

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 Création d'une table (LDD)
- 6 Langage de manipulation des données (LMD)
- 7 Interrogation de la base

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 Création d'une table (LDD)
- 6 Langage de manipulation des données (LMD)
- 7 Interrogation de la base

Présentation de SQL

SQL = Structured Query Language

= langage d'interrogation structuré

Langage de gestion de bases de données relationnelles pour

- interroger
- mettre à jour (LMD ; Langage de Manipulation des Données)
- définir les données (LDD ; Langage de Définition des Données gérées)
- contrôler l'accès aux données (LCD ; Langage de Contrôle de l'accès aux Données)

Définition (Norme SQL92 (ou SQL2))

Norme adoptée en 1992 par l'ISO (International Organisation for Standardization)

Presque complètement implémentée par les principaux SGBD : Oracle, DB2, Informix, MySQL, PostgreSQL, Access, SQL Server,...

Depuis, de nouvelles normes (SQL99 (ou SQL3), SQL:2003 et SQL:2008) ont été proposées. Elles ajoutent certaines fonctionnalités mais elles ne sont pas toutes implémentées dans tous les SGBD.

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 Création d'une table (LDD)
- 6 Langage de manipulation des données (LMD)
- 7 Interrogation de la base

- SGBDR (Relationnel) qui utilise le langage SQL
- Langage procédural suivant la norme SQL:2003

Nombreux programmes utilitaires

- mysql : SQL interactif en ligne de commandes
- MySQL workbench : saisir/voir des données, créer/administrer des serveurs, ...
- phpmyadmin pour l'interface avec le Web
- ...

Identificateurs pour les objets manipulés

- 30 caractères au plus : lettres, chiffres, _, \$ ou # (commence par une lettre)
- pas un mot clef

Quelques mots clefs : ASSERT, ASSIGN, AUDIT, COMMENT, DATE, DECIMAL, DEFINITION, FILE, FORMAT, INDEX, LIST, MODE, OPTION, PARTITION, PRIVILEGES, PUBLIC, SELECT, SESSION, SET, TABLE.

Relations stockées sous forme de tables composées de lignes et de colonnes

SQL92 : le nom d'une table est précédé du nom d'un schéma (pour réunir tous les objets liés à un même thème) Par défaut, le schéma est le nom de l'utilisateur connecté

Exemple (Exemple de table : DEPT)

| Dept | NomD | Lieu |
|------|-----------|----------|
| 10 | Finances | Paris |
| 20 | Recherche | Grenoble |
| 30 | Ventes | Lyon |

Exemple (Exemple de table : EMP)

| Matr | Nom | Sal | Com | Sup | Dept |
|------|---------|------|-----|------|------|
| 1200 | Dupond | 2500 | 300 | 2200 | 10 |
| 2200 | Durand | 3000 | 500 | | 10 |
| 1780 | Boisier | 2500 | | 2200 | 20 |

- Toutes les données d'une colonne sont d'un même type
- Identificateur unique pour les colonnes d'une table, mais 2 colonnes dans 2 tables différentes peuvent avoir le même nom
- Le nom complet d'une colonne comprend le nom de la table à laquelle elle appartient (obligatoire en cas d'ambiguïté) :
DEPT.dept, BERNARD.DEPT.dept

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 Création d'une table (LDD)
- 6 Langage de manipulation des données (LMD)
- 7 Interrogation de la base

- Types numériques
- Types pour les chaînes de caractères
- Types temporels (dates, heures, minutes,...)
- Types fichiers
- SQL ne permet pas à l'utilisateur de créer ses propres types

- Nombres entiers :
 - SMALLINT sur 2 octets
 - INTEGER sur 4 octets
- À virgule flottante :
 - REAL
 - DOUBLE PRECISION (ou FLOAT)
- Constantes : 253.8, -10, 1.3E-5

- Nombres décimaux à nombre fixe de décimales :
 - DECIMAL(nbChiffres, nbDécimales)
 - NUMERIC(nbChiffres, nbDécimales)
 - Les deux sont identiques en MySQL
- NUMERIC(8, 2) ou DECIMAL(8, 2) : 6 chiffres avant la virgule et 2 après
- Constantes : 253.8, -10

- CHAR(longueur) chaînes de caractères avec un nombre fixe de caractères
- VARCHAR(longueurMaximum) chaînes de caractères avec un nombre variable de caractères (mais un nombre maximum de caractères)
- CHAR(5) : chaîne de 5 caractères
- VARCHAR(20) : chaîne de 20 caractères au plus
- Constante : 'Comptabilité', 'Aujourd''hui'

- DATE pour les dates
- TIME pour les heures, minutes et secondes
- TIMESTAMP pour un moment précis : date et heures, minutes et secondes, avec une précision jusqu'à la microseconde (un millionième de seconde)

- BIT permet d'enregistrer un bit
- Exemples : BIT(1), BIT(4)

- Une valeur de type BLOB est un objet binaire de grande taille
- TINYBLOB, BLOB, MEDIUMBLOB, et LONGBLOB ne diffèrent que par la taille maximale de données qu'ils peuvent stocker
- TINYTEXT, TEXT, MEDIUMTEXT, et LONGTEXT correspondent aux types BLOB
- différences au niveau des tris et comparaisons : une valeur TEXT est une valeur BLOB insensible à la casse.

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 Création d'une table (LDD)
- 6 Langage de manipulation des données (LMD)
- 7 Interrogation de la base

Interrogations simples

Définition (Select simple)

```
SELECT expression1, expression2, ...  
FROM table  
WHERE prédicat
```

Exemple

```
select nomE, poste from emp;  
select * from dept;  
select nomE, sal + comm from emp;  
select matr, nomE, sal * 1.15 from emp where sal + comm >= 12500;
```

Expressions

Ces expressions se trouvent à la suite du select ou du where ; elles peuvent comporter des

- noms de colonnes
- constantes
- opérateurs arithmétiques
- concaténations de chaînes de caractères (||)
- calculs sur les dates (+ et -)
- fonctions

Valeur NULL

- Valeur attribuée aux attributs qui n'ont pas reçu de valeur
- Les expressions qui contiennent la valeur NULL ont la valeur NULL :
comm + sal est NULL si comm est NULL
- Sauf la fonction COALESCE

COALESCE

- COALESCE(expr1, expr2, ...)
retourne la valeur de la 1ère expression non NULL parmi les expressions expr1, expr2, ...
- COALESCE(comm, 0)
- COALESCE(comm, salaire, 0)
- COALESCE(poste, 'simple soldat')

NVL

- **NVL(expression, valeur)**
renvoie la valeur de l'expression si elle n'est pas NULL, et
valeur sinon
- Exemple : NVL(comm, 0)

NULLIF

- Permet de récupérer des données d'une BD qui n'utilisait pas la valeur NULL
- Exemple : select nomE, NULLIF(comm, -1) from emp

Signification de NULL

- NULL signifie qu'une donnée est manquante et qu'on ne connaît donc pas sa valeur Cependant, dans la pratique il peut signifier (entre autres) :
 - **pas applicable** (seuls les commerciaux touchent une commission) ; à éviter, mais possible pour simplifier le schéma de la base
 - **valeur 0** : on peut considérer que la valeur NULL correspond à la valeur 0 ; à éviter !

Logique à 3 valeurs

- La valeur NULL implique une logique à 3 valeurs :
 - vrai
 - faux
 - on ne sait pas si c'est vrai ou faux
- Par exemple, la condition « salaire > 1000 » pour un employé peut être
 - vrai, si le salaire est 1200
 - faux, si le salaire est 700
 - on ne sait pas, si le salaire est NULL

Implication de cette logique

- Soit une liste L qui contient NULL : (1, 8, NULL, 78, 7)
- Est-ce que 1 appartient à L ?
- Évidemment oui
- Est-ce que 10 appartient à L ?
- On ne sait pas !
- Est-ce que 10 n'appartient pas à L ?
- On ne sait pas !

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 **Création d'une table (LDD)**
 - Contraintes d'intégrité
 - Dictionnaire des données
- 6 Langage de manipulation des données (LMD)
- 7 Interrogation de la base

Création d'une table

Définition

Création d'une table CREATE TABLE table (
colonne1 type1,
colonne2 type2,
...
...)

Exemple

```
create table article (  
ref char(5) not null,  
nom varchar(20),  
prix numeric(9,2),  
dateMAJ date);
```

Option **not null** si la colonne doit obligatoirement être renseignée.

Valeur par défaut

On peut donner une valeur par défaut pour une colonne :

Exemple

```
create table dept (
    numDept integer not null,
    nomDept varchar(20),
    ville varchar(30) default 'Nice');
```

On peut aussi donner une fonction comme valeur par défaut ; par exemple, **default sysdate()**

DESCRIBE

Cette commande est à la base développée pour Oracle mais est maintenant intégrée dans MySQL. Elle affiche une description des colonnes d'une table :

Exemple

```
SQL > describe article;
```

| Name | Null ? | Type |
|---------|----------|-------------|
| REF | Not null | CHAR(5) |
| NOM | | VARCHAR(20) |
| PRIX | | NUMBER(9,2) |
| DATEMAJ | | DATE |

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 **Création d'une table (LDD)**
 - Contraintes d'intégrité
 - Dictionnaire des données
- 6 Langage de manipulation des données (LMD)
- 7 Interrogation de la base

Contraintes d'intégrité

Définition

Contraintes d'intégrité

- Une contrainte d'intégrité est une contrainte que doivent vérifier les données d'une table
- Une commande est annulée par le SGBD si son exécution viole une des contraintes

Types de contraintes d'intégrité

- PRIMARY KEY : clé primaire
- FOREIGN KEY ... REFERENCES : clé étrangère
- UNIQUE : 2 lignes ne peuvent avoir la même valeur pour les colonnes spécifiées
- CHECK : contrainte de domaine, ou autre ; porte sur une seule ligne

Types de contraintes d'intégrité

2 types de contraintes :

- contrainte de colonne (concerne une seule colonne)
- contrainte de table

Définition des contraintes

- Les contraintes sont définies dans les commandes CREATE (ou ALTER) TABLE
 - à l'intérieur des définitions de colonnes pour les contraintes de colonne
 - au même niveau que les définitions de colonnes pour les contraintes de table
- Pour les contraintes de table : **CONSTRAINT**
nomContrainte définitionContrainte

Clé primaire

- Si la clé primaire n'est formée que d'une seule colonne, le plus simple est d'ajouter une contrainte de colonne :

Exemple

```
create table emp (
    matr integer primary key,
    ...
```

- Sinon, il faut ajouter une contrainte de table :

Exemple

```
create table participation (
    matr integer,
    codeP integer,
    ....
constraint pkpar primary key(matr, codeP))
```

Une erreur à ne pas faire

- Si une table a une clé primaire formée de 2 colonnes, il ne faut pas déclarer 2 contraintes de colonne
- Il faut déclarer une seule contrainte de table portant sur les 2 colonnes :
constraint pkpar primary key(matr, codeP)
- Aucune des colonnes de la clé primaire ne peut avoir la valeur **null**

Contrainte UNIQUE

- 2 lignes de la table ne pourront avoir la même valeur (sauf NULL)
- Correspond à un identificateur (clé candidate si minimal), si on ajoute une contrainte NOT NULL

Clé étrangère

Si une seule colonne forme la clé étrangère, le plus simple est d'utiliser une contrainte de colonne :

Exemple

```
create table emp (
    ...
    dept integer references dept(dept))
```

(dept) : Optionnel si la colonne référencée est clé primaire

Clé étrangère

Peut être une contrainte de table :

Définition

```
FOREIGN KEY (colonne1, colonne2,...)  
REFERENCES table-ref [(col1, col2,...)]
```

Exemple

```
create table emp (  
....  
dept integer,  
constraint r_dept foreign key(dept) references dept(dept))
```

Clé étrangère

Les colonnes de l'autre table référencées (col1, col2,...) doivent avoir la contrainte PRIMARY KEY ou UNIQUE
constraint r_dept references dept(dept)
dept doit être clé primaire, ou unique

Option ON DELETE CASCADE (sans)

Exemple

```
create table emp (
...
dept integer references dept)
```

ou

Exemple

```
create table emp (
...
dept integer
constraint r_dept foreign key (dept) references dept)
```

Attention On ne peut supprimer un département s'il est référencé par une ligne de la table **emp**

Option ON DELETE CASCADE (avec)

Exemple

```
create table emp (
...
dept decimal(2,0),
constraint r_dept foreign key (dept) references dept on delete
cascade)
```

Attention La suppression d'un département entraîne automatiquement la suppression de toutes les lignes de la table **emp** qui référencent ce département.

Autres options pour les clés étrangères

- on delete set null
- on delete set default
- on update cascade
- on update set null
- on update set default

Exemples divers de contraintes

Exemple

```
create table emp (
    matr integer primary key,
    nomE varchar(30) unique,
    dept smallint references dept check (dept in (10, 20, 30, 40)),
    constraint nom_unique check (nomE = upper(nomE)));
```

Exemple

```
create table participation (
    matr integer references emp,
    codeP varchar(5) references projet,
    ...,
    constraint pkpart primary key(matr, codeP));
```

Modification des contraintes

Exemple

```
ALTER TABLE emp
DROP CONSTRAINT nom_unique
ADD (CONSTRAINT sal_min check(sal + coalesce(comm, 0) >
5000))
RENAME CONSTRAINT truc TO machin;
```

Attention On ne peut ajouter que des contraintes de table.

Vérification des contraintes

- En fonctionnement normal les contraintes sont vérifiées à chaque requête SQL.
- Cette vérification peut être gênante, en particulier lors de l'ajout de plusieurs lignes de données.
- Exemple : si on a cette contrainte sur la colonne SUP de la table EMP : constraint sup_ref_emp foreign key (SUP) references EMP
- La contrainte oblige à ajouter les supérieurs en premier.

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 **Création d'une table (LDD)**
 - Contraintes d'intégrité
 - Dictionnaire des données
- 6 Langage de manipulation des données (LMD)
- 7 Interrogation de la base

Dictionnaire des données

Définition

- Tables qui stockent les descriptions des objets de la base
- Tenues à jour automatiquement par le SGBD
- Peuvent être consultées au moyen du langage

INFORMATION_SCHEMA

- Ce dictionnaire est stocké dans la base
INFORMATION_SCHEMA

Exemple

```
SELECT table_name, table_type, engine FROM
information_schema.tables;
```

- Il est aussi possible d'utiliser la commande **SHOW** (ex : show tables)

Tables de INFORMATION_SCHEMA :

- SCHEMATA : informations sur les bases de données
- TABLES : informations sur les tables
- COLUMNS : informations sur les colonnes dans les tables
- USER_PRIVILEGES : informations sur les droits globaux
- SCHEMA_PRIVILEGES : informations sur les droits des schémas
- TABLE_PRIVILEGES : informations sur les droits des tables
- COLUMN_PRIVILEGES : informations sur les droits reliés aux colonnes
- TABLE_CONSTRAINTS : informations sur les tables qui ont des contraintes
- KEY_COLUMN_USAGE : descriptions des contraintes sur les colonnes
- ROUTINES : informations sur les procédures stockées
- VIEWS : informations sur les vues

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 Création d'une table (LDD)
- 6 Langage de manipulation des données (LMD)
 - INSERT/UPDATE/DELETE
 - Transactions
- 7 Interrogation de la base

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 Création d'une table (LDD)
- 6 Langage de manipulation des données (LMD)
 - INSERT/UPDATE/DELETE
 - Transactions
- 7 Interrogation de la base

Langage de manipulation des données (LMD)

Commandes de manipulation des données :

- INSERT pour ajouter des lignes
- UPDATE pour modifier des lignes
- DELETE pour supprimer des lignes

Insertion

Définition

```
INSERT INTO table [(colonne1, colonne2,...)] VALUES (valeur1,  
valeur2,...)
```

ou

Définition

```
INSERT INTO table [(colonne1, colonne2,...)] select ...
```

La liste des colonnes est optionnelle ; par défaut, toutes les colonnes sont dans l'ordre donné lors de la création de la table. Si la commande comporte une liste, les colonnes qui ne sont pas dans la liste auront la valeur par défaut

Dans les programmes, il faut toujours donner la liste des colonnes dont on donne les valeurs pour faciliter la maintenance de l'application

Insertion

Exemple

```
insert into dept values (10, 'Finance', 'Paris');
insert into dept (lieu, nomD, dept) values ('Grenoble', 'Recherche',
20);
insert into emp (matr, nomE, dept, sal)
select matr + 100, nomE, 60, sal * 0.15
from emp
where dept = 10;
```

Modification

Définition

```
UPDATE table SET colonne1 = expr1, colonne2 = expr2, ... [  
WHERE prédicat ]
```

Définition

```
UPDATE table [ synonyme ] SET (colonne1, colonne2, ...) =  
(select ...) [ WHERE prédicat ]
```

Le **select** ne doit renvoyer qu'une seule ligne

Exemples

Exemple

```
update emp set dept = 10 where nomE = 'Martin';  
update emp set sal = sal * 1.1 where poste = 'Commercial';
```

Exemple

```
update emp set sal = (select avg(sal) * 1.1 from emp where poste  
= 'Secrétaire') where nomE = 'Clément';  
update emp E set (sal, comm) = (select avg(sal), avg(comm) from  
emp where dept = E.dept) where poste = 'Secrétaire';
```

Suppressions

Définition

`DELETE FROM table [WHERE prédicat]`

Attention S'il n'y a pas de clause WHERE, toutes les lignes sont supprimées.

Exemple

```
delete from emp where dept = 10;
```

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 Création d'une table (LDD)
- 6 Langage de manipulation des données (LMD)
 - INSERT/UPDATE/DELETE
 - Transactions
- 7 Interrogation de la base

Transaction

- Une requête forme un tout indivisible
- Si une erreur survient pendant l'exécution d'une requête SQL, toutes les modifications déjà effectuées par la requête sont annulées automatiquement.
- On peut généraliser ce comportement à un ensemble de requêtes SQL en utilisant les transactions
- Ensemble de modifications de la base qui forment un tout indivisible : à la fin de la transaction, toutes les modifications effectuées pendant la transaction sont sauvegardées ou annulées

Début d'une transaction

- Dans la norme SQL2 toute modification appartient à une transaction
- Par défaut, MySQL est lancé en mode autocommit (transaction effectuée après chaque opération)
- Une transaction démarre automatiquement après la fin d'une transaction (pas de commande pour démarrer une transaction)
- Une transaction démarre par `START TRANSACTION`
- La structure des transactions est « plate » : les transactions ne peuvent se chevaucher

Terminer une transaction

- Pour terminer une transaction on peut
 - valider la transaction (COMMIT) : toutes les modifications deviennent effectives
 - annuler la transaction (ROLLBACK) : toutes les modifications sont annulées
- Les ordres DDL (create table par exemple) provoquent un COMMIT automatique

Propriétés des transactions - ACID

- **Atomicité** : un tout indivisible
- **Cohérence** : une transaction doit laisser la base dans un état cohérent ; elle ne doit pas mettre les données dans un état « anormal »
- **Isolation** : une transaction est isolée des autres transactions (dans une certaine mesure...)
- **Durabilité** : le SGBD doit garantir que les modifications d'une transaction validée seront conservées, même en cas de panne

Propriétés des transactions - ACID

- **A**ID est du ressort du système transactionnel du SGBD
- **C** est du ressort de l'utilisateur mais il est aidé
 - par **I**, car il n'a pas à considérer les interactions avec les autres transactions
 - par la vérification automatique des contraintes d'intégrité par le SGBD
- **I** est assuré par le système de contrôle de la concurrence du SGBD et **AD** sont supportés par les procédures de reprise après panne du SGBD

Transaction

Exemple d'annulation d'une transaction

Exemple

```
start transaction;  
insert into dept values (10, 'Finance', 'Paris');  
delete from emp;  
rollback;
```

Isolation des transactions

En fonctionnement standard les modifications effectuées par une transaction T ne sont connues par les autres transactions qu'après validation de T. En fait, il existe plusieurs niveaux d'isolation (voir cours sur la concurrence)

Transactions longues

- Exemple : organisation par une agence de voyage d'un voyage Nice – Wuhan (Chine)
- Nécessite la réservation de plusieurs billets d'avion : Nice – Paris ; Paris – Beijing ; Beijing – Wuhan
- On commence par réserver les 2 premiers mais si on ne peut trouver de Beijing – Wuhan, il faut tout annuler
- On met donc toutes ces réservations dans une transaction ; ça peut être long si l'agence discute avec le client pendant la transaction

Problèmes avec les transactions longues

- Manque de souplesse : si on ne trouve pas de voyage Beijing – Wuhan, on annule tout
- On aurait pu garder le Nice – Paris et essayer de passer par Shanghai pour aller à Wuhan, en annulant seulement le Paris – Beijing
- Autre problème : le contrôle de la concurrence effectue des blocages sur les tables et les lignes qui ne sont relâchés qu'à la fin de la transaction
- Un problème de communication peut provoquer l'annulation des premières réservations alors qu'on pourrait simplement réessayer le lendemain

Transactions emboîtées

- Extension de la notion de transaction « plate »
- Évite les annulations complètes de transactions
- Apporte plus de souplesse dans les transactions longues et multi-sites
- Permet de limiter la durée des blocages des ressources système

Définition des transactions emboîtées

- Une transaction globale (mère) peut contenir des soustractions filles qui, elles-mêmes, peuvent avoir des filles
- L'annulation d'une transaction n'annule pas nécessairement la transaction mère ; celle-ci peut
 - décider d'un traitement substitutif
 - reprendre la transaction annulée
 - s'annuler
 - ou même ignorer l'annulation (traitement pas indispensable)
- L'annulation d'une transaction provoque l'annulation automatique de toutes ses transactions filles

Points de reprise

- Sans passer au modèle des transactions emboîtées, on peut assouplir le modèle des transactions plates
- Désigner des points de reprise dans une transaction : **savepoint *nomPoint***
- Possible d'annuler toutes les modifications effectuées depuis un point de reprise : **rollback to *nomPoint***
- Évite d'annuler toute la transaction et permet d'essayer de pallier le problème si c'est possible

Points de reprise

Exemple

```
start transaction;  
insert into ...;  
savepoint p1;  
delete from ...;  
update ...;  
savepoint p2;  
insert into ...; – Problème !  
rollback to p2;  
insert into ...; – on essaie autre chose  
commit;
```

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 Création d'une table (LDD)
- 6 Langage de manipulation des données (LMD)
- 7 Interrogation de la base
 - Fonctions de groupe

Syntaxe générale

Définition

SELECT ... FROM ...

WHERE ...

GROUP BY ...

HAVING ...

ORDER BY ...

- L'ordre des clauses est imposé
- SELECT et FROM sont obligatoires

Clauses SELECT

Définition

```
select [distinct] expression1 [ [AS] nom1], expression2 [ [AS]  
nom2], . . .
```

nom1 :

- en-tête de la colonne (entre guillemets si mot-clé ou si contient plusieurs mots)
- alias pour désigner la colonne dans une autre partie du select

select * : Toutes les colonnes

Exemple

```
select distinct poste from emp;  
select nomE, sal + coalesce(comm, 0) as "Salaire Total" from emp;
```

select dans une expression

Exemple

```
select nomE, sal / (select sum(sal) from emp) * 100 from emp
```

Le select de l'expression peut être synchronisé avec le select principal :

Exemple

```
select nomE, (select count(*) from emp where sal > e1.sal) + 1 as  
rang from emp e1
```

e1 est un synonyme de emp pour lever l'ambiguïté dans le select interne

select dans une expression

On peut utiliser l'alias pour trier par ordre décroissant des salaires :

Exemple

```
select nomE, (select count(*) from emp where sal > e1.sal) + 1 as  
rang from emp e1 order by rang
```

Clause FROM

Définition

FROM table1 [synonyme1], table2 [synonyme2], ...

- Produit cartésien des tables s'il y en a plusieurs
- Possible de se restreindre à un sous-ensemble du produit cartésien (voir jointure)

Clause FROM

Exemple

| | DEPT | NOMD |
|--|----------------|--------------------------------|
| select B.dept, A.nomD from dept A, dept B | 10 20 30 | VENTES RECHERCHE FINANCE |

| | DEPT | NOMD |
|--|-------------|-------------|
| | 30 | VENTES |
| | 20 | VENTES |
| | 10 | VENTES |
| | 30 | RECHERCHE |
| | 20 | RECHERCHE |
| | 10 | RECHERCHE |
| | 30 | FINANCE |
| | 20 | FINANCE |
| | 10 | FINANCE |

Clause FROM

Certains SGBDs (et la norme SQL-2) permettent l'utilisation d'un SELECT à la place du nom d'une table :

Exemple

```
select nomE, sal,  
sal / total * 100 Pourcentage  
from emp,  
(select sum(sal) as total from emp);
```

Clause WHERE

La clause WHERE comporte de nombreuses possibilités :

- opérateurs de comparaison
- opérateurs logiques
- jointures
- sous-interrogations

Opérateurs de comparaison

- $=$, \neq , $<$, $>$, \leq , \geq , BETWEEN, LIKE, NOT LIKE, IN, NOT IN, IS NULL, IS NOT NULL
- LIKE permet d'utiliser des jokers :
 - % pour une chaîne de caractères de longueur quelconque
 - _ pour un seul caractère
- Attention, expression = NULL n'est jamais vrai, il faut utiliser expression IS NULL

Opérateurs de comparaison

Exemple

```
select * from emp where poste = 'Secrétaire';
select * from emp where sal between 10000 and 15000;
select * from emp where dept in (10, 30);
select * from emp where comm is not null;
select * from emp where nomE like '%A%';
```

Opérateurs logiques

- AND, OR, NOT

Exemple

```
select nomE from emp  
where dept = 30  
and (sal > 10000 or comm is null);
```

Exemple

```
select * from emp  
where not (poste = 'Directeur'  
or poste = 'Secrétaire');
```

Jointures

Traduction de l'équi-jointure « emp \bowtie dept » :

Exemple

```
select nomE, nomD  
from emp, dept  
where emp.dept = dept.dept
```

Autre syntaxe :

Exemple

```
select nomE, nomD  
from emp JOIN dept  
ON emp.dept = dept.dept
```

Jointure de plus de 2 tables

Exemple

```
select nomE, nomP  
from emp, participation, projet  
where emp.matr = participation.matr  
and participation.codeP = projet.codeP
```

Autre syntaxe :

Exemple

```
select nome, nomp  
from emp  
join participation  
on emp.matr = participation.matr  
join projet  
on participation.codep = projet.codep
```

Jointure naturelle

- La jointure s'effectue sur toutes les colonnes qui ont le même nom dans les 2 tables ; ces colonnes ne sont pas répétées dans la jointure
- Les colonnes qui participent à la jointure ne doivent être préfixées par un nom de table

Exemple

```
select nomE, nomD, dept from emp NATURAL JOIN dept;  
select nome, nomp from emp NATURAL JOIN participation  
NATURAL JOIN projet;
```

Jointure d'une table avec elle-même

Alias indispensable pour le nom de la table afin de lever l'ambiguïté sur les colonnes :

Exemple

```
select emp.nomE "Employé",
       supe.nomE "Supérieur"
  from emp join emp supe
    on emp.sup = supe.matr
```

Jointures « non équi »

Les jointures « non équi » peuvent être traduites comme les équi-jointures, en utilisant d'autres opérateurs de comparaison

Exemple

```
select emp1.nomE, emp2.nomE  
from emp emp1  
join emp emp2  
on emp1.sal < emp2.sal
```

Jointure externe

- Dans une jointure n'apparaissent que les lignes qui ont une ligne correspondante dans l'autre table
- Dans l'exemple suivant, un département qui n'a pas d'employé n'apparaîtra pas :

Exemple

```
select nomE, nomD from emp join dept on emp.dept = dept.dept
```

- Si on veut qu'il apparaisse, on doit utiliser une jointure externe

Syntaxe de la jointure externe

Exemple

```
select nomE, nomD  
from emp RIGHT OUTER JOIN dept  
ON emp.dept = dept.dept
```

- RIGHT indique que l'on veut afficher toutes les lignes de la table de droite (dept)
- Ça revient à ajouter une « ligne fictive » dans l'autre table emp
- Cette ligne fictive aura toutes ses colonnes à null, sauf la colonne de jointure
- Il existe de même LEFT OUTER JOIN et FULL OUTER JOIN

Sous-interrogations

- Une clause WHERE peut comporter un ordre SELECT emboîté :

Exemple

```
select nomE from emp  
where poste = (select poste from emp where nomE = 'Martin');
```

- Cette sous-interrogation doit ramener une seule ligne et une seule colonne
- Remplace le select interne par NULL s'il ne renvoie aucune ligne (ou erreur, suivant les SGBD)

Sous-interrogations

Des variantes de sous-interrogations ramènent plusieurs colonnes ou plusieurs lignes

Sous-interrogation ramenant 1 ligne, 1 colonne

Définition

WHERE expression op (SELECT ...)

où op est un des opérateurs de comparaison =, !=, <, >, <=, >=

Exemple

```
select nomE from emp  
where sal >=  
(select sal from emp  
where nomE = 'Mercier')
```

Sous-interrogation ramenant plusieurs lignes

Définition

WHERE expression op ANY (SELECT ...)

WHERE expression op ALL (SELECT ...)

WHERE expression IN (SELECT ...)

WHERE expression NOT IN (SELECT ...)

où op est un des opérateurs de comparaison =, !=, <, >, <=, >=

- **ANY** : vrai si la comparaison est vraie pour **au moins** une des valeurs ramenées par le SELECT
- **ALL** : vrai si la comparaison est vraie pour **toutes** les valeurs ramenées par le SELECT

Remarque

- `=ANY` est équivalent à `IN`
- `!=ALL` est équivalent à `NOT IN`

Exemple

```
select nomE, sal from emp  
where sal > all (select sal from emp where dept = 30);  
select nomE, sal from emp  
where sal > all (select sal  
from emp  
where dept = 38888)
```

Réflexion sur ALL

- Quand « $x > \text{all } (x_1, x_2, \dots, x_n)$ » est faux ?
- Quand \exists un x_i tel que $x_i \geq x$
- Si \nexists un x_i tel que $x_i \geq x$, l'expression est vraie
- Donc si la liste (x_1, x_2, \dots, x_n) est vide, l'expression est toujours vraie

Retour sur NULL

- La condition **expression not in (expr1, expr2, null)** n'est jamais vérifiée. Pourquoi ?
- Est-ce que la condition **expression in (expr1, expr2, null)** peut être vérifiée ?
- Rappel : la logique de SQL n'utilise pas seulement vrai et faux mais aussi « je ne sais pas », représenté par NULL
- Utile à savoir pour les sous interrogations qui renvoient NULL pour une des lignes

Sous-interrogations ; optimisation

- Soit un select qui comporte une sous-interrogation :

Exemple

```
select nom from emp  
where dept in  
(select dept from dept  
where lieu = 'NICE')
```

- Pour chaque employé, le select peut lancer la sousinterrogation pour savoir si l'employé est dans un département qui se trouve à Nice
- En fait, le moteur de recherche du SGBD va optimiser en lançant d'abord la sous-interrogation et en conservant en mémoire les départements de Nice

Sous-interrogations synchronisées

- Cette optimisation n'est pas possible quand la sous-interrogation utilise une des valeurs ramenées par l'interrogation principale
- On dit que la sous-interrogation est synchronisée avec l'interrogation principale
- Notation pointée utilisée pour se référer dans la sous-interrogation à une colonne de l'interrogation principale : sous-interrogation

Exemple

```
select nomE from emp E  
where dept != (select dept from emp  
where matr = E.sup);
```

Sous-interrogation ramenant 1 ligne de plusieurs colonnes

Définition

WHERE (expr1, expr2,...) op (SELECT ...)

où op est = ou != (mais pas <, >, <=, >=) et où le select renvoie 1 seule ligne

Exemple

```
select nomE from emp  
where (poste, sal) =  
(select poste, sal from emp  
where nomE = 'Mercier');
```

Relation d'ordre

- Il n'y a pas de relation d'ordre totale « naturelle » sur les tuples
- $5 > 3, 12 > 7$
- On ne peut comparer $(5, 7)$ et $(3, 12)$

Sous-interrogation ramenant plusieurs lignes de plusieurs colonnes

Définition

WHERE (expr1, expr2,...) op ANY (SELECT ...)

WHERE (expr1, expr2,...) op ALL (SELECT ...)

WHERE (expr1, expr2,...) IN (SELECT ...)

WHERE (expr1, expr2,...) NOT IN (SELECT ...)

où op est = ou != (mais pas <, >, <=, >=)

Exemple

```
select nomE from emp  
where (poste, sal) in  
(select poste, sal from emp  
where dept = 10);
```

EXISTS

- La clause « EXISTS(select ...) » est vraie si le select renvoie au moins une ligne
- La sous-interrogation est le plus souvent synchronisée :

Exemple

```
select nomE from emp E  
where exists  
(select null from emp  
where sup = E.matr);
```

EXERCICE : Division avec not exist

Exemple (Tables)

R (A, B)
S (A)

Exemple (Requête SQL "R ÷ S")

```
select distinct R1.B from R R1
where not exists (
select S.A from S
where not exists (
select * from R
where R.B = R1.B and R.A = S.A ) );
```

Exemple (Afficher la liste des numéros des départements qui ont tous les postes)

La solution est $R \div_{\text{poste}} S$ où

$R = \text{EMP} [\text{dept}, \text{poste}]$

$S = \text{EMP} [\text{poste}]$

select distinct dept from emp E

where not exists (

select poste from emp E2

where not exists (

select dept, poste from emp

where dept = E.dept

and poste = E2.poste));

Chapitre III

SQL

- 1 Présentation de SQL
- 2 MySQL
- 3 Types de données
- 4 Interrogations simples
- 5 Création d'une table (LDD)
- 6 Langage de manipulation des données (LMD)
- 7 Interrogation de la base
 - Fonctions de groupe

Fonctions de groupe

Les fonctions de groupe peuvent apparaître dans une expression du select ou du having :

- AVG moyenne
- SUM somme
- MIN plus petite valeur
- MAX plus grande valeur
- VARIANCE variance
- STDDEV écart type
- COUNT(*) nombre de lignes
- COUNT(col) nombre de valeurs non NULL dans la colonne
- COUNT(DISTINCT col) nombre de valeurs distinctes

Fonctions de groupe

Exemple

```
select count(*) from emp;  
select count(comm) from emp where dept = 10;  
select sum(sal) from emp where dept = 10;  
select max(sal) from emp where poste = 'INGENIEUR';
```

Niveaux de regroupement

- A un niveau de profondeur (relativement aux sous interrogations) d'un SELECT, les fonctions de groupe et les colonnes doivent être toutes du même niveau de regroupement
- Par exemple, si on veut le nom et le salaire des employés qui gagnent le plus dans l'entreprise, la requête suivante provoquera une erreur :

Exemple

```
select nome, sal from emp  
where sal = max(sal)
```

- Solution :

Exemple

```
select nome, sal from emp  
where sal = (select max(sal) from emp)
```

Clause GROUP BY

Permet de regrouper des lignes qui ont les mêmes valeurs pour des expressions :

Définition

GROUP BY expression1, expression2,...

Il n'est affiché qu'une seule ligne par regroupement de lignes

Exemple

```
select dept, count(*) from emp group by dept;
```

Clause GROUP BY

Exemple

```
select dept, poste, count(*) from emp group by dept, poste;  
select dept, count(comm) from emp group by dept;
```

Schéma à retenir pour obtenir des optima sur des regroupements :

Exemple

```
select nome, dept, sal from emp  
where (dept, sal) in (select dept, max(sal) from emp group by  
dept);
```

Sans doute moins performant :

Exemple

```
select nome, dept, sal from emp E  
where sal = (select max(sal) from emp  
where dept = E.dept);
```

Clause GROUP BY

Exemple

```
select nomD, avg(sal) from emp natural join dept group by nomD;
```

Exemple

```
select dept, count(*) from emp where poste = 'SECRETAIRE'  
group by dept;
```

Exemple

```
select count(*) "Nbre employés", count(comm) "Nbre  
commissions", count(comm) / count(*) "Ratio" from emp;
```

Restrictions pour les expressions

Dans la liste des expressions du select ne peuvent figurer que des caractéristiques liées aux groupes :

- des fonctions de groupes
- des expressions figurant dans le GROUP BY

Exemple (Ceci est interdit)

```
select dept, nomE, sal  
from emp  
group by dept;
```

Restrictions pour les expressions

- Ceci ne fonctionne pas :

Exemple

```
select nomd, sum(sal)
from emp natural join dept
group by dept.dept
```

- Il aurait fallu écrire :

Exemple

```
select nomd, sum(sal)
from emp natural join dept
group by nomd
```

Clause HAVING

- Cette clause sert à sélectionner des groupes : HAVING prédicat
- Le prédicat ne peut porter que sur des caractéristiques de groupe

Clause HAVING

Exemple

```
select dept, count(*) from emp  
where poste = 'Secrétaire'  
group by dept  
having count(*) > 1;
```

Exemple

```
select nomd "Département",  
count(*) "Nombre de secrétaires"  
from emp natural join dept  
where poste = 'Secrétaire'  
group by nomd  
having count(*) = (select max(count(*)) from emp  
where poste = 'Secrétaire'  
group by dept);
```

Clause HAVING

Exemple

```
select dept, count(*) from emp group by dept having count(*) =  
max(count(*));
```

Interdit ! car pas le même niveau de regroupement que le reste du select

Exemple

```
select dept, count(*) from emp  
group by dept  
having count(*) = (select max(count(*)) from emp  
group by dept);
```

Clause ORDER BY

La clause ORDER BY précise l'ordre des lignes d'un SELECT :

Définition

ORDER BY expr1 [DESC], expr2 [DESC], ...

On peut aussi donner le numéro de la colonne qui servira de clé de tri (**Déprécié**)

Exemple

```
select dept, nomD from dept order by nomD;  
select dept, nomD from dept order by 2; -- Déprécié
```

Clause ORDER BY

Exemple

```
select nome, poste from emp order by dept, sal desc;
```

```
select dept, sum(sal) "Total salaires" from emp group by dept  
order by "Total salaires";
```

```
select dept, sum(sal) "Total salaires" from emp group by dept  
order by sum(sal);
```

Fonctions

- Fonctions arithmétiques : `abs(n)`, `mod(m, n)`, `power(n, e)`, `round(n, p)`, `trunc(n, p)`, `sign(n)`, `sqrt(n)`, `greatest(n1, n2, ...)`, `least(n1, n2, ...)`
- Conversions nombre - chaîne de caractères : `to_char(n, format)`, `number(chaîne)`
- Fonctions date : `datediff(date1, date2)`, `timediff(time1, time2)`, `curdate()`, `now()`
- Conversions date - chaîne de caractères : `date_format(date, format)`, `time_format(time, format)`, `str_to_date(chaîne, format)`

Fonctions

- Fonctions sur les chaînes de caractères : `length(chaîne)`, `substr(chaîne, position, longueur)`, `locate(sous-chaîne, chaîne)`, `upper(chaîne)`, `lower(chaîne)`, `lpad(chaîne, long, car)`, `rpad(chaîne, long, car)`, `ltrim(chaîne, car)`, `rtrim(chaîne, car)`, `replace(chaîne, ancienne, nouvelle)`

Fonctions

Exemple

```
select nomE, date_format(datemb, '%d/%m/%Y')
from emp
where datediff(curdate(), datemb) > 3;
```

Conditionnelle

2 variantes :

Définition

CASE

WHEN condition1 THEN resultat1

WHEN condition2 THEN resultat2

ELSE resultat3

END

Définition

CASE expression

WHEN valeur1 THEN resultat1

WHEN valeur2 THEN resultat2

ELSE resultat3

END

Conditionnelle

Exemple

```
SELECT nome, poste FROM emp  
order by  
CASE  
WHEN poste = 'Président' THEN 1  
WHEN poste = 'Directeur' THEN 2  
ELSE 3  
END;
```

Exemple

```
SELECT nome, poste FROM emp  
order by  
CASE poste  
WHEN 'Président' THEN 1  
WHEN 'Directeur' THEN 2  
ELSE 3 END;
```

Opérateurs ensemblistes

- On peut effectuer des opérations sur plusieurs select considérés comme des ensembles de lignes :

Définition

select ... **UNION** select ...

select ... **INTERSECT** select ...

select ... **MINUS** select ...

- Ensembles au sens mathématique : si 2 lignes des 2 select d'une union ont les mêmes valeurs, l'union n'aura qu'une seule ligne avec ces valeurs

Exemple d'opérateur ensembliste

Exemple (PostgreSQL)

```
select dept from dept  
minus  
select dept from emp;
```

Exemple (MySQL)

```
select dept from dept  
not in (select dept from emp);
```

Exemple

```
select nomE, 'salaire' as TYPE, sal as MONTANT from emp union  
select nomE, 'commission' as TYPE, comm as MONTANT from  
emp where comm is not null;
```

| nomE | TYPE | MONTANT |
|-------------|-------------|----------------|
| Toto | salaire | 1200 |
| Bibi | salaire | 5000 |
| Toto | commission | 240 |

Opérateurs ensembliste

- UNION ALL : Pour conserver les doublons avec UNION
- Opérateurs ensembliste et order by : Seul le dernier select peut recevoir une clause order by

Limiter le nombre de lignes

Problème de portabilité si on ne veut afficher qu'une partie des lignes récupérées par un select

Exemple

```
SELECT matr, nomE  
FROM emp  
LIMIT 10
```