COURS SQL

Présenté par : Ali SADEG

Versions

Version	Date	Auteur	Modifications
T1.0	03/07/2017	Ali SADEG	Version Initiale

Sommaire

1 Le langage SQL

- SQL (Structured Query Language / langage de requête structurée)

C'est un langage informatique normalisé servant à exploiter des bases de données relationnelles.

Il est créé en 1974, normalisé depuis 1986, le langage est reconnu par la grande majorité des SGBDR

Le langage SQL contient quatre parties, à savoir

- La partie langage de manipulation des données (LMD) : rechercher, ajouter, modifier ou de supprimer des données dans les bases de données relationnelles.

  – Interrogation : select

– Manipulation : insert, update, delete

- La partie langage de définition des données (LDD) :

- créer et modifier l'organisation des données dans la base de données.

– Create, alter, drop

- La partie langage de contrôle de transaction (LCT) : commencer/ terminer des transactions - commit, rollback

- La partie langage de contrôle des données (LCD): autoriser/ interdire l'accès à certaines données à certaines personnes.

- Grant, revoke

2 Le langage de définition des données(LDD)

2.1 Contraintes d'intégrité

- Les contraintes d'intégrité sont des règles qui doivent être vérifiées en permanence par le SGBD, quel que soit l'opération effectuée sur la base.

  Parmi ces contraintes d'intégrité :

  - la définition des clés primaires et des clés étrangères.

  - l'indication des valeurs possibles pour les attributs.

  C'est lors de la création d'une table, qu'il est possible des spécifier une contrainte associée à une colonne. Les contraintes sont de natures différentes :

  • La colonne ne peut pas contenir de valeur nulle (NOT NULL)

  • La valeur d'un attribut doit être unique (une même colonne ne doit pas contenir deux valeurs identiques dans des tuples différents)

  • La condition générale (clause CHECK) qui permet de spécifier un intervalle ou une zone de validité de la valeur d'un attribut.

- La clé primaire
- La clé étrangère

2.1.1 Principales contraintes

2.1.1.1 NOT NULL : permet donc d'indiquer qu'un attribut ne peut pas prendre la valeur NULL, sinon (dans le cas d'une tentative d'insertion de valeur nulle), le SGBD renvoie un message d'erreur.

Exemple : Création de la table «matiere» avec une contrainte sur le libellé, qui doit toujours être renseigné !

```
CREATE TABLE matiere
        ( ….
                libelle-m VARCHAR(20) NOT NULL
        ) ;
```

2.1.1.2 UNIQUE : permet de ne pas stocker deux données identiques pour des tuples différents. Par exemple, ce type de clause peut être appliqué pour un numéro INSEE.

Exemple : Numéro de sécurité sociale unique.

```
CREATE TABLE etudiant
        (
                Nom VARCHAR(20) NOT NULL,
                Prenom VARCHAR(20),
                Num_INSEE CHAR(15) UNIQUE
        ) ;
```

2.1.1.3 CHECK : permet, lors de la création d'une table, de spécifier une contrainte associée à une colonne, pour valider les informations insérées dans cette colonne.

Exemple : Un champ sexe-et ne peut contenir que 'M' ou 'F'.

```
CREATE TABLE etudiant
(       Nom_et VARCHAR(20) NOT NULL,
        Prenom_et VARCHAR(20),
        CodeSexe_et CHAR
        CONSTRAINT CK_Sexe CHECK
        (CodeSexe_et IN ('M', 'F') )
) ;
```

Rmq : La colonne CodeSexe_et ne peut prendre que l'une des 2 valeurs 'M' ou 'F'. Si un utilisateur tente d'insérer une autre valeur, le système retourne un message d'erreur.

CONSTRAINT CK_Sexe permet de donner un nom à la contrainte, dans le but de pouvoir utiliser ce nom pour une éventuelle suppression de cette contrainte

2.1.1.4 Clé primaire : Une clé primaire est un ensemble de colonnes (ou une seule colonne) dont les valeurs (la valeur) identifient de manière unique chaque tuple (ligne) de la table.

Syntaxe :

```
CREATE TABLE nom_table
(
        Nom_colonne1 TYPE_DE_DONNEES [NOT NULL], ...
        Nom_colonneN TYPE_DE_DONNEES [NOT NULL],
        [CONSTRAINT nom_contrainte_clé_primaire]
        PRIMARY KEY (nom_colonneA, ..., nom_colonneX)
) ;
```

Exemple :

```
CREATE TABLE etudiant
(
Code_et INTEGER,
Nom_et VARCHAR(20) NOT NULL,
Prenom_et VARCHAR(20),
CodeSexe_et CHAR CONSTRAINT CK_Sexe
CHECK (CodeSexe_et IN ('M', 'F')),
CONSTRAINT PK_Code_et PRIMARY KEY
(Code_et)
) ;
```

Remarque : Si la clé primaire est composée d'une seule colonne, il est possible d'indiquer la clause PRIMARY KEY au niveau de sa définition.

Exemple :

```
CREATE TABLE etudiant
( Code_et INTEGER PRIMARY KEY,
Nom_et VARCHAR(20) NOT NULL,
Prenom_et VARCHAR(20),
CodeSexe_et CHAR CONSTRAINT CK_Sexe
CHECK (CodeSexe_et IN ('M', 'F'))
) ;
```

2.1.1.5 La clé étrangère : Pour créer une clé étrangère, on utilise la clause REFERENCES suivie de la table référencée et du champ concerné.

Syntaxe :

```
CREATE TABLE nom_table
(Nom_col_1 TYPE_DE_DONNEES [NOT NULL], ...
Nom_col_n TYPE_DE_DONNEES [NOT NULL],
[CONSTRAINT nom_contrainte_clé_étrangère]
FOREIGN KEY (nom_colonneF1, ...,
nom_colonneFX)
REFERENCES table_referencée (nom_colonneP1, ...,
nom_colonnePX),
[CONSTRAINT nom_contrainte_clé_primaire]
PRIMARY KEY (nom_colonneA, ..., nom_colonneX)
) ;
```

Exemple : création d'une table représentant la notion d'atelier, puis création d'une table employe qui fait référence à la table atelier dans lequel travaille l'employé.
Commençons par créer la table atelier :

```
CREATE TABLE atelier
(at_code CHAR,
at_intitule VARCHAR(12),
```

CONSTRAINT PK_at_Code
PRIMARY KEY (at_code)) ;

Il est maintenant possible de créer la table employe quifait référence à la table atelier.

Note importante :L'ordre de création des tables est important : créer d'abord la table référencée (ici atelier, qui a été déjà créée), puis on crée employe :

CREATE TABLE employe
(Code_e INTEGER,
Nom_e VARCHAR(20) NOT NULL,
Prenom_e VARCHAR(20),
at_code CHAR,
CONSTRAINT FK_at-code FOREIGN KEY (at_code)
REFERENCES atelier (at_code),
[ou bien : at_code char references atelier (at_code),]
CONSTRAINT PK_Code_e PRIMARY KEY (Code_e)
) ;

2.1.1.6 Valeur par défaut : **La clause DEFAULT permet, lors de la définition d'une colonne, de déclarer une valeur par défaut.**

Lors de l'insertion d'un tuple, si aucune information n'est spécifiée pour cette colonne, c'est la valeur par défaut qui sera enregistrée.

Exemple
CREATE TABLE employe (
Code_e INTEGER,
Nom_e VARCHAR(20) NOT NULL,
Prenom_e VARCHAR(20),
atelier_e CHAR DEFAULT 'A',
CONSTRAINT FK_atelier_e FOREIGN KEY (atelier_e)
REFERENCES atelier (at_code)
CONSTRAINT PK_Code_e PRIMARY KEY (Code_e)) ;

2.1.2 Contrainte d'intégrité : Résumé

- PRIMARY KEY :définit une clé primaire (la valeur est différente de NULL et unique dans la table)
⇒ toutes les lignes sont différentes
- FOREIGN KEY :définit une clé étrangère. Attribut (ou groupe) qui fait référence à une clé primaire dans une autre table
⇒ «contrainte d'intégrité référentielle».
- **NOT NULL :**la valeur de l'attribut ne doit pas être à NULL (elle doit être renseignée).
- UNIQUE :chaque tuple de la table doit avoir une valeur différente de celle des autres ou NULL, pour l'attribut qui a cette option.
- CHECK :définit un ensemble de valeurs possibles pour l'attribut.
- DEFAULT :donne une valeur par défaut (A lacréation du tuple, c'est la valeur par défaut qui serafournie, sauf saisie d'une valeur).

Les contraintes **PRIMARY KEY, NOT NULL, UNIQUE** et **CHECK** ont le même type de conséquence :
● Si on cherche à donner une valeur à un attribut qui n'est pas conforme à ce qui est précisé dans la définition de l'attribut :
- une valeur NULL s'il est défini NOT NULL ou PRIMARY KEY,
- une valeur existant déjà s'il est défini UNIQUE ou PRIMARY KEY,
- une valeur n'appartenant pas au domaine spécifié par CHECK,

●ALORS le SGBD renvoie un message d'erreur et ne modifie pas la base de données.

FOREIGN KEY : contrainte d'intégrité référentielle : ordre de création/suppression

1. Si on crée une table avec une clé étrangère, il faut que la table à laquelle on fait référence soit déjà créée => ordre logique de création des tables (l'ordre INVERSE pour la suppression).
2. Même chose pour la création et la suppression de tuples (lignes) : même ordre que création/suppression.
3. **Remarque :** on doit supprimer d'abord le tuple qui référence, ensuite le tuple référencé

2.2 CREATION D'UNE TABLE

CREATE TABLE nom_tab (<definition_colonne>
[, <definition_colonne>] …);
où <definition_colonne> est :
nom-col type [DEFAULT val_defaut] [NOT NULL] [UNIQUE] [autre]
où autre peut être :
… REFERENCES nom_tab [(nom_col [, nom_col] …)] …;
… CHECK condition;
Remarque : On peut nommer les contraintes

Exemple
CREATE TABLE livre (
code_l INTEGER PRIMARY KEY,
titre_l VARCHAR(30) NOT NULL,
num_aut INTEGER references auteur(num_aut)
) ;
Avec le nom de la contrainte de clé primaire :
CREATE TABLE auteur (
num_aut INTEGER,
nom_aut VARCHAR(30),
CONSTRAINT cle_auteur PRIMARY KEY (num_aut)
);

CREATE TABLE – Contraintes : Rappel

NOT NULL :si on ne spécifie rien pour la valeur d'un attribut, il prend la valeur NULL (indéterminée). Si la clause NOT NULL est présente, on doit spécifier une valeur non nulle pour cet attribut.

DEFAULT :on peut spécifier une valeur par défaut pour un attribut à la création de la table (cette valeur doit être du type de l'attribut).

UNIQUE :spécifie qu'un attribut (ou groupe) est une clé candidate. Un seul tuple correspond à une valeur de cette clé. Il est conseillé de spécifier NOT NULL pour un attribut déclaré UNIQUE (qui n'est pas clé primaire).

Exemple
Create table fournisseurs
(… nom_fou char(25) NOT NULL UNIQUE
…);
CHECK :spécifie une contrainte qui doit être vérifiée à tout moment par les tuples concernés.
Exemple :
Create table …
(cli_type char(16) DEFAULT 'PARTICULIER'
CHECK (cli_type IN
('PARTICULIER', ' PME ', 'PMI')) ;

qte_liv integer default CHECK (qte_liv <= qtç_com….)
NOT NULL : ... CHECK (nom_fou IS NOT NULL) …
Intervalle : . CHECK val_prix BETWEEN 10 AND 40

Contrainte d 'entité : PRIMARY KEY
1- Create table client
(numcli char(5) not null primary key,
…
);
2- Si la clé est composée (ici 2 attributs) :
Create table lign_comm
(ref char(5) not null,
design char(10) not null,
primary key (ref, design),
…
);

CI référentielle
=> gestion correcte des modifications de la clé étrangère dans la table qui référence et de la clé primaire dans la table référencée.
Soient les relations :
client(numcl, ….) et commande(numcom, …numcl,..)
si on ajoute une commande, il faut que le client associéexiste.

2.2.1 Vérification des CI référentielles

Pour gérer les Contraintes d'intégrité référentielles : vérification assurée par le SGBD automatiquement : la contrainte est déclarée explicitement une fois pour toutes (valable pour tout
programme et tout utilisateur).
=> adoptée par la norme SQL-92 ou SQL-2 (et donc dans beaucoup de SGBD)

Exemple
Create table client
(num_cl char(5) not null primary key, ...);
Create table commande
(numcomm integer not null primary key, .
num_com char(5) not null REFERENCES client, ...);

Si la clé étrangère = plusieurs attributs
⇒ utiliser FOREIGN KEY.
… FOREIGN KEY (atr1_cle, attr2_cle)
REFERENCES (cle_prim1).

2.3 Modification de la structure d'une table :

INSTRUCTION ALTER TABLE
Pour modifier la structure d'une table, on utilise l'instruction ALTER TABLE :
• ajouter un champ (une colonne) à une table (ADD COLUMN)
• modifier un champ (une colonne) (ALTER COLUMN)
• supprimer une colonne (DROP COLUMN)
• ajouter une contrainte (ADD CONSTRAINT)
• supprimer une contrainte (DROP CONSTRAINT)

ALTER TABLE nom_tab ADD nom_col type ;
ALTER TABLE nom_tab DROP COLUMN nom_col ;

ALTER TABLE nom_tab ALTER nom_tab type ;
●EX : ALTER TABLE emp ADD (diplome char(20));
ALTER TABLE EMP ALTER (sal number (10,3));

2.4 Suppression d 'une table : DROP

DROP TABLE nom-tab;
Ex : Drop table bonus;

Autres exemples
• Ajout d'un champ Age de type décimal
ALTER TABLE etudiant ADD COLUMN Age DECIMAL;
• Modification du type du champ
ALTER TABLE etudiant ALTER COLUMN Age INTEGER;

Ajout d'une contrainte sur ce champ
ALTER TABLE etudiant ADD CONSTRAINT CK_age CHECK (age > 18);
• Suppression de la contrainte
ALTER TABLE etudiant DROP CONSTRAINT CK_age;
• Suppression de la colonne
ALTER TABLE etudiant DROP COLUMN Age;

3 Langage de manipulation de données (LMD)

3.1 Interrogation : select

● Permet d'interroger une BD (requêtes sur les tables de la BD)
● Syntaxe générale :
Select … from … where .. <autres clauses>
– où autres clause : group by, order by, ...

Le résultat d'un SELECT est souvent un ensemble de lignes de (ou des) table(s) sur laquelle (lesquelles) porte le SELECT.
● Il existe plusieurs options pour la commande
SELECT : tri, regroupement, projection, sélection, sous-interrogation, conditions, ….

On utilise comme exemple une BDR contenant 2 tables EMP et DEPT et une table vide DUAL ayant un seul attribut.
EMP (empno, ename, job, mgr, hiredate, sal, comm, deptno)
DEPT (deptno, dname, loc)
EMP et DEPT ne sont pas vides.
DUAL (dummy) table vide à une colonne

SELECT de base
Select * from nom_tab;

– Lister toutes les lignes de la table
Select col1, col2, …. from nom_tab

– projection sur les attributs col1, col2, …
● ex. liste des infos sur les départements :
Select * from dept;
● ex. liste des noms et salaires des employés :
Select ename, sal from emp;

3.2 Eliminer les doublons

Quand on projette sur un ou plusieurs attributs, les valeurs peuvent être dupliquées
=> utiliser **DISTINCT** pour supprimer les doublons

Liste des métiers (job) :
Select job from emp; - - valeur de job est dupliquée :

Accès aux attributs dans un SELECT
● Parfois, on a besoin de préfixer les noms d'attributs par le nom de la table : tab.col
● ex.
Select ename, job from emp;
ou
Select emp.ename, emp.job from emp;

3.3 Conditions de recherche : WHERE

● C'est l'operateur de sélection (ou restriction) de l'algèbre relationnelle
● Permet d'extraire les lignes d'une table satisfaisant une ou plusieurs conditions.
Select … from …where [not] cond1 [and | or] cond2 …;
● ex. les employés du département numéro 20 :
Select * from emp where deptno = 20;
● ex. employés dont le métier est 'OPERATEUR' :

Select * from emp where job = 'CLERCK';

Les employés dont le salaire est > 2000 et qui travaillent dans le département numéro 20 :
Select * from emp where
sal>2000 and deptno=20;
Employés qui travaillent dans les départements 10 ou 30 et qui sont 'MANAGER' :
select * from emp where
(deptno = 10 OR deptno = 20)
AND job = 'MANAGER';

Autres exemples
Remarque :les conditions peuvent être reliées par les connecteurs logiques **NOT, AND, OR** (prio(not)>prio(and)>prio(or))

-Les informations sur les départements numeros 30 et 20 :
Select * from dept where deptno=30 or deptno=20;
Dans les conditions, les operateurs sont : = , <>, < , > , <= et >=
Ces operateurs s'appliquent aux types numérique, chaine de
caractères et dates.

La valeur NULL
● Correspond à l'absence de valeur pour un attribut
● Est différent de la valeur 0 (zéro)
● Se teste avec : attibut IS [NOT] NULL
● ex. les employés dont le commission n'est pas renseignée :
select * from emp where comm is null;
● ex. Les employés qui ont un chef :
select * from emp where mgr is not null;

Intervalle de valeurs : [not] between
● ex. les noms et salaires des employés dont le salaire est compris entre 1000 et 2000

(bornes incluses) :
select ename, sal from emp
Where sal between 1000 and 2000;
● Equivalent a :
select ename, sal from emp where
sal >= 1000 and sal <= 2000;

Remaque: on peut comparer avec **NOT BETWEEN**
● ex. les employés dont le salaire est <= 1250 et > 2500 :

select * from emp
Where sal not between 1250 and 2500 or sal = 1250;

liste de valeurs : [not] in
● ex. les noms et salaires des employés qui travaillent dans l'un des départements 10, 20, ou 30 :
select ename, sal from emp
Where deptno in (10, 20, 30);
● Equivalent a :
select ename, sal from emp
Where deptno=10 or deptno=20 or deptno=30;
Les employés qui ne sont ni du département 20, ni du département 30 :
Select * from emp where deptno NOT IN (20, 30)

3.4 Renommer les colonnes ou les tables

EX: 1. Lister les noms des départements :
select dname AS «NOM DEPARTEMENT»
from dept;
/* nom est composé => mettre entre guillemets */

2. Lister les noms et salaires des employés :
select ename AS NOM, sal AS SALAIRE
from emp;

Opérateurs arithmétiques
+, -, *, / : combines pour faire des expressions.
EX : Afficher les employés avec leur salaire total (salaire +
commission quand elle existe).
Select ename, sal, comm, sal+comm as «salaire total»
where comm is not null;

Tri des résultats
Les résultats d'une sélection peuvent être triés, sur une ou plusieurs colonnes, par ordre croissant (par défaut) ou décroissant.
Syntaxe :
Select nom_col [,nom_col ...] from nom_tab
where …..
order by nom_col [asc|desc] [,nom_col [asc|desc] ...] ;

Exemples
Employés dont le salaire est < 1500, triés par ordre alphabétique croissant :
Select * from emp where sal < 1500 ORDER BY ename;
/* Ordre croissant par défaut */
Employés par ordre alphabétique inverse du nom :

Select * from emp where sal < 1500 ORDER BY ename desc;

Correspondances de chaînes :
LIKE , NOT LIKE
Les caractères jokers :
% : toute chaîne de caractère (même vide)
_ : un et un seul caractère
[ ] un caractère de l'intervalle ou de l'ensemble spécifié
[^] un caractère en dehors de l'intervalle ou de l'ensemble spécifié

Exemples
Liste des employés dont le nom commence par 'a' et se termine par 'e' :
Select * from emp
where ename like 'a%e' or ename like 'A%E';
On peur utiliser des fonctions (transforment le mot en majuscules ou minuscules)
Select * from emp
where ename like upper('a%e')
or ename like lower('A%E');

Exemples
Liste des employés dont le nom commence possède exactement 8 caractères et se termine par 'e' :
Select * from emp
where ename like upper('_______e') ;

Exemples
Noms des employés se terminant par 's' :
SELECT * FROM emp WHERE ename LIKE '%s';
Nom des répartements commençant par 't' ou 'T' et se terminant par 's':
SELECT * FROM dept
WHERE dname LIKE 't%s' OR dname LIKE 'T%s';
Nom des départements commençant par 't' ou 'T' et se terminant par 's'
et ayant 5 caractères exactement:
SELECT * FROM dept
WHERE dname LIKE 't___s' OR dname LIKE 'T___s';

Exemples
Noms des employés contenant la chaîne 'abc' au début du nom :
SELECT * FROM emp WHERE ename LIKE 'abc%';
Nom des répartements commençant par 'B' et ayant 7 caractères :
SELECT * FROM dept
WHERE dname LIKE upper('b_____');
Noms des employés ne contenant pas la lettre 'e' :
SELECT * FROM emp WHERE ename NOT LIKE '%e%';

Exemples
Employés dont les noms commencent par b, s ou p :
SELECT * FROM emp WHERE ename LIKE '[bsp]%';
Villes dont les noms finissent par a, b, c ou d :
SELECT * FROM dept WHERE loc LIKE ' %[a-d]';
Employés dont les noms ne commencent pas par b, t ou s :
SELECT * FROM emp WHERE ename LIKE '[!bsp]%';

4 Les Jointures

- Association de lignes de plusieurs tables en fonction d 'un critère (de jointure).
Select … from nom_tab [nom_alias]
where <critère de jointure>;
- EX. Afficher les noms des employés (table emp) avec les noms de leur département (table dept) :
Select ename, dname from emp, dept
where emp.deptno = dept.deptno ;
Rmq : colonne commune deptno

Remarque :
Le nom de colonne étant identique => on le préfixe par le nomde la table.

4.1 Jointure interne

Pour retourner les enregistrements quand la condition est vraie dans les 2 tables. C'est l'une des jointures les plus communes. Elle est désignée par le mot clé **INNER JOIN**

4.2 Jointure externe

Pour effectuer une jointure externe entre table, il faut utiliser la syntaxe conforme au standard SQL (disponible aussi Oracle), **LEFT OUTER JOIN** et **RIGHT OUTER JOIN**:

Ex : SELECT * FROM table1
LEFT OUTER JOIN table2
ON table1.id=table2.id;

Tous les n-uplets de table1 apparaitront dans l'ensemble des n-uplets résultats. Les n-uplets de table1 pour lesquelles il n'est pas possible de faire une jointure avec des n-uplets de la table2 auront NULL comme valeur pour les attributs de table2.

SELECT * FROM table1
RIGHT OUTER JOIN table2
ON table1.id=table2.id;

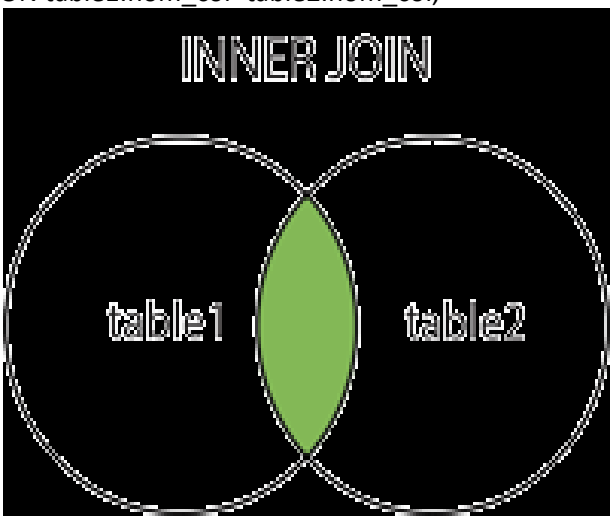
Tous les n-uplets de table2 apparaitront dans l'ensemble des n-uplets résultats. Les n-uplets de table2 pour lesquelles il n'est pas possible de faire une jointure avec des n-uplets de la table1 auront NULL comme valeur pour les attributs de table1.

Récapitulatif

- **INNER JOIN** : jointure fermée, les données doivent être à la fois dans les deux tables.
- **LEFT [OUTER] JOIN :** jointure ouverte, on lit les données de la table de gauche en y associant éventuellement celle de la table de droite.
- **RIGHT [OUTER] JOIN :** jointure ouverte, on lit les données de la table de droite en y associant éventuellement celle de la table de gauche.

SELECT nom_col
FROM table1 INNER JOIN table2
ON table1.nom_col=table2.nom_col;
Ou:
SELECT nom_col
FROM table1 JOIN table2

ON table1.nom_col=table2.nom_col;



Exemple : inner join

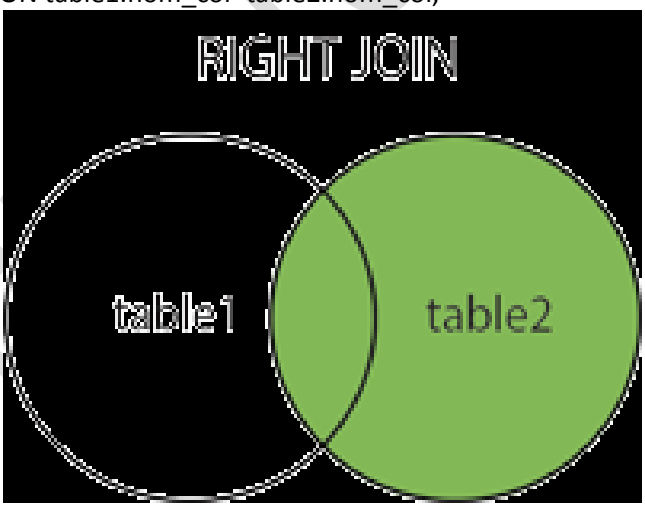
```
SELECT emp.ename, dept.dname
FROM emp INNER JOIN dept
ON emp.deptno=dept.deptno
ORDER BY emp.ename;
```

Note: INNER JOIN sélectionne toutes les lignes des 2 tables emp et dept pour les colonnes où il y a correspondance entre les colonnes deptno des 2 tables. S'il y a des lignes de emp qui ne correspondent pas aux lignes de dept, les lignes de emp n'apparaissent pas.

```
SELECT nom_col
FROM table1 RIGHT JOIN table2
ON table1.nom_col=table2.nom_col;
```

Ou:

```
SELECT nom_col
FROM table1 RIGHT OUTER JOIN table2
ON table1.nom_col=table2.nom_col;
```



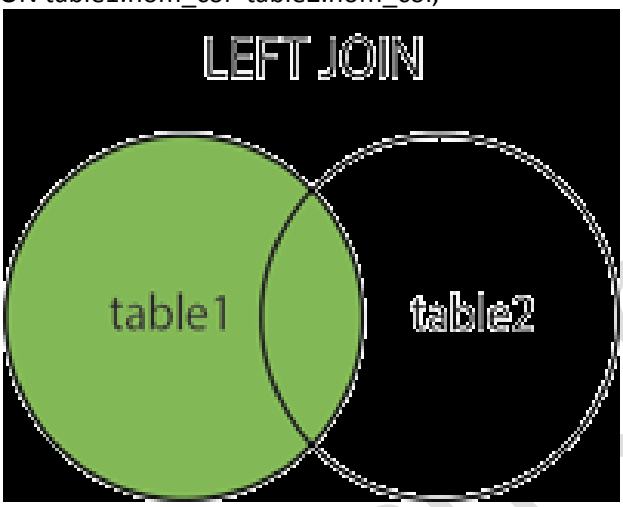
Exemple : right join

```
SELECT emp.ename, dept.dname
FROM emp RIGHT JOIN dept
ON emp.deptno=dept.deptno
ORDER BY emp.ename;
```

Note: RIGHT JOIN sélectionne toutes les lignes des 2 tables emp et dept pour les colonnes où il y a correspondance entre les colonnes deptno des 2 tables.

S'il y a des lignes de dept qui ne correspondent pas aux lignes de emp, ces lignes de dept apparaissent dans le résultat.

```
SELECT nom_col
FROM table1 LEFT JOIN table2
ON table1.nom_col=table2.nom_col;
ou:
SELECT nom_col
FROM table1 LEFT OUTER JOIN table2
ON table1.nom_col=table2.nom_col;
```



Exemple : left join

```
SELECT emp.ename, dept.dname
FROM emp LEFT JOIN dept
ON emp.deptno=dept.deptno
ORDER BY emp.ename;
```

Note: LEFT JOIN sélectionne toutes les lignes des 2 tables emp et dept pour les colonnes où il y a correspondance entre les colonnes deptno des 2 tables. S'il y a des lignes de emp qui ne correspondent pas aux lignes de dept, ces lignes de emp apparaissent dans le résultat.

5 Sous-interrogation

```
Select … from …
where nom_col <opérateur>
(Select … From … Where…);
```

EX: Afficher les employés qui sont dans le même departement que 'ALLEN' : d'abord recherche du département de « ALLEN », puis connaissant son numéro (30), on cherche les employés qui y travaillent.

```
Select * From emp Where
deptno = (Select deptno from emp
where ename like ‘ALLEN’);
```

Sous-interrogation rapportant plusieurs lignes : IN, ANY, ALL

Le résultat d’une sous-interrogation peut comporter plusieurs lignes. Dans ce cas les

opérateurs =, <, > … ne conviennent plus. L'égalité sera traitée par l'operateur IN (un '=' par rapport à une suite de valeurs), et Les inégalités seront traitées par l'operateur ANY et ALL.

5.1 Opérateur IN

Exemple : Les employés travaillant dans le même département que l'un des employés dépendant du président.

```
Select * from emp
Where deptno in
    (select deptno
     from emp
     Where mgr = (Select empno from emp
    Where job=' PRESIDENT '
    )
);
```

5.2 Opérateur ANY

Le résultat de la comparaison est VRAI, s'il l 'est pour au moins un élément de l 'ensemble (de la sous-interrogation) EX: Afficher les employés ayant un salaire supérieur a celui de l 'un des employés travaillant dans le département 10

```
Select * from emp
where sal > ANY (select sal
from emp
where deptno=10
);
```

5.3 Opérateur ALL

Le résultat de la comparaison est VRAI, s'il l 'est pour tous les éléments de l 'ensemble (sous-interrogation) EX: Afficher les employés ayant un salaire supérieur à celui des employés travaillant dans le département 20.

```
Select * from emp
where sal > ALL (select sal
from emp
where deptno=20
);
```

Extraction de lignes si le résultat de la sous-interrogation comporte au moins une ligne : [NOT] EXIST

```
Select … From ….
 Where [NOT] EXISTS
(Select … From … Where …);
Ex1: lister les employes s'il y en un parmi eux qui a une
commission > 1000;
Select * from emp where
EXISTS
(select * from emp where comm > 1000);
```

Extraction de lignes si le résultat dela sous-interrogation comporte au moinsune ligne : [NOT] EXIST

●Ex2 : lister les employes s'il n 'y a aucun parmi eux qui a une

commission > 1000
Select * from emp where
NOT EXISTS
(select * from emp where comm > 1000);

6 Fonctions de groupes

→applique des fonctions sur un ensemble de données :

6.1 Min :

Retourne le minimum

6.2 Max :

Retourne le maximum

6.3 Avg :

Retourne la moyenne arithmétique

6.4 Count :

Retourne le nombre (de lignes)

6.5 Sum :

Retourne le total (la somme)

On utilise la clause GROUP BY, pour indiquer au moteurSQL que la fonction porte sur une sélection de données(groupe de lignes).

Exemple
Afficher la sommes des salaires par département :
Select deptno, sum(sal) as “somme salaires”
from emp
group by deptno;

Exemple
Ex:
1. Afficher le total des salaires :
Select sum(sal) as total_sal from emp;
2. Afficher le total et la moyenne des commissions :
Select sum(comm) as «somme_comm,
avg(comm) as moy_comm
from emp;
●Rmq : la valeur NULL n ’entre pas dans les différents calculs.
Ex : la moyenne de la colonne COMM est calculée en cumulant le nombre de commissions divisée par le nombre de valeurs non NULL.

Ex. Afficher le nombre de commissions :
Select count(comm) from emp;
●**Remarque :** Il est impossible d ’utiliser a la fois une projection et

une fonction de groupe : Select ename, sum(sal) from emp;
SERA REJETEE.
●**Remarque :** COUNT (*) : nombre de lignes non NULL.

Conditions de sélection sur les groupes :
HAVING
Créer des critères de sélection qui portent sur un ensemble de données :
→ associer la clause GROUP BY a la clause
HAVING (ou NOT HAVING).
EX. Les départements qui ont des salaires > 5000 :
Select deptno, sum(sal) as “somme salaires”
from emp
group by deptno having sum(sal) > 5000;

7 Opérateurs sur plusieurs tables

union, différence, intersection
Soit 2 tables de même structure et de contenus différents :
depot1 et deptot2:
Structure :
●numprod, design, conditionnement, stock, rayon, pu, tva

7.1 L'union : UNION

En faisant l'union de 2 SELECT, on obtient les lignes
de l'une ou de l'autre table (Les doublons sont élimines)
Ex : Lister les produits en stocks dans lun ou l'autre des
2 dépôts :
Select numpro, design, pu, from depot1
UNION
Select numpro, design, pu, from depot2;
Rmq : les noms de colonnes du résultat sont les noms
figurant dans le 1er SELECT

7.2 La différence :

MINUS (Oracle), EXCEPT (Posgresql)
En faisant la différence entre 2 SELECT, on obtient les lignes se
trouvant dans la 1ere table mais pas dans la 2eme.
Ex : Lister les produits en stocks seulement dans le 1er dépôt :
Select numpro, design, pu, from depot1
EXCEPT
Select numpro, design, pu, from depot2;
Rmq : les noms de colonnes du résultat sont les noms figurant dans le 1er SELECT

7.3 Intersection : INTERSECT

En faisant l'intersection de 2 SELECT, on obtient les lignes communes aux 2 tables
Ex : Lister les produits en stocks dans les 2 depots :
Select numpro, design, pu, from depot1
INTERSECT

Select numpro, design, pu, from depot2;
Rmq : les noms de colonnes du résultat sont les noms figurant dans le 1er SELECT

8 Les types de données :

8.1 LES ENTIERS

On choisit un type d'entier plutôt qu'un autre, selon la taille de l'entier a stocker.
* Un entier court : [-32 768 , 32767]
* Un entier : [-2 147 483 648 , 2 147 483 647]
* Un entier long : [-9 223 372 036 854 775 808 , 9 223 372 036 854 775 807.

8.2 Les REELS

• reel simple : FLOAT4 ou REAL
• reel double : FLOAT8 ou FLOAT ou DOUBLE PRECISION

Remarque :

un type ≪ float ≫ ne permet pas de stocker la valeur precise d'unnombre reel. Il stocke une valeur approchee (nombre a virguleflottante).

8.3 Les DECIMAUX PRECIS

Contrairement au type ≪ float ≫, le type NUMERIC (ou DECIMAL) permet de stocker la valeur exacte d'un réel.
Syntaxe :
NUMERIC (long, dec) ou DECIMAL (long, dec)
long : nombre maximum de chiffres a stocker
dec : nombre de chiffres apres la virgule
Exemple NUMERIC (9, 3)
permet de stocker un nombre a 9 chiffres dont 3 après la virgule.

8.4 Les CHAINES DECARACTERES

• chaine fixe : CHAR ou CHARACTER
CHAR (n) ou CHARACTER (n)
=> Stockage de n caractères
• chaine variable : VARCHAR ou CHARACTER VARYING
VARCHAR (n) ou CHARACTER VARYING (n)
=> Stockage de n caractères au maximum
• chaine illimitée : TEXT
TEXT, stockage d'un nombre illimité de caractères entre apostrophes (ou quottes)

Exemples

• En déclarant un type CHAR(8), la chaine ≪lundi≫ (5 caractères) et la chaine ≪vendredi≫ (8 caractères)

occuperont chacune 8 caractères.
• Les données stockées dans une colonne de type VARCHAR occuperont exactement l'espace nécessaire a leur stockage.
• VARCHAR2 (sensiblement équivalent au VARCHAR)
Est spécifique au SGBD Oracle.

Exemples :
BOOLEAN
Le type booléen (BOOLEAN ou BOOL), ce type peut contenir :
pour vrai : true, t, yes, y ou 1
Pour faux : false, f, no, n ou 0

8.5 DATES ET HEURES

Il existe plusieurs façons de stocker une date :
• Le type DATE (code sur 4 octets) permet de stocker une date au jour prés.
• TIMESTAMP (code sur 8 octets) permet de stocker une date au millième de seconde près.
• TIME (code sur 8 octets) permet de stocker une heure au millième de seconde près.
• INTERVAL (code sur 12 octets) permet de stocker un intervalle de temps au millième de seconde près.

Remarque :
Les dates peuvent être présentées différemment :
l'ordre du jour du mois et de l'année peut être différent selon la présentation européenne ou non. Le caractère séparateur peut également être différent :/ ou . par exemple.

9 Expressions et Fonctions

9.1 Une expression :

combinaison de variables (contenu d'une colonne), de constantes et d'autres expressions a l 'aide desopérateurs : +, -, *, /

9.2 Une fonction :

Routine ayant des arguments et ramenant unrésultat (un argument peut être une expression)
Il y a 3 types d'expressions : arithmétiques, chaines decaractères et date.
A chaque type ==> il y a des opérateurs spécifiques.

9.2.1 Fonction NVL (Oracle), Coalesce (Postgres)

NVL : NULL VALUE (Oracle) : permet de remplacer une valeur NULL par une valeur significative.
Syntaxe : NVL(exp1, exp2) :

→ renvoie la valeur de exp1, si exp1 non NULL et de exp2 sinon.
Ex: Select ename, sal, comm, sal+NVL(comm, 0) from emp;

Coalesce : PostrgreSQL
COALESCE(arg1, arg2, …, argn) ou argi est une colonne d'une table.
La valeur de la fonction est la première valeur NOT NULL.
EX. Select ename, sal, comm, sal+COALESCE(comm,0)
from emp;

9.2.2 Concaténation

Met bout à bout 2 chaines de caractères.
EX. Afficher les noms des employés concatène à leur métier.
Select ename||' - '||job AS 'NOM – METIER' from emp;

9.2.3 ROUND (n, m) :

Permet d'arrondir le nombre n avec m décimales.
Exp. Noms et salaires horaires des employés du département 20, en renommant les colonnes.

```
Select ename as «nom, ROUND(sal / 150, 2)
as « salaire_horaire »
from emp where deptno = 20;
```

9.2.4 TRUNC (n, m) :

Permet de tronquer le nombre n à m décimales.

```
SELECT TRUNC (15.79,1) "Truncate" FROM DUAL;
Truncate ----------
15.7
```

9.2.5 Opérateur de choix : CASE

```
CASE WHEN cond1 THEN arg1
WHEN cond1 THEN arg2 ….
ELSE arg-n
END
```

EX. Dans emp, afficher le salaire seul si deptn = 10, le salaire plein si deptno = 20, le numéro de département sinon.

```
SELECT CASE
WHEN deptno=10 THEN sal
WHEN deptno = 20 THEN sal+coalesce(comm,0)
ELSE deptno
END
FROM emp;
```

9.2.6 Fonctions sur Chaînes de caractères

9.2.6.1 SUBSTR (chaine, pos, n) :

Extrait de la chaîne 'chaine', à partir de la position pos, n caractères (ou le reste de la chaîne, si n absent)

9.2.6.2 UPPER (chaine) ou LOWER (chaîne) :

Transforme en majuscules (ou minuscules) la chaîne

9.2.6.3 LENGTH (chaine) :

Donne le nombre de caractères de lachaîne

9.2.7 Les fonctions de DATES

9.2.7.1 TO_CHAR (col, format)

col est une colonne d'une table, ou une valeur
de type DATE.
Met col (correspondant a une date) selon le format : YYYY année,
YY : 2 chiffres de l'année, MM : mois, HH24 : heure sur 24 heures,
…
Exp. Transformer le date d'embauche au format 'DD/MM/YYYY'.
Select TO_CHAR(hiredate, 'DD-MM-YYYY') as date from emp;
MONTH , DAY : mois (jour) en lettres
Autres fonctions DATE
CURRENT_DATE : date du jour

10 Commandes de modifications des données

Principales commandes :
- INSERT pour ajouter les données
- UPDATE pour modifier les données
- DELETE pour supprimer les données

10.1 INSERT

INSERT INTO nom_tab [(nom_col,...)]
VALUES ({expr | DEFAULT},...)
Si les colonnes sont présentes, leur nombre doit correspondre
au nombre d'expressions (ou valeurs) .

Ex1: Insérer un nouveau département : 50, 'EDUCATION', 'MIAMI' :
Insert INTO dept VALUES (50, 'EDUCATION' , 'MIAMI');
Rmq : On n 'a pas a citer les colonnes quand on insère des valeurs pour toutes les colonnes !
Ex2:Insérer un employé en ne connaissant que le numéro, le
nom, le job et le numéro de département :
Insert into EMP (empno, ename, job, deptno)
VALUES (7950, ' JOHN ', ' TRAINER ', 50);

10.2 Mise à jour : UPDATE

UPDATE nom_tab SET nom_col1=exp1 [,nom_col2=exp2 ...]
[WHERE conditions] [LIMIT nb_lignes]

- SET permet d'attribuer une nouvelle valeur à un attribut.

- On peut mettre à jour plusieurs attributs en même temps.
- WHERE permet de préciser quelles données on veut mettre à jour.

- Si clause WHERE absente, alors toutes les données de la table sont mises a jour.

- LIMIT permet de limiter le nombre de lignes à modifier.

Ex: Remplacer dans le département 30, la localité 'CHICAGO'par 'LOS ANGELES' :
UPDATE dept
SET loc = 'LOS ANGELES'
Where deptno=30;
Ex. Augmenter la commission de 10% pour tous les employés :
Update emp SET comm = comm * 1.1;
Ex. Mettre le numéro de département à 10 et leur salaire augmenté de 100 pour les employés du département 20 :
Update emp SET deptno=10, sal=sal+100
where deptno=20;

10.3 Suppression : DELETE

DELETE FROM nom_tab
[WHERE conditions] [LIMIT nb_lignes]

- WHERE permet de préciser quelles données on veut supprimer

- Si clause WHERE absente, alors toutes les données de la table sont supprimées (sauf si clause LIMIT)
- LIMIT permet de limiter le nombre de lignes à supprimer.

EX1. Supprimer toutes les lignes de la table BONUS :
delete from bonus;
EX2. Supprimer les employés ayant une commission non NULL :
delete from emp where comm is not null;

11 QUELQUES RAPPELS, EXEMPLES

Jointures : Associer plusieurs tables dans une même requête
Types de jointures
Les jointures internes :

- L'équijointure : Lorsque le prédicat de jointure est « = »

```
select *
from REALISATEUR inner join FILM
on REALISATEUR .NUMERO_REALISATEUR = FILM .NUMERO_REALISATEUR;
```

- Les jointures naturelles : S'il y a au moins une colonne qui porte le même nom entre les 2 tables SQL

```
select FILM.NUMERO_REALISATEUR
from REALISATEUR natural join FILM
```

Les jointures externes :

- RIGHT JOIN : Retourner tous les enregistrements de la table de droite même si la condition n'est pas vérifiée dans l'autre table.

```
s e l e c t *
  from REALISATEUR right outer join FILM
  on REALISATEUR .NUMERO_REALISATEUR = FILM .NUMERO_REALISATEUR;
```

- LEFT JOIN : Retourner tous les enregistrements de la table de gauche même si la condition n'est pas vérifiée dans l'autre table.

```
s e l e c t *
from REALISATEUR left outer join FILM
on REALISATEUR .NUMERO_REALISATEUR = FILM .NUMERO_REALISATEUR;
```

Jointure externe complète (FULL)
La table résultat conserve les lignes des deux tables en complétant au besoin par NULL

```
s e l e c t *
from FILM full outer join REALISATEUR
on REALISATEUR .NUMERO_REALISATEUR = FILM .NUMERO_REALISATEUR
```

UNION : Combiner un ensemble des résultats de deux requêtes
[instructions SQL 1] UNION [instructions SQL 2];

Table Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999

Table Internet_Sales

Txn_Date	Sales
07-Jan-1999	250
10-Jan-1999	535
11-Jan-1999	320

```
SELECT Txn_Date FROM Store_Information
UNION
SELECT Txn_Date FROM Internet_Sales;
```

Txn_Date
05-Jan-1999
07-Jan-1999
08-Jan-1999
10-Jan-1999
11-Jan-1999

UNION ALL : Combiner les résultats de deux requêtes. La différence entre UNION

ALL et UNION est que : UNION sélectionne seulement des valeurs distinctes et UNION ALL sélectionne toutes les valeurs.

[instructions SQL 1] UNION ALL [instructions SQL 2];

avec le même exp :

SELECT Txn_Date FROM Store_Information
UNION ALL
SELECT Txn_Date FROM Internet_Sales;

Txn_Date
05-Jan-1999
07-Jan-1999
08-Jan-1999
08-Jan-1999
07-Jan-1999
10-Jan-1999
11-Jan-1999

INTERSECT() : La différence entre les deux commandes est la suivante:
UNION agit comme OR (OU) ; INTERSECT agit comme AND (ET)
(valeur sélectionnée seulement si elle apparaît dans les 2 instructions).
[instructions SQL 1] INTERSECT [instructions SQL 2];

Table Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999

Table Internet_Sales

Txn_Date	Sales
07-Jan-1999	250
10-Jan-1999	535
11-Jan-1999	320

SELECT Txn_Date FROM Store_Information
INTERSECT
SELECT Txn_Date FROM Internet_Sales;

Txn_Date
07-Jan-1999

MINUS : prend tous les résultats de la première instruction SQL,
puis soustrait ceux de la deuxième instruction SQL. Si la deuxième instruction SQL comprend des résultats qui ne sont pas inclus dans la première instruction SQL, ils seront ignorés.
[instructions SQL 1] MINUS [instructions SQL 2];
Table Store_Information Table Internet_Sales

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999

Txn_Date	Sales
07-Jan-1999	250
10-Jan-1999	535
11-Jan-1999	320

```
SELECT Txn_Date FROM Store_Information
MINUS
SELECT Txn_Date FROM Internet_Sales;
```

Txn_Date
05-Jan-1999
08-Jan-1999

CASE : permet de faier un choix selon des conditions

```
SELECT CASE ("nom de colonne")
  WHEN "condition1" THEN "résultat1"
  WHEN "condition2" THEN "résultat2"
 ...
  [ELSE "résultatN"]
 END
FROM "nom de table";
```

Table Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

```
SELECT Store_Name, CASE Store_Name
 WHEN 'Los Angeles' THEN Sales * 2
  WHEN 'San Diego' THEN Sales * 1.5
 ELSE Sales
  END
"Nouvelles Ventes",
Txn_Date
FROM Store_Information;
```

Store_Name	Nouvelle vente	Txn_Date
Los Angeles	3000	05-Jan-1999
San Diego	375	07-Jan-1999
Los Angeles	300	08-Jan-1999

EXISTS : teste simplement si la requête interne retourne une ligne. Si elle le fait, la requête externe peut s'exécuter. Sinon, la requête externe ne s'exécutera pas, et l'instruction SQL entière ne retournera aucun résultat.

```
SELECT "nom de colonne 1"
FROM "nom de table 1"
WHERE EXISTS
        (SELECT *
         FROM "nom de table 2"
         WHERE "condition"
        );
```

Exp : Table Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999

Table Geography

Region_Name	Store_Name
East	Boston
East	New York
West	Los Angeles

```
SELECT SUM(Sales)
FROM Store_Information
WHERE EXISTS
(SELECT * FROM Geography
WHERE Region_Name = 'West');
```

SUM (Sales)
2050

LES PREDICATS (ALL, ANY, EXISTS (NOT EXISTS)

ALL : Teste si la valeur d'un attribut satisfait un critère de comparaison avec tous les résultats d'une sous-requête

```
SELECT colonne1
 FROM table1
WHERE colonne1 > ALL
                    ( SELECT colonne1
                FROM table2 )
```

ANY : Teste si la valeur d'un attribut satisfait un critère de comparaison avec au moins un résultat d'une sous-requête

```
SELECT B.nb, B.nom
FROM Buveurs B, Commanes C
WHERE B.nb = C.nb AND C.qte > ANY
(SELECT qte
                    FROM Commandes)
```

EXISTS : Teste si la réponse à une sous-requête est vide "Viticulteurs ayant produit au moins un vin"

```
SELECT P.*
FROM Producteurs P
WHERE EXISTS
        (SELECT R.*
         FROM Recoltes R
        WHERE P.num = R.nprod)
```

12 LES FONCTIONS

12.1 LES FONCTIONS D'AGREGATION

12.1.1 COUNT() :

Compter le nombre d'enregistrements dans une table

```
SELECT COUNT(*) FROM table
```

Exp : Liste les utilisateurs d'un site web d'e-commerce :

Id	nom	ville
1	Marie	Paris
2	Louis	Marseille
3	Paul	Lyon

```
SELECT COUNT(*) FROM utilisateur
```

Résultat : 3

12.1.2 SUM() :

Calculer la somme totale d'une colonne contenant des valeurs numériques

```
SELECT SUM(nom_colonne)
FROM table
```

Exp :

Id	facture_id	produit	prix
1	1	calculatrice	17
2	1	ciseaux	3
3	2	agenda	15

```
SELECT SUM(prix) AS prix_total
FROM facture
WHERE facture_id = 1
```

Résultat : 20

12.1.3 AVG() :

Calculer une valeur moyenne sur un ensemble d'enregistrements de type numérique.
Syntaxe : SELECT AVG(nom_colonne)
FROM nom_table
Exp :

Id	Client	Tarif	date
1	Pierre	102	2012-10-23
2	Simon	47	2012-10-27
3	Pierre	160	2012-12-03

SELECT client, AVG(tarif)
FROM achat
GROUP BY client

client	AVG(tarif)
Pierre	131
Simon	47

12.1.4 MAX()/MIN() :

Retourner la valeur maximale/minimale d'une colonne dans un ensemble d'enregistrements.
Syntaxe : SELECT MAX(nom_colonne) FROM table
SELECT MIN(nom_colonne) FROM table

Id	Nom	Prix
1	clavier	50
2	souris	21

SELECT MAX(prix) FROM produit
50
SELECT MIN(prix) FROM produit
21

12.2 LES FONCTIONS DES CHAINES DE CARACTÈRES

12.2.1 TRIM() :

supprimer des caractères au début et en fin d'une chaîne de caractère
SELECT TRIM(' Exemple ');
TRIM('x', 'xxxExemplexxx');
Résultat : Exemple

12.2.2 LTRIM :

supprimer des caractères au début d'une chaîne de caractère
SELECT LTRIM(' Exemple ');
Résultat : 'Exemple ‘

12.2.3 RTRIM :

supprimer des caractères en fin d'une chaîne de caractère
SELECT RTRIM(' Exemple ');
Résultat : ‘ Exemple'

12.2.4 UPPER() :

permet de transformer tous les caractères en minuscules d'une chaîne de caractère en majuscules
SELECT UPPER('Exemple');
Résultat : EXEMPLE

12.2.5 LOWER() :

permet de transformer tous les caractères d'une chaîne de caractère en minuscules.
SELECT LOWER('BONJOUR tout le Monde');
Résultat : bonjour tout le monde

12.2.6 CONCAT() :

permet de concaténer les valeur de plusieurs colonnes pour ne former qu'une seule chaîne de caractère
CONCAT(colonne1, colonne2)

12.2.7 LENGTH() :

permet de calculer la longueur d'une chaîne de caractères (MySQL, PostgreSQL et Oracle) [LEN() SQL Server]
SELECT LENGTH('exemple');
Résultat : 7

12.2.8 SUBSTRING() (ou SUBSTR()) :

segmenter une chaîne de caractère. Autrement dit extraire une partie d'une chaîne.
SUBSTR(string, start, length)

Table pays

Id	nom_fr_frnom_en_gb	
1	FRANCE	FRANCE
2	ESPAGNE	SPAIN
3	ALLEMAGNE	GERMANY
4	CHINE	CHINA

SELECT id, nom_fr_fr, SUBSTR(nom_fr_fr, 1, 2)
FROM pays

13 Optimisation SQL

Plusieurs façons pour optimiser la performance d'une base de données

13.1 La bonne définition de la structure de la base :

- Standardiser les colonnes : Déclarer des champs identiques du même type et bien choisir les types de variables
- Si la table est petite il vaut mieux la mettre dans une autre table
- Privilégier VARCHAR plutôt que CHAR
- Eviter les valeurs nulles (problème d'indexer les valeurs NULL)
- Eviter les données redondantes en utilisant les clés primaires
- Eviter les clés composées
- Numériser l'information (type numeric)
- Si une colonne est utilisée beaucoup dans la clause WHERE il vaut mieux mettre longueur fixe et non variable

13.2 Normalisation de la base de données

- Les formes normales
- Dé-normalisation dans le cas des requêtes complexes

13.3 Réseau

Limitation de l'accès à la base, structurer l'accès et la distribution des cartes réseau

13.4 Utilisation des index :

Create index nom_index on nom_table(nom_champs1,nom_champs2,…..nom_champsn)

⇒ **2 types :**

B-Tree : C'est l'index par défaut

Bitmap : Dans le cas où le nombre de modalités est réduit (sexe, qualité (M, Mme, Mlle)….)

13.5 Partitionnement : fractionner une table (volumineuse)en partitions plus petites en utilisant des méthodes de partitionnement

13.6 Utilisation des clauses

- **Select** : mettre les noms des champs sectionnés (éviter le *)
- Opter pour BETWEEN que LIKE
- Eviter les in et not in, et les remplacer par EXIST, et or NOT EXIST

13.7 Optimisation des jointures:

Commencer toujours par les tables les moins volumineuses

13.8 Le ROW ID (Oracle):

C'est un ID unique pour chaque ligne il est unique dans toutes les BDD (oracle)

13.9 Les tables globales (Oracle) :

- Table temporaire,
- L'information supprimée suivant deux conditions (DELETE ROW et COMMIT PRESERVE ROW)
- Enregistrée dans la RAM