Formation MySQL: Requêtes

Romain Bohdanowicz

Sommaire

- Le moteur MySQL
- Les types de données
- Les requêtes
  - Extraction de données
  - Modification de structures et de données
  - Triggers

Moteur MySQL

Types de tables

- Les différents types de tables
  - MyISAM
    - Performant et consomme peu de mémoire
    - Supporte la recherche « full-text »
  - InnoDB
    - Transactions
    - Contrainte de clé étrangère

- Le type de table peut être choisit à la création

Types de données

Types de données (1/9)

- Les nombres entiers

TINYINT	Entier 8 bits signé
SMALLINT	Entier 16 bits signé
MEDIUMINT	Entier 24 bits signé
INT, INTEGER	Entier 32 bits signé
BIGINT	Entier 64 bits signé
SERIAL	BIG INT AUTO_INCREMENT NOT NULL PRIMARY KEY
BOOL	TINYINT

Types de données (2/9)

- Les nombres à virgule flottante ou fixe

FLOAT	Flottant sur 32 bits
DOUBLE	Flottant sur 64 bits
REAL	DOUBLE
DECIMAL(p, s)	Nombre à virgule fixe - p : nombre de chiffre total - s : nombre de chiffre après la virgule
NUMERIC	DECIMAL

Types de données (3/9)

- Date et heure

DATE	Date (au format 'yyyy-mm-dd')
TIME	Heure (au format 'hh:mm:ss')
DATETIME	Combinaison de DATE + TIME
YEAR	Année (entre 1900 et 2155)
TIMESTAMP	Date et heure avec mise à jour automatique

Types de données (4/9)

- Les chaînes de caractères

CHAR(n)	Chaine de longueur fixe (longueur maximale : 255)
VARCHAR(n)	Chaine de longueur variable stockée dans l'enregistrement (longueur maximale : 65 535)
TINYTEXT	Chaine de longueur variable (longueur maximale : 255 caractères)
TEXT	(longueur maximale : 2^16-1)
MEDIUMTEXT	(longueur maximale : 2^24-1)
LONGTEXT	(longueur maximale : 2^32-1)

Types de données (5/9)

- Chaîne de caractères
  - Pour manipuler une chaîne de caractères, il faut connaître le jeu de caractère utilisé et le tri
  - Par exemple, latin 1 (ISO-8859-1) ou UTF-8

- Définition de l'ordre de tri: collation
  - Par exemple, latin1_general_ci
  - L'extension « ci » signifie « case insencitive »

Types de données (6/9)

- Données binaires
  - Type BLOB
  - Les données binaires ne sont pas directement stockées dans l'enregistrement

Types de données (7/9)

- Définition de listes de valeurs
  - ENUM ou SET

- Exemple

```
CREATE TABLE TestEnum
(
    couleur ENUM('Rouge', 'Vert', 'Bleu')
);
```

Types de données (8/9)

- Les différents attributs
  - AUTO_INCREMENT
    - Applicable à une seule colonne par table
    - Autorisé seulement si le champ est déclaré comme unique ou comme clef primaire
    - La dernière valeur attribuée peut être récupérée avec LAST_INSERT_ID()
  - NULL / NOT NULL
    - Le champ accepte ou non de ne pas avoir de valeur
    - Par défaut, NULL

Types de données (9/9)

- DEFAULT « valeur »
  - Permet de définir une valeur par défaut
- PRIMARY KEY
  - Définit une colonne ou un ensemble de colonnes comme clef primaire
- FOREIGN KEY / REFERENCES
  - Définit une ou plusieurs colonnes de la table comme clef étrangère faisant référence à une clef primaire
- CHARACTER SET name COLLATE sort
  - Définit le jeu de caractère et l'ordre de tri

Conversions

- Pour convertir de le format d'une colonne, il faut utiliser la fonction « CAST »

```
CAST(expression AS TYPE)
```

- Types possibles

BINARY
CHAR
DATE / TIME / DATETIME
SIGNED [INTEGER] / UNSIGNED [INTEGER]

Les valeurs NULLs

- Quel que soit le type, l'absence d'information est représentée avec NULL

- Pour tester si un champ est NULL
  - IS NULL ou IS NOT NULL
  - IFNULL(champ1, champ2)
  - COALESCE(champ1, champ2, champ3, …)

- Attention aux fonctions d'agrégation de données

Fonctions de chaînes

CHAR_LENGTH(str)	Retourne la longueur de la chaîne
CONCAT(str1, str2, …)	Concatène des chaînes
CONCAT_WS(sep, str1, str2, …)	Concatène avec un séparateur entre les chaînes
INSERT(str, pos, len, newstr)	Insertion d'une sous chaîne dans une autre
INSTR(str, substr)	Recherche d'une sous chaîne
LCASE ou LOWER	Retourne la chaîne en minuscule
LEFT(str, len)	Retourne les N caractères de gauche
MID(str, pos, len)	Retourne une sous chaîne
RIGHT(str, len)	Retourne les N caractères de droite
UCASE ou UPPER	Retourne la chaîne en majuscule

Manipulation de l'heure/date

ADDDATE(expr, days)	Ajoute des jours à une date
CURDATE	Retourne la date courante
CURTIME	Retourne l'heure courante
DATE(expr)	Extraction de la date
DATEDIFF(expr1, expr2)	Retourne le nombre de jours entre les 2 dates
DATE_ADD / DATE_SUB (date, INTERVAL expr type)	Calculs sur les dates Valeurs possibles: SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, YEAR ...
DATE_FORMAT(date, format)	Mise en forme d'une date
NOW()	DATETIME courant

Requêtes d’extraction

Composition d'une requête SQL

- Les différentes clauses qui compose une requête d'extraction de données existantes ou calculées

SELECT	Liste des champs à extraire
FROM	Liste des tables intervenant dans la requête
WHERE	Filtre portant sur le contenu des tables
GROUP BY	Regroupement des données
HAVING	Filtre portant sur le résultat des regroupements
ORDER BY	Tri des données du résultat

- Seul SELECT est obligatoire

La clause « SELECT » (1/3)

- Extraction de tous les champs d'une table avec l'opérateur * (étoile)

```
SELECT * FROM MaTable
```

- Extraction de champs en formant une liste de champs séparés par des virgules. Les champs peuvent être préfixés par le nom de la table

```
SELECT champ_1, MaTable.champ_2 FROM MaTable
```

La clause « SELECT » (2/3)

- Calculs sur les champs
  - Opérations mathématiques (+, -, /, *)
  - Opérations sur les chaînes de caractères (||)

- Donner un nouveau nom (alias) aux données récupérées ou calculées avec le mot clef AS

```
SELECT prix_1+prix_2 AS Somme,
       concat('Mr ', nom) AS NomPersonne
FROM   MaTable
```

La clause « SELECT » (3/3)

- Lorsque le SGBD construit la réponse à une requête, il récupère tous les lignes, généralement dans l'ordre ou il les trouve, même si il y a des doublons
- C'est le comportement par défaut qui correspond au mot clef ALL, pour éliminer les doublons, il faut utiliser DISTINCT

```
SELECT [ALL] * FROM MaTable
SELECT DISTINCT * FROM MaTable
```

La clause « FROM »

- Clause FROM
  - Permet de définir la ou les tables utilisées par la requête d'extraction de données
  - Les noms de tables doivent être séparés avec des virgules
  - On peut renommer les tables (avec un alias)

```
SELECT *
FROM   MaTable_1,
       MaTable_2 Alias,
       MaTable_3
```

La clause « ORDER BY » (1/2)

- Permet de trier le résultat de la requête
  - Définit un tri pour les colonnes de la réponse en précisant le nom des champs ou les numéros d'ordre dans l'énumération des champs
  - Ordre de tri
    - ASC : ascendant (par défaut)
    - DESC : descendant
  - Cette clause est obligatoirement à la fin de la requête parce qu'elle agit sur son résultat

La clause « ORDER BY » (2/2)

- Exemple de syntaxe

```
SELECT          champ_1, champ_2
FROM            MaTable
ORDER BY        champ_1 ASC, champ_2 DESC

SELECT          champ_1, champ_2
FROM            MaTable
ORDER BY        1, 2 DESC
```

La clause « WHERE » (1/4)

- Permet de filtrer les enregistrements (ou lignes) pour lesquels la condition est vérifiée

- Exemple: extraire d'une table de contacts toutes les personnes nommées « Dupont »

```
SELECT          *
FROM            Contact
WHERE           Nom = 'Dupont'
ORDER BY        Prenom
```

La clause « WHERE » (2/4)

- Composition d'une expression (booléenne)
  - Valeurs littérales
  - Opérateurs de comparaisons
    - (=, <, <=, >, >=, <>)
  - Opérateurs logiques
    - (OR, AND, NOT)

```
SELECT Nom, Prenom
FROM   Personne
WHERE  Age > 20 AND Age < 50
```

La clause « WHERE » (3/4)

- Opérateur « IN »
  - Permet de rechercher si une valeur se trouve dans un ensemble donné
  - Possibilité d'inverser le fonctionnement de cet opérateur avec NOT

```
SELECT Nom, Prenom
FROM   Personne
WHERE  Prenom IN ('Martin', 'Arthur', 'Julien')
```

La clause « WHERE » (4/4)

- Opérateur « BETWEEN »
  - Permet de rechercher si une valeur est dans un intervalle donné

```
SELECT Nom, Prenom
FROM   Personne
WHERE  Age BETWEEN 20 AND 50
```

- Opérateur « LIKE »
  - Permet d'effectuer une comparaison partielle avec l'utilisation de 2 jokers: % et _

Limiter les résultats

- Limiter le nombre d'enregistrements
  - LIMIT n, m
    - A la fin de la requête, permet de limiter le nombre d'enregistrements retournés
  - SQL_CALC_FOUND_ROWS / FOUND_ROWS()
    - Après avoir utilisé le premier mot clef dans une requête, la fonction « FOUND_ROWS » retourne le nombre d'enregistrements sélectionnés

Jointure simple (1/6)

- Principe d'une jointure
  - Extraction de données depuis plusieurs tables
  - Généralement basée sur une condition d'égalité entre une clef primaire et étrangère

- Jointure simple (interne)
  - Il suffit de spécifier plusieurs tables avec FROM
  - Par défaut, le résultat de la requête correspond au produit cartésien des différentes tables

Jointure simple (2/6)

Table1 : Nom
Martin
Durand

Jointure
(produit cartésien)

Table 2: Prénom
Julien
Arthur
Bernard

Nom	Prénom
Martin	Julien
Durand	Arthur
Martin	Bernard
Durand	Julien
Martin	Arthur
Durand	Bernard

Jointure simple (3/6)

- Combinaison des enregistrements
  - Egalité entre la clef primaire et une référence à cette clef (clef étrangère)

```
SELECT P.Nom, P.Prenom, E.Entreprise
FROM   Entreprises E , Personnes P
WHERE  E.ClefPrimaire=P.ClefEtrangere
```

  - Dans cet exemple, les autres combinaisons entre les deux tables n'ont pas de sens: tout le monde ne travaille pas dans toutes les entreprises…

Jointure simple (4/6)

- Cette jointure simple est appelé équijointure, c'est-à-dire jointure basée sur une égalité

- La clause « WHERE » peut bien sur venir filtrer le résultat de la requête

```
SELECT P.Nom, P.Prenom, E.Entreprise
FROM   Personnes P, Entreprises E
WHERE  E.ClefPrimaire=P.ClefEtrangere
AND    P.Prenom LIKE 'Julien'
```

Jointure simple (5/6)

- La syntaxe normalisée des jointures utilise le mot clef JOIN et permet
  - D'améliorer la lisibilité
  - Au SGBD d'optimiser l'exécution des requêtes
  - De séparer clairement les conditions de jointure du filtrage

- Par défaut, le mot clef JOIN désigne une jointure interne

Jointure simple (6/6)

- Une jointure interne s'effectue entre les tables sur les champs précisés dans la condition (après le mot clef ON)

```
SELECT          P.Nom, P.Prenom, E.Entreprise
FROM            Personnes P
INNER JOIN      Entreprises E ON P.Clef=E.Clef
WHERE           P.Prenom LIKE 'Julien'
```

- C'est la jointure par défaut, le mot clef INNER est donc optionnel

Jointure externe (1/4)

- Exemple de jointure interne
  - On récupère toutes les personnes habitant à Lille
  - C'est-à-dire que l'on estime qu'une personne dont on ignore la ville n'habite pas à Lille !

- Au contraire, une jointure externe permet en cas d'absence de lien entre les deux tables, de ne pas rejeter l'enregistrement

Jointure externe (2/4)

- Jointure externe
  - On peut récupérer toutes les personnes avec leurs villes
  - Les enregistrements de la table personne pour lesquels la ville n'est pas renseignée ne respecte pas la condition de jointure mais sont retournés

```
SELECT                  P.Nom, P.Prenom, V.Ville
FROM                    Personnes P
LEFT OUTER JOIN         Villes V ON P.Clef=V.Clef
```

Jointure externe (3/4)

- Les types de jointures externes (OUTER JOIN)
  - Droite (RIGHT)
  - Gauche (LEFT)
  - Complète (FULL) > pas encore supportée

- Exemple: une jointure externe gauche retourne tous les enregistrements de la table à gauche de l'opérateur même si ils n'ont pas de correspondances dans la table de droite

Jointure externe (4/4)

- A l'inverse, la jointure externe droite retourne tous les enregistrements de la table à droite de l'opérateur même sans correspondance

- Les deux types de jointures sont échangeables

```
FROM                    Personnes P
LEFT OUTER JOIN         Villes V ON P.Clef=V.Clef

FROM                    Villes V
RIGHT OUTER JOIN        Personnes P ON P.Clef=V.Clef
```

Conditions de jointures

- Conditions de jointures
  - Equijointure : avec une condition d'égalité
  - Peut ne pas porter sur les clefs des tables
    - Exemple: extraction des personnes dont le nom est celui d'une ville
  - Peut porter sur plusieurs champs des différentes tables
  - Les conditions d'inégalité sont aussi possibles
    - Exemple: connaître les élèves qui ont une meilleure note en math que 'Arthur'

Auto-jointure

- Jointure d'une table avec elle-même, exemples:
  - Dans une table de personnes, retrouver l'autre « moitié » d'un couple marié
  - Dans une table des employées, connaître le supérieur hiérarchique de tout employé

```
SELECT E1.Nom, E1.Prenom, E2.Nom
FROM   Employes E1
JOIN   Employes E2
ON     E1.Superieur=E2.EmployeID
```

Les fonctions d'agrégation

- Les fonctions d'agrégation calculent une valeur unique à partir d'un ensemble de valeurs
  - COUNT / MIN / MAX / AVG / SUM
  - Par exemple avec COUNT: connaître le nombre d'employé habitant à Lille

```
SELECT COUNT(*)
FROM   Employes E1
WHERE  Ville='Londres'
```

La clause « GROUP BY » (1/2)

- La clause GROUP BY permet d'utiliser les fonctions d'agrégat en « découpant » un résultat de requête en groupes

```
SELECT          E1.Nom, AVG(E1.Salaire)
FROM            Employes E1
GROUP BY        E1.Nom
```

- Attention: seuls les champs avec lesquels ont forme les groupes et les fonctions d'agrégations peuvent être utilisés dans la clause SELECT

La clause « GROUP BY » (2/2)

- GROUP_CONCAT
  - Concatène les chaînes (non nulles) d'un groupe

- GROUP BY – WITH ROLLUP
  - Permet d'avoir des sous totaux par groupe

```
SELECT     E.NUM_ELEVE, AVG(R.POINTS) AS Moyenne
FROM       ELEVES E
JOIN       RESULTATS R ON E.NUM_ELEVE = R.NUM_ELEVE
WHERE      E.ANNEE = 1
GROUP BY   E.NUM_ELEVE WITH ROLLUP;
```

La clause « HAVING »

- La clause HAVING est l'équivalent de la clause WHERE mais pour les lignes agrégées
- Elle permet de filtrer les résultats d'une requête après avoir utilisé des fonctions d'agrégation

```
SELECT          E1.Nom, COUNT(*) AS Homonymes
FROM            Employes E1
GROUP BY        E1.Nom
HAVING          COUNT(*)>5
```

Les fonctions d'ensembles

- Union des résultats
  - UNION / UNION ALL
- Ensemble vide ou non vide
  - EXISTS / NOT EXISTS

- Présence d'une valeur dans un ensemble

```
SELECT          Nom
FROM            Employes
WHERE           Nom IN ('Dupont', 'Arthur');
```

Les variables utilisateurs

- Les variables utilisateurs commencent par un @

```
SET    @var = 1;
SELECT @var AS maVariable;
SELECT 'Hello' INTO @var;
```

- La valeur par défaut est NULL
- Les variables peuvent contenir des valeurs de type réel, entier ou chaîne

Structure conditionnelle

- Mots clefs « IF » et « CASE »

```
-- Fonction IF
IF(condition, result1, result2)
-- Exemple
SELECT nom, IF(age>=18, 'majeur', 'mineur')
FROM   etudiant;

-- Structure conditionnelle CASE
CASE expr
       WHEN valeur1 THEN resultat1
       WHEN valeur2 THEN resultat2
       ELSE resultat3
END
```

Fonction « RAND() »

- Tirage aléatoire d'un nombre entre 0 et 1

- Permet par exemple d'extraire d'une table les enregistrements dans un ordre aléatoire

```
SELECT      E.NOM, E.PRENOM
FROM        ELEVES E
WHERE       E.ANNEE = 1
ORDER BY    RAND();
```

Requêtes de modification

Modifications de structures

- Comme pour l'extraction de données, toutes les modification de la structure des bases, tables et enregistrements se font avec des requêtes SQL

- Les différentes modifications de structures
  - CREATE: création
  - ALTER: modification
  - DROP: suppression

Les bases et les tables (1/4)

- Exemple sur les bases

```
-- Création de la base 'MaBase'
CREATE DATABASE IF NOT EXISTS MaBase
-- Suppression
DROP DATABASE IF EXISTS MaBase
```

- L'expression « IF [NOT] EXISTS » permet de tester l'existence ou l'absence de l'objet concerné avant d'exécuter l'instruction

Les bases et les tables (2/4)

- Exemple sur les tables

```
-- Création de la table 'MaTable'
CREATE TABLE IF NOT EXISTS MaTable
(
  id       INT AUTO_INCREMENT PRIMARY KEY,
  entier   BIGINT CHECK(entier>=0 AND entier<=20),
  texte    VARCHAR(64)
);

-- Suppression
DROP TABLE IF EXISTS MaTable
```

Les bases et les tables (3/4)

- Les tables peuvent aussi être modifiées

```
-- Ajout d'un champ à la table 'MaTable'
ALTER TABLE MaTable ADD nouveau DECIMAL;
-- Suppression d'un champ
ALTER TABLE MaTable DROP nouveau;
-- Modification d'un champ
ALTER TABLE MaTable CHANGE nouveau montant FLOAT;
```

- Attention: les bases et les tables peuvent être supprimées même lorsqu'elles contiennent des données

Intégrité référentielle (1/2)

- Lorsqu'une clef étrangère est définie, on peut choisir le comportement en cas de mise à jour ou suppression de la clef primaire

CASCADE	Supprime en cascade tous les éléments faisant référence à la clef primaire
SET NULL	Toutes les clefs étrangères sont remplacées par la valeur NULL
NO ACTION	Autorise la suppression d'une clef primaire sans se soucier des clefs étrangères
RESTRICT	Par défaut, impossible de supprimer une clef primaire référencée par des clefs étrangères

Intégrité référentielle (2/2)

- Exemple de syntaxe

```
CREATE TABLE client
(
  id  INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  nom VARCHAR(32)
);

CREATE TABLE facture
(
  idClient INT NOT NULL,
  CONSTRAINT fk_cascade FOREIGN KEY (idClient)
  REFERENCES client (id) ON DELETE CASCADE
);
```

Les enregistrements (1/2)

- Insertion

```
-- Insertion avec des valeurs pour tous les champs
INSERT INTO MaTable
       VALUES(1, 23, 'bonjour');
-- Insertion avec des valeurs par défaut
INSERT INTO MaTable
       VALUES(DEFAULT, 19, 'hello world');
-- Insertion avec des valeurs pour certains champs
INSERT INTO MaTable(entier, texte)
       VALUES(37, 'nouveau');
-- Insertion basée sur une requête
INSERT INTO Copie_MaTable
       SELECT * FROM MaTable;
```

Les enregistrements (2/2)

- Mise à jour

```
-- Mise à jour d'un champ pour toute la table
UPDATE MaTable SET texte = 'modification';
-- Mise à jour des enregistrements filtrés
UPDATE MaTable SET entier = entier * 3
       WHERE id = 2;
```

- Suppression

```
-- Suppression des enregistrements filtrés
DELETE FROM MaTable
       WHERE id = 2;
```

Métadonnées (1/2)

- Commandes SHOW
  - DATABASES
  - TABLES
  - COLUMNS [FROM tablename]
  - INDEX [FROM tablename]
  - COLLATIONS

- Base de données « INFORMATION_SCHEMA »

Métadonnées (2/2)

- Quelques exemples de tables systèmes

schemata	Informations sur les bases de données
tables	Propriétés des tables
columns	Propriétés des colonnes
statistics	Statistiques sur les index
user_privileges	Privilèges des utilisateurs
collations	Liste des ordres de tri disponibles

Index

- Les index permet d'améliorer la recherche de données dans une table donc les performances

- Exemple

```
-- Indexation de toute la colonne
CREATE INDEX idx_nom ON personne (nom)

-- Indexation partielle
CREATE INDEX idx_partiel_nom ON personne (nom(8))
```

Recherche Full-Text (1/3)

- Full-Text
  - Permet d'effectuer une recherche sur l'ensemble d'un texte
  - Seulement utilisable avec les tables MyISAM

- Pour cela, il faut créer un index « full-text »

```
ALTER TABLE maTable ADD FULLTEXT(champ1, champ2)
```

Recherche Full-Text (2/3)

- L'index peut ensuite être utilisé dans n'importe quel requête sur la table

```
SELECT *
FROM   maTable
WHERE  MATCH(champ1, champ2) AGAINST('recherche')
       > 0.001;
```

- L'expression retourne un nombre à virgule en fonction de la pertinence de recherche

Recherche Full-Text (3/3)

- Recherche booléenne
  - Permet de lier les différents mots de la recherche avec un ET logique au lieu d'un OU logique
  - Permet d'utiliser les caractères spéciaux suivants
  - Le résultat de la recherche est alors booléen

`+mot`	Le mot doit obligatoirement être présent
`-mot`	Le mot ne doit pas être présent
`>mot` `<mot`	Augmente ou diminue l'importance d'un mot dans la recherche
`"mot1 mot2"`	Recherche précise

Les transactions

- Permet d'assurer l'exécution d'un ensemble de requêtes
- Débuter une transaction
  - START TRANSACTION
- Terminer une transaction
  - Valider: COMMIT
  - Annuler: ROLLBACK

Requêtes préparées

- Exemple

```
-- Préparation de la requête.
PREPARE select_eleve FROM
"SELECT nom, prenom FROM eleves WHERE num_eleve=?";

-- Utilisation.
SET @id = 1;
EXECUTE select_eleve USING @id;

-- Suppression.
-- Toujours supprimée en fin de session.
DEALLOCATE PREPARE select_eleve;
```

Trigger (1/2)

- Trigger
  - Permet d'exécuter du code avant (BEFORE) ou après (AFTER) les requêtes de modification des données pour une table données
  - Les mots clefs « OLD » et « NEW » permettent d'accéder au contenu de l'enregistrement en train d'être modifié avant ou après modification
  - Impossible de définir plusieurs triggers pour la même action sur la même table
  - Une erreur dans un trigger, annule l'action

Trigger (2/2)

- Exemple

```
-- Table qui contient des lignes de compte
CREATE TABLE account
(
	acct_num INT,
	amount DECIMAL(10,2)
);

-- A chaque insertion, on met à jour un compteur
CREATE TRIGGER ins_sum BEFORE INSERT ON account
FOR EACH ROW
	SET @sum = @sum + NEW.amount;
```