MySQL - Mise en oeuvre, configuration, administration

François Gannaz – INP Grenoble Formation Continue

Introduction aux bases de données et à MySQL

Qu'est-ce qu'une base de données ?

Plusieurs sens suivant le contexte :

- Un **jeu de données** particulier
  Ex : les données d'une application web de blog
  (peut représenter des millions d'enregistrements)
- Les fichiers qui contiennent ces données
- Le **système** qui les gère
  Ex : MySQL

Eviter les abus...

- Le logiciel qui gère des données
  Ex : une application de gestion de bibliothèque

On parlera pour cela de **client** de base de données.

Le monde des bases de données (SGBD)

Les applications bureautiques "tout-en-un"

- File Maker Pro
- MS Access
- ...

Les systèmes de bases de données

- Souvent de structure **client-serveur**
- Presque toujours de modèle **relationnel**
- langage standard pour l'accès : **SQL** (Structured Query Language)
- API dans des langages de programmation divers

Les BdD Orientées Objet (donc non relationnelles) sont rares.

Principaux SGBDR du marché

SGBDR propriétaires

- Oracle
- DB2 (IBM)
- SQL Server (MS)

SGBDR libres

- MySQL
- PostgreSQL
- SQLite (embarqué et non client-serveur)
- Firebird (fork de Borland InterBase)

MySQL c'est...

- une base de donnée **relationnelle** créée en 1995
- modèle **client-serveur**
- une application **légère** dans le monde des SGBD
- développée par une société suédoise (ABSoft)
  Rachetée par Sun Microsytems début 2008.
- Le plus répandu des SGBDR libres
  Particulièrement utilisé pour le web (LAMP)
- diffusée sous **double licence**
  - libre (GPL) pour un usage interne ou libre
  - propriétaire payant pour un usage propriétaire
- Principales versions :
  - 4.1 stable depuis octobre 2004
  - 5.0 stable depuis octobre 2005
  - 5.1 stable depuis novembre 2008

Caractéristiques de MySQL

Avantages

- Multi plates-formes : Linux, Windows, OSX, etc.
- Gratuit pour un usage libre ou non commercial
- Bonne documentation de référence (HTML, PDF)
  `http://dev.mysql.com/doc/refman/5.0/en/`
- SGBD performant
- Plusieurs moteurs internes suivant les besoins
- Interfacée avec la plupart des langages de programmation

Inconvénients

- Partiellement conforme au standard SQL:2003
- Quelques fonctionnalités absentes ou très faibles :
  - analyse à la volée (OLAP)
  - traitement du XML
  - données géographiques (GIS)
  - triggers (déclencheurs) et curseurs
  - ...

Installation

Installation Windows - 1

Composants

- MySQL Windows
  - Serveur MySQL (mysqld)
  - Clients console : shell (mysql)
  - Clients console : utilitaires (mysqladmin, mysqldump...)
  - Instance Manager (obsolète)
- MySQL GUI Tools (optionnel)
  - MySQL Administrator
  - MySQL Query Browser
  - MySQL Migration Toolkit
    Migration d'un SGBD étranger vers MySQL
- MySQL Workbench (optionnel)
  Conception et diagrammes des bases MySQL

Installation Windows - 2

Structure des répertoires

`C:\Program Files\mySQL\MySQL Server 5.0`

- bin : les exécutables binaires
- data : les fichiers bases de données
- docs
- examples
- include : en-têtes pour la programmation C
- lib : les bibliothèques dynamiques
- **configuration** "`my.ini`" dans `C:\windows`
  (fichier nommé "`my.cnf`" sous Linux)

Gestion des services

Par le gestionnaire de services de Windows

Installation WAMPServer

- Un pack de logiciels libres configurés ensemble
  - Apache : serveur Web
  - MySQL : serveur de base de données
  - PHP : langage de programmation web
  - PHPMyAdmin : interface web de gestion de MySQL, écrite en PHP
  - SQLiteManager
- réalisé par Anaska (sté française), sous licence GPL v2.0
- concurrents : EasyPHP, xAMP...

Parcourir l'arborescence installée : trouver les fichiers de configuration de Apache, MySQL, phpMyAdmin

L'architecture client-serveur

Réseau : utilisation du protocole IP

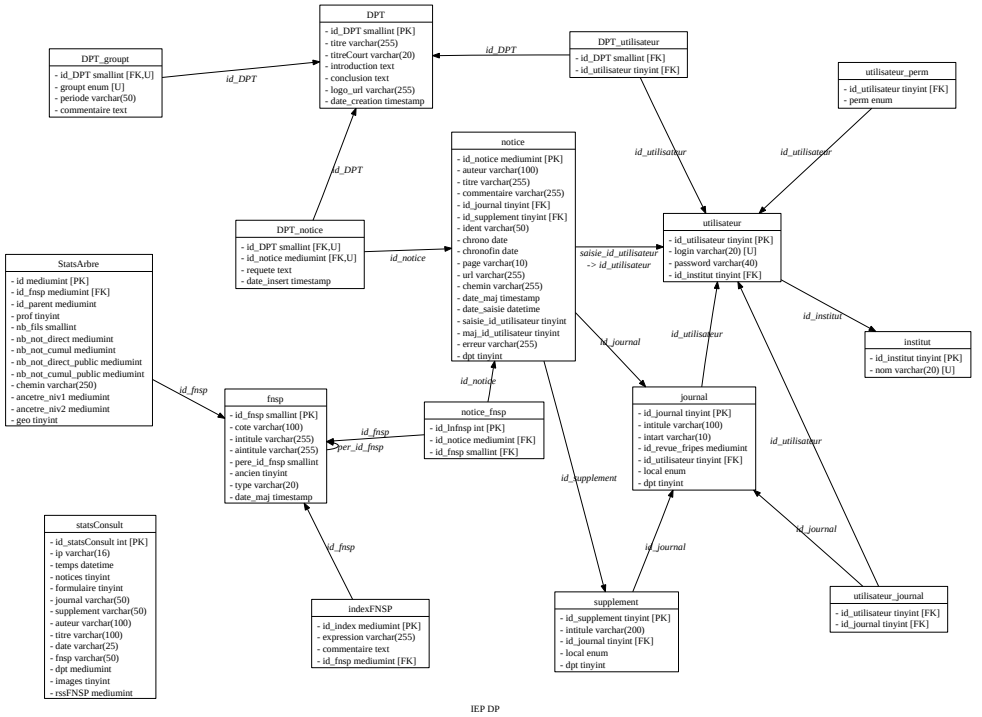
- une adresse IP, ex. 192.168.1.100
- un nom de machine, ex. pc101-01.cuefa.inpg.fr
- un port (=protocole) ; 3306 par défaut pour MySQL

Cas particulier : client-serveur en local

- localhost : IP=127.0.0.1 (universel)
- utilisation des canaux nommés (Windows NT...)
- utilisation des sockets Unix

Modélisation d'une base de données

Le modèle relationnel



IEP DP

Les tables

Une base de données (par ex. *discotheque*) est faite de **tables**.

Table **Disques**		
Titre	**Compositeur**	**Date**
Cantates	Bach J.S.	2006
Sonates	Beethoven	2005
Concerto	Dvořak	2000

Chaque ligne est un **enregistrement** (ou tuple, ou n-uplet).
Le nom d'une colonne est dit **champ** (ou **attribut**).

Les colonnes sont typées

Numérique	BOOLEAN, INT, DOUBLE ...
Texte	VARCHAR(taille), TEXT ...
Listes	ENUM(liste), SET(liste)
Date/Heure	DATE, TIMESTAMP ...

Conception d'une base de données

Les données de l'exemple

Partita, *Bach & Busoni*, Harmonia Mundi, 1986.
Concerto, *Dvořak*, Sony, 1980.

Première étape (normalisation 0)

Lister les données à stocker
Les structurer en entités–attributs (tables–champs) avec une information par champ.

Application à l'exemple

Comment organiser ces données ?

Disques
titre
compositeur**s**
label
date de sortie

Disques
titre
compositeur1
compositeur2
label
date de sortie

Première forme normale

Avant

titre	compositeur1	compositeur2	label	date
Partita	Bach	Busoni	Harmonia	1986
Concerto	Dvořak		Sony	1980

Règles de normalisation

- Une table pour chaque groupe de données associées,
- Pas de colonnes au contenu similaire,
- Chaque enregistrement doit avoir une **clé primaire** (identifiant unique).

Application à l'exemple

id	titre	compositeur	label	date
1	Partita	Bach	Harmonia	1986
2	Partita	Busoni	Harmonia	1986
3	Concerto	Dvořak	Sony	1980

Disques
id [PK]
titre
compositeur
label
date de sortie

Deuxième forme normale

Règles de normalisation

- Si plusieurs lignes ont des contenus similaires, la table doit être découpée en sous-tables,
- Ces tables doivent être reliées par des **clés étrangères** (référence à une clé primaire).

Application à l'exemple

Disques
id_disque
titre
label
date de sortie

⟵

Compositeurs
id_compositeur
id_disque [FK]
nom

Doublons dans la table *Compositeurs* $\Longrightarrow$ **Normalisation ratée !**

Deuxième forme normale : application

Il faut créer une **table de relation** entre les tables *Disques* et *Compositeurs*.

Disques
id_disque
titre
label
date de sortie

⟵

Rel_Disq_Comp
id_disque
id_compositeur

⟶

Compositeurs
id_compositeur
nom

Un compositeur n'est défini qu'une seule fois, mais peut être mis en relation avec plusieurs disques.

Troisième forme normale

Règle de normalisation

- Les colonnes qui ne sont pas intrinsèquement liées à la clé primaire doivent être dans une table séparée.

Application à l'exemple

La colonne *Disques.label* contredit la règle.

Labels
id_label
nom

⟵

Disques
id_disque
titre
id_label
date de sortie

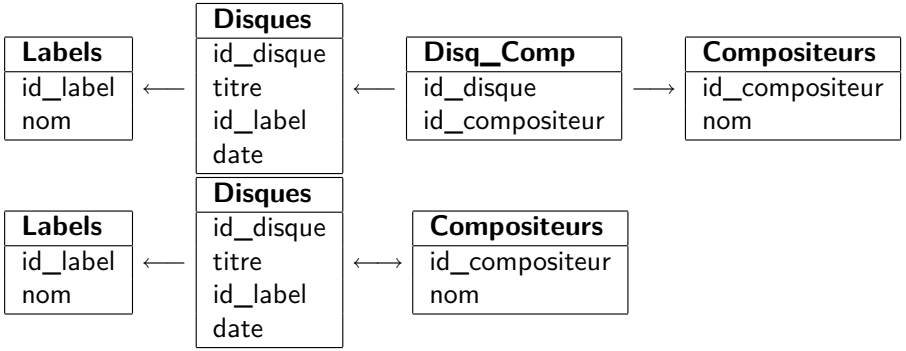
Le modèle relationnel

On peut lier les tables par des **relations**, classées en 3 types.

Chaque **clé étrangère** induit une relation entre 2 tables.

Un Diagramme Entité-Relation (**ERD**) est une aide précieuse.

Exemple de la gestion d'une liste de CD



Relations (1)

Relation 1:1

Chaque élément de la première table est lié à **au plus un** élément de la seconde, et **réciproquement**.

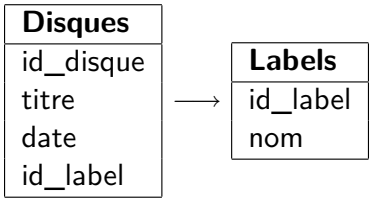
Cette relation est rare, elle scinde une table sans normalisation.

Relation 1:N

Chaque élément de *Disques* est lié à **au plus un** élément de *Labels*.

Et un élément de *Labels* peut correspondre à **plusieurs** disques.

$\Longrightarrow$ *Disques* a une clé étrangère sur la clé primaire de *Labels*.



Relations (2)

Relation N:M

Chaque élément de *Disques* est lié à **plusieurs** éléments de *Compositeurs*, et **réciproquement**.

Cette relation a besoin d'une table de relation avec 2 clés étrangères.

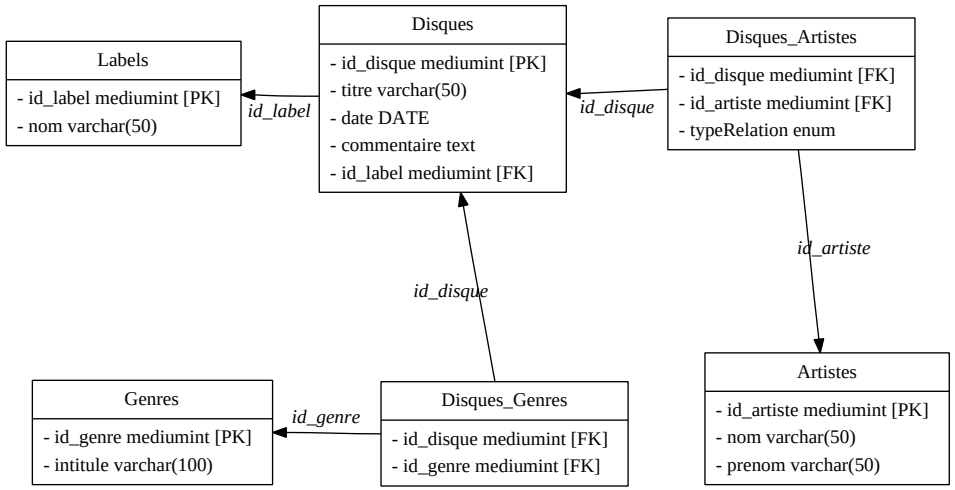
TP - Disques

Adapter le modèle de la base de données pour stocker des disques, chacun décrit avec les informations :

- titre
- date de sortie
- commentaire du diffuseur
- label
- interprètes
- compositeurs
- genres

Tracer un ERD de la base.

Schéma final pour l'exemple



Bonnes pratiques

- Normaliser est une recommandation générale.
  Parfois, il y a des exceptions (par ex. pour la performance).
- Prendre le temps de bien concevoir son modèle.
  Les changements structurels sur une base en production peuvent être calamiteux.
- Éviter en général les noms de type *objet1*, *objet2*.
  Utiliser une table dédiée.
- Tester sa base avec un jeu de données.
  Surtout si la performance est importante.
- Utiliser un schema de la base (ERD).

Conventions de nommage

Pas de convention universelle.
Il faut s'en fixer une et s'y tenir.

- Noms en ASCII
  On évite ainsi les problèmes d'encodage ("é" en latin1/utf-8/...)
- Fixer une règle sur le singulier et le pluriel
  Table *user* ou *users* ?
- Composition des mots dans les noms
  Champ *camelCase* ou *with_underscores* ?
- Attention aux majuscules !
  Sous Windows, le système de fichiers est indifférent à la casse
  $\Longrightarrow$ Les noms des bases et des tables sont concernés.

Clé primaire

- Déclarée dans la table avec le mot réservé **PRIMARY KEY**
- Chaque table devrait en avoir une (recommandé)
- Au plus 1 PK par table (obligation)
- Les valeurs sont forcément **uniques**
- PK le plus compacte possible pour être performante
  $\Longrightarrow$ presque toujours numérique (INT)
- Attribut **AUTOINCREMENT** pour que MySQL numérote automatiquement les nouveaux enregistrements
  `SELECT LAST_INSERT_ID()` renvoie la valeur utilisée pour la PK
- peut être construite à partir de 2 champs :
  `PRIMARY KEY (colonne1, colonne2)`

Clé étrangère et contraintes

Clé étrangère

- Champ qui référence une clé primaire d'une autre table
- Pas forcément explicite dans la structure

  MyISAM Pas de déclaration des clés étrangères

  InnoDB `FOREIGN KEY (parent_id) REFERENCES parent(id)`

Contraintes

Avec InnoDB, les clés étrangères sont déclarées.
Les modifications des données doivent conserver la cohérence.

En cas d'insertion d'une clé étrangère sans clé primaire associée :
`Cannot add or update a child row : a foreign key constraint fails`

TP - Agence immobilière (simplifiée)

Modéliser la situation suivante : on veut représenter l'organisation d'un ensemble d'immeubles en appartements et décrire les informations sur les propriétaires et les occupants.

- une personne occupe un seul appartement
- un appartement peut être occupé par plusieurs personnes (couples, colocataires)
- une personne peut posséder plusieurs appartements
- un appartement peut appartenir à plusieurs personnes (chacun avec quote-part)

Créer une base InnoDB adaptée et tracer son schema.

Approche objet

MySQL n'est pas un SGBD Orienté Objet...

Mais on peut émuler des fonctionnalités OO en relationnel.

Comment remplacer les disques par une gestion de médiathèque ?
Modéliser :

- les attributs communs aux (disques, livres, images)
- les attributs distincts

Outils MySQL et Interfaces Utilisateur

Prise en main

Les interfaces utilisateur

- Ligne de commande
  - la “console” mysql
  - les utilitaires : mysqldump, mysqladmin...
  - options communes :
    -u <user> -p<password> -h <hote> -P <port>...
- Les “clients lourds” graphiques
  - MySQL Administrator
  - MySQL Query Browser
- L’interface web
  - PhpMyAdmin : interface unifiée

La documentation

Le manuel de référence

- `http://dev.mysql.com/doc/`
- multiples versions, multiples langues
- mises à jour régulières
- attention à la synchro des versions

Formats :

- En PDF : imprimable...
- En format CHM (aide Windows)
- en ligne de commande (terminal) : `HELP ...` ;
- En ligne : HTML
  - Commentaires utilisateurs

Authentification

- Des comptes ad-hoc, indépendants de l’OS
  Un compte n’est rien sans des attributions de **privilèges**.
- Le login administrateur : root
  Pour changer le mot de passe root (à l’installation du serveur), 2 possibilités :
  - `mysqladmin -u root password <secret>`
  - `SET PASSWORD FOR root = PASSWORD('<secret>')`
- Gestion des utilisateurs
  - Création
    `CREATE USER username [IDENTIFIED BY '<secret>']`
  - Suppression
    `DROP USER user`

Premier contact - privilèges

- `SHOW DATABASES` ;
- `USE mysql` ;
- `SHOW TABLES` ;
- `SHOW PRIVILEGES` ;
- `DESCRIBE user` ;
- Puis `DESCRIBE` avec les tables : db, host, table-priv, column-priv

GRANT et REVOKE

GRANT Attribue des privilèges à un utilisateur (doc 12.5.1)
REVOKE Enlève des privilèges

Les différents niveaux auxquels s'appliquent les privilèges

- serveur (ou global)
- base de données
- table
- colonne

Restriction aussi sur l'hôte de provenance du client (IP ou nom)

Exemples

- Tous les privilèges sur une base :
  `GRANT ALL ON mabase.* TO 'paul'@'%.cuefa.fr' ;`
- Création du compte en même temps qu'un privilège global :
  `GRANT SELECT ON *.* TO 'nouveau'@'%' IDENTIFIED BY 'secret' ;`

Gestion de la structure de données

Au niveau global

- `CREATE DATABASE mabase ;`
- `DROP DATABASE mabase ;`

Au niveau base de données

- **CREATE TABLE** fournisseur (
  id **INT NOT NULL** auto_increment ,
  nom **VARCHAR**(255) **NOT NULL** ,
  url **VARCHAR**(255) **DEFAULT NULL** ,
  comment TEXT,
  PRIMARY KEY (id)
) ENGINE=InnoDB CHARSET=utf8 ;
- `DROP TABLE matable ;`
- `RENAME TABLE matable TO latable ;`
- `ALTER TABLE matable ...`

Les types de données

dualité représentation interne / affichage (ex. TINYINT(3))

- Entiers : INT, TINYINT, SMALLINT, MEDIUMINT, BIGINT
  Avec les options [UNSIGNED] [ZEROFILL]
- Décimaux : FLOAT, DOUBLE, DECIMAL
- Heure et date : DATE, TIME, DATETIME, TIMESTAMP, YEAR
- Texte : CHAR, VARCHAR(0 à 255), TEXT...
- Listes : ENUM('homme','femme'), SET('a','b','c')
- Extensions : SPATIAL...
- la valeur **NULL** (champ vide, ni '', ni 0)
  Tout champ de type quelconque admet ou interdit `NULL`.

Manipulation des données
Requêtes SQL

Lire des données : SELECT

SELECT renvoie une "table" : résultat en lignes/colonnes.

Syntaxe simplifiée

SELECT expression **FROM** matable **WHERE** condition ;

Une expression (et une condition) est composée de

constantes : 3.14, 'chaine'

attributs : date, nom

fonctions : CONCAT(nom,' ',prenom)

Exemples :

- `SELECT * FROM commandes ;`
- `SELECT numcommande FROM commandes WHERE date>'2006-01-01' ;`

Compléments sur SELECT

- **ORDER BY** : Trier les résultats
  - `SELECT * FROM articles ORDER BY nom ASC`
  - `SELECT * FROM articles ORDER BY prix DESC, nom ASC`
- **LIMIT** : Limiter le nombre de résultats
  - `SELECT * FROM articles LIMIT 3`
  - `SELECT * FROM articles LIMIT 6,3`
- **DISTINCT** : Supprimer tout doublon dans les résultats
  - `SELECT DISTINCT nom FROM clients`

Quelques fonctions

- opérateurs : $= < > != * / +$ etc.
- la comparaison de texte est sans casse et sans accents (interclassement par défaut)
- **LIKE** : chaînes contenant un motif donné
  `SELECT nom FROM clients WHERE prenom LIKE 'A%'`

Exercices

Sur la base facsys :

1. Trouver les articles de plus de 50 euros.
2. Lister les noms des articles, triés par prix. Les trier par catégorie, puis par stock pour une même catégorie.
3. Quelle différence entre `SELECT nom, idcategorie, description FROM categories` et `SELECT * FROM categories` ?
4. Afficher toutes les commandes de 2004. Les 3 commandes les plus récentes.
5. Que donne `SELECT COUNT(*) FROM articles` ? Quelle différence avec `SELECT COUNT(articles.codearticle) FROM articles` ?
6. Combien d'articles de squash a-t-on ?

Jointures

Le but : interroger plusieurs tables à la fois

Exemple :

```
SELECT articles.nom FROM articles
  JOIN categories
    ON articles.idcategorie = categories.idcategorie
  WHERE categories.nom = 'squash'
```

Variantes

- `SELECT a.nom FROM articles AS a JOIN categories AS c ON a.idcategorie=c.idcategorie WHERE c.nom LIKE 'squash'`
- `SELECT a.nom FROM articles a JOIN categories c USING (idcategorie) WHERE c.nom = 'squash'`
- implicite : `SELECT a.nom FROM articles a, categories c WHERE a.idcategorie = c.idcategorie AND c.nom = 'squash'`

Jointures : exemple

`SELECT * FROM Joueurs`

nom	id_pays
Federer	1
Nadal	2
Ferrer	2

`SELECT * FROM Pays`

id_pays	pays
1	Suisse
2	Espagne
3	France

`SELECT * FROM Pays JOIN Joueurs USING (id_pays)`

id_pays	pays	nom
1	Suisse	Federer
2	Espagne	Nadal
2	Espagne	Ferrer

Exercices

1. Quels articles ont été commandés par Pierre Durand ?
2. Combien d'articles ont été expédiés à Paris ?
3. Lister les clients ayant commandé au moins deux fois ?
   Au moins trois articles différents ?
4. Afficher tous les clients avec leurs articles associés ?
   Avec leur article le plus cher ?

Les jointures externes

2 types de jointures :

INNER JOIN = jointure standard
Correspondance de 2 tables sur une valeur commune

OUTER JOIN = jointure externe
l'une des deux tables est prioritaire (LEFT, RIGHT)
$\Longrightarrow$ toujours citée pour toutes ses valeurs

Exemple : liste des clients n'ayant jamais commandé

- `SELECT nom, prenom, numcommande FROM clients LEFT JOIN commandes USING (idclient)`
- ... `WHERE numcommande IS NULL`

Utilisations fréquentes : contrôle de cohérence d'une base, nettoyage

Travailler avec NULL

`http://dev.mysql.com/doc/refman/5.0/en/working-with-null.html`

- Origines
  - données : champs non remplis
  - certaines erreurs : 1/0
  - certaines fonctions : `OUTER JOIN, ROLLUP`...
- Impact
  - sur COUNT(col), mais pas sur COUNT(*)
- Traitement : logique tri-valuée (TRUE, FALSE, UNKNOWN)
  - comparaison : `val IS (NOT) NULL, ISNULL(val)`
  - comparaison : `val1 <=> val2` : prend en compte NULL
  - `IFNULL(v1, vdef)` : si v1 est NULL, remplacée par vdef
  - `NULLIF(val1, val2)` : retourne NULL si val1=val2
  - `COALESCE(v1,v2,...)` : retourne la première valeur non NULL

Les agrégats - GROUP BY

- But : regrouper les résultats par valeur de colonne(s)
- Processus :
  1. Partitionnement du résultat (GROUP BY)
  2. Calcul des agrégats (fonctions COUNT, MIN...)
  3. Filtrage : clause optionnelle `HAVING`
  4. Sous-totaux : clause optionnelle `WITH ROLLUP`
- Attention : distinguer `HAVING` et `WHERE`

Exemple

```
SELECT a.idcategorie, a.nom, a.codearticle,
SUM(quantite) AS TotArt, SUM(d.prix*d.quantite) AS
PrixTot
FROM articles a JOIN details d USING (codearticle)
JOIN commandes c USING (numcommande)
GROUP BY a.idcategorie, a.codearticle WITH ROLLUP
```


INSERT

Insérer une ligne dans une table

2 syntaxes directes :

- `INSERT INTO clients (idclient,nom,prenom) VALUES ('SOR01','Sorel','Julien'), ...`
  Permet d'insérer plusieurs enregistrements
- `INSERT INTO clients SET nom='Sorel', prenom='Julien'`
  Syntaxe commune avec UPDATE

Si un champ n'a pas de valeur :

- s'il est en AUTO_INCREMENT, il vaudra 1 de plus que le dernier (compteur interne)
- sinon, il prend la valeur par défaut (souvent NULL ou '')

INSERT... SELECT

But : alimenter une table à partir d'une (ou plusieurs) autres
`INSERT INTO table [(col1, ...)] SELECT ...`

Exemple

C'est Noël : cadeau promotionnel pour tous les clients qui ont passé une commande cette année, sous la forme d'une commande fictive gratuite, avec un cadeau unique référencé **CAD08**.

1. `INSERT INTO commandes(idclient, date) SELECT DISTINCT idclient, '2008-12-25' FROM commandes WHERE date>='2008'`
2. `INSERT INTO details(numcommande, numordre, codearticle, quantite, prix) SELECT c.numcommande, 1, 'CAD08', 1, 0.00 FROM commandes c WHERE c.date='2008-12-25'`

INSERT... et contrainte d'unicité

Contraintes d'unicité sur les enregistrements d'une table

- sur la clé primaire (forcément unique), ou
- sur un index unique (éventuellement plusieurs)

Trois façons de régler le problème

1. `INSERT IGNORE INTO table ...`
   - conserve l'ancien enregistrement, oublie le nouveau
   - transforme les erreurs (bloquantes) en avertissements (non bloquants)
2. `REPLACE INTO table` (3 mêmes syntaxes qu'INSERT)
   - remplace l'ancien enregistrement par le nouveau
   - compte comme une (insert) ou deux opérations (delete+insert)
3. `ON DUPLICATE KEY UPDATE col1=expr, ...`
   - remplace l'INSERT par un UPDATE si nécessaire
   - compte comme une ou deux opérations

INSERT... les clauses particulières

- Priorité : à utiliser avec précaution
  - LOW_PRIORITY : insertion remise à plus tard, quand plus aucun accès en lecture ne sera en cours ; bloquant.
  - HIGH_PRIORITY : outrepasse le LOW_PRIORITY défini au niveau serveur.
  - DELAYED : insertion remise à plus tard, quand le serveur aura le temps ; non bloquant. Tous les INSERT DELAYED du tampon sont groupés.

Mise à jour d'enregistrements : UPDATE

UPDATE : 2 syntaxes, mono-table et multi-tables

- `UPDATE [...] table SET col1=expr1 [, col2=expr2 ...] [WHERE ...] [ORDER BY ...] [LIMIT ...]`
- `UPDATE table-refs SET col1=expr1 [, col2=expr2 ...] [WHERE ...]`

Exemples :

- `UPDATE clients SET codepostal='38000',ville='Grenoble' WHERE (nom,prenom)=('Sorel','Julien')`
- `UPDATE articles SET stock=stock+2 WHERE stock<5 AND prix<30`

Suppression d'enregistrements : DELETE

DELETE : 3 syntaxes

- `DELETE [...] FROM table [WHERE ...] [ORDER BY ...] [LIMIT N]`
- **ATTENTION : deux syntaxes multi-tables**
  - `DELETE tcible1 [, tcible2] ... FROM table-refs [WHERE ...]`
  - `DELETE FROM tcible1 [, tcible2] ... USING table-refs [WHERE ...]`

Exemples :

- `DELETE FROM clients WHERE (nom,prenom)=('Sorel','Julien')`
- Exercice : effacer de la table clients tous ceux qui n'ont jamais commandé ! Remarque : il est conseillé de s'aider d'une vue ou d'une table temporaire

Remarque : `TRUNCATE TABLE matable` permet de vider la table

Les index : améliorer les performances

Fonctionnement d'un SELECT ou JOIN

`SELECT a.nom, c.nom FROM articles a JOIN categories c USING (idcategorie) WHERE a.stock>2005`

S'il n'y a pas d'index, MySQL parcourt

- toute la table *articles* pour trouver les *stocks*
- toute la table *categorie* pour trouver les *idcategorie*

Index

Permet à MySQL de trouver rapidement une valeur

- Clé primaire $\Longrightarrow$ index
- Clé étrangère $\Longrightarrow$ index (presque toujours)
- Contrainte d'unicité : index `UNIQUE`

Les auto-jointures

- possible de référencer plusieurs fois la même table dans une requête SQL
- utilisation indispensable des alias (ex. : `matable AS mt`)

Exemples

- Afficher tous les articles classés dans la même catégorie que le tuba
  ```
  SELECT a1.codearticle, a1.nom, a1.prix FROM
  articles a1 JOIN articles a2 USING (idcategorie)
  WHERE a2.nom = "tuba" ;
  ```
- Afficher tous les articles moins chers que le tuba
  ```
  SELECT a1.codearticle, a1.nom, a1.prix FROM
  articles a1 JOIN articles a2 ON (a1.prix <
  a2.prix) WHERE a2.nom = "tuba" ;
  ```

Usages fréquents : les hiérarchies, arbres...

Les sous-requêtes

`http://dev.mysql.com/doc/refman/5.0/en/subqueries.html`

- résultat scalaire : `SELECT MAX(prix) FROM articles ;`
- résultat colonne : `SELECT prix FROM articles ;`
- résultat table : sous-table

Peu performant en MySQL (mieux vaut utiliser des jointures).

Exemples

- `SELECT nom, codearticle, prix FROM articles WHERE prix=(SELECT MAX(prix) FROM articles) ;`
- `SELECT prix FROM details WHERE prix NOT IN (SELECT prix FROM articles) ;`
- `SELECT * FROM articles WHERE prix IN (SELECT prix FROM articles GROUP BY prix HAVING COUNT(codearticle)>1) ;`

SELECT ... UNION

`SELECT ... UNION [ALL | DISTINCT] SELECT ...`

- DISTINCT : par défaut (union ensembliste)
- ALL : lignes répétées

Compatibilité avec l'ordonnancement

- `(SELECT a FROM t1 ORDER BY a LIMIT 5) UNION (SELECT a FROM t2 ORDER BY a LIMIT 5) ;`
- `(SELECT a FROM t1) UNION (SELECT a FROM t2) ORDER BY a LIMIT 10 ;`
- ordre des lignes non garanti par l'opération UNION
- contournement : ajout de colonnes explicites

Exemples

- `(SELECT codearticle, prix FROM articles A) UNION (SELECT codearticle, prix from details D) ORDER BY prix ASC ;`
- ex. Trouver les différences entre prix catalogue et commandes

Exercice : trouver l'article le plus cher (code, nom, prix)

Par le plus grand nombre de méthodes différentes !

Réponses

1. `SELECT nom, codearticle, prix FROM articles ORDER BY prix DESC LIMIT 1 ;` Inconvénient ?
2. `SET @maxi=(SELECT MAX(prix) FROM articles) ; SELECT nom, codearticle, prix FROM articles WHERE prix=@maxi ;`
3. `SELECT nom, codearticle, prix FROM articles WHERE prix=(SELECT MAX(prix) FROM articles) ;`
4. `SELECT a1.codearticle, a1.nom, a1.prix FROM articles a1 LEFT JOIN articles a2 ON (a1.prix < a2.prix) WHERE a2.codearticle IS NULL ;`
5. `SELECT nom, codearticle, prix FROM articles WHERE prix >= ALL(SELECT prix FROM articles) ;`
6. ...

Travaux Pratiques : améliorer la base FacSys

1. Ajouter à chaque client un champ datecreation
2. L'initialiser à la date de sa première commande
3. Implémenter un parrainage d'un client par un autre. Chaque année, envoyer des cadeaux aux trois plus "gros" parrains.
4. Donner la possibilité de classer un article dans plusieurs catégories.
5. Organiser un suivi des commandes avec historique, en 4 étapes : commande reçue -> saisie -> confectionnée -> expédiée, et une table opérateurs.
6. Faire une vue facture
7. Ajouter un prix d'achat et une table des fournisseurs.

Administration MySQL

Variables - généralités

- Noms alphanumériques
- Noms insensibles à la casse (depuis 5.0)
- Portée : Globale ou Session
- Type : Système ou Utilisateur
- Définies par la commande `SET` :
  `SET @var := 1`

Variables système (5.1.5)

- définies au lancement du serveur mysqld
  - fichier de configuration
  - ligne de commande, ex. `mysqld -key-buffer=16M ...`
- `SHOW VARIABLES [LIKE "..."]`
- statiques (certaines) ou dynamiques (la plupart) -> SET
- portée de la variable :
  - globale : ex. `connect_timeout`
  - session (ou locale) : ex. `autocommit`
  - ou les deux : ex. `default_week_format`
    $\Longrightarrow$ la variable session hérite de la variable globale

Réglage des variables système

Trois possibilités complémentaires :

1. Dans le fichier de configuration, section **[mysqld]**
   ex. `log_bin_trust_function_creators = 1`

2. En ligne de commande, au lancement du serveur
   ex. `mysqld --key-buffer=16M ...`

3. (éventuellement) dynamiquement, en cours d'exécution
   - session : `SET SESSION var :=` ou `SET @@var :=`
   - globale : `SET GLOBAL var :=` ou `SET @@global.var :=`
     $\Longrightarrow$ privilège SUPER nécessaire pour la modification

Messages d'erreur de MySQL

- Erreurs Serveur (Annexe B-3)
  - ex. `ERROR 1193 (HY000) : Unknown system variable 'hop'`
  - un numéro d'erreur mysqld, entre 1000 et 1477
  - un code SQLSTATE sur 5 caractères, ANSI SQL
  - un message d'erreur
- Erreurs Client (Annexe B-4)
  - un numéro, entre 2000 et 2055 (ex : 2034)
  - un message, ex. *Invalid parameter number*
- Erreurs système (rare)
  - un message de type *ERROR '...' not found (errno : 23)*
  - errno entre 1 et 152 $\Longrightarrow$ commande `perror`

Commandes `SHOW ERRORS [LIMIT ...]` et `SHOW WARNINGS [LIMIT ...]`

Le fichier de configuration : my.cnf / my.ini

- Localisation
  - sous Unix : `/etc/my.cnf`
  - sous Windows : `C :\my.ini` ou `INSTALLDIR\my.ini`
- Organisation en sections :

```
  client : options passées à tous les clients
 mysqld  : options passées au serveur
  mysql  : options spécifiques à la console mysql
mysqldump : options spécifiques au client de dump
    ...
```

- Syntaxe générale : cle = valeur, ex.
  `key_buffer = 16M`

Les fichiers de log

Quatre types différents :

- `log` (general query) : toutes les requêtes reçues par le serveur
- `log-bin` : les requêtes modifiant le contenu des bases
  - utilisé pour la réplication de serveur
  - plus compact que le précédent (binaire)
- `log-slow-queries` : les requêtes longues
  utilisé pour le débogage ou le profilage
- `log-error` : les messages d'erreur du serveur

Import / Export de données - fichiers

SELECT INTO OUTFILE

LOAD DATA INFILE : exemple importation .CSV

```
LOAD DATA INFILE 'donnees.csv'
INTO TABLE TableDonnees
CHARACTER SET utf8
FIELDS TERMINATED BY ',' optionally ENCLOSED BY '"'
IGNORE 1 LINES ;
```


Import / Export de données

- `mysqlhotcopy`
- `mysqldump`
  chaînage possible avec `mysql`
- `SELECT INTO DUMPFILE / LOAD DATA INFILE`
  export / import de type CSV
- `BACKUP TABLE / RESTORE TABLE`
  limitée à MyISAM

Surveillance des processus

- Liste des processus
  - `SHOW [ FULL ] PROCESSLIST ;`
  - FULL = colonne info complète (tronquée à 100 sinon)
  - environ 30 commandes, dont *sleep, query, execute...*
  - env. 60 états d'exécution, dont *sending data, locked...*
  - privilège PROCESS nécessaire pour voir les autres clients
- Interruption d'un processus
  - `KILL [CONNECTION | QUERY] id ;`
  - QUERY = interrompre la requête
  - CONNECTION = couper la connexion (par défaut)
  - privilège SUPER nécessaire pour tuer les autres clients
- Équivalent ligne de commande : `mysqladmin`
  `mysqladmin processlist` et `mysqladmin kill`

FLUSH et RESET

- `FLUSH HOSTS` : vide le cache des hôtes (chgt IP)
- `FLUSH LOGS` : ferme et rouvre tous les fichiers de logs
- `FLUSH PRIVILEGES` : relit les privilèges dans la base mysql
- `FLUSH QUERY CACHE` : optimise le cache des requêtes
- `FLUSH STATUS` : réinitialise les variables de connexion (session)
- `FLUSH TABLES [...]` : vide le cache des tables...
- `FLUSH TABLES WITH READ LOCK` : idem + verrou en lecture
- `FLUSH USER_RESOURCES` : remet à zéro les quotas utilisateurs
- `FLUSH` :
- `RESET QUERY CACHE` : vide le cache des requêtes
- `mysqladmin flush-...` : équivalent partiel en ligne de commande

Le dictionnaire : INFORMATION_SCHEMA

- métadonnées normalisées, interrogeables en SQL
- base virtuelle (contrairement à mysql)
- calcul des droits spécifique
- exemple :

```
SHOW TABLES FROM information_schema ;
DESC information_schema.tables;
SELECT table_schema, table_name
  FROM information_schema.tables;
```

Ex. d'application : une requête similaire à DESC, en plus détaillé.

Pour aller plus loin...

Internationalisation...

... des messages de MySQL

- Changer la langue des messages d'erreur, etc.
  Au démarrage du serveur : `mysqld --language=french`

... des fonctions MySQL

- dates en français avec : `SET lc_time_names = 'fr_FR' ;`

... du texte dans MySQL

Dépend de 3 paramètres :

- Le **jeu de caractères** utilisé (*charset*)
  Exemples : Alphabet latin, Unicode
- L'**encodage** du texte (*encoding*)
  Exemples : latin1 (ISO-8859-1), UTF-8, UTF-16
- La règle d'**interclassement** (*collation*)
  Détermine l'ordre de tri, le mode de comparaison, etc.

On emploie souvent abusivement *charset* pour *encoding*.

Jeux de caractères

2 niveaux de choix :

- *charset* de stockage dans la table
- *charset* du client

MySQL convertit à la volée si ces 2 encodages sont différents.

Quel encodage choisir ?

- 36 disponibles
- 2 principaux : *latin1* et *utf8* (recommandé)
- Parfois, choisir *ascii* ou *binary* pour éviter toute conversion

Comment imposer le charset des requêtes et résultats ?

- `SET NAMES 'utf8' ;` passe le client en UTF-8
- `SHOW VARIABLES LIKE 'char%' ;` liste les paramètres de configuration
- cf doc 9.1.4. Connection Character Sets and Collations

L'interclassement (collation)

Rôle

- "ordre alphabétique" étendu pour comparaison, tri...
- classes de caractères équivalents, ex. e, é, è, ê, E, É, È, Ê

Impact sur

- les opérateurs =, >, `BETWEEN`, `LIKE`...
- les commandes `GROUP BY`, `ORDER BY`

Principes de fonctionnement

- plusieurs collations par jeu de caractère, dont 1 par défaut
- nom de type *charset_collation_var* (ex. : utf8_general_ci)
  avec var $\in$ { ci, cs, bin } = { sans casse, avec, pas de classes }
- possibilité d'ajouter une *collation personnalisée*

En pratique et en résumé

- À la création de la base, fixer charset et collation par défaut des tables (et éventuellement de la base) :
  ```
  CREATE TABLE t1 (...)
      CHARACTER SET utf8 COLLATE utf8_general_ci ;
  ```
- Suivant les cas, adopter des valeurs différentes pour certains champs (mots de passe, identifiants textuels...) :
  ```
  CREATE TABLE t1 (
    ref VARCHAR(8) CHARACTER SET ascii COLLATE ascii_bin
  ```
- À chaque connexion du client, déclarer l'encodage souhaité :
  `SET NAMES 'latin1' ;`
  Parfois l'API le permet directement.
- Si on utilise `DATE_FORMAT()` et consort, les précéder de :
  `SET lc_time_names = 'fr_FR' ;`
- Relire la doc MySQL

Optimisation - principes

Exécution d'une requête SQL

1. Analyse et traduction
   - analyse et vérification syntaxique
   - vérification de la validité (existences...)
   - vérification des permissions
   - produit une liste d'opérations
2. Optimisation
   - utilise le dictionnaire : index, taille des tables...
   - produit le **plan d'exécution** (arbre)
3. Exécution de l'arbre de la requête

Plan d'exécution - principe

- Données intermédiaires : pipelining vs matérialisation
  - pas de stockage
  - retour immédiat des premiers résultats au client
- Opérations bloquantes :
  - tris : ORDER BY
  - dédoublonnage : DISTINCT
  - certaines fonctions d'aggrégation globales : MIN(), MAX(), SUM()...
  - partitionnement : GROUP BY
- Arbre d'exécution

EXPLAIN - syntaxe

```
EXPLAIN SELECT ... ;

EXPLAIN EXTENDED SELECT ... ;
SHOW WARNINGS ;
```

- ne s'applique qu'à SELECT $\Longrightarrow$ reformuler les UPDATE, INSERT...
- EXPLAIN : affiche une vue du plan d'exécution
- EXPLAIN EXTENDED : reconstruit un SQL canonique

- `EXPLAIN SELECT * from articles WHERE prix >50.0 \G`

- limites et imprécisions d'EXPLAIN

EXPLAIN - colonnes

colonnes d'EXPLAIN	
id	le numéro de SELECT dans la requête
select_type	type de SELECT, simple ou complexe...
table	la table concernée, ou l'alias
type	le type d'accès à cette table choisi par MySQL
possible_keys	les clés utilisables (à première vue)
key	la clé choisie par MySQL pour l'opération
key_len	la longueur de clé utilisée en octets
ref	la colonne référencée par la clé choisie
rows	nombre de lignes parcourues (estimation)
Extra	infos complémentaires, selon champs précédents

EXPLAIN - colonnes id, select_type, table

- colonne *id* : 1, 2, 3... et NULL, non unique

- colonne *select_type*

SIMPLE	le SELECT ne contient ni sous-requête ni UNION
PRIMARY	requête principale
SUBQUERY	sous-requête autre qu'apparaissant dans le FROM
DERIVED	sous-requête apparaissant dans le FROM
UNION	2e partie (et suivantes) d'une UNION
UNION RESULT	encapsule tous les SELECT d'une UNION

- colonne *table* :
  - nom (ou alias) de la table concernée
  - *derivedN* : en cas de sous-requête dans FROM
  - *unionX,Y...* : en cas d'UNION
  - l'ordre des lignes indique l'ordre du plan d'exécution

EXPLAIN - colonne type

ALL	parcourt toutes les lignes de la table
index	() parcourt toutes les lignes dans l'odre de l'index
index	(Extra=Using index) parcourt tout l'index
range	parcourt un intervalle d'index
ref	accès indexé direct - valeurs multiples possibles
eq_ref	accès indexé direct - au plus une valeur de retour
const, system	remplacé par une constante dans l'optimiseur
NULL	résolu immédiatement par l'optimiseur

- Note : ALL systématique pour les petites tables

EXPLAIN - colonnes clés

- colonne *possible_keys* (informatif)
  - liste déterminée à la phase d'analyse
  - peut rester inutilisée après optimisation
- colonne *keys*
  - souvent une clé de la liste
  - parfois aucune ne convient $\Longrightarrow$ NULL
  - parfois une clé extérieure à la première liste
- colonne *key_len*
  - longueur utilisée en octets
  - si clé multicolonnes, peut être inférieure au total

EXPLAIN - colonnes ref, rows, Extra

- colonne *ref*
  - utilisée si une clé est déclarée
  - référence des champs des lignes précédentes
  - ou des constantes
- colonne *rows*
  - nombre de lignes (estimé) à parcourir
  - relatif au point courant du plan d'exécution
  - estimation dépend des statistiques sur la table (cf plus loin)
  - néglige les LIMIT (jusqu'à v.5.1)
- colonne *Extra*

Using index	utilise un index couvrant : évite l'accès à la table
Using where	post-filtrage des lignes retournées
Using temporary	utilisation d'une table temporaire (tri...)
Using filesort	tri externe, en mémoire ou sur disque

Métadonnées et statistiques utiles

- `SHOW TABLE STATUS LIKE 'table' \G`
- `ANALYZE TABLE table ;`
- que peut-on prévoir comme optimisation de la strucure de communes ?

Benchmark

- Commande `BENCHMARK`
- Limitée à une évaluation d'expression
- Limitée à l'exécution de la requête par le serveur
- Exemples

```
SET @input := "mon mot de passe secret";
SELECT BENCHMARK(1000000, MD5(@input));
SELECT BENCHMARK(1000000, SHA1(@input));

SELECT BENCHMARK(10,(SELECT MAX(naiss) FROM naissances));
SELECT BENCHMARK(10,(SELECT @v:=MAX(naiss) FROM naissances));
```

Profiling

- Recherche des étapes longues dans un processus
- Analyse a posteriori
  utiliser `log-slow-queries`
- Analyse en direct : commandes SQL
  - `SET @@profiling :=1 ;`
  - `SHOW PROFILES ;`
  - `SHOW PROFILE [ ALL ] FOR QUERY ... ;`
  - types : BLOCK IO, CPU, MEMORY, PAGE FAULTS, SWAPS...
  - paramètre : `profiling_history_size` (=15)

Les index

- Généralités
  - porte sur une ou plusieurs colonnes de la table
  - possède un nom distinctif (PRIMARY pour la clé primaire)
- Les types d'index - pour l'utilisateur
  - Clé primaire : unique pour une table + contrainte d'unicité
  - INDEX simple : pour les recherches...
  - UNIQUE INDEX : recherche + contrainte d'unicité
  - FULLTEXT : index plein texte
  - SPATIAL : index géométrique - extension SPATIAL
- Les type d'index interne
  - HASH : fonction de hachage (par défaut en MyISAM)
  - B-Tree : arbre équilibré (par défaut en InnoDB)
  - R-Tree : index spatial
  - FULLTEXT

Index - création

Création d'index

```
ALTER TABLE table
    ADD PRIMARY KEY [index-type] (index-col,...)
  | ADD UNIQUE [INDEX] [index-name] [index-type] (index-col,...)
  | ADD [FULLTEXT|SPATIAL] INDEX [index-name] (index-col,...)

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index-name
    [ USING [BTREE | HASH]]
    ON tbl_name (index_col_name,...)
```

Note : préfixe d'index (chaînes de caractères)

Suppression

```
ALTER TABLE table
    DROP PRIMARY KEY
  | DROP {INDEX|KEY} index_name
```


Index - utilisation

Désactivation temporaire pour insertion massive

```
ALTER TABLE table  DISABLE KEYS;
 ...
ALTER TABLE table  ENABLE KEYS;
```

Utilisation courante

- utilisation automatique pour JOIN, ORDER... (cf EXPLAIN)
- utilisation forcée sur JOIN, ex. :
  `SELECT * FROM t1 USE INDEX (col1, col2)`
- indication = USE | IGNORE | FORCE

Cache d'index

- CACHE INDEX
- LOAD INDEX INTO CACHE

Verrous - généralités

```
LOCK TABLES
    nom-table [[AS] alias] lock-type
    [, nom-table [[AS] alias] lock-type] ...
( lock-type: READ | [LOW_PRIORITY] WRITE )
...
UNLOCK TABLES
```

- Priorité : WRITE > LOCAL > LOW_PRIORITY WRITE
- READ : empêche l'écriture ; tout le monde peut lire
- WRITE : empêche tous les autres accès
- Notes
  - doit porter sur **toutes** les tables utilisées, même multiples
  - privilège LOCK TABLE nécessaire en complément du SELECT
  - s'applique aussi sur les vues
  - pas de sens sur une table temporaire

Verrous - les pièges

- Déverrouillages implictes
  - pose d'un nouveau verrou
  - début de transaction
  - perte de connexion client - serveur
- **Attention** aux interactions verrou - transactions
- `FLUSH TABLES WITH READ LOCK` : verrou global
  - prioritaire
  - nécessite le privilège RELOAD

Les moteurs de stockage

MyISAM le moteur par défaut, d'origine ABSoft
- très rapide pour des requêtes et des tables simples
- faible empreinte disque

InnoDB moteur "sophistiqué" : intégrité, transactions
- développé par InnoBASE, rachetée par Oracle
- moteur plus complexe

Memory tout le stockage en RAM ; perdu à l'arrêt serveur

Archive prévu pour la journalisation
$\Longrightarrow$ INSERT et SELECT seulement

Merge fusion virtuelle de plusieurs tables MyISAM

Maria (dév.) successeur prévu pour MyISAM

Falcon (dév.) successeur prévus pour InnoDB

Particularités de MyISAM

- limitations : ni clés étrangères, ni transactions
- cache mémoire des index seulement
- index non plaçant (B-Tree)
- stockage disque en 3 fichiers par table : .frm (structure), .MYD (données), .MYI (index)
- tables et index très compacts sur disque
- indexation spécifique : FULLTEXT et SPATIAL

Particularités d'InnoDB

- déclaration possible des clés étrangères
- vérification de l'intégrité référentielle
- support des transactions, avec 4 niveaux d'isolation
- utilisation d'index plaçant (clustering) sur la clé primaire (B-Tree+)
- cache mémoire des données aussi
- stockage disque en un fichier par table : .frm
- tables et index plus volumineux sur disque

Intégrité référentielle

- Définition des clés étrangères

```
CREATE TABLE table | ALTER TABLE table ADD

[CONSTRAINT symb] FOREIGN KEY [i-fkey-id] (col1, ...)
    REFERENCES nom-table (col1, ...)
    [ON DELETE [RESTRICT | CASCADE | SET NULL ]]
    [ON UPDATE [RESTRICT | CASCADE | SET NULL ]]
```

RESTRICT rejette la modification, avec un message d'erreur
CASCADE répercute la modif sur la table référencée
SET NULL effectue l'action et anNULLe la clé sur la table référençante

- Activation : `SET FOREIGN_KEY_CHECKS := 0|1 ;`

Transactions

Théorie : propriétés ACID pour les transactions

- **Atomicité**
  - règle du "tout ou rien" sur une séquence d'opérations
  - inclut la réversibilité des opération
- **Cohérence**
  - respect des règles de cohérence après la transaction, quel que soit le résultat
- **Isolation**
  - les données dans un état intermédiaire ne sont pas visibles des autres sessions
  - assure la cohérence des données entre transactions
  - minimise l'impact sur les performances
- **Durabilité**
  - une fois terminée, la transaction ne peut être remise en cause

Transactions - utilisation

```
START TRANSACTION
COMMIT   [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [AND [NO] CHAIN] [[NO] RELEASE]
SET AUTOCOMMIT = [0 | 1]
```

- Options :

CHAIN enchaîne immédiatement une autre transaction
RELEASE coupe la connexion à la fin de la transaction

```
SAVEPOINT identifier
ROLLBACK TO identifier
RELEASE SAVEPOINT identifier
```

Transaction - verrous en lecture

- Deux types de verrous sur les lignes
  - verrou partagé (S) : permet à tous de lire la ligne, et de poser (S)
  - verrou exclusif (X) : interdit aux autres de poser (S) et (X) sur la ligne
- verrous implicites posés en lecture
  - `SELECT ... LOCK IN SHARE MODE`
    pose un verrou (S) sur chaque ligne lue
  - `SELECT ... FOR UPDATE`
    pose un verrou (X) sur chaque ligne lue
  - verrou valide jusqu'à la fin de la transaction

Transaction - isolation

- Commande
  `SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL niveau`
- Niveaux d'isolation, du plus faible au plus fort :
  - READ UNCOMMITTED
    accès en lecture aux autres transactions, où qu'on soit
  - READ COMMITTED
    accès en lecture aux transactions validées
  - **REPEATABLE READ**
    accès en lecture identique au premier SELECT de la transaction
  - SERIALIZABLE
    comme précédemment, mais tous les SELECT sont LOCK IN SHARE MODE

Vues, fonctions et procédures stockées

MySQL - modules stockés

- Fonctionnalités apparues majoritairement avec MySQL 5.0
  stabilisation longue
- "stockés" : enregistrés sur le serveur
- Les vues : tables dynamiques
- Les commandes "préparées"
- Les routines
  - fonctions définies par l'utilisateur
  - procédures stockées
  - curseurs
  - gestionnaires d'erreur (handlers)
  - langage procédural (sous-ensemble de SQL/PSM)
- Les déclencheurs (triggers)

Vues - Généralités

Idée

Vue = requête (SELECT) stockée, présentée comme une table (table dynamique).
Exemple :

```
CREATE VIEW ClientsBref AS
     SELECT prenom, nom, idclient, ville FROM clients ;
SELECT * FROM ClientsBref ;
```

Commandes :

- CREATE [OR REPLACE] VIEW
- ALTER VIEW
- DROP VIEW
- SHOW FULL TABLES [WHERE Table_type="VIEW"]
- SHOW CREATE VIEW

Vues - Exemples

- Définir une vue Vcommandes, qui augmente la table commande avec le nom du client, le nb de lignes, les champs quantite et montant
- Définir une vue Vfactures, qui donne les mêmes informations, plus les détails : une ligne par détail, plus une ligne de totalisation (astuce : GROUP BY WITH ROLLUP)

Vues - Paramètres avancés

Clause ALGORITHM=

- MERGE : vue remplacée par sa définition dans requête d'appel
- TEMPTABLE : utilisation d'une table temporaire
- UNDEFINED : MySQL fait le meilleur choix (MERGE si possible)

Privilèges et sécurité

- CREATE VIEW : nécessaire pour créer une vue
- SHOW CREATE VIEW : pour voir la définition d'une vue
- utilisation des vues : donner des droits partiels limités à certains utilisateurs
- Ex. : accorder les droits sur ClientsBref à un utilisateur "lambda". Vérifier l'action de SQL SECURITY (DEFINER, INVOKER).

Vues modifiables

- Certaines vues acceptent des UPDATE, DELETE, INSERT :
  - la modification ne s'applique qu'à une table ET
  - correspondance univoque entre colonnes tables / vue ET
  - les champs non référencés ont des valeurs par défaut définies
  - ... cf *21.4.3 Updatable and Insertable Views*
  - cf `SELECT * FROM information_schema.views ;`
- WITH CHECK OPTION
  - UPDATE ou INSERT doivent vérifier la clause WHERE de la vue
  - CASCADED (défaut) : vérif. étendue aux vues référencées, récursivement
  - LOCAL : vérif. limitée à la vue affectée

Tables temporaires et tables en mémoire

Tables temporaires

- `CREATE TEMPORARY TABLE nom-table`
- syntaxe identique à une création standard
- temporaire : existence limitée à la durée de la connexion
- isolée : nom de table local à la connexion
- privilège nécessaire : `CREATE TEMPORARY TABLES`

Tables en mémoire

- `CREATE TABLE nom-table (...) ENGINE MEMORY ;`
- existence limitée à la durée du serveur
- table partagée entre tous les clients
- privilège nécessaire : `CREATE TABLES`

Syntaxe des commentaires

- après un #, jusqu'à la fin de la ligne
- après une séquence `"-- "` (tiret-tiret-espace), jusqu'à la fin de la ligne
- **/* syntaxe C */** éventuellement multiligne
- /***!** variante syntaxe C */
  exécutée par MySQL, ignorée par les autres SGBD SQL
  `SELECT * FROM articles /* WHERE prix<10 */ ;`

Variables - généralités

- Noms alphanumériques
- Noms insensibles à la casse (depuis 5.0)
- Portée : Globale ou Session
- Type : Système ou Utilisateur
- Définies par la commande `SET` :
  `SET @var := 1`

Variables système (5.1.5)

- définies au lancement du serveur mysqld
  - ligne de commande `mysqld -var1=val1...`
  - fichier de configuration
- statiques (certaines) ou dynamiques (la plupart) -> SET
- portée : Global, Session, ou les deux
  - globale : `SET GLOBAL var` ou `SET @@global.var`
    $\Longrightarrow$ privilège SUPER nécessaire pour la modification
  - session : `SET SESSION var` ou `SET @@var`

Variables utilisateurs (8.4)

- ex. **@var**
- locales = portée toujours limitée à la session (connexion)

Affectation

- Affectation directe
  - `SET @a := 4, @b := "Dupont" ;`
  - `SET @c := LEFT(@b, @a) ;`
- Affectation par requête
  - `SET @p1 := (SELECT MIN(prix) FROM articles) ;`
  - `SELECT @p2 := MIN(prix), @p3 := MAX(prix) FROM articles ;`
  - `SELECT MIN(prix), MAX(prix) INTO @p4, @p5 FROM articles ;` obligatoire pour les routines

Commandes préparées

Contexte d'utilisation normal

- API pour les langages prévus :
  - natifs : C, Java (Connector/J), .NET
  - surcouches à l'API C : mysqli (PHP)...
- SQL pour mise au point / débogage

Syntaxe des commandes

- PREPARE stmt-name FROM preparable-stmt
- EXECUTE stmt-name [USING @var-name [, @var-name] ...]
- DROP PREPARE stmt-name

Exemple

- PREPARE clientsNom FROM "SELECT nom, prenom, ville FROM clients WHERE Nom > ? " ;
- SET @nomdeb := "E" ;
- EXECUTE clientsNom USING @nomdeb ;

Commandes préparées - Usages

Usages

- Optimisation : requête à paramètres, précompilée sur le serveur
- Méta-programmation : construction d'une requête en SQL
  - Utilisation d'une chaîne quelconque pour créer un PREPARE

Routines

- Fonctionnalités MySQL 5.0, en évolution
- Deux types : procédures stockées et fonctions
- Utilisent des paramètres typés
  - procédures : en entrée et sortie
  - fonctions : en entrée seulement
- Retours :
  - paramètres de sortie + retour des commandes (SELECT...)
  - valeur unique, typée

Routines - syntaxe commune

```
CREATE
      FUNCTION | PROCEDURE nom ([param1, param2...])
      [ RETURNS type ]
 LANGUAGE SQL  | [NOT] DETERMINISTIC |
 {CONTAINS SQL | NO SQL | {READS | MODIFIES} SQL DATA}
 | SQL SECURITY { DEFINER | INVOKER }
 | COMMENT 'chaine'
[label:] BEGIN
...
END [label]
```

- SHOW [PROCEDURE | FUNCTION] STATUS ;
- SHOW CREATE [PROCEDURE | FUNCTION] nom ;

Fonctions - syntaxe

```
CREATE  FUNCTION
  nomfonc ([para1 type1, para2 type2...])
RETURNS type
 LANGUAGE SQL  | [NOT] DETERMINISTIC |
 {CONTAINS SQL | NO SQL | {READS | MODIFIES} SQL DATA}
 | SQL SECURITY { DEFINER | INVOKER }
 | COMMENT 'chaine'
[label:] BEGIN
... RETURN <valeur> ...
END [label]
```

- SHOW FUNCTION STATUS ;
- SHOW CREATE FUNCTION nomfonc ;

- variable système **login_bin_trust_function_creators**

Fonctions - exemples

Fonctions simples

- Ecrire une fonction **Majuscule** qui prend une chaîne, et la retourne en minuscules, sauf la première lettre en majuscules.
- A l'aide de la précédente, écrire une fonction PreNom, qui prend deux chaînes et affiche "Prénom Nom" bien typographiés.

Fonctions “requêtes”

- Ecrire une fonction **MontantCumule** qui retourne le montant total commandé par un client de la base facsys.
- À partir du nom d'un nouveau client, retourner un nouveau idclient unique (rappel : 3 premiers caractères du nom, suivis d'un numéro, par ex. DUR005).

Contrôle de flot - les tests IF et CASE

```
IF (condition) THEN ... ;
[ ELSEIF (cond2) THEN ... ; ]
[ ELSE ... ; ]
END IF
```

```
CASE valeur
    [ WHEN valeur1 THEN ... ; ]  xN
    [ ELSE ...; ]
END CASE
```

```
CASE
    [ WHEN condition1 THEN ... ; ]  xN
    [ ELSE ... ; ]
END CASE
```

Ne pas confondre avec les **fonctions** IF() et CASE.

Contrôle de flot - les boucles

```
[label:] LOOP
   ...
END LOOP [label]

[label:] REPEAT
   ...
UNTIL (condition)
END REPEAT [label]

[label:] WHILE (condition) DO
   ...
END WHILE [label]
```

Les échappements

- LEAVE label : quitte la boucle
- ITERATE label : recommence la boucle

Procédures stockées - syntaxe

```
CREATE PROCEDURE
  nompro (IN par1 T1, OUT par2 T2, INOUT par3 T3...)
  LANGUAGE SQL  | [NOT] DETERMINISTIC |
 {CONTAINS SQL | NO SQL | {READS | MODIFIES} SQL DATA}
 | SQL SECURITY { DEFINER | INVOKER }
 | COMMENT 'chaine'
[label:] BEGIN
...
END [label]
```

- SHOW PROCEDURE STATUS ;
- SHOW CREATE PROCEDURE nompro ;

Procédures stockées - exemples

- Définir une procédure **Cumul** qui retourne le montant cumulé ET le nombre d'articles commandés.
- Définir une routine **Mode** qui affiche le mode (valeur la plus fréquente) de la colonne prix de la table articles.
- Ajouter un paramètre de sortie qui indique le nombre de modes.
- Définir une routine qui affiche la médiane d"une liste de valeurs. (Plusieurs méthodes possibles).

Curseurs

- Généralités
  - Autorisés à l'intérieur des "routines" : procédures, fonctions, triggers
  - Passage à un parcours classique d'une liste de résultats : boucle sur les lignes
  - Chaque curseur est associé à un SELECT
- Commandes
  - DECLARE mon-curseur CURSOR FOR SELECT...
  - OPEN mon-curseur
  - FETCH mon-curseur INTO var1, var2, ...
  - CLOSE mon-cuseur

Curseurs - exemple

- Définir une procédure qui affiche la somme des montants des N articles les plus chers et la somme totale du stock correspondant
- Définir une fonction qui affiche la médiane d"une liste de valeurs

Handlers - gestion d'erreur

```
DECLARE handler-type HANDLER
    FOR h-condition1 [, h-condition2]
    [ instruction  |  BEGIN ... END ] ;

h-type: CONTINUE | EXIT | UNDO

h-condition:
    SQLSTATE  valeur | mysql-code-erreur
  | SQLWARNING | NOT FOUND | SQLEXCEPTION
  | nom-condition
```

- méthode MySQL pour intercepter les erreurs
- souvent associé aux curseurs (NOT FOUND), mais pas seulement
- souvent l'instruction positionne un booléen (SET fini :=1)

Conditions définies pour le Handler

```
DECLARE nom-condition CONDITION
  FOR valeur-condition

valeur-condition:
    SQLSTATE  valeur
  | mysql-code-erreur
```

Façon de définir un “alias” pour une erreur ou une famille d'erreurs.

Messages d'erreur de MySQL

- Erreurs Serveur (Annexe B-3)
  - `ERROR 1193 (HY000) : Unknown system variable 'hop'`
  - un numéro d'erreur mysqld, entre 1000 et 1477
  - un code SQLSTATE sur 5 caractères, ANSI SQL
  - un message d'erreur
- Erreurs Client (Annexe B-4)
  - un numéro, entre 2000 et 2055 (ex : 2034)
  - un message, ex. *Invalid parameter number*
- Erreurs système (rare)
  - un message de type *ERROR '...' not found (errno : 23)*
  - errno entre 1 et 152 $\Longrightarrow$ commande `perror`

Curseurs et handlers - exemples

- Implémenter une fonction maximum avec plafond sur les prix des articles : ne prend pas en compte les valeurs supérieures au plafond donné en paramètre.

Déclencheurs (Triggers)

But : déclencher une action complémentaire lors de la modification (Insert, Delete, Update) d'un enregistrement dans une table.

```
CREATE TRIGGER nom t-moment t-modif ON table
  FOR EACH ROW instruction ;
| FOR EACH ROW   BEGIN ... END ;

t-moment = BEFORE | AFTER
t-modif = INSERT | UPDATE | DELETE
```

- Complément : `DROP TRIGGER [IF EXISTS] nom` et `SHOW TRIGGERS`
- Valeurs : NEW.champ (Insert, Update) $\Longrightarrow$ modifiable
  OLD.champ (Delete, Update) $\Longrightarrow$ lecture seule.

Triggers - Exemples

- Gestion du stock des articles lors de la commande :

```
CREATE TRIGGER majstock AFTER INSERT ON facsys.details
FOR EACH ROW
    UPDATE articles SET stock := stock - NEW.quantite
    WHERE codearticle=NEW.codearticle ;
```

- Faire un compteur des montants et des commandes du jour (variable session).
- Si on commande des balles de squash : 5 au minimum.
- Ex. courant : journalisation des actions critiques sur une table
- Ex. : définir une valeur par défaut dynamique pour une colonne. Ex. 75 $\Longrightarrow$ Paris.

En résumé

Déclarations dans un bloc BEGIN... END

1. Variables : DECLARE <var> <type> DEFAULT <valeur>, ...
2. Conditions : DECLARE <nom-cond> CONDITION FOR ...
3. Curseurs : DECLARE <nom-curs> CURSOR FOR SELECT ...
4. Handlers : DECLARE <handler-type> HANDLER FOR ...

Bibliographie complémentaire

- MySQL Reference Manual 5.0, AB Soft (plus complet en anglais)
- Maîtriser MySQL 5, O'Reilly France (2005), Darmaillac et Rigaux
- MySQL Cookbook, 2nd Ed (2006), O'Reilly
- MySQL Stored Procedure Programming (2006-03), O'Reilly
- High Performance MySQL, 2nd Ed. (2008-06), O'Reilly
- Expert MySQL 5, Apress (2007-01)

Informations utiles

Pour garder le contact :

`francois.gannaz@silecs.info`

Les documents utilisés sont disponibles en ligne :

`http://silecs.info/dld/MySQL/`

- Transparents
- Énoncés et corrections des exercices

Licence