

Base de données MYSQL

Simon Mielcarek

Plan de la formation

- ▶ Introduction aux SGBD
- ▶ Principes et concepts du modèle relationnel
- ▶ Langage SQL => Requêtes sur une table
- ▶ Langage SQL => Requêtes sur plusieurs tables
- ▶ Mises à jour des tables
- ▶ Sous requêtes

Introduction aux SGBD

Définitions

- ▶ Base de données définition :
 - ▶ Ensemble de données organisé en vue de son utilisation par des programmes correspondant à des applications distinctes et de manière à faciliter l'évolution indépendante des données et des programmes.
- ▶ Décrit correctement la base de données mais très limité
- ▶ Une base de données doit ne sert à rien sans son SGBD : Système de Gestion de base de données
 - ▶ Outil logiciel qui assure l'interface entre les utilisateurs et les base de données
 - ▶ MySQL
 - ▶ PostgreSQL

Définitions

- ▶ Base de donnée : endroit ou est stockée l'information
- ▶ SGBD : ce qui permet d'y accéder
- ▶ Plusieurs générations se sont succédées avant d'arriver aux modèles actuels
 - ▶ SGBD/R => Relationnel
 - ▶ NoSQL => Big Data

Définitions

- ▶ **Pilote :**
 - ▶ Interface d'accès entre le SGBD et le programme
 - ▶ Permet de normaliser l'accès aux BDD peu importe le SGBD
 - ▶ Ils permettent :
 - ▶ De réaliser des connexions vers les BDD
 - ▶ D'effectuer des requêtes
 - ▶ De récupérer les résultats et les erreurs
 - ▶ Plusieurs technologies disponibles : ODBC, JDBC, PDO

Fonctions d'un SGBD

- ▶ Décrire les données indépendamment des applications : Data Definition Language (DDL)
- ▶ Manipuler les données (mise à jour et interrogations) : Data manipulation Language (DML)
- ▶ Contrôler les données (DCL):
 - ▶ Vérifier l'intégrité des données : pas de doublons ou d'orphelins, pas de données incohérentes
 - ▶ Confidentialité
- ▶ Gérer la concurrence : transactions
- ▶ Sécuriser les données : possibilité de reprise après un plantage

Concepts et principes du modèle relationnel

Le modèle relationnel

- ▶ Produit cartésien entre des éléments
 - ▶ On appelle « relation » un ensemble d'attributs qui caractérisent une proposition ou une combinaison de propositions
 - ▶ Exemple : un employé a un matricule, il a un nom, il a un employeur
- ▶ Chaque combinaison de propositions ainsi formée est appelée tuple

<i>SALARIE</i>	Prénom	Fonction
	Paul	CDP
	Pierre	AP
	André	P

Ligne (ou tuple) →

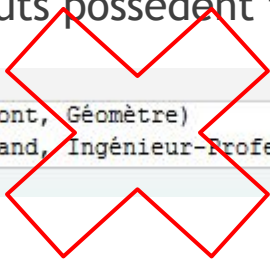
↑
Colonne ou Attribut

Table

- ▶ Une table est une implémentation de la relation
 - ▶ Elle est définie par :
 - ▶ Son degré : nombre d'attributs
 - ▶ Sa cardinalité : nombre de ligne
 - ▶ Son schéma : nom de la relation, liste d'attributs

Formes normales

- ▶ Pour garantir l'intégrité des données nous utilisons les formes normales
 - ▶ Il y a 7 Formes normales au total
 - ▶ Nous allons voir les 3 principales
 - ▶ Relations spécifiques
- ▶ 1^{ère} forme normale :
 - ▶ « Est en première forme normale, une relation (ayant par définition une clé) dont les attributs possèdent tous une valeur **sémantiquement atomique** »



1	(Dupont, Géomètre)
2	(Durand, Ingénieur-Professeur)

1	(Dupont, Géomètre)
2	(Durand, Ingénieur)
3	(Durand, Professeur)

Formes normales

- ▶ 2^{ème} forme normale
 - ▶ Respecte la deuxième forme normale, la relation respectant la première forme normale et respectant le principe suivant :
 - ▶ Les attributs d'une relation sont divisés en deux groupes :
 - ▶ le premier groupe est composé de la clé (un ou plusieurs attributs).
 - ▶ Le deuxième groupe est composé des autres attributs (éventuellement vide).
 - ▶ La deuxième forme normale stipule que tout attribut du deuxième groupe ne peut pas dépendre d'un sous-ensemble (strict) d'attribut(s) du premier groupe.
 - ▶ En d'autres termes : « Un attribut non clé ne dépend pas d'une partie de la clé mais de toute la clé. »

Formes normales

- ▶ 3^{ème} forme normale
 - ▶ Les attributs d'une relation sont divisés en deux groupes :
 - ▶ le premier groupe est composé de la clé (un ou plusieurs attributs)
 - ▶ Le deuxième groupe est composé des autres attributs (éventuellement vide)
 - ▶ Respecte la troisième forme normale, la relation respectant la deuxième forme normale et respectant le principe suivant :
 - ▶ tout attribut du deuxième groupe ne peut pas dépendre que d'un sous-ensemble (strict et excluant l'attribut considéré) d'autres attribut(s) du second groupe

Pour résumer

- ▶ Une relation est 3NF quand
 - ▶ Elle a une clé primaire qui identifie chaque ligne de manière unique
 - ▶ Tout attribut non clé dépend :
 - ▶ De la clé
 - ▶ De toute la clé
 - ▶ Rien que de la clé

Les valeurs Null

- ▶ Un attribut peut être dit « null »
 - ▶ Cela veut dire qu'il est « vide »
 - ▶ Il y a 2 sous cas :
 - ▶ Soit l'attribut n'a pas de sens
 - ▶ Soit l'attribut n'est pas connu il est marqué A-Mark
 - ▶ Dans tous les cas il sera marqué comme « null »
 - ▶ ATTENTION : « null » ne veut pas dire = 0
 - ▶ Intérêt ?
 - ▶ Permet de sélectionner des ensemble ou là valeur n'est pas connu
 - ▶ Exemple : date d'obtention du BAC

Index

Structure de donnée qui permet de retrouver facilement des données

Exemple : les numéro de pages d'un bouquin

=> Permet de se repérer rapidement dans son livre

Structure triée

Cela permet d'accélérer les opérations de recherche

“Pour trouver un livre dans une bibliothèque,
au lieu d'examiner un par un chaque livre (*ce qui correspond à une recherche séquentielle*),
il est plus rapide de consulter le catalogue où ils sont classés par thème, auteur et titre”

PRIMARY KEY FOREIGN KEY

Les clés (key) sont des index à part entière. Ce sont même des synonymes

Ils peuvent être de plusieurs types :

Primary key : clé primaire, c'est la principale qui sera utilisée pour identifier une ligne. Elle est unique.

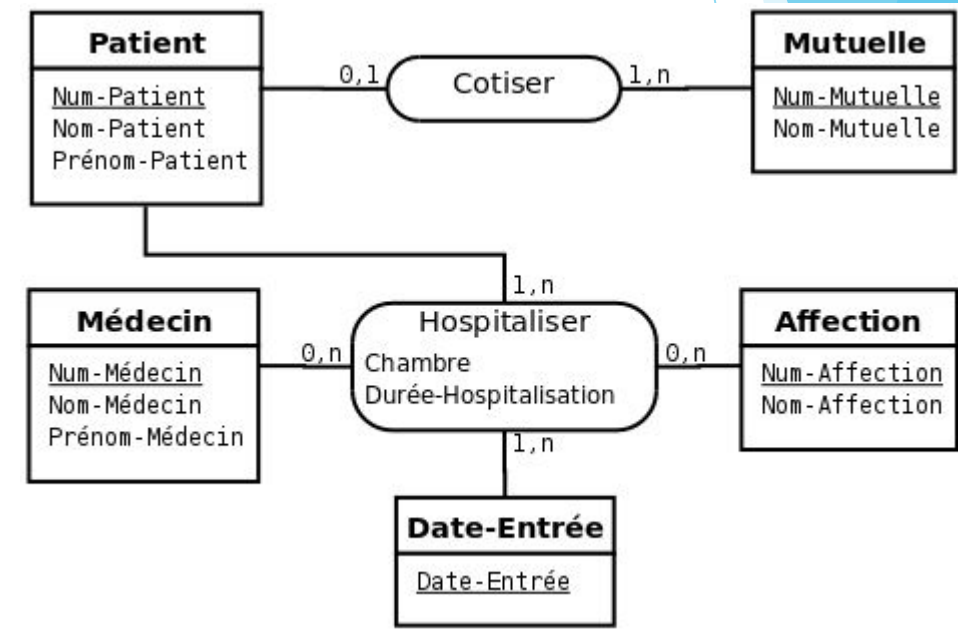
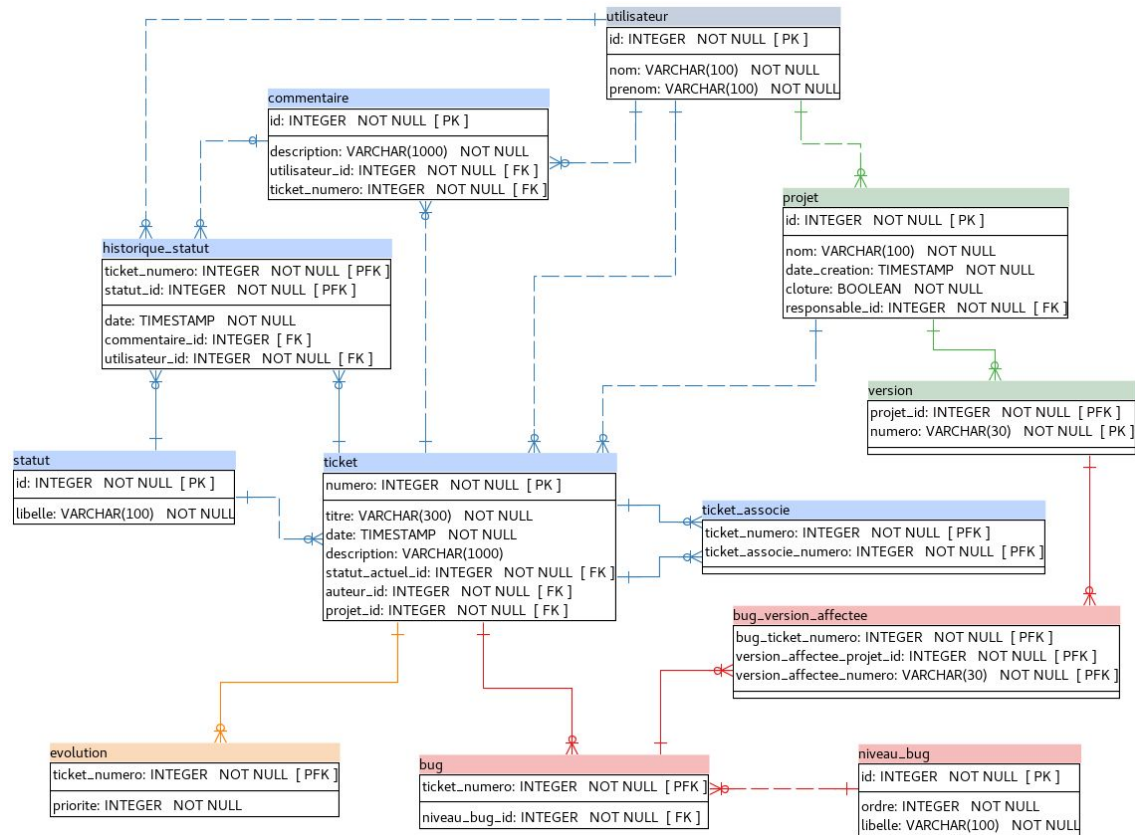
Foreign key : clé étrangère, elle fait référence à une autre table (primary key)

Permet de rechercher rapidement mais aussi d'ajouter des contraintes pour garder une base de données "cohérente".

Modélisation

- ▶ Clé primaire : clé permettant d'identifier une ligne
- ▶ Clé étrangère : Clé appartenant à une autre table mais qui permet de lier les deux
 - ▶ Exemple : Client / Facture
 - ▶ Facture / Ligne
- ▶ UML : Unified modeling language
- ▶ Merise

Exemple d'UML



Exercice : modélisation d'une BDD

- ▶ Modéliser la base de données suivante :
 - ▶ Nous souhaitons un système de gestion de facture
 - ▶ Les clients sont représentés par : un code client, nom de société, adresse, code postal, ville, numéro de téléphone
 - ▶ Les factures :
 - ▶ Appartiennent à un client
 - ▶ Ont un numéro de facture
 - ▶ Une date
 - ▶ Plusieurs lignes correspondant à des produits
 - ▶ On peut appliquer une réduction et une TVA sur les lignes produits et une quantité
 - ▶ Les produits sont représentés par un code produit, un libellé, un prix et un stock

Modélisation de la BDD

- ▶ Déterminer les clés primaires et les attributs
- ▶ Déterminer les relations entre les tables
- ▶ Faire le schéma UML
- ▶ Modéliser dans MySQL Workbench

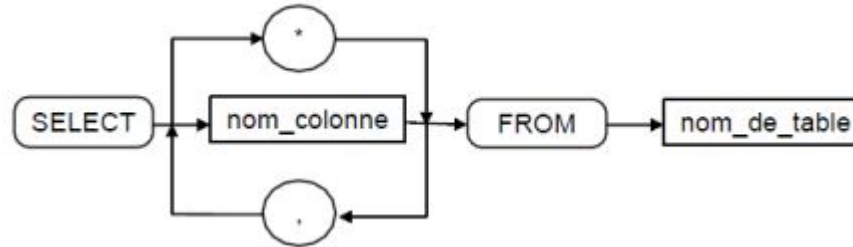
Le langage SQL : requêtes sur une table

Qu'est ce que SQL ?

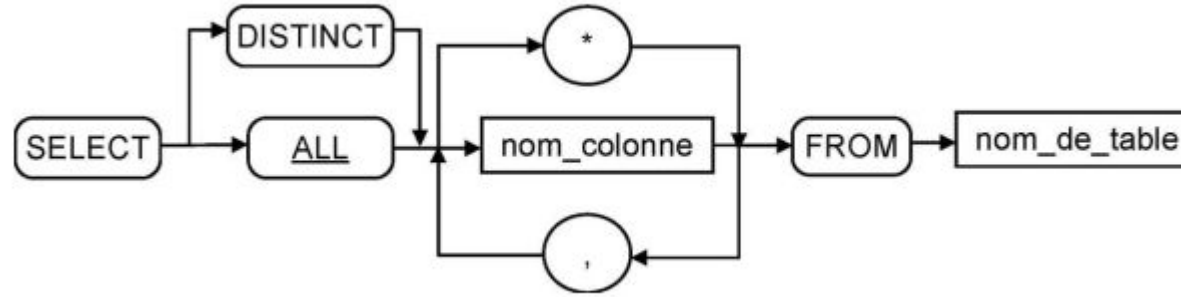
- ▶ Structured Query Language
- ▶ Langage Non-procédural de manipulation de données
 - ▶ Fait l'objet d'une normalisation
 - ▶ Des différences peuvent être trouvées entre les SGBD
- ▶ Exploitation des base de données Relationnelles
- ▶ Evolution vers le Big Data : NoSQL

La projection (SELECT)

- ▶ SELECT est la commande dont vous vous servirez le plus
- ▶ Permet de sélectionner un ensemble de données
- ▶ SELECT permet de sélectionner des colonnes FROM une ou plusieurs tables
 - ▶ **SELECT** col1,col2,col3
FROM table1
 - ▶ ***SELECT** adresse,cp
FROM clients*



DISTINCT



- ▶ Elimination des doublons :

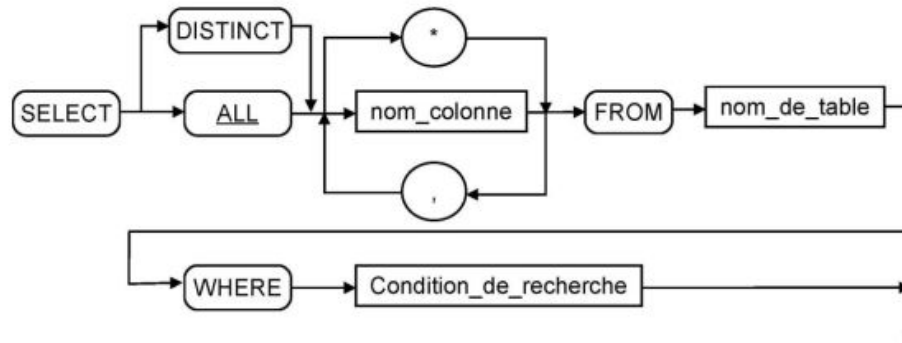
- ▶ Mot clé DISTINCT

- ▶ **SELECT DISTINCT** col1,col2
FROM table1

- ▶ **SELECT** cp, ville
FROM clients

Permet de récupérer toutes les villes et les code postaux (ville ou CEDEX par exemple) dans notre BDD

WHERE



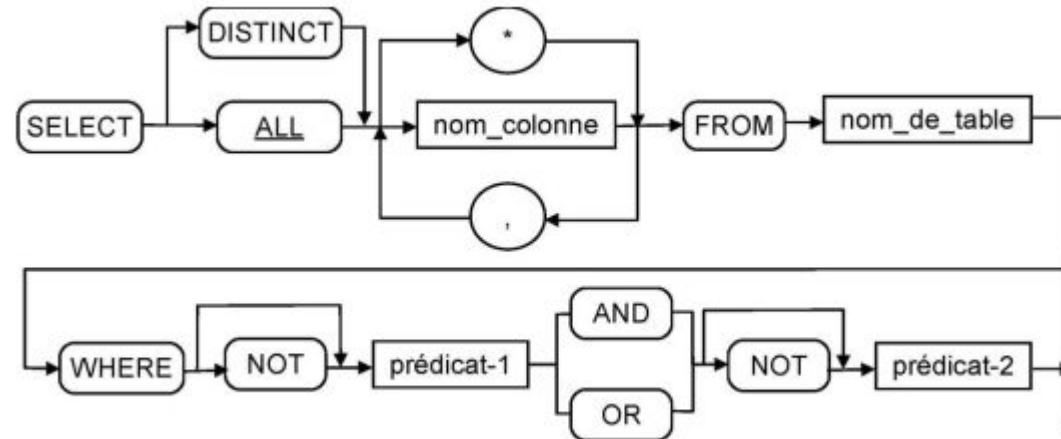
- ▶ SQL nous permet de faire des filtres pour sélectionner seulement une partie des informations
 - ▶ Exemple tout à l'heure : toutes les personnes avec une date de BAC nulle
 - ▶ Mot clé : WHERE
 - ▶ **SELECT** col1,col2,col3
FROM table1
WHERE condition
 - ▶ **SELECT** adresse,cp
FROM clients
WHERE codcli > 1000
- SELECT** adresse,cp
FROM clients
WHERE ville LIKE '%PARIS%'

WHERE opérateurs

Restriction souhaitée		Opérateur
Égal		=
Différent		<> ou !=
Inférieur		<
Inférieur ou égal		<= ou ! >
Supérieur		>
Supérieur ou égal		>= ou ! <
OU logique		OR
ET logique		AND
NON logique		NOT
Fourchette de valeurs (bornes incluses)		BETWEEN valeur1 and valeur2 NOT BETWEEN valeur1 and valeur2
Liste de valeurs		IN (liste de valeurs) NOT IN (.....)
Recherche de caractères	1 caractère quelconque	LIKE 'DUPON_' NOT LIKE 'DUPON_'
	Chaîne de caractères quelconques	LIKE 'DUP%' NOT LIKE 'DUP%'
Sélection d'une colonne "null"		nom_de_colonne IS NULL nom_de_colonne IS NOT NULL

WHERE

- WHERE accepte plusieurs conditions :



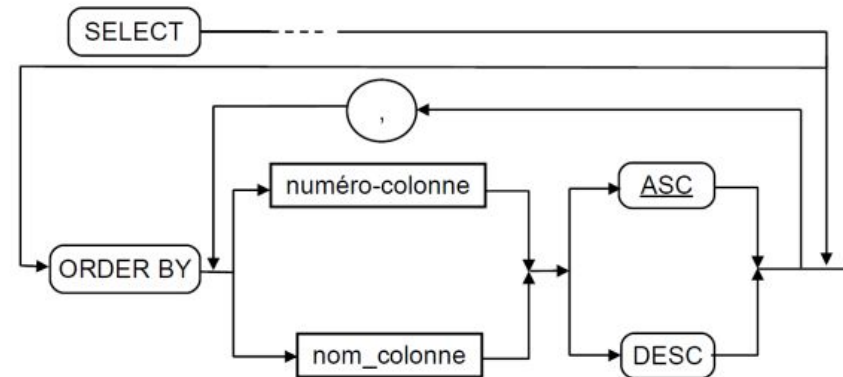
SELECT *adresse,cp*
FROM *clients*
WHERE *ville LIKE '%PARIS%' OR ville LIKE '%LILLE%'*

ORDER BY

- ▶ Il est possible de trier les résultats grâce à l'opérateur ORDER BY

- ▶ **SELECT** libelle, codpro
FROM produits
ORDER BY libelle

- ▶ On peut mettre ASC ou DESC selon si on veut organiser de manière croissance ou décroissante



LIMIT

Permet de limiter les résultats :

LIMIT 10 => Limiter le nombre de résultats à 10

On peut aussi “sauter” des éléments :

LIMIT 0,10 => Sauter 0 résultats, et limiter les résultats à 10

LIMIT 10,10 => Sauter 10 résultats, et limiter les résultats à 10

Exemple : des pages produits ($\text{pageActuelle} * \text{NbElements}, \text{NbElements}$)

Opérateurs arithmétiques

- ▶ Il est possible de faire des opérations sur les colonnes
- ▶ Par exemple transformer un prix en prix T.T.C
- ▶ **SELECT** prix * 1.2
FROM produits
- ▶ Trouver les prix qui coutent plus de 5€ T.T.C
- ▶ **SELECT** codpro, libelle
FROM produits
WHERE prix * 1.2 > 5

Le mot clé AS

- ▶ Dans tout langage SQL il est possible de donner un alias à une colonne, un resultat ou une table
- ▶ Mot clé AS :

```
SELECT prix * 1,2 AS prix_ttc  
FROM produits
```

```
SELECT p.prix * 1,2 AS 'prix ttc'  
FROM produits AS p
```

Cela va nous permettre d'identifier plus simplement les colonnes ou tables soit dans nos requêtes soit dans nos résultats

Exerçons nous !

- ▶ Plusieurs séries d'exercices à disposition

Requêtes sur plusieurs tables : Jointures

Jointures simples :

2 tables

- ▶ Les jointures naturelles sont des jointures dites « internes »
- ▶ Par défaut elles éliminent les résultats « vides » (nous allons voir cela en détail)
- ▶ Comment faire une jointure ?
- ▶ Il y a 2 écritures possible, une normalisée et une non normalisée.
 - ▶ Pour les exercices, entraînez vous à faire les deux !!!
 - ▶ Ecriture normalisée :
`SELECT nom_societe, no_facture, date`
`FROM clients`
`INNER JOIN factures`
`ON clients.codclient = factures.codclient`
 - ▶ Ecriture non normalisée
`SELECT nom_societe, no_facture, date`
`FROM clients, factures`
`WHERE clients.codclient = factures.codclient`
- ▶ Remarquez qu'on précise une condition de jointure. Si on ne le fait pas ce sera simplement un produit cartésien des deux tables !

Ajout de critères : WHERE, ORDER BY

- ▶ Bien sûr les conditions que nous avons vu précédemment sont possibles !
- ▶ Facture depuis le premier janvier 2018
 - ▶ Ecriture normalisée :
SELECT nom_societe, no_facture, date
FROM clients
INNER JOIN factures
ON clients.codclient = factures.codclient
WHERE date > '2018-01-01'
 - ▶ Ecriture non normalisée
SELECT nom_societe, no_facture, date
FROM clients, factures
WHERE clients.codclient = factures.codclient
AND date > '2018-01-01'
- ▶ Notez que date n'a pas de prefix alors que codclient en à un !
 - ▶ Le même nom de colonne étant présent dans les deux tables il faut préciser sur quelle colonne nous faisons la comparaison !

Jointures :

Plus de 2 tables

- ▶ Ecriture normalisée :
`SELECT libelle, f.no_facture
FROM factures AS f
INNER JOIN lignes_factures AS lf
ON f.no_facture = lf.facture
INNER JOIN produits AS p
ON lf.codproduit = p.codproduit`
- ▶ Ecriture non normalisée :
`SELECT libelle, f.no_facture
FROM factures AS f, lignes_factures AS lf, produits AS p
WHERE f.no_facture = lf.facture
AND lf.codproduit = p.codproduit`
- ▶ Possibilité d'ajouter d'autres conditions derrière
Exemple : `WHERE f.date > '2010-01-01'`

Auto-jointure

- ▶ Jointure d'une table sur elle-même
- ▶ Par exemple, les factures émises le même jour :
SELECT f1.no_facture, f2.no_facture
FROM factures **AS** f1
INNER JOIN factures **AS** f2
ON f1.date = f2.date
WHERE f1.no_facture > f2.no_facture
- ▶ On rajoute f1.no_facture > f2.no_facture pour éviter le doublons
En effet on fait un produit cartésien des deux tables donc la facture f1.X aura forcément la même date que la facture f2.X (X étant la clé primaire)

Thêta-jointure

- ▶ L'opérateur de comparaison sur la colonne de jointure est quelconque
- ▶ Exemple :
 - ▶ Liste des produits ayant un prix supérieur au produit numéro 5
 - ▶ `SELECT p1.codpro, p1.libelle, p1.prix`
`FROM produits AS p1`
`INNER JOIN produits AS p2`
`ON p1.prix > p2.prix`
`WHERE p2.codproduit = 5`

Exercices

- ▶ Partie 1 des jointures

Jointures externes

- ▶ Les jointures externes, contrairement aux jointures interne ne vérifient pas que les conditions soient remplies pour afficher un résultat
- ▶ On distingue 3 cas : LEFT, RIGHT et FULL
 - ▶ Exemple :
SELECT nom_societe, no_facture, date
FROM clients
LEFT OUTER JOIN factures
ON clients.codclient = factures.codclient
 - ▶ INNER JOIN : n'affiche que les clients ayant des factures et leurs factures associées
 - ▶ LEFT OUTER JOIN : affiche tous les clients même s'ils n'ont pas de facture
 - ▶ RIGHT OUTER JOIN : affiche toutes les factures même si elles n'ont pas de client
 - ▶ FULL OUTER JOIN : affiche tous les clients et les factures
- ▶ **IMPORTANT** : si une valeur ne peut pas être affichée elle est remplacée par NULL

Exercices

- ▶ Partie 2 des exercices

Sous-requêtes

Sous-requêtes : imbrication

- ▶ Dans la condition d'une requête il est possible d'utiliser le résultat d'une seconde requête : SOUS REQUÊTE ou requête imbriquée
- ▶ Attention on ne peut pas faire n'importe quoi selon l'opérateur !!

```
SELECT codpro  
FROM lignes_facture  
WHERE no_facture IN  
      ( SELECT no_facture  
        FROM factures  
        WHERE codcli = 11)
```

- ▶ Cette requête va retourner tous les code produit que le client 11 à commandé
- ▶ En réalité SQL va traduire cela par une jointure

Requêtes corrélées

```
▶ SELECT no_facture  
FROM facture AS f  
WHERE 7 IN  
    (SELECT codpro  
     FROM lignes_facture  
     WHERE f.no_facture = no_facture)
```

Pour chaque facture, nous allons sélectionner ses lignes_factures. Puis nous allons vérifier si le code produit est égal à 7. Si c'est le cas nous allons sélectionner cette facture et l'afficher

Opérateurs

- ▶ Pour les opérateurs de type :
 - ▶ =, !=, >, <, >=, <= : le résultat de la sous requête ne doit fournir qu'une valeur
 - ▶ Pour : ALL ou ANY il est possible d'avoir plusieurs résultats, les opérateurs ci-dessus peuvent être combinés avec ces opérateurs :
 - ▶ > ALL => supérieur à tous les résultats fournis
 - ▶ = ANY : Une idée ? D'équivalence avec un opérateur connu ?

Exists

- ▶ **SELECT** no_facture
FROM facture AS f
WHERE EXISTS
 (SELECT 'coucou'
 FROM lignes_facture
 WHERE f.no_facture = no_facture
 AND codpro = 7)
- ▶ Retourne aussi les no_facture contenant le produit numéro 7
- ▶ On vérifie que c'est « vrai », tout ce qui n'est pas null est vrai

Not exists

- ▶ **SELECT** no_facture
FROM facture AS f
WHERE NOT EXISTS
 (SELECT 'allo'
 FROM lignes_facture
 WHERE f.no_facture = no_facture
 AND codpro = 7)
- ▶ Toutes les facture ne contenant pas le produit numéro 7

Sous-requêtes : exercices

- ▶ A vous de jouer !

Fonctions récapitulatives et partitionnement

COUNT

- ▶ COUNT : permet de compter le nombre d'éléments
- ▶ **SELECT COUNT(*)**
FROM clients
- ▶ Nombre de clients dans la base

- ▶ **SELECT COUNT(DISTINCT ville)**
FROM clients
- ▶ Nombre de ville dans lequel nous avons des clients

MAX et MIN

- ▶ **SELECT MAX(prix)**
FROM produits
- ▶ Le prix le plus haut référencé

- ▶ **SELECT MIN(stock)**
FROM produits
- ▶ Le stock le plus bas dans nos produits

AVG et SUM

- ▶ **SELECT AVG(quantite)**
FROM ligne_produits
- ▶ La moyenne des quantités commandés
- ▶ **SELECT SUM(quantite), no_facture**
FROM ligne_produits
GROUP BY no_facture
- ▶ Le nombre de produits commandés PAR facture

Partitionnement

- ▶ GROUP BY : permet de regrouper des résultats selon une colonne choisie
- ▶ C'est ce qu'on appelle le partitionnement logique !
- ▶ HAVING : permet lorsqu'on utilise le partitionnement, de faire l'équivalent d'un « WHERE », il permet d'éliminer les lignes qui ne répondent pas aux critères demandés
- ▶ Exemple :
- ▶ **SELECT** AVG(quantite), no_facture
FROM ligne_produits
GROUP BY no_facture
HAVING SUM(quantite) > 5
- ▶ Permet de récupérer les quantités commandées par facture, si le nombre de produit de la facture est supérieur à 5

Résumé de SELECT

SELECT ...	←	Projection
FROM ... AS ...		
INNER JOIN ...	←	Jointure
ON ...		
WHERE ...	←	Restriction
(SELECT ...		
FROM ...	←	Sous-requête
WHERE ...)		
GROUP BY ...		
HAVING ... >	←	Restriction
(SELECT ...)	←	Sous-requête
ORDER BY ...	←	Tri

Exercices

- ▶ Exercices sur fonctions de partitionnement

Autre opérateurs

UNION

- ▶ On peut combiner des requêtes entre elles grâce à UNION
- ▶ La seule condition c'est que les colonnes aient le même type

- ▶

```
SELECT nom_societe  
FROM clients  
WHERE nom_societe LIKE '%SARL'
```

UNION

```
SELECT nom_societe  
FROM clients  
WHERE ville LIKE 'PARIS'
```

- ▶ Nous retournera la liste des sociétés de type SARL
Ainsi que celles se trouvant sur Paris

INNERSECT et MINUS

- ▶ Pas implémenté en MySQL
- ▶ INNERSECT permet de retourner les listes présentes dans les deux requêtes
- ▶ MINUS la différence entre les deux requêtes

Mise à jour d'une base

INSERT : Ajout de lignes

- ▶ Syntaxe :
- ▶ **INSERT INTO** *table*
[(*colonne1*, *colonne2*, *colonne3*...)]
VALUES (*valeur1*, *valeur2*, *valeur3*...)[,
(*valeur1'*, *valeur2'*, *valeur3'*)]

UPDATE : mise à jour d'une ligne

- ▶ **UPDATE** nom_table
SET *colonne1* = valeur1, *colonne2* = valeur2
WHERE condition
- ▶ ATTENTION il est important de préciser une condition !!
- ▶ Pas de conditions = mise à jour de toutes les lignes de la table

DELETE

- ▶ **DELETE FROM** nom_table
WHERE condition
- ▶ IDEM QU'UPDATE !! Pas de condition = suppression de **TOUTES** les lignes

TRANSACTIONS

- ▶ Qu'est ce qu'une transaction ?
 - ▶ Groupe de requêtes liées entre elles
 - ▶ Elles sont **TOUTES** exécutées ou **AUCUNE**
- ▶ Exemple : Jean envoie 50€ à Jeanne par virement
 - ▶ **BEGIN TRANSACTION;**
UPDATE comptes
SET solde = (solde - 50)
WHERE nom **LIKE** 'Jean';
UPDATE comptes
SET solde = (solde + 50)
WHERE nom **LIKE** 'Jeanne';
COMMIT;
 - ▶ Les deux soldes doivent être mis à jour « simultanément »
 - ▶ **COMMIT** permet de valider
 - ▶ **ROLLBACK** permet d'annuler une transaction

TRIGGER

Permet l'automatisation de tâches dans SQL :

Exemple :

```
CREATE TRIGGER `ins_film` AFTER INSERT ON `film` FOR EACH ROW BEGIN  
    INSERT INTO film_text (film_id, title, description)  
    VALUES (new.film_id, new.title, new.description);  
END;;
```

Permettra d'ajouter un élément à film_text à l'insertion d'un élément dans film

<https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html>