
CREATE PROCEDURE `Capitalize` (INOUT str VARCHAR(1024))
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE myc, pc CHAR(1);
    DECLARE outstr VARCHAR(1000) DEFAULT str;
    WHILE i <= CHAR_LENGTH(str) DO
        SET myc = SUBSTRING(str, i, 1);
        SET pc = CASE WHEN i = 1 THEN ' '
                     ELSE SUBSTRING(str, i - 1, 1)
                END;
        IF pc IN (' ', '&', '!', ' ', '_', '?', ';', ':', '!', ',', '-', '/', '(', '.') THEN
            SET outstr = INSERT(outstr, i, 1, UPPER(myc));
        END IF;
        SET i = i + 1;
    END WHILE;
    SET str = outstr;
END$$
DELIMITER ;

SET @str = 'mysql stored procedure tutorial';CALL Capitalize(@str);SELECT
@str;
```

Instructions de contrôle: IF

- **Commande IF:**
 - IF expression THEN commands [ELSEIF expression THEN commands] [ELSE commands]
END IF;
- Les commandes associées au THEN sont exécutées si l'expression est vraie.
- Les commandes associées au ELSE sont exécutées si l'expression est fausse.

Attention à vérifier que l'expression n'est jamais NULL!

Instructions de contrôle: CASE

- Il s'agit d'un IF multiple:

```
CASE case_expression
```

```
  WHEN when_expression THEN commands
```

```
  WHEN when_expression THEN commands ... ELSE commands
```

```
END CASE;
```

Instructions de contrôle

- **Commande REPEAT:**
 - REPEAT statement(s) UNTIL search_condition END REPEAT
- **WHILE**
 - WHILE search_condition DO statement(s) END WHILE

Exemple: WHILE

```
DELIMITER $$
DROP PROCEDURE IF EXISTS WhileLoopProc$$
CREATE PROCEDURE WhileLoopProc()
BEGIN
    DECLARE x INT;
    DECLARE str VARCHAR(255);
    SET x = 1;
    SET str = "";
    WHILE x <= 5 DO
        SET str = CONCAT(str,x,',');
        SET x = x + 1;
    END WHILE;
    SELECT str;
END$$
DELIMITER ;
```

Exemple : REPEAT

```
DELIMITER $$
DROP PROCEDURE IF EXISTS RepeatLoopProc$$
CREATE PROCEDURE RepeatLoopProc()
BEGIN
    DECLARE x INT;
    DECLARE str VARCHAR(255);
    SET x = 1;
    SET str = '';
    REPEAT
        SET str = CONCAT(str,x,',');
        SET x = x + 1;
    UNTIL x > 5
    END REPEAT;
    SELECT str;
END$$
DELIMITER ;
```

Curseurs

- Un curseur est une ligne d'une requête
- Des curseurs simples sont supportés dans les routines

A	B	C	D
A	B	C	D
Aa	Bb	Cc	Dd
Aaa	Bbb	Ccc	Ddd



Le curseur est une ligne d'une requête. Ce dernier peut "avancer" ou "reculer" dans le résultat de la requête.

Curseurs

- **Déclaration des curseurs**
 - `DECLARE cursor_name CURSOR FOR sql_statement`
- **Commande de curseur OPEN**
 - `OPEN cursor_name`
- **Commande de curseur FETCH**
 - `FETCH cursor_name`
- **Commande de curseur CLOSE**
 - `CLOSE cursor_name`

HANDLER

- Permet de gérer une « erreur »

```
DECLARE handler_action HANDLER FOR condition_value [, condition_value] ...  
statement
```

handler_action: CONTINUE | EXIT | UNDO

condition_value: *mysql_error_code* | SQLSTATE [VALUE] *sqlstate_value* |
condition_name | SQLWARNING | NOT FOUND | SQLEXCEPTION

Exemple

```
DECLARE stop INT DEFAULT 0;  
DECLARE cur_product CURSOR FOR SELECT productCode FROM products;  
DECLARE CONTINUE HANDLER FOR NOT FOUND SET stop= 1;  
...  
...  
REPEAT  
...  
UNTIL stop = 1  
END REPEAT;
```

Curseurs/Handler

```
DELIMITER $$
DROP PROCEDURE IF EXISTS CursorProc$$
CREATE PROCEDURE CursorProc()
BEGIN
  DECLARE no_more_products, quantity_in_stock INT DEFAULT 0;
  DECLARE prd_code VARCHAR(255);
  DECLARE cur_product CURSOR FOR SELECT productCode FROM products;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_products = 1;
  OPEN cur_product;
  FETCH cur_product INTO prd_code;
  REPEAT
  SELECT quantityInStock INTO quantity_in_stock FROM products
  WHERE productCode = prd_code;

  IF quantity_in_stock < 100 THEN INSERT INTO infologs(msg) VALUES
  (prd_code); END IF;
  FETCH cur_product INTO prd_code;
  UNTIL no_more_products = 1
  END REPEAT;
  CLOSE cur_product;
END$$
```



SQL procédural

Triggers

- Contenu
 - Qu'est ce qu'un trigger?
 - Comment programmer un trigger?
- Objectifs
 - Cette fonctionnalité est cruciale et complexe.
 - Le but de cette section est de mettre le « pied » à l'étrier



Déclencheur/Triggers

- Un déclencheur est un objet de base de données nommé, qui est associé à une table et qui s'active lorsqu'un événement particulier survient dans une table.
- Utilisation:
 - Intégrité de la base de données
 - Sécurité

Déclencheur/Triggers

- Il s'agit d'une fonctionnalité importante des Bases de données.
- Disponible depuis MySQL 5.0.2 avec des limitations:
 - Il n'est pas possible d'utiliser une procédure stockée dans un triggers
 - Il n'est pas possible de définir une procédure stockée pour une vue ou une table temporaire
 - Il n'est pas possible de faire des transactions dans un trigger
 - Il n'y a pas de « return »
 - Invalide les caches de requêtes

CREATE TRIGGER

- CREATE TRIGGER *trigger_name* *trigger_time* *trigger_event* ON *tbl_name* FOR EACH ROW *trigger_stmt*
- *trigger_time* est le moment d'action du déclencheur: BEFORE (avant) ou AFTER (après)
- *trigger_event*: INSERT, UPDATE ou DELETE
- *trigger_stmt* : commande à exécuter lorsque le déclencheur s'active
- Deux variables implicites NEW et OLD correspondant aux nouvelles/anciennes valeurs de la ligne

Exemple simple

```
CREATE TABLE `users` (  
  `nom`      varchar(20),  
  `prenom`   varchar(20),  
  `password` varchar(20)  
) ENGINE = InnoDB;  
DELIMITER |  
  
CREATE DEFINER = 'root'@'localhost' TRIGGER `insert_password`  
  BEFORE INSERT  
  ON `users`  
  FOR EACH ROW  
  BEGIN  
    SET NEW.password=MD5(NEW.password);  
  END|  
  
DELIMITER ;
```

Exemple simple

```
insert into users values ('moi','toi','password');
select * from users;
mysql> select * from users;
+-----+-----+-----+
| nom | prenom | password |
+-----+-----+-----+
| moi | toi | 5f4dcc3b5aa765d61d83 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

Exemples: les traces

- Tracer les modifications d'une table d'employés :

```
DELIMITER $$
CREATE TRIGGER before_employee_update
BEFORE UPDATE ON employees
FOR EACH ROW BEGIN
INSERT INTO employees_audit
SET action = 'update',
employeeNumber = OLD.employeeNumber,
lastname = OLD.lastname,
changedon = NOW(); END$$
DELIMITER ;
```

Connexions, droits d'accès, sécurité

- Introduction et rappels
- Pratique du SQL avec MySQL
- Modèle relationnel, conception et création d'une base
- Tables transactionnelles InnoDB
- SQL procédural
- **Connexions, droits d'accès, sécurité**
 - Niveaux de privilèges et vérification des droits.
 - Gestion des utilisateurs et de leurs privilèges.
 - Sécurisation des procédures stockées et des vues.
- Introduction à l'administration

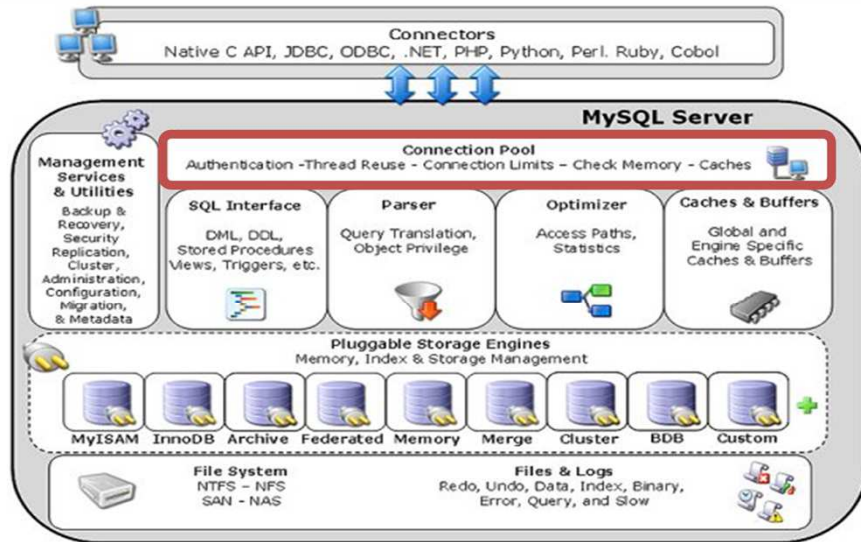


Connexions, droits d'accès, sécurité

Gestion des utilisateurs et de leurs privilèges.

- Contenu
 - Comment créer un utilisateur?
 - Comment MySQL gère la sécurité des connexions
 - Qu'est ce qu'un privilège?
- Objectifs
 - Créer un utilisateur
 - Gérer les habilitations

Connexion et droits



Niveaux de privilèges

- Le système de droits de MySQL s'assure que les utilisateurs font exactement ce qu'ils sont supposés pouvoir faire dans la base. Lorsque vous vous connectez au serveur, votre identité est déterminée par :
 - *l'hôte d'où vous vous connectez :*
 - *le nom d'utilisateur que vous spécifiez.*
 - Le système donne les droits en fonction de votre identité et de *ce que vous voulez faire.*

Niveaux de privilèges

- Le contrôle d'accès de MySQL se fait en deux étapes :
- Etape 1 : Le serveur vérifie que vous êtes autorisé à vous connecter.
- Etape 2 : En supposant que vous pouvez vous connecter, le serveur vérifie chaque requête que vous soumettez, pour vérifier si vous avez les droits suffisants pour l'exécuter. Par exemple, si vous sélectionnez des droits dans une table, ou effacez une table, le serveur s'assure que vous avez les droits de SELECT pour cette table, ou les droits de DROP, respectivement.

Niveaux de privilèges

- Les privilèges sont stockés dans la base Mysql dans 5 grandes tables:
 - **User** : contient les comptes utilisateurs ainsi que les privilèges globaux sur la totalité des bases
 - **Db** : contient les privilèges sur d'un utilisateur sur une base de données particulière
 - **Table_priv, column_priv**: contient les privilèges sur une table et une colonne particulière
 - **Procs_priv**: contient les privilèges sur les procédures stockées

Création d'un utilisateur

- La création se fait via la commande:
 - `CREATE USER user IDENTIFIED BY password`
- Exemple:
 - `CREATE USER 'dbadmin'@'localhost' IDENTIFIED BY 'CrEate-User';`
 - `CREATE USER 'superadmin'@'%' IDENTIFIED BY 'Secured';`
- Il est possible "d'attaquer" directement la table user:
 - `INSERT INTO user
(host,user,password)VALUES('localhost','dbadmin',PASSWORD('CrEate-User'));`

Mise à jour d'un mot de passe

- Via une requête UPDATE

```
USE mysql;
```

```
UPDATE user SET password = PASSWORD('Secret1970')WHERE user = 'mysqletutorial'
```

```
AND host = 'mysqletutorial.org';
```

```
FLUSH PRIVILEGES;
```

- Via une requête PASSWORD

```
SET PASSWORD FOR 'mysqletutorial'@'mysqletutorial.org' = PASSWORD('Secret1970')
```

Commande GRANT

- La commande GRANT :
 - `GRANT privileges (column_list)ON [object_type] privilege_levelTO account [IDENTIFIED BY 'password'] [REQUIRE encryption] WITH with_options`
- Privilèges_level: le type de privilèges à fournir
- Column_list: Une liste de colonnes
- Account : le compte utilisateur

Commande GRANT

- CREATE USER 'super'@'localhost' IDENTIFIED BY 'SecurePass1';
- GRANT ALL ON *.* TO 'super'@'localhost' WITH GRANT OPTION;

- CREATE USER 'rfc'@'%' IDENTIFIED BY 'SecurePass3';
- GRANT SELECT, UPDATE, DELETE ON classicmodels.* TO 'rfc'@'%';

Type de privilèges

Privilège	Description
ALL	Tous les privilèges
ALTER	Autorise les requêtes ALTER
CREATE	Permet de créer des tables
CREATE USER	Permet de créer des utilisateurs
CREATE VIEW	Permet de construire des vues
EXECUTE	Permet d'exécuter les store proc
INSERT	Autorise les INSERT
TRIGGER	Autorise les triggers
UPDATE	Permet de faire des Update

En pratique

Server Access Management | Schema Privileges

User	From Host
-anonymo...	localhost
sql_job1	localhost
root	localhost
root	127.0.0.1
root	:::1

Details for account sql_job1@localhost

Login | Administrative Roles | Account Limits

Login Name: You may create multiple accounts with the same name to connect from different hosts.

Limit Connectivity to Hosts Matching: % and _ wildcards may be used.

Password: Type a password to reset it.

Confirm Password: Enter password again to confirm.

En pratique

Server Access Management | Schema Privileges

User Accounts

User	From Host
-anonymo.	localhost
sql_lab1	localhost
root	localhost
root	127.0.0.1
root	:::1

Details for account sql_lab1@localhost

Login | Administrative Roles | Account Limits

Role	Description
<input type="checkbox"/> DBA	grants the rights to perform all tasks
<input checked="" type="checkbox"/> MaintenanceAdmin	grants rights needed to maintain server
<input checked="" type="checkbox"/> ProcessAdmin	rights needed to assess, monitor, and k
<input checked="" type="checkbox"/> UserAdmin	grants rights to create users logins and r
<input checked="" type="checkbox"/> SecurityAdmin	rights to manage logins and grant and r
<input checked="" type="checkbox"/> MonitorAdmin	minimum set of rights needed to monit
<input checked="" type="checkbox"/> DBManager	grants full rights on all databases
<input checked="" type="checkbox"/> DBDesigner	rights to create and reverse engineer an
<input checked="" type="checkbox"/> ReplicationAdmin	rights needed to setup and manage repl
<input checked="" type="checkbox"/> BackupAdmin	minimal rights needed to backup any da

Global Privileges Assigned to User

- ALTER
- ALTER ROUTINE
- CREATE
- CREATE ROUTINE
- CREATE TABLESPACE
- CREATE TEMPORARY TABLES
- CREATE USER
- CREATE VIEW
- DELETE
- DROP
- EVENT
- EXECUTE
- FILE
- GRANT OPTION
- INDEX
- INSERT
- LOCK TABLES
- PROCESS
- REFERENCES
- RELOAD
- REPLICATION CLIENT
- REPLICATION SLAVE
- SELECT
- SHOW DATABASES
- SHOW VIEW
- SHUTDOWN

Add Account Remove Revoke All Privileges Revert Apply Refresh

En pratique

Server Access Management Schema Privileges

Select a user and pick the privileges it has for a given Schema and Host combination.

Host	Schema	Privileges
%	photo_browser	none

Schema and Host fields may use % and _ wildcards.
The server will match specific entries before wildcarded ones.

The user 'sql_lab1', when connecting from any host, will have the following access rights to schemas matching 'photo_browser':

Object Rights	DDL Rights	Other Rights
<input type="checkbox"/> SELECT	<input type="checkbox"/> CREATE	<input type="checkbox"/> GRANT OPTION
<input type="checkbox"/> INSERT	<input type="checkbox"/> ALTER	<input type="checkbox"/> CREATE TEMPORARY TABLES
<input type="checkbox"/> UPDATE	<input type="checkbox"/> REFERENCES	<input type="checkbox"/> LOCK TABLES
<input type="checkbox"/> DELETE	<input type="checkbox"/> INDEX	
<input type="checkbox"/> EXECUTE	<input type="checkbox"/> CREATE VIEW	
<input type="checkbox"/> SHOW VIEW	<input type="checkbox"/> CREATE ROUTINE	
	<input type="checkbox"/> ALTER ROUTINE	
	<input type="checkbox"/> DROP	
	<input type="checkbox"/> TRIGGER	

Select "ALL" Unselect All Revert Save Changes

Introduction à l'administration

- Introduction et rappels
- Pratique du SQL avec MySQL
- Modèle relationnel, conception et création d'une base
- Tables transactionnelles InnoDB
- SQL procédural
- Connexions, droits d'accès, sécurité
- **Introduction à l'administration**
 - Exportation de données.
 - Sauvegardes, la commande mysqldump.
 - Replication
 - Clusterisation



Introduction à l'administration

Sauvegardes, la commande mysqldump.

- Contenu
 - La commande mysqldump
 - Backup/Restore d'une base de données
- Objectifs
 - Savoir faire une sauvegarde d'une base de données
 - Savoir restaurer une base de données

MySQL Dump

- Utilitaire qui permet d'exporter une base ou un groupe de bases vers un fichier texte, pour la sauvegarde ou le transfert entre deux serveurs (pas nécessairement entre serveurs MySQL). L'export contiendra les requêtes SQL nécessaires pour créer la table et la remplir.
- shell> **mysqldump [options] db_name [tables]** shell> **mysqldump [options] --databases DB1 [DB2 DB3...]** shell> **mysqldump [options] --all-databases**
- **mysqldump --help.**

MySQL Dump

- `--add-drop-table`
- Ajoute une commande drop table avant chaque requête de création de table.
- `--all-databases, -A`
- Exporte toutes les tables. C'est l'équivalent de l'option `--databases` avec toutes les bases de données sélectionnées.
- `--compatible=name` : Produit un résultat qui est compatible avec les autres bases de données, ou avec d'anciennes versions de MySQL. Les valeurs possibles de name sont `mysql323`, `mysql40`, `postgresql`, `oracle`, `mssql`,

MySQL Dump

- `--complete-insert, -c`
- Utilise des commandes INSERT complètes, avec les noms de colonnes.
- `--host=host_name, -h host_name`
- Exporte les données depuis le serveur MySQL vers l'hôte indiqué. L'hôte par défaut est localhost.
- `--lock-tables, -l`
- Verrouille toutes les tables avant de commencer l'export. Les tables sont verrouillées avec READ LOCAL pour permettre des insertions concurrentes sur les tables MyISAM.
- `--password`
- `--user=user_name`

MySQL Dump

- Sauvegarde
 - `mysqldump --databases database1 [database2 ...] > my_databases.sql`
- Restauration
 - `mysql database < backup-file.sql`

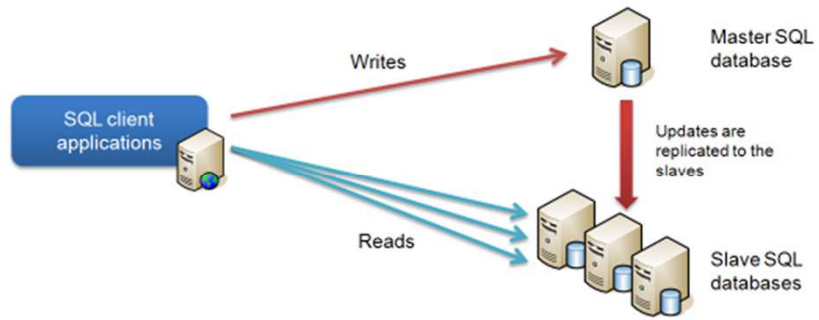


Introduction à l'administration

Mise en place de la réplication

- **Contenu:**
 - Qu'est ce que la réplication
 - Paramètres des répliques
- **Objectifs:**
 - Utilisation de la réplication pour optimiser la montée en charge

Réplication



Mise en place

- Définition d'un MySQL maître (pour les écritures)
 - Mise en place d'un « user » dans le maître pour le MySQL esclave
 - Bloquer les écritures sur le maître
 - Faire un dump du maître
- Définition d'un MySQL esclave
 - Mise en place des paramètres du MySQL maître
 - Rattrapage des données
 - Libérer les écritures sur le maître

Master (My.ini)

- Définition d'un MySQL maître (pour les écritures) (Windows)

```
[mysqld]
basedir=D:/MyProjects/MySQL Replication/mysql-5.5.32-winx64-master
datadir=D:/MyProjects/MySQL Replication/mysql-5.5.32-winx64-
master/data
port                = 3306
log-bin=mysql-bin
server-id = 1
binlog-do-db=clone
```

- Démarrage serveur

```
.\bin\mysqld --defaults-file=my.ini --console
```

Master (SQL)

- Définition d'une Base, Table et d'un utilisateur pour l'esclave

```
CREATE DATABASE CLONE;
CREATE TABLE A (col1 VARCHAR(20))
INSERT INTO A values ('maman')

GRANT REPLICATION SLAVE ON *.* TO replicateur@'localhost'
IDENTIFIED BY 'replicateur';

GRANT REPLICATION SLAVE ON *.* TO replicateur@'%' IDENTIFIED BY
'replicateur';
```

- Dump (après lock des tables)
- Récupération des informations de logs

```
mysql> show master status;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000002 |      1143 | clone        |                   |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

Esclave (My.ini)

- Définition d'un MySQL esclave (pour les lecture) (Windows)

```
[mysqld]
basedir=D:/MyProjects/MySQL Replication/mysql-5.5.32-winx64-slave
datadir=D:/MyProjects/MySQL Replication/mysql-5.5.32-winx64-
slave/data
port                = 3307
log-bin=mysql-bin
server-id = 2
```

- Démarrage serveur

```
.\bin\mysqld --defaults-file=my.ini --console
```

Esclave (SQL)

- Définition de la base, rattrapage avec le dump
- Définition du master

```
mysql> change master to MASTER_HOST='localhost', MASTER_PORT=3306,  
MASTER_USER='replicateur', MASTER_PASSWORD='replicateur',  
MASTER_LOG_FILE='mysql-bin.000002',MASTER_LOG_POS=1143;
```

- Redémarrage de l'esclave

```
130614 12:28:58 [Note] .\bin\mysqld: ready for connections.  
Version: '5.5.32-log' socket: '' port: 3307 MySQL Community  
Server (GPL)  
130614 12:28:58 [Note] Slave I/O thread: connected to master  
'replicateur@localh  
ost:3306',replication started in log 'mysql-bin.000002' at  
position 1143
```

Test

- Ecriture dans le master
- Lecture sur l'esclave



Introduction à l'administration

Mise en place d'un cluster

- **Contenu:**
 - Voir les étapes d'un mini cluster
- **Objectifs:**
 - Se sensibiliser aux clusters mysql



MySQL Cluster

- 3 type de nœuds:
 - Nœud de management
 - Nœuds de données
 - Nœuds SQL

- Dans notre cas, 2 nœuds de données (mysql_data1 et mysql_data2), 2 nœuds SQL (mysql_mysqlid1,mysql_mysqlid2), 1 nœud de management (mysql_management)

Pour chaque serveur

- Par de sécurité (c'est assez complexe comme cela):
 - SELINUX=disabled
 - service iptables stop
 - chkconfig iptables off
- Mise en place de fichiers hosts:
 - mysql-mysqld1 192.168.1.1
 - mysql-mysqld2 192.168.1.2
 - mysql-management 192.168.1.3
 - mysql-data1 192.168.1.4
 - mysql-data2 192.168.1.5

Nœud de management

- MySQL-Cluster-gpl-management-7.0.35-1.rhel5.x86_64.rpm
- MySQL-Cluster-gpl-tools-7.0.34-1.rhel5.x86_64.rpm
- Dans /usr/local/src faire:
 - rpm -Uvh MySQL-Cluster-gpl-management-7.0.35-1.rhel5.x86_64.rpm
 - rpm -Uvh MySQL-Cluster-gpl-tools-7.0.34-1.rhel5.x86_64.rpm
 - mkdir -p /var/lib/mysql-cluster
 - Touch /var/lib/mysql-cluster/config.ini

Fichier de configuration du manageur

```
[ndb_mgmd default]
DataDir=/var/lib/mysql-cluster

[ndb_mgmd] HostName=mysql-management

[ndbd default] NoOfReplicas=2
DataMemory=256M IndexMemory=128M
DataDir=/var/lib/mysql-cluster

[ndbd] HostName=mysql-data1

[ndbd] HostName=mysql-data2

[mysqld] HostName=mysql-mysqld1

[mysqld] HostName=mysql-mysqld2
```

Nœud de données

- MySQL-Cluster-gpl-storage-7.0.35-1.rhel5.x86_64.rpm

- Fichier My.cnf

```
[mysqld]
```

```
ndbcluster
```

```
ndbcluster ndb-connectstring=mysql-management
```

```
[mysql_cluster]
```

```
ndb-connectstring=mysql-management
```

Nœud de SQL

- MySQL-Cluster-client-gpl-7.2.8-1.el6.x86_64.rpm
- MySQL-Cluster-shared-gpl-7.2.8-1.el6.x86_64.rpm
- MySQL-Cluster-server-gpl-7.2.8-1.el6.x86_64.rpm

- Fichier My.cnf

```
[mysqld]
```

```
ndbcluster ndb-connectstring=mysql-management
```

```
default_storage_engine=ndbcluster
```

```
[mysql_cluster]
```

```
ndb-connectstring=mysql-management
```

Démarrage

- Sur le nœud de management :

- `ndb_mgmd -f /var/lib/mysql-cluster/config.ini`

```
MySQL Cluster Management Server mysql-5.1.63 ndb-7.0.35
```

```
2013-06-24 23:36:55 [MgmtSrvr] INFO -- The default config directory '/usr/mysql-cluster' does not exist. Trying to create it...
```

```
2013-06-24 23:36:55 [MgmtSrvr] INFO -- Successfully created config directory
```

Démarrage

- Sur les nœud de données:

- Ndbd --initial

MySQL Cluster Management Server mysql-5.1.63 ndb-7.0.35

2013-06-24 23:46:55[ndbd] INFO – Pilou connected to 'mysql-management:1186'

2013-06-24 23:46:55 [MgmtSrvr] INFO – Pilou allocated nodeid: 2