



SQL

Jean-Henri Raoelina



Vue d'ensemble de la formation

- Chapitre 1 : Introduction
- Chapitre 2 : Les Objets et leurs manipulations avec le LDD
Base de Données , Tables, Contraintes, Index, Vues
Procédures Stockées & Fonctions, Triggers
- Chapitre 3 : Les Données et leurs manipulations avec le LMD
Insert & Replace, Delete & Truncate, Update, Select
- Chapitre 4 : Les jointures
- Chapitre 5 : Les Agrégats
- Chapitre 6 : Les Requêtes ensemblistes
- Chapitre 7 : Les Requêtes imbriquées
- Chapitre 8 : Les Requêtes Corréllées
- Chapitre 9 : Les Tableaux croisés
- Chapitre 10 : Les transactions
- Chapitre 11 : Les événements



Vue d'ensemble de la formation

- **Chapitre 1 :Introduction**
- Chapitre 2 : Les Objets et leurs manipulations avec le LDD
Base de Données , Tables, Contraintes, Index, Vues
Procédures Stockées & Fonctions, Triggers
- Chapitre 3 : Les Données et leurs manipulations avec le LMD
Insert & Replace, Delete & Truncate, Update, Select
- Chapitre 4 : Les jointures
- Chapitre 5 : Les Agrégats
- Chapitre 6 : Les Requêtes ensemblistes
- Chapitre7 : Les Requêtes imbriquées
- Chapitre8 : Les Requêtes Corréllées
- Chapitre9 : Les Tableaux croisés
- Chapitre10 : Les transactions
- Chapitre11 : Les évènements



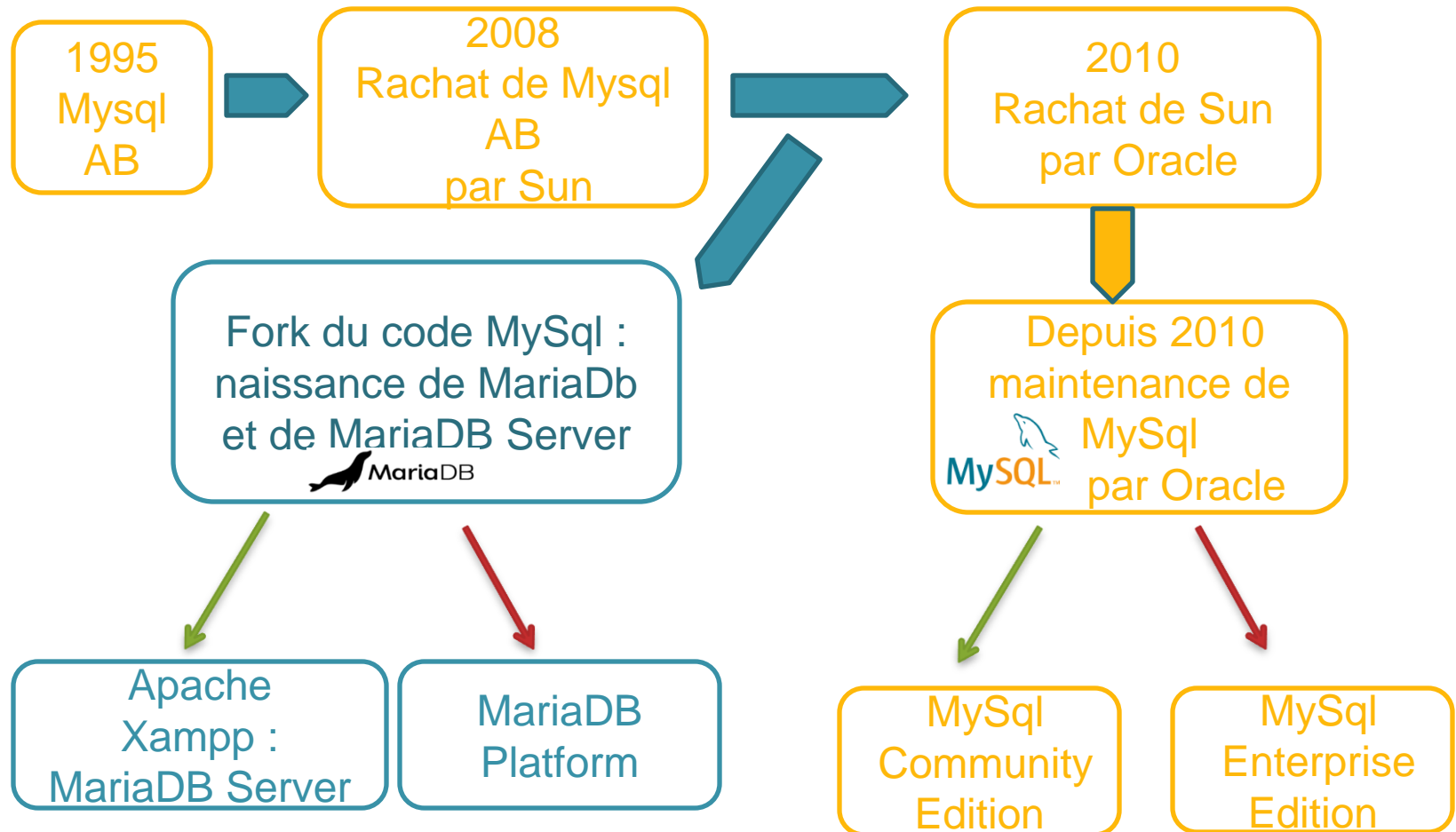
INTRODUCTION

MySql est un SGBD-R (RDBMS en Anglais)

- Système de Gestion de Bases de Données Relationnelles
- Relationnelles : tables reliées entre elles qui modélisent un domaine du SI

Introduction

- Historique de MySql





Introduction

Evolutions récentes de MySQL et MariaDB

- Prise en charge de données au format JSON dans un champ dédié de type JSON
- Disponibilité dans le cloud



INTRODUCTION

Un SGBD-R a son Langage : le **SQL**

- Langage basé sur des prédicats
- 4 sous-Langages:
 - **LDD** : crée, modifie, supprime les Objets de la Base de Données
 - **LMD** : crée, modifie, supprime les données
 - **LCD** : gère les droits sur les données
 - **LCT** : gère les transactions

INTRODUCTION

Historique du SQL

Année	Événement
1970	Edgar Frank Codd publia l'article «A Relational Model of Data for Large Shared Data Banks» ("Un modèle de données relationnel pour de grandes banques de données partagées") dans la revue Communications of the ACM (Association for Computing Machinery).
1970	Donald Chamberlain et Raymond Boyce ont conçu chez IBM System R. Le langage est nommé SEQUEL.
1975	Naissance de SQL
1979	Relational Software, Inc. (actuellement Oracle Corporation) présente la première version commercialement disponible de SQL.
1986	SQL est adopté comme recommandation par l'Institut de normalisation américaine (ANSI), puis comme norme internationale par l'ISO en 1987 sous le nom de ISO/CEI 9075



INTRODUCTION

Les Normes **SQL**

Année	Normes
1986	SQL-86
1989	SQL-89 ou SQL1
1992	SQL-92 ou SQL2
1999	SQL-99 ou SQL3
2003	SQL:2003
2008	SQL:2008
2011	SQL:2011



Vue d'ensemble de la formation

- Chapitre 1 : Introduction
- **Chapitre 2 : Les Objets et leurs manipulations avec le LDD**
 - Base de Données , Tables, Contraintes, Index, Vues
 - Procédures Stockées & Fonctions, Triggers
- Chapitre 3 : Les Données et leurs manipulations avec le LMD
 - Insert & Replace, Delete & Truncate, Update, Select
- Chapitre 4 : Les jointures
- Chapitre 5 : Les Agrégats
- Chapitre 6 : Les Requêtes ensemblistes
- Chapitre 7 : Les Requêtes imbriquées
- Chapitre 8 : Les Requêtes Corréllées
- Chapitre 9 : Les Tableaux croisés
- Chapitre 10 : Les transactions
- Chapitre 11 : Les évènements



Base de Données

Espace qui Contient tous les Objets

- Syntaxe de création

```
CREATE DATABASE [IF NOT EXISTS] nom_de_base  
[DEFAULT CHARACTER SET jeu_de_caractères  
COLLATE collation];
```



Base de Données

- Exemple

```
CREATE DATABASE IF NOT EXISTS base_exos  
DEFAULT CHARACTER SET utf8  
COLLATE utf8_general_ci;
```

Base de Données

- Charset
 - Ensemble de Symboles et de Codes
 - Utf8, latin1, greek, hebrew
 - Utf8 encode plus de 1 million de caractères sur 1,2 ou 3 octets. Permet de stocker des textes dans 650 langues différentes
 - ISO 8859-1, Alphabet de l'Europe Occidentale encode 191 caractères, dont les caractères accentués de la langue française

Base de Données

- Collate

- Algorithme utilisé pour la comparaison de chaînes de caractères
 - Pour utf8 : utf8_general_ci, utf8_bin, utf8_unicode_ci, ...
 - Pour latin1 : latin1_swedish_ci, latin1_bin, latin1_general_cs, latin1_general_ci, ...
- Une comparaison binaire est une comparaison exacte
 - A est différent de a
- Une comparaison générale est plus tolérante
 - A est équivalent à a



Base de Données

- Modification d'une Base de Données
 - Syntaxe
 - ALTER DATABASE nom_de_base DEFAULT CHARACTER SET jeu COLLATE collation;
 - Exemple
 - ALTER DATABASE base_exos DEFAULT CHARACTER SET utf8 COLLATE utf8_general_cs;



Base de Données

- Suppression d'une Base de Données
 - Syntaxe
 - DROP DATABASE [IF EXISTS] nom_de_base;
 - Exemple
 - DROP DATABASE IF EXISTS base_exos;



Les Tables

- Caractéristiques d'une Table
 - Composée de Colonnes et de Lignes
 - Chaque colonne possède un type correspondant à la donnée qu'elle va stocker
 - Elle doit posséder une clé primaire
 - Composée d'une ou plusieurs colonnes
 - La clé primaire permet d'identifier chaque ligne



Les Tables

- Les Types de Données
 - Numérique
 - Caractère
 - Date
 - Binaire
 - Ensemble (Set)*
 - Enumérations*
 - Géométrie

* Disponible uniquement pour MySql

Les Tables

- Syntaxe de création

```
CREATE TABLE nom_de_table (  
  nom_de_colonne TYPE CONTRAINTE  
  [, col2 ... ,  
  [Contrainte de table]])  
[ENGINE moteur_de_table]  
[DEFAULT CHARSET=jeu_de_caractères  
  COLLATE=collation];
```

Nous verrons les contraintes en détail dans les slides suivants

Les Tables

- Exemple de création

```
CREATE TABLE IF NOT EXISTS clients (  
  id_client INT(5) NOT NULL AUTO_INCREMENT ,  
  nom VARCHAR(50) NOT NULL ,  
  prenom VARCHAR(50) NULL ,  
  adresse VARCHAR(100) NULL ,  
  date_naissance DATE NULL ,  
  cp CHAR(5) NOT NULL ,  
  PRIMARY KEY (id_client), INDEX(nom)  
) ENGINE = InnoDB  
DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;
```



Les Tables

Manipulation des Colonnes

- Syntaxe d'ajout de Colonne
`ALTER TABLE nom_de_table ADD ...[, ADD ...]`
- Syntaxe de suppression de Colonne
`ALTER TABLE nom_de_table DROP
nom_de_colonne [, DROP nom_de_colonne];`
- Syntaxe de modification
`ALTER TABLE nom_de_table CHANGE ...`



Les Tables - outils de contrôle

- Pour vérifier le bon fonctionnement d'une table

CHECK TABLE nom_de_table

- Pour réparer une table

REPAIR TABLE nom_de_table

- Pour optimiser une table

L'optimisation permet de défragmenter une table

OPTIMIZE TABLE nom_de_table



Les Contraintes

- Elles permettent de contrôler l'intégrité des données d'une table à la volée
- Elles sont définies à la création de la table ou a posteriori
- Elles peuvent concerner une colonne ou une Table
- Elles peuvent s'appuyer sur des opérations
- logiques

Les Contraintes

- Exemple

Création d'une table comportant les colonnes c1 et c2, avec la contrainte que $c1 + c2 \leq 100$ dans tous les cas

- Application

```
CREATE TABLE testcheck (
```

```
id int(11) NOT NULL AUTO_INCREMENT, c1 int(11) DEFAULT 0,  
  c2 int(11) DEFAULT 0, PRIMARY KEY (id), CHECK (c1 + c2 <=  
100)
```

```
);
```




Les Contraintes

Ajout d'une Contrainte

- Syntaxe d'ajout de Contraintes

ALTER TABLE nom_de_table

ADD CONSTRAINT nom_contrainte TypeContrainte
[, **ADD CONSTRAINT** ...];

- Types Contrainte

PRIMARY KEY, FOREIGN KEY, UNIQUE ,
CHECK

Les Contraintes

Ajout d'une contrainte

- Exemple

Sur la table testcheck, il faut que la colonne c1 soit toujours strictement positive (>0)

- Application

```
ALTER TABLE testcheck ADD  
CONSTRAINT c1_positif CHECK (c1 > 0)
```



Les Contraintes

Suppression d'une Contrainte

- Syntaxe suppression d'une Contrainte

```
ALTER TABLE nom_de_table DROP  
TYPE_DE_CONTRAINTE [nom_de_contrainte];
```

- Types de Contrainte

Ne fonctionne que pour

PRIMARY KEY, FOREIGN KEY et CHECK.

Pour UNIQUE , il faut trouver le nom de l'index et le supprimer

Les Contraintes

- Exemple

Dans la table test_check, suppression de la contrainte que nous avons appelée “c1_positif” sur la colonne c1 .

- Application

```
ALTER TABLE testcheck DROP CHECK c1_positif;
```

Les Index

- Objectifs
 - Accélérer l'exécution des requêtes de lecture
 - Peut s'appliquer à une ou plusieurs colonnes

- Syntaxe de création

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX  
nom_d_index ON nom_de_table  
(nom_de_colonne[(n)],...)
```



Les Index

- Limitations

- FULLTEXT et SPATIAL ne peuvent être utilisés que sur des tables MyISAM
- Utilisés sur un SELECT comportant une clause WHERE sur une colonne indexée
- En cas d'index sur plusieurs colonnes, ne fonctionne que la si clause Where contient au moins la première colonne de l'index
- FULLTEXT ne fonctionne qu'avec les fonctions MATCH () et AGAINST()



Les Vues

- Caractéristiques

- Une vue est une requête stockée
- Les autorisations sont données sur des vues plutôt que sur des tables
- Elles garantissent l'intégrité des données ainsi que la confidentialité
- Elles s'utilisent comme une table : select, insert, modifications peuvent être effectuées

Les Vues

- Création

CREATE OR REPLACE VIEW nom_de_vue

AS SELECT * | colonnes FROM nom_de_table [WHERE
condition]

[WITH CHECK OPTION];

- CHECK OPTION permet de contrôler les MAJ à partir des vues
- Insertion et modification ne sont possibles que si les prédicats de la vue correspondent à ceux de la table.

- Exemple

CREATE VIEW clients_parisiens AS SELECT * FROM clients
WHERE cp LIKE '75%';



Les Vues

- **Suppression**

`DROP VIEW nom_de_vue`

- **Modification**

`ALTER VIEW nom_de_la_vue AS SELECT ... ;`

- **Stockage**

- fichiers .frm stockés dans le dossier /mysql/data/bd.

- **Limites**

- Les vues matérialisées (snapshots) n'existent pas chez MySQL.
- Les mises à jour ne sont possibles que sur les vues mono-table



Procédures Stockées & Fonctions

- Objectif

- Effectuer des traitements ou des calculs sur des données
- Les fonctions peuvent être appelées depuis une commande (insert, update, select ...)
- Les Procédures Stockées sont appelés par un « call »

- Syntaxe

- CREATE PROCEDURE sp_name ([parameter[,...]])
[characteristic ...] routine_body
- CREATE FUNCTION sp_name ([parameter[,...]]) [RETURNS
type] [characteristic ...] routine_body

Procédures Stockées & Fonctions

- Exemple

```
DELIMITER |
```

```
#Exemple de procédure calculant X exposant n
```

```
CREATE PROCEDURE Math_Puissance(IN X INT UNSIGNED, IN exposant  
TINYINT, OUT resultat BIGINT UNSIGNED)
```

```
BEGIN
```

```
DECLARE o BIGINT UNSIGNED DEFAULT 1;
```

```
    WHILE exposant > 0 DO
```

```
        SET o := o * X;
```

```
        SET exposant := exposant - 1;
```

```
    END WHILE ;
```

```
SELECT o INTO resultat;
```

```
END|
```

```
DELIMITER ;
```

```
CALL Math_Puissance(3,3,@a); #appel de la procédure
```

```
SELECT @a; #affichage du résultat
```

Trigger

- Objectif

- Déclencher une action avant ou après (Before / After) un événement concernant des lignes d'une table:
 - Insertion (insert), Mise à jour (update), Suppression (delete)
- Utile pour créer des pistes d'audit (historique des modifications) ou calculer des valeurs de colonnes en fonction des autres colonnes avant l'insert

Trigger

- Limites

- Ne se déclenche pas sur Drop Table ou sur Truncate Table
- L'utilisateur doit disposer des privilèges sur la table concernée

- Syntaxe

- CREATE [DEFINER = { *user* | CURRENT_USER }]

TRIGGER *trigger_name* { BEFORE | AFTER } { INSERT |
UPDATE | DELETE }

ON *tbl_name*

FOR EACH ROW

trigger_body

Trigger

- Exemple

```
delimiter //
```

```
CREATE TRIGGER recettes_vendeurs_insert_t  
BEFORE INSERT ON recettes_vendeurs  
FOR EACH ROW  
BEGIN  
    INSERT INTO recettes_jour_mat (rc_date, rc_montant) VALUES (NEW.rc_date, NEW.rc_montant)  
    ON DUPLICATE KEY UPDATE rc_montant = rc_montant + NEW.rc_montant;  
  
    INSERT INTO recettes_mois_mat (rc_year, rc_month, rc_montant)  
    VALUES (YEAR( NEW.rc_date ), MONTH( NEW.rc_date ), NEW.rc_montant)  
    ON DUPLICATE KEY UPDATE rc_montant = rc_montant + NEW.rc_montant;  
  
    INSERT INTO recettes_vendeur_mois_mat (rc_year, rc_month, vd_id, rc_montant)  
    VALUES (YEAR( NEW.rc_date ), MONTH( NEW.rc_date ), NEW.vd_id, NEW.rc_montant)  
    ON DUPLICATE KEY UPDATE rc_montant = rc_montant + NEW.rc_montant;  
END//
```



Vue d'ensemble de la formation

- Chapitre 1 : Introduction
- Chapitre 2 : Les Objets et leurs manipulations avec le LDD
 - Base de Données , Tables, Contraintes, Index, Vues
 - Procédures Stockées & Fonctions, Triggers
- **Chapitre 3 : Les Données et leurs manipulations avec le LMD**
 - Insert & Replace, Delete & Truncate, Update, Select
- Chapitre 4 : Les jointures
- Chapitre 5 : Les Agrégats
- Chapitre 6 : Les Requêtes ensemblistes
- Chapitre 7 : Les Requêtes imbriquées
- Chapitre 8 : Les Requêtes Corréliées
- Chapitre 9 : Les Tableaux croisés
- Chapitre 10 : Les transactions
- Chapitre 11 : Les événements

CRUD

Verbe générique	Verbe SQL	Fonctionnalité
Create	INSERT	Pour ajouter un ou des enregistrements.
Read	SELECT	Pour extraire un ou des enregistrements.
Update	UPDATE	Pour modifier un ou des enregistrements.
Delete	DELETE	Pour supprimer un ou des enregistrements.

Insert & Replace

- Objectif

- Insert

- Ajouter un ou plusieurs enregistrements dans une table

- Replace

- insérer des données si la clef précisée n'existe pas (PRIMARY) ou si la valeur des champs UNIQUE n'existe pas
- remplacer la ligne si la clef précisée existe déjà (PRIMARY) ou si la valeur des champs UNIQUE existe déjà
- Spécifique à Mysql : n'existe pas dans tous les RDBMS

Insert & Replace (Syntaxe)

- Syntaxe

- INSERT INTO nom_de_table [(col1, col2, ...)] VALUES (valeur1, valeur2, ...);
- INSERT INTO nom_de_table(col1, col2, ...) VALUES (valeur1, valeur2, ...) , (valeur3, valeur4, ...) ...;
- INSERT INTO table1 [(col1, col2, ...)] SELECT col1, col2, ... FROM table2 ...;
- REPLACE INTO nom_de_table(colonne1 [, colonne2]) VALUES(valeur1 [, valeur2]);
- REPLACE INTO table1 [(col1, col2, ...)] SELECT col1, col2, ... FROM table2;
- REPLACE INTO nom_de_table SET id=n, colonne2="xx", colonne3="yy";

Insert & Replace

- Exemple

```
INSERT INTO clients(nom, prenom, adresse , date_naissance ,  
cp) Values ("Durand","Robert","3, rue des tuyas","2019-03-  
19","75015");
```

```
REPLACE INTO clients(nom, prenom, adresse ,  
date_naissance , cp) Values ("Dupont","Pierre","15, Rue des  
Vosges","2017-06-09","75011");
```

```
REPLACE INTO clients(id_client,nom, prenom, adresse ,  
date_naissance , cp) Values (2,"Dubois","Pierre","15, Rue  
des Vosges","2017-06-09","75018");
```

```
REPLACE INTO clients SET  
id_client=2,nom="Delaporte",prenom="Pierre",adresse="15,  
Rue des Vosges",date_naissance="2017-06-09",cp="75018";
```

Delete & Truncate

- Objectif

- DELETE : Supprimer un ou plusieurs enregistrements dans une table.
 - Plus lent car provoque un recalcul des index si il en existe
 - peut déclencher un trigger
- TRUNCATE supprime toutes les données de la table
 - Plus rapide car pas de recalcul des index
 - Ne déclenche pas de trigger

- Syntaxe

- DELETE FROM nom_de_table [WHERE condition];
- DELETE FROM nom_de_table WHERE colonne Opérateur (SELECT ...);
- TRUNCATE TABLE nom_de_table;;

Delete & Truncate

- Exemple

- DELETE FROM Clients where id_client =5;
- TRUNCATE TABLE Clients;

Update

- Objectif

- Modifier un, plusieurs ou tous les enregistrements

- Syntaxe

- UPDATE nom_de_table SET col1 = valeur1 [, col2 = valeur2] [WHERE condition];
- UPDATE nom_de_table SET col1 = valeur1 [, col2 = valeur2] WHERE colonne Opérateur (SELECT ...);

- Exemple

- UPDATE clients SET nom = UPPER(nom) WHERE cp IN
(SELECT cp FROM villes
WHERE UPPER(nom_ville) LIKE UPPER('lille%'));

Update

- Example

- UPDATE clients SET nom = UPPER(nom) WHERE UPPER(nom) LIKE UPPER('durand%');

Select

- Objectif

- Extraire des enregistrements d'une ou plusieurs tables

- Syntaxe simple

- `SELECT [DISTINCT] * | col1 [[AS] alias de colonne], col2 [[AS] alias de colonne],`

`FROM nom_de_table [[AS] alias de table] [WHERE condition];`

- Syntaxe avec un tri

- `SELECT * | col1, col2, ... FROM nom_de_table`

`ORDER BY col1 [ASC | DESC] [, col2 ...];`

Select

- Opérateurs de comparaison

OPERATEUR	DESCRIPTION
=	Egal
!= , <>	Différent de
>	Supérieur à
>=	Supérieur ou égal à
<	Inférieur à
<=	Inférieur ou égal à

- Opérateurs logiques

Opérateur	Description
AND ou &&	Et logique
OR ou	Ou logique
NOT	La négation logique

Select

- Opérateurs ensemblistes

Opérateur	Descripteur
[NOT] IN(V1, V2, ...)	Egal à n'importe quelle valeur d'une liste de valeurs
[NOT] BETWEEN x AND y	$x \geq \text{valeur} \leq y$
[NOT] LIKE	Comparer deux chaînes de caractères avec l'utilisation des caractères génériques : _ pour un caractère et % pour une chaîne de caractères
IS [NOT] NULL	Tester la valeur NULL (ou non) dans une colonne Le NULL est la valeur dite indéterminée indépendamment du type

- Exemples

- SELECT col1, col2, ...FROM nom_de_table
 - WHERE colonne IN (v1, v2, ...);
 - WHERE colonne BETWEEN v1 AND v2;
 - WHERE colonne LIKE '...%';
 - WHERE colonne IS NULL | IS NOT NULL;

Select

- Requêtes Calculées
- Fonctions Numériques
- `SELECT col1, col2 opérateur col | opérande, ...FROM nom_de_table`
 - `SELECT designation, prix "Prix HT", FORMAT(prix * 0.196, 2) "TVA",
FORMAT(prix * 1.196, 2) "Prix TTC FROM produits p;`
- Fonctions sur les chaînes de caractères
- `SELECT CONCAT(col1, col2) FROM nom_de_table`
 - `SELECT id_client, CONCAT(nom, "-", prenom) "Nom et prénom
" FROM clients ORDER BY nom DESC;`



Select

- Requêtes Calculées

Fonctions Sur les Dates

- `SELECT date_cde, DATE_ADD(date_cde, INTERVAL 31 DAY) FROM cdes;`
- `SELECT DATEDIFF(date_cde, NOW()), date_cde, NOW() FROM cdes;`

Fonctions logiques

- `IFNULL(colonne, 'Texte' | valeur)`
 - `SELECT nom, IFNULL(date_naissance, 'Date inconnue') FROM clients;`

Recherche FULL TEXT

- `SELECT * | colonne MATCH (colonne) AGAINST ('valeur_recherchee')`
`FROM nom_de_table WHERE MATCH (colonne)`
`AGAINST('valeur_recherchee');`
- Ne fonctionne que sur les table ISAM sur lesquelles un index FULLTEXT a été créé sur les colonnes du match

Select

- Exemple

```
SELECT nom,prenom, DATEDIFF(date_naissance, NOW()), date_naissance,  
NOW() FROM clients;
```

```
SELECT id_client, CONCAT(nom, "-", prenom) "Nom et prénom " FROM clients  
ORDER BY nom DESC;
```

Select

- Exemple FullText

```
SELECT titre, MATCH (texte, commentaire, titre) AGAINST ('peuple') AS cpt  
FROM example_fulltext WHERE MATCH (texte, commentaire, titre)  
AGAINST ('peuple') ORDER BY cpt DESC;
```



Vue d'ensemble de la formation

- Chapitre 1 : Introduction
- Chapitre 2 : Les Objets et leurs manipulations avec le LDD
Base de Données , Tables, Contraintes, Index, Vues
Procédures Stockées & Fonctions, Triggers
- Chapitre 3 : Les Données et leurs manipulations avec le LMD
Insert & Replace, Delete & Truncate, Update, Select
- **Chapitre 4 : Les jointures**
- Chapitre 5 : Les Agrégats
- Chapitre 6 : Les Requêtes ensemblistes
- Chapitre 7 : Les Requêtes imbriquées
- Chapitre 8 : Les Requêtes Corréllées
- Chapitre 9 : Les Tableaux croisés
- Chapitre 10 : Les transactions
- Chapitre 11 : Les évènements

Préparation pour les jointures

Création d'une table

- `CREATE TABLE `base_exos`.`villes` (`id_ville` INT NOT NULL AUTO_INCREMENT, `cp` CHAR(5) NOT NULL, `nom_ville` VARCHAR(45) NOT NULL, PRIMARY KEY (`id_ville`), UNIQUE INDEX `cp_UNIQUE` (`cp` ASC) VISIBLE);`

Insertion des données

- `INSERT INTO `base_exos`.`villes` (`cp`, `nom_ville`) VALUES ('75001', 'Paris 01'), ('75015', 'Paris 15'), ('75016', 'Paris 16'), ('75011', 'Paris 11'), ('75020', 'Paris 20');`

Les jointures

- Objectif

- concaténation de tous les enregistrements d'une table T1 avec chaque enregistrement d'une table T2 quand une condition de jointure est satisfaite.

- Plusieurs Types de Jointure

- **Equi-jointure** quand l'opérateur de comparaison est =,
- **La jointure naturelle**,
- **Theta-jointure** quand l'opérateur de comparaison est différent de =,
- **Auto-jointure** quand la jointure s'effectue sur la même table,
- **Jointures externes** quand tous les enregistrements d'une table T1 sont dans le résultat même s'ils n'ont pas de correspondant dans une table T2.

Equi-jointures

- Syntaxe simplifiée

- SELECT clients.id_client, clients.nom, clients.cp,
villes.nom_ville

FROM clients, villes

WHERE clients.cp = villes.cp;

- Syntaxe ANSI (JOIN ou INNER JOIN)

- SELECT c.id_client, c.nom , v.nom_ville

FROM clients c **JOIN** villes v

ON c.cp = v.cp;

Jointure Naturelle

- Jointure sur 2 ou N tables sans spécifier les colonnes de jointure
 - Les colonnes des tables portant le même nom sont jointes automatiquement, peu importe le typage dans chaque table
- Syntaxe simplifiée et ANSI
 - `SELECT *`
`FROM villes NATURAL JOIN clients;`

Auto-Jointure

- Jointure sur la même table
 - Permet de trouver des dépendances entre les données d'une table
- Syntaxe simplifiée
 - `SELECT v1.id_vendeur, v1.nom, v1.chef, v2.nom
"Nom du chef"`
`FROM vendeurs v1, vendeurs v2`
`WHERE v1.chef = v2.id_vendeur;`

Auto-Jointure

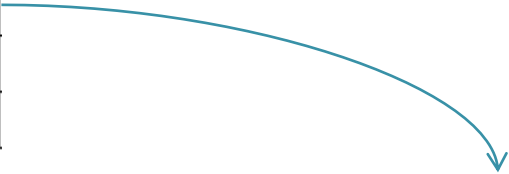
- Syntaxe ANSI

- SELECT v1.id_vendeur, v1.nom, v1.chef, v2.nom
"Nom du chef"

FROM **vendeurs** v1 **JOIN vendeurs** v2

ON v1.chef = v2.id_vendeur;

id_vendeur	nom	chef
1	Lucky	0
2	Dalton	1
3	Mickey	1
4	Donald	2



id_vendeur	nom	chef	Nom du chef
2	Dalton	1	Lucky
3	Mickey	1	Lucky
4	Donald	2	Dalton

Jointure Externe

- Objectif

- Extraire les enregistrements d'une table T1 et d'une Table T2 répondant à une condition , mais aussi ceux de T1 ne répondant pas à la condition.
- La jointure externe peut être à gauche (LEFT OUTER JOIN) ou à droite (RIGHT OUTER JOIN)

- Syntaxe ANSI

- SELECT v.nom_ville, c.nom

FROM villes v **LEFT OUTER JOIN** clients c

ON v.cp = c.cp

ORDER BY nom;



Vue d'ensemble de la formation

- Chapitre 1 : Introduction
- Chapitre 2 : Les Objets et leurs manipulations avec le LDD
Base de Données , Tables, Contraintes, Index, Vues
Procédures Stockées & Fonctions, Triggers
- Chapitre 3 : Les Données et leurs manipulations avec le LMD
Insert & Replace, Delete & Truncate, Update, Select
- Chapitre 4 : Les jointures
- **Chapitre 5 : Les Agrégats**
- Chapitre 6 : Les Requêtes ensemblistes
- Chapitre 7 : Les Requêtes imbriquées
- Chapitre 8 : Les Requêtes Corréllées
- Chapitre 9 : Les Tableaux croisés
- Chapitre 10 : Les transactions
- Chapitre 11 : Les évènements

Agrégats

- Objectifs

- Faire des calculs sur des ensembles d'enregistrements

- Syntaxe simplifiée

- `SELECT fonction_agregat([DISTINCT | ALL] col1
| *), ...`

`FROM nom_de_table;`

Agrégats - Fonctions

Fonction	Description
COUNT(*)	Nombre de lignes d'une table ou satisfaisant une condition.
COUNT([DISTINCT ALL] colonne)	Nombre de valeurs NOT NULL de la colonne.
SUM([DISTINCT ALL] colonne)	Somme des valeurs de la colonne. La colonne est nécessairement numérique. Avec DISTINCT somme les valeurs uniques (sans les doublons).
AVG([DISTINCT ALL] colonne)	Moyenne des valeurs d'une colonne. La colonne est nécessairement numérique. Avec DISTINCT moyenne des valeurs uniques (sans les doublons).
MAX([DISTINCT ALL] colonne)	Maximum des valeurs de la colonne. La colonne n'est pas nécessairement numérique.
MIN([DISTINCT ALL] colonne)	Minimum des valeurs de la colonne. La colonne n'est pas nécessairement numérique.
STDDEV([DISTINCT ALL] colonne)	Ecart type des valeurs de la colonne. La colonne est nécessairement numérique.
VARIANCE([DISTINCT ALL] colonne)	La variance des valeurs de la colonne. La colonne est nécessairement numérique.

Agrégats

- Exemples

- `SELECT COUNT(*) "Nombre de villes" FROM villes;`
- `SELECT COUNT(DISTINCT date_naissance) FROM clients;`
- `SELECT COUNT(DISTINCT nom) "Nombre de noms de client différents " FROM clients;`
- `SELECT MAX(date_naissance) "Naissance la plus récente" FROM clients;`
- `SELECT COUNT(*) "Nombre de clients a Paris" FROM clients c JOIN villes v ON c.cp = v.cp WHERE UPPER(v.nom_ville) LIKE 'PARIS%';`

Agrégats – Group by

- Objectif

- Regrouper les enregistrements lorsque l'on applique une fonction agrégat.

- Syntaxe ANSI

- SELECT v.nom_ville "Ville", **COUNT(*)** "Nombre de clients"

FROM clients c JOIN villes v

ON c.cp = v.cp

GROUP BY v.nom_ville;

Ville	Nombre de clients
Lille	1
Milan	1
Paris 11	5
Paris 12	2
Rome	1

Agrégats – Having

- Objectif

- Appliquer une condition à un ensemble une fois l'agrégation effectuée

- Syntaxe ANSI

- SELECT cp, **COUNT(*)** "Nombre de clients"

FROM clients

GROUP BY cp

HAVING COUNT(cp) > 1;

Retourne les cp dont le nombre de clients > 1

cp	Nombre de clients
75011	5
75012	2

Agrégats – Group_Concat

- Objectif

- Concaténer les valeurs d'une colonne correspondant à un regroupement
- Spécifique à MySql (non standard)

- Syntaxe ANSI

- SELECT v.nom_ville, **GROUP_CONCAT**(c.nom SEPARATOR '+') AS clients FROM villes v INNER JOIN clients c

ON v.cp = c.cp

GROUP BY c.cp, v.nom_ville

nom_ville	clients
Lille	Washington
Paris 11	Lahideb+Napoléon-Bonaparte+Fournier de Sarlovèse+Abou+Lahideb
Paris 12	Tintin+Milou
Rome	Sordi
Milan	Muti

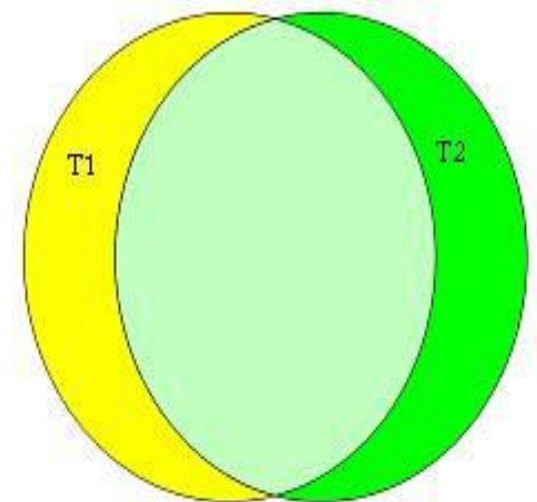


Vue d'ensemble de la formation

- Chapitre 1 : Introduction
- Chapitre 2 : Les Objets et leurs manipulations avec le LDD
Base de Données , Tables, Contraintes, Index, Vues
Procédures Stockées & Fonctions, Triggers
- Chapitre 3 : Les Données et leurs manipulations avec le LMD
Insert & Replace, Delete & Truncate, Update, Select
- Chapitre 4 : Les jointures
- Chapitre 5 : Les Agrégats
- **Chapitre 6 : Les Requêtes ensemblistes**
- Chapitre 7 : Les Requêtes imbriquées
- Chapitre 8 : Les Requêtes Corréliées
- Chapitre 9 : Les Tableaux croisés
- Chapitre 10 : Les transactions
- Chapitre 11 : Les événements

Requêtes ensemblistes

- Types de requêtes
 - L'UNION (**UNION**) renvoie l'ensemble des enregistrements des tables de l'union (Tout)
 - L'INTERSECTION (**INTERSECT**) renvoie l'ensemble des enregistrements communs aux tables de l'intersection (Vert Clair)
 - La différence (**MINUS** ou **EXCEPT**) renvoie l'ensemble des enregistrements qui appartiennent à une table et seulement à cette table. (Jaune ou Vert foncé)



Requêtes ensemblistes-Union

- **Syntaxe**

- SELECT schéma FROM table1

UNION

SELECT schéma FROM table2;

- **Limites**

- Les schéma des tables doivent être identiques
 - Rien ne garantit l'unicité des lignes dans le résultat



Requêtes ensemblistes- Intersect

- Syntaxe

- SELECT schéma FROM table1

INTERSECT

SELECT schéma FROM table2;

- Limites

- N'existe pas chez MySql

- Existe chez MariaDB

Requêtes ensemblistes-Minus

- Syntaxe

- SELECT schéma FROM table1 ...

MINUS | EXCEPT

SELECT schéma FROM table2 ...;

- Limites

- N'existe pas chez MySql

- Existe chez MariaDB



Vue d'ensemble de la formation

- Chapitre 1 : Introduction
- Chapitre 2 : Les Objets et leurs manipulations avec le LDD
Base de Données , Tables, Contraintes, Index, Vues
Procédures Stockées & Fonctions, Triggers
- Chapitre 3 : Les Données et leurs manipulations avec le LMD
Insert & Replace, Delete & Truncate, Update, Select
- Chapitre 4 : Les jointures
- Chapitre 5 : Les Agrégats
- Chapitre 6 : Les Requêtes ensemblistes
- **Chapitre7 : Les Requêtes imbriquées**
- Chapitre8 : Les Requêtes Corréliées
- Chapitre9 : Les Tableaux croisés
- Chapitre10 : Les transactions
- Chapitre11 : Les évènements



Requêtes Imbriquées

- Définition

- Requête qui utilise dans sa clause WHERE le résultat d'une sous- requête

- Retours de la sous-requête

- Une seule ligne et une seule colonne (une seule valeur),
- Une seule colonne (plusieurs valeurs),
- Une seule ligne (plusieurs valeurs),
- Plusieurs lignes et plusieurs colonnes (plusieurs valeurs).

Requêtes Imbriquées

- La sous-requête renvoie un seul résultat et une seule colonne
- Syntaxe
 - `SELECT col1, col2, ... FROM table1`
`WHERE colonne =`
`(SELECT colonne`
`FROM table2`
`WHERE condition)`

Requêtes Imbriquées

- La sous-requête renvoie une seule colonne et plusieurs résultats
- Syntaxe
 - SELECT col1, col2, ... FROM table1
WHERE colonne **IN**
(SELECT colonne
FROM table2
WHERE condition)

Requêtes Imbriquées

- La sous-requête renvoie un seul résultat avec plusieurs colonnes
- Syntaxe
 - SELECT col1, col2, ... FROM table1
WHERE (**col1,col2,...** =
(SELECT **colonne1, colonne2,...**
FROM table2
WHERE condition)

Requêtes Imbriquées

- La sous-requête renvoie plusieurs résultats avec plusieurs colonnes

- Syntaxe

- SELECT col1, col2, ... FROM table1

JOIN

(SELECT **colonne1, colonne2,...**

FROM table2

WHERE condition) **As SR1**

**ON (table1.col1 = SR1.colonne1 AND table1.col2 =
SR1.colonne2)**

Requêtes Imbriquées

- La sous-requête renvoie vrai ou faux
 - La clause EXISTS teste la présence ou l'absence de résultats de la requête imbriquée pour chaque enregistrement du premier SELECT. Elle renvoie True ou False.
 - Permet de Contrôler qu'une valeur existe dans une table avant de faire quelque chose.
- Syntaxe
 - SELECT colonnes FROM table(s) WHERE [NOT] EXISTS (SELECT colonne(s) FROM table(s) [WHERE (conditions)]);

Requêtes Imbriquées

- EXISTS et les contraintes

- Permet de Contrôler qu'une valeur existe dans une table avant de faire quelque chose.

- Syntaxe

- INSERT INTO villes(cp, nom_ville)

SELECT **'75011'**, 'Paris 11'

FROM **dual**

WHERE **NOT EXISTS**

(SELECT *

FROM villes

WHERE cp = '75011');

NOT EXISTS(FALSE)
signifie que l'insertion
peut être exécutée

Requêtes Imbriquées


- EXISTS et les contraintes
 - N'insérera pas l'enregistrement Clients si '75016' n'existe pas dans la table villes.
- Syntaxe

```
INSERT INTO clients(nom, prenom, adresse,  
date_naissance, cp)
```

```
SELECT 'Dandy', 'Germain', 'Rue de  
Passy', '2008-06-25', '75016' FROM dual
```

```
WHERE EXISTS
```

```
(SELECT * FROM villes WHERE cp =  
'75016');
```



Si Retourne
False,
l'insertion
n'est pas
exécutée

Requêtes Imbriquées

- Les Opérateurs ALL & ANY

- combinés aux opérateurs =, >, >=, < et <=.
- **ALL** : la condition est vraie si la comparaison est vraie pour **CHACUNE des valeurs** ramenées par la sous-requête.
- **ANY** : la condition est vraie si la comparaison est vraie pour **AU MOINS une des valeurs** ramenées par la sous-requête..

- Syntaxe

- SELECT liste_expression FROM liste_de_tables
WHERE

<expression> **<opérateur_de_comparaison> {ALL | ANY}**

(SELECT liste_expression FROM liste_de_tables

[WHERE (conditions)]);

Requêtes Imbriquées

- Exemples

- Trouver commande dont le total est le plus élevé.

```
SELECT lc.id_cde "Code commande", SUM(prix * qte) "Total de  
la commande"
```

```
FROM ligcdes lc JOIN produits p
```

```
ON lc.id_produit = p.id_produit
```

```
GROUP BY lc.id_cde
```

```
HAVING SUM(prix*qte) >= ALL
```

```
(SELECT SUM(prix * qte) FROM ligcdes l JOIN produits p
```

```
ON l.id_produit = p.id_produit
```

```
GROUP BY l.id_cde
```

```
);
```



Vue d'ensemble de la formation

- Chapitre 1 : Introduction
- Chapitre 2 : Les Objets et leurs manipulations avec le LDD
Base de Données , Tables, Contraintes, Index, Vues
Procédures Stockées & Fonctions, Triggers
- Chapitre 3 : Les Données et leurs manipulations avec le LMD
Insert & Replace, Delete & Truncate, Update, Select
- Chapitre 4 : Les jointures
- Chapitre 5 : Les Agrégats
- Chapitre 6 : Les Requêtes ensemblistes
- Chapitre 7 : Les Requêtes imbriquées
- **Chapitre 8 : Les Requêtes Corréliées**
- Chapitre 9 : Les Tableaux croisés
- Chapitre 10 : Les transactions
- Chapitre 11 : Les événements

Requêtes Corréliées

- Définition

- Une requête corréliée est une sous-requête qui utilise la requête principale.

- Exemple

- SELECT c.id_commentaire, c.titre_commentaire,
c.date_commentaire, c.id_article

FROM sdp.commentaires **c**

WHERE c.date_commentaire =

(SELECT MAX(date_commentaire)

FROM sdp.commentaires **cInterne**

WHERE **cInterne.id_article = c.id_article**);



Vue d'ensemble de la formation

- Chapitre 1 : Introduction
- Chapitre 2 : Les Objets et leurs manipulations avec le LDD
Base de Données , Tables, Contraintes, Index, Vues
Procédures Stockées & Fonctions, Triggers
- Chapitre 3 : Les Données et leurs manipulations avec le LMD
Insert & Replace, Delete & Truncate, Update, Select
- Chapitre 4 : Les jointures
- Chapitre 5 : Les Agrégats
- Chapitre 6 : Les Requêtes ensemblistes
- Chapitre 7 : Les Requêtes imbriquées
- Chapitre 8 : Les Requêtes Corréliées
- **Chapitre 9 : Les Tableaux croisés**
- Chapitre 10 : Les transactions
- Chapitre 11 : Les événements

Tableaux Croisés

- Définition

- Un TC (Tableau croisé) permet de passer d'une représentation 1D à une représentation 2D.
- Il s'agit d'un TC Statique. (Colonnes définies en dur). La création d'un TC vraiment dynamique passe par un langage de programmation

- Exemple

- ```
CREATE VIEW tcd.v_ventes_tcd_statique_table AS
SELECT nom_vendeur,
SUM(IF(designation = 'Evian' , vente, 0)) AS 'Evian',
SUM(IF(designation = 'Graves' , vente, 0)) AS 'Graves',
SUM(IF(designation = 'Badoit' , vente, 0)) AS 'Badoit'
FROM tcd.ventes_croisees
GROUP BY nom_vendeur;
```



# Vue d'ensemble de la formation

- Chapitre 1 : Introduction
- Chapitre 2 : Les Objets et leurs manipulations avec le LDD  
Base de Données , Tables, Contraintes, Index, Vues  
Procédures Stockées & Fonctions, Triggers
- Chapitre 3 : Les Données et leurs manipulations avec le LMD  
Insert & Replace, Delete & Truncate, Update, Select
- Chapitre 4 : Les jointures
- Chapitre 5 : Les Agrégats
- Chapitre 6 : Les Requêtes ensemblistes
- Chapitre 7 : Les Requêtes imbriquées
- Chapitre 8 : Les Requêtes Corréliées
- Chapitre 9 : Les Tableaux croisés
- **Chapitre 10 : Les transactions**
- Chapitre 11 : Les événements

# Transactions

- Garantir les propriétés ACID
  - Atomicité : toutes les actions élémentaires sont effectuées ou aucune.
  - Cohérence : A tout moment l'état des données est cohérent
  - Isolation : d'autres actions ne peuvent accéder aux données pendant la transaction.
  - Durabilité : une fois la transaction validée , les données sont persistées . Elle ne peut plus être défaite.

# Transactions

- Paramétrage

- Par défaut MySQL est lancé avec l'option AutoCommit=true .  
Toutes les MAJ sont immédiatement persistées.
  - Possibilité de modifier My.ini init\_connect='SET autocommit=0'
  - L'utilisateur 'root' est toujours en autocommit=true
- Changer le paramétrage dynamiquement
  - SET AUTOCOMMIT = 0;

# Transactions

- Les étapes d'une transaction :

**START TRANSACTION;**

**SAVEPOINT sp1;**

INSERT INTO villes(cp, nom\_ville) VALUES('75031','Paris 31');

**SAVEPOINT sp2;**

INSERT INTO villes(cp, nom\_ville) VALUES('75032','Paris 32');

**ROLLBACK TO SAVEPOINT sp2;**

**COMMIT;**

# Transactions

- Verrouillage des tables pendant la transaction

## Syntaxe

- LOCK TABLES nom\_de\_table verrouillage [, nom\_de\_table verrouillage];
  - LOCK TABLES villes READ: autorise lecture, interdit écriture
  - LOCK TABLES villes WRITE : interdit lecture et écriture

## Exemple

| Utilisateur                                                                                                                                                                    | Autre utilisateur                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| <pre>SELECT * FROM villes;  START TRANSACTION;  LOCK TABLES villes <b>WRITE</b>;  UPDATE villes SET nom_ville = 'Marsiglia' WHERE cp = '13000';  UNLOCK TABLES;  COMMIT;</pre> | <pre>SELECT * FROM villes; -- <b>KO</b>  UPDATE villes SET nom_ville = 'Marsilia' WHERE cp = '13000'; -- <b>KO</b></pre> |

# Transactions

- Verrouillage de lignes pendant la transaction
- ## Syntaxe
- `SELECT * FROM nomDeTable WHERE condition FOR UPDATE;`

## Exemple

| Utilisateur                                                                                                                                          | Autre utilisateur                                                                                                       |
|------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <pre>START TRANSACTION;  SELECT * FROM pays WHERE id_pays = '033' FOR UPDATE;  UPDATE pays SET nom_pays = 'FR' WHERE id_pays = '033';  COMMIT;</pre> | <pre>START TRANSACTION;  UPDATE pays SET nom_pays = 'fr' WHERE id_pays = '033'; -- Attente bloquante (*)  COMMIT;</pre> |



# Vue d'ensemble de la formation

- Chapitre 1 : Introduction
- Chapitre 2 : Les Objets et leurs manipulations avec le LDD  
Base de Données , Tables, Contraintes, Index, Vues  
Procédures Stockées & Fonctions, Triggers
- Chapitre 3 : Les Données et leurs manipulations avec le LMD  
Insert & Replace, Delete & Truncate, Update, Select
- Chapitre 4 : Les jointures
- Chapitre 5 : Les Agrégats
- Chapitre 6 : Les Requêtes ensemblistes
- Chapitre 7 : Les Requêtes imbriquées
- Chapitre 8 : Les Requêtes Corréliées
- Chapitre 9 : Les Tableaux croisés
- Chapitre 10 : Les transactions
- **Chapitre 11 : Les évènements**



# Les évènements

- Objectifs

- Créer un événement pour exécuter une action, une commande SQL.
- L'événement peut gérer une action récurrente (un archivage tous les mois) ou unique (une suppression de données dans 2 minutes ...).

- Syntaxe

- CREATE EVENT [bd.]nom\_d\_evènement  
ON SCHEDULE horaire  
DO commandeSQL;
- horaire
  - ON SCHEDULE AT heure [+ INTERVAL intervalle]
  - ON SCHEDULE intervalle

# Les évènements

- Exemples

- CREATE EVENT cours.tous\_les\_mois  
ON SCHEDULE EVERY 1 MONTH  
DO DELETE FROM cours.villes\_bis;

- CREATE EVENT cours.dans\_2\_minutes  
ON SCHEDULE AT CURRENT\_TIMESTAMP + INTERVAL 2  
MINUTE  
DO DELETE FROM cours.villes\_bis;