SQL
PL/SQL

Professeur Sofia Belkhala

Année Scolaire 2024/2025

Bases de données : Conception& Modélisation

Langage SQL

Introduction au SQL

Plusieurs langages permettant de définir, de manipuler et de contrôler les données relationnelles ont été proposés :

- QUEL (QUEry Language)
- QBE (Query By Example)
- **SQL** (Structured Query Language)

- Le langage SQL est basé sur un ensemble d'opérations à effectuer sur des tables.
- Ces opérations constituent d'une part l'algèbre relationnelle et d'autre part des opérateurs très utiles tels que le tri et les calculs (agrégats).

Introduction au SQL

- Les Systèmes de Gestion de Bases de Données (SGBD) présentent une interface externe sous forme de langage de requêtes.

- Celui-ci permet de spécifier les données à sélectionner ou à mettre à jour.

- Il n'est pas nécessaire de spécifier comment retrouver les informations (le SGBD s'en charge) mais seulement de quelles informations on a besoin.

Introduction au SQL

- Aujourd'hui le langage SQL est normalisé, et est le standard d'accès aux bases de données relationnelles

- SQL est un langage simple et concis

- Il est utilisable directement sur un terminal grâce à l'utilitaire interactif SQL ou dans un langage hôte.

Introduction au SQL

Le langage SQL ne présente intentionnellement qu'un nombre limité de verbes ou mots-clés, qui se répartissent en trois familles fonctionnellement distinctes :

1. Le **Langage de Définition de Données** (**LDD** ou DDL) permet la description de la structure de la base de données (tables, vues, attributs, index...). Les mots clés utilisés sont: CREATE, DROP et ALTER.

2. Le **Langage de Manipulation de Données** (**LMD** ou DML) permet la manipulation des tables et des vues avec ses quatre verbes correspondant aux opérations fondamentales sur les données: INSERT, DELETE, UPDATE et SELECT

3. Le **Langage de Contrôle de Données** (**LCD** ou DCL) contient les primitives de gestion des transactions: COMMIT et ROLLBACK et des privilèges d'accès aux données GRANT et REVOKE

Langage SQL :

La Définition des donnés

Création des Tables

- La création de la Base de Données (BD) se fait simplement en créant les tables qui la composent.
- Il n'est pas nécessaire d'attribuer un nom à un ensemble de tables (schéma), mais c'est possible.

Création de la BDD

Création simple:

CREATE TABLE nom_table (Nom_col1 TYPE1, Nom_col2 TYPE2, …);

- **NUMBER(x) :**pour les entiers (optionnel : de longueur x)
- **NUMBER(x,y) :** réels (optionnel : de longueur totale x dont y décimales)
- **CHAR(n) :** chaines de caractères de longueur n
- **VARCHAR2(n) :** chaines de caractères de longueur variable, n max date, de la forme
- **DATE :** 01/11/81 ou 01-nov-81
- **TIMESTAMP :**date et heure

Création des Tables : Contraintes

- Les contraintes sont les règles appliquées aux colonnes de données d'une table. Celles-ci sont utilisées pour limiter le type de données pouvant aller dans une table. Cela garantit l'exactitude et la fiabilité des données de la base de données.
- Le contrôle de la validité des données se fait par la définition de contraintes
- Il y a deux types de contraintes :
  - Les contraintes de colonne portent sur une seule colonne
  - Les contraintes de table portent sur un ensemble de colonnes

Création des Tables : Contraintes

Les contraintes les plus communes sont :

- NOT NULL
- DEFAULT
- UNIQUE
- CHECK
- PRIMARY KEY
- FOREIGN Key

Contraintes : NOT NULL

- Par défaut, une colonne peut contenir des valeurs **NULL**. Si on ne souhaite pas qu'une colonne ait une valeur **NULL**, on doit définir une telle contrainte sur cette colonne en spécifiant que **NULL** n'est plus autorisé pour cette colonne.

```
CREATE TABLE Employes(
    Id NUMBER                NOT NULL,
    Nom VARCHAR (20)     NOT NULL,
    Age  NUMBER              NOT NULL,
    Salaire   DECIMAL (18, 2)
  );
```

- Si la table Employes a déjà été créée, pour ajouter une contrainte NOT NULL à la colonne Salaire dans MySQL, la requête s'écrit comme suit :

```
  ALTER TABLE Employes
  MODIFY Salaire  DECIMAL (18, 2) NOT NULL;
```

Contraintes : DEFAULT

- La contrainte **DEFAULT** fournit une valeur par défaut à une colonne lorsque l'instruction **INSERT INTO** ne fournit pas de valeur spécifique.

```
CREATE TABLE Employes(
    Id NUMBER                   NOT NULL,
    Nom VARCHAR (20)     NOT NULL,
     Age  NUMBER                 NOT NULL,
    Salaire   DECIMAL (18, 2) DEFAULT 5000.00,
    PRIMARY KEY (Id)
);
```

- Pour supprimer une contrainte **DEFAULT**, on utilise la requête suivante :

```
ALTER TABLE Employes
MODIFY Salaire DEFAULT NULL;
```

Contraintes: UNIQUE

- La contrainte **UNIQUE** empêche que deux enregistrements aient des valeurs identiques dans une colonne.

```
CREATE TABLE Employes(
    Id NUMBER                   NOT NULL,
    Nom VARCHAR (20)     NOT NULL UNIQUE,
    Age  NUMBER                 NOT NULL,
    Salaire   DECIMAL (18, 2),
    PRIMARY KEY (Id)
);
```

Contraintes: CHECK

- La contrainte **CHECK** active une condition permettant de vérifier la valeur saisie dans un enregistrement. Si la condition est évaluée à false, l'enregistrement viole la contrainte et n'est pas entré dans la table.
- Par exemple, la requête SQL suivante crée la même table **Employes**, mais dans ce cas, la colonne **Age** est définie sur **CHECK**, de sorte qu'on ne peut pas avoir un employé de moins de 18 ans.

```
CREATE TABLE Employes(
    Id NUMBER                NOT NULL,
    Nom VARCHAR (20)     NOT NULL,
    Age  NUMBER                 NOT NULL CHECK (Age >= 18),
    Salaire   DECIMAL (18, 2),
    PRIMARY KEY (Id)
);
```

Contraintes: PRIMARY KEY

- Une colonne de clé primaire ne peut pas avoir de valeur **NULL**.
- Une table ne peut avoir qu'une seule clé primaire, qui peut consister en un ou plusieurs champs. Lorsque plusieurs champs sont utilisés comme clé primaire, ils sont appelés clé composite.
- Si une table a une clé primaire définie sur un ou plusieurs champs, vous ne pouvez pas avoir deux enregistrements ayant la même valeur pour pour ces champs.
- Syntaxe pour définir l'attribut Id en tant que clé primaire dans une table Employes.

```
CREATE TABLE Employes(
  Id NUMBER                NOT NULL,
    Nom VARCHAR (20)     NOT NULL,
    Age  NUMBER          NOT NULL,
    Salaire   DECIMAL (18, 2),
    PRIMARY KEY (Id)
);
```

Contraintes: PRIMARY KEY

- Pour créer une contrainte **PRIMARY KEY** sur la colonne "**Id**" alors que la table Employes existe déjà, on utilise la syntaxe SQL suivante :

```
ALTER TABLE Employes
ADD PRIMARY KEY (ID);
```

- Pour supprimer les contraintes de clé primaire de la table, on utilise la syntaxe donnée ci-dessous.

```
ALTER TABLE Employes DROP PRIMARY KEY ;
```

Contraintes: FOREIGN KEY

- Une clé étrangère est une clé utilisée pour relier deux tables. Ceci est parfois appelé aussi clé de référencement.
- Une clé étrangère est une colonne ou une combinaison de colonnes dont les valeurs correspondent à une clé primaire dans une autre table.

```
CREATE TABLE Employes(
    Id NUMBER                NOT NULL,
    Nom VARCHAR (20)     NOT NULL,
    Age  NUMBER              NOT NULL,
    Salaire   DECIMAL (18, 2),
    PRIMARY KEY (Id)
);
```

```
CREATE TABLE Conges(
   IDc NUMBER                NOT NULL,
    Date_debut  DATE     NOT NULL,
    Date_fin  DATE       NOT NULL,
    ID_EMP  NUMBER,
    PRIMARY KEY (IDc),
    FOREIGN KEY (ID_EMP ) REFERENCES
Employes(Id)
);
```

Contraintes: Suppression

- Toute contrainte que vous avez définie peut être supprimée à l'aide de la commande **ALTER TABLE** avec l'option **DROP CONSTRAINT**.

```
ALTER TABLE Nom_table DROP CONSTRAINT nom_contrainte;
```

Modification de tables

- La commande **ALTER TABLE** permet d'ajouter/de modifier une ou plusieurs **colonnes** dans une tables existante. Idéal pour ajouter une colonne, supprimer une colonne ou modifier une colonne existante. Sa syntaxe est comme suit :

```
ALTER TABLE table ADD/MODIFY
(
attribut_1 [<type> <contraintes>],
...
attribut_n [<type> <contraintes>]
);
```

Modification de tables

Ajouter une colonne

- L'ajout d'une colonne dans une table est relativement simple et peut s'effectuer à l'aide de la requête ressemblant suivante :

```
ALTER TABLE nom_table
ADD nom_colonne type_donnees
```

Supprimer une colonne

- Il y a 2 manières totalement équivalente pour supprimer une colonne en suivant l'un des requêtes suivantes:

```
ALTER TABLE nom_table
DROP nom_colonne
```

ou

```
ALTER TABLE nom_table
DROP COLUMN nom_colonne
```

Modification de tables

Modifier une colonne

- Requête pour modifier une colonne, comme par exemple changer le type d'une colonne:

```
ALTER TABLE nom_table
MODIFY nom_colonne type_donnees
```

Renommer une colonne

- Pour renommer une colonne, il convient d'indiquer l'ancien nom de la colonne et le nouveau nom de celle-ci.

```
ALTER TABLE nom_table
RENAME COLUMN colonne_ancien_nom TO colonne_nouveau_nom;
```

Modification de tables

Gestion des contraintes :

- Pour ajouter une ou plusieurs contraintes dans une tables existante :

```
ALTER TABLE table ADD <contraintes>
```

- Pour ajouter ou desupprimer des valeurs par défaut dans une tables existante :

```
ALTER TABLE table MODIFY attribut SET valeur;

ALTER TABLE table MODIFY attribut DEFAULT NULL;
```

destruction de tables

- La commande DROP TABLE en SQL permet de supprimer définitivement une table d'une base de données. Cela supprime en même temps les éventuels index, trigger, contraintes et permissions associées à cette table.

- Pour supprimer une table "nom_table" il suffit simplement d'utiliser la syntaxe suivante :

```
DROP TABLE nom_table [CASCADE CONSTRAINTS];
```

L'option CASCADE CONSTRAINTS permet de supprimer les clés étrangères des autres tables qui pointent sur la table à supprimer.

Une base de données pour gérer les employés et leurs projets.

1.Table Employes :

Id (clé primaire, non null, nombre entier)
Nom (chaîne de caractères, 50 caractères maximum, non null)
Age (nombre entier, doit être supérieur ou égal à 18)
Salaire (nombre décimal avec deux décimales, par défaut 3000.00)

2.Table Projets :

Id_projet (clé primaire, non null, nombre entier)
Nom_projet (chaîne de caractères, 100 caractères maximum, unique)
Lieu
Date_debut (date, non null)
Date_fin (date, non null)

3.Table Employe_Projets :

Id_employe (clé étrangère faisant référence à **Employes.Id**)
Id_projet (clé étrangère faisant référence à **Projets.Id_projet**)
Role (chaîne de caractères, 30 caractères maximum, non null)

1. Crée les trois tables avec les colonnes et contraintes spécifiées..
2. Modifie le salaire d'un employé pour le rendre non null
3. Ajouter une colonne email dans la table Employes :.

Langage SQL :

La Manipulation des donnés

Insertion

- L'insertion de données dans une table s'effectue à l'aide de la commande INSERT INTO. Cette commande permet au choix d'inclure une seule ligne à la base existante ou plusieurs lignes d'un coup.

Insertion d'une ligne à la fois

Pour insérer des données dans une base, il y a 2 syntaxes principales :

- Insérer une ligne en indiquant les informations pour chaque colonne existante (en respectant l'ordre)
- Insérer une ligne en spécifiant les colonnes que vous souhaiter compléter. Il est possible d'insérer une ligne renseignant seulement une partie des colonnes

Insertion

Insérer une ligne en spécifiant toutes les colonnes

La syntaxe pour remplir une ligne avec cette méthode est la suivante :

```
INSERT INTO table VALUES ('valeur 1', 'valeur 2', ...)
```

Cette syntaxe possède les avantages et inconvénients suivants :

• Obliger de remplir toutes les données, tout en respectant l'ordre des colonnes

• Il n'y a pas le nom de colonne, donc les fautes de frappe sont limitées. Par ailleurs, les colonnes peuvent être renommées sans avoir à changer la requête

• L'ordre des colonnes doit resté identique sinon certaines valeurs prennent le risque d'être complétée dans la mauvaise colonne

Insertion

Insérer une ligne en spécifiant seulement les colonnes souhaitées

Cette deuxième solution est très similaire, excepté qu'il faut indiquer le nom des colonnes avant "VALUES". La syntaxe est la suivante :

```
INSERT INTO table (nom_colonne_1, nom_colonne_2, ... VALUES
('valeur 1', 'valeur 2', ...)
```

A noter : il est possible de ne pas renseigner toutes les colonnes. De plus, l'ordre des colonnes n'est pas important.

Insertion

Insertion de plusieurs lignes à la fois

Il est possible d'ajouter plusieurs lignes à un tableau avec une seule requête. Pour ce faire, il convient d'utiliser la syntaxe suivante :

```
INSERT INTO client (prenom, nom, ville, age) VALUES ('Rébecca',
'Armand', 'Saint-Didier-des-Bois', 24), ('Aimée', 'Hebert',
'Marigny-le-Châtel', 36), ('Marielle', 'Ribeiro', 'Maillères', 27),
('Hilaire', 'Savary', 'Conie-Molitard', 58);
```

A noter : lorsque le champ à remplir est de type VARCHAR ou TEXT il faut indiquer le texte entre guillemet simple. En revanche, lorsque la colonne est un numérique tel que INT ou BIGINT il n'y a pas besoin d'utiliser de guillemet, il suffit juste d'indiquer le nombre.

Insertion

Exemples :

```
INSERT INTO     CENTRE (NUMC, NOMC, VILC, COUTINSC)
VALUES ( ‘103’, ’Pleine Forme’, ‘PARIS’, 400);

INSERT INTO     CENTRE
VALUES ( ‘107’, ’Pleine Forme’, ‘PARIS’, 420);

INSERT INTO     CENTRE (NUMC, NOMC, COUTINSC)
VALUES ( ‘111’, ’Le nouveau’, 400);

INSERT INTO     PROPOSE (NUMC, SPORT)
VALUES ( ‘103’, ’FOOTBALL’);
```

Modification

- La commande UPDATE permet d'effectuer des modifications sur des lignes existantes. Très souvent cette commande est utilisée avec WHERE pour spécifier sur quelles lignes doivent porter la ou les modifications.

```
UPDATE    table
SET       attribut_1          =       valeur_1,
          attribut_2          =       valeur_2,
          …
          attribut_n          =       valeur_n
[WHERE    <condition>] ;
```

- Cette syntaxe permet d'attribuer une nouvelle valeur à la colonne nom_colonne_1 pour les lignes qui respectent la condition stipulé avec WHERE. Il est aussi possible d'attribuer la même valeur à la colonne nom_colonne_1 pour toutes les lignes d'une table si la condition WHERE n'était pas utilisée.

Modification

Exemples :

- **Augmenter le coût d' inscription, pour tous les centres, de 5% :**

```
UPDATE CENTRE
SET   COUTINSC = COUTINSC * 105%;
```

- **Baisser le coût d' inscription, dans le centre 101, de 20% :**

```
UPDATE CENTRE
SET   COUTINSC = COUTINSC * 80%
WHERE NUMC = '101';
```

Suppression

La commande DELETE en SQL permet de supprimer des lignes dans une table. En utilisant cette commande associé à WHERE il est possible de sélectionner les lignes concernées qui seront supprimées.

La syntaxe pour supprimer des lignes est la suivante :

```
DELETE FROM `table` WHERE condition
```

- **Supprimer une ligne**
  - Il est possible de supprimer une ligne en effectuant la requête SQL suivante :

```
DELETE FROM `utilisateur` WHERE `id` = 1
```

Suppression

Supprimer plusieurs lignes

Si l'ont souhaite supprimer les utilisateurs qui se sont inscrit avant le **10/04/2012**, il va falloir effectuer la requête suivante :

```
DELETE FROM `utilisateur`
WHERE `date_inscription` < '2012-04-10'
```

Supprimer toutes les données

Pour supprimer toutes les lignes d'une table il convient d'utiliser la commande DELETE sans utiliser de clause conditionnelle.

```
DELETE FROM `utilisateur`
```

Suppression

Supprimer toutes les données : DELETE ou TRUNCATE

- Pour supprimer toutes les lignes d'une table, il est aussi possible d'utiliser la commande TRUNCATE, de la façon suivante :

```
TRUNCATE TABLE `utilisateur`
```

- Cette requête est similaire. La différence majeure étant que la commande TRUNCATE va réinitialiser l'auto-incrémente s'il y en a un. Tandis que la commande DELETE ne réinitialise pas l'auto-incrément.

Sélection

L'utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s'effectue grâce à la commande SELECT, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table.

SELECT	*Je veux une telle chose*
FROM	*de cette source*
WHERE	*Ayant telles caractéristiques*
ORDER BY	*avec cette ordre*

Sélection

Syntaxe :

```
SELECT nom_du_champ
FROM nom_du_tableau
```

- **Obtenir plusieurs colonnes :**

```
SELECT nom_du_champ_1, nom_du_champ_2
FROM nom_du_tableau
```

- **Obtenir toutes les colonnes d'un tableau**

```
SELECT * FROM client
```

Sélection

Exemple :

Table « client » :

identifiant	prenom	nom	ville
1	Pierre	Dupond	Paris
2	Sabrina	Durand	Nantes
3	Julien	Martin	Lyon
4	David	Bernard	Marseille
5	Marie	Leroy	Grenoble

Si l'ont veut avoir la liste de toutes les villes des clients, il suffit d'effectuer la requête suivante :

```
SELECT ville FROM client
```

ville
Paris
Nantes
Lyon
Marseille
Grenoble

Sélection

L'utilisation de la commande SELECT en SQL permet de lire toutes les données d'une ou plusieurs colonnes. Cette commande peut potentiellement afficher des lignes en doubles. Pour éviter des redondances dans les résultats il faut simplement ajouter DISTINCT après le mot SELECT.

L'utilisation basique de cette commande consiste alors à effectuer la requête suivante :

```
SELECT DISTINCT ma_colonne
FROM nom_du_tableau
```

Cette requête sélectionne le champ « ma_colonne » de la table « nom_du_tableau » en évitant de retourner des doublons.

- **WHERE**

La commande WHERE dans une requête SQL permet d'extraire les lignes d'une base de données qui respectent une condition. Cela permet d'obtenir uniquement les informations désirées.

La commande WHERE s'utilise en complément à une requête utilisant SELECT. La façon la plus simple de l'utiliser est la suivante :

```
SELECT nom_colonnes FROM nom_table
WHERE condition
```

- **ORDER BY**

La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.

```
SELECT colonne1, colonne2
 FROM table
ORDER BY colonne1
```

Par défaut les résultats sont classés par ordre ascendant, toutefois il est possible d'inverser l'ordre en utilisant le suffixe DESC après le nom de la colonne.

```
SELECT colonne1, colonne2
 FROM table
ORDER BY colonne1 DESC
```

- **ORDER BY**

Par ailleurs, il est possible de trier sur plusieurs colonnes en les séparant par une virgule.

```
SELECT colonne1, colonne2, colonne3
FROM table
ORDER BY colonne1 DESC, colonne2 ASC
```

- **AND & OR**

Une requête SQL peut être restreinte à l'aide de la condition WHERE. Les opérateurs logiques AND et OR peuvent être utilisées au sein de la commande WHERE pour combiner des conditions

Les opérateurs sont à ajoutés dans la condition WHERE. Ils peuvent être combinés à l'infini pour filtrer les données comme souhaités.

L'opérateur AND permet de s'assurer que la condition1 ET la condition2 sont vrai :

Sélection

- **AND & OR**
- L'opérateur AND permet de s'assurer que la condition1 ET la condition2 sont vrai :

```
SELECT nom_colonnes FROM nom_table
WHERE condition1 AND condition2
```

- L'opérateur OR vérifie quant à lui que la condition1 OU la condition2 est vrai :

```
SELECT  nom_colonnes  FROM  nom_table
WHERE condition1 OR condition2
```

Sélection

De nombreux operateurs et fonctions peuvent être utilisés pour définir une condition, on peut distinguer trois types:

- **Comparaison** entre attributs(colonnes) et/ou valeurs, au moins une par condition, doit être vraie ou fausse pour chaque ligne(n-uplet)
- **Transformation** des attributs avant comparaison
- **Combinaison** logique de comparaisons

Sélection : opérateurs de comparaison

- **Opérateurs de comparaisons :** Ces operateurs peuvent être utilisés pour comparer des valeurs numériques, des dates ou des chaines de caractères

Opérateur	Description
=	Égale
<>	Pas égale
!=	Pas égale
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donnée (utile pour les nombres ou dates)
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot.
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

Sélection : opérateurs de comparaison

- **LIKE**

- L'opérateur LIKE permet d'effectuer une comparaison entre une chaine de caractère et un modèle particulier:
    - **LIKE '%a'** : se termine par un «a»
    - **LIKE 'a%'** : commence par un «a»
    - **LIKE '%a%'** : contient le caractère «a» (n'importe où)
    - **LIKE 'pa%on'** : commence par «pa», se termine par «on»
    - **LIKE 'a_c'** : contient une seule lettre entre «a» et «c»

- «%» remplace tous les autres caractères
- •«_» remplace un caractère uniquement
- •LIKE est sensible à la casse

Sélection : opérateurs de comparaison

- **BETWEEN**

L'opérateur BETWEEN est utilisé dans une requête SQL pour sélectionner un intervalle de données dans une requête utilisant WHERE. L'intervalle peut être constitué de chaînes de caractères, de nombres ou de dates. L'exemple le plus concret consiste par exemple à récupérer uniquement les enregistrements entre 2 dates définies.

```
SELECT * FROM table
WHERE nom_colonne BETWEEN 'valeur1' AND 'valeur2'
```

Sélection : opérateurs de comparaison

- **BETWEEN**

id	nom	date_inscription
1	Maurice	2012-03-02
2	Simon	2012-03-05
3	Chloé	2012-04-14
4	Marie	2012-04-15
5	Clémentine	2012-04-26

- Quelle requête pour obtenir les membres qui se sont inscrit entre le 1 avril 2012 et le 20 avril 2012

Sélection : opérateurs de comparaison

- **IN**

L'opérateur logique IN dans SQL s'utilise avec la commande WHERE pour vérifier si une colonne est égale à une des valeurs comprise dans set de valeurs déterminés. C'est une méthode simple pour vérifier si une colonne est égale à une valeur OU une autre valeur OU une autre valeur et ainsi de suite, sans avoir à utiliser de multiple fois l'opérateur OR.

```
SELECT nom_colonne FROM table
WHERE nom_colonne IN ( valeur1, valeur2, valeur3, ... )
```

Sélection : opérateurs de comparaison

- **IN**

Soit cette requête :

```
SELECT prenom FROM utilisateur
WHERE prenom = 'Maurice' OR prenom = 'Marie'
OR prenom = 'Thimoté'
```

Requête équivalent avec l'opérateur IN :

```
SELECT prenom FROM utilisateur
WHERE prenom IN ( 'Maurice', 'Marie', 'Thimoté')
```

Sélection : opérateurs de comparaison

- **Exemple**

id	id_utilisateur	addr_rue	addr_code_postal	addr_ville
1	23	35 Rue Madeleine Pelletier	25250	Bournois
2	43	21 Rue du Moulin Collet	75006	Paris
3	65	28 Avenue de Cornouaille	27220	Mousseaux-Neuville
4	67	41 Rue Marcel de la Provoté	76430	Graimbouville
5	68	18 Avenue de Navarre	75009	Paris

Quelle requête doit on utiliser si on souhaite obtenir les enregistrements des adresses de Paris ou de Graimbouville

Sélection : opérateurs de comparaison

- **Exemple**

id	nom	categorie	stock	prix
1	ordinateur	informatique	5	950
2	clavier	informatique	32	35
3	souris	informatique	16	30
4	crayon	fourniture	147	2

- **SELECT * FROM produit WHERE categorie = 'informatique' AND stock < 20**
- **SELECT * FROM produit WHERE nom = 'ordinateur' OR nom = 'clavier'**
- **SELECT * FROM produit WHERE (categorie = 'informatique' AND stock < 20) OR (categorie = 'fourniture' AND stock < 200)**

Sélection : Transformation numérique

Operateurs de transformation de valeurs numériques:

- + : addition
- -: soustraction
- * : multiplication
- / : division

Exemple :

```
SELECT* FROM produits WHERE prix_vente > prix_achat + prix_achat* 20/100;
```

Operateurs de transformation de chaines de caractères: ||

```
SELECT * FROM clients WHERE nom||prenom LIKE '%David%';
```

Sélection : Transformation numérique

Fonctions de transformation de valeurs numériques :

- SQRT(N) : racine carrée de N
- ABS (N) : valeur absolue de N
- ROUND (N), FLOOR (N), CEILING (N) : arrondi de N, entier inf, entier sup
- POW(N, P), EXP (N) : A puissance P, exponentiel de N
- …

```
SELECT * FROM produits WHERE ABS (gain) > SQRT (prix_achat );
```

Sélection : Transformation numérique

Fonctions de transformation de chaines de caractères:

- LOWER(C) : met C en minuscule
- UPPER (C) : met C en majuscule
- SOUNDEX (C) : transforme C en phonétique
- LENGTH (C) : calcule le nombre de caractères de C
- TO_CHAR (N) : transforme le nombre N en caractères
- …

```
SELECT * FROM clients WHERE LOWER (Nom) = 'dupuis';
```

Sélection : Agrégation

Les fonctions d'agrégation dans le langage SQL permettent d'effectuer des opérations statistiques sur un ensemble d'enregistrement. Étant données que ces fonctions s'appliquent à plusieurs lignes en même temps, elle permettent des opérations qui servent à récupérer l'enregistrement le plus petit, le plus grand ou bien encore de déterminer la valeur moyenne sur plusieurs enregistrement.

Exemple de fonctions d'agrégat :

- Moyenne
- Somme
- Maximum
- Comptage
- Minimum

Sélection : Agrégation

- **SUM(attribut)** : total des valeurs (numériques) d'un attribut
- **AVG(attribut)** : moyenne des valeurs (num) d'un attribut
- **MIN(attribut)** : plus petite valeur (num, date) d'un attribut
- **MAX(attribut)** : plus grande valeur (num, date) d'un attribut
- **COUNT(*)** : nombre de lignes
- **COUNT(DISTINCT attribut)** : nombre de valeurs différentes de l'attribut

Remarque : les valeurs NULL sont ignorées.

Sélection : Agrégation

- La fonction d'agrégat s'utilise dans la clause SELECT à la place d'un nom de colonne.

Exemples:

- Prix moyen des articles : `SELECT AVG(prix) FROM articles ;`
- Note maximale : `SELECT MAX(note) FROM resultats ;`
- Nombre d'étudiants : `SELECT COUNT(*) FROM etudiants ;`

• Regroupement des lignes : GROUP BY

- Il est possible de calculer l'agrégation non pas sur toutes les lignes mais par catégories.

Syntaxe:

```
SELECT fonction_agregation FROM nom_table
GROUP BY liste_attributs;
```

- Les lignes sont regroupées si elles ont les mêmes valeurs sur le ou les attributs de la clause GROUP BY.

Sélection : Agrégation

Exemple :

Table Achat

Commande	CodeProduit	Quantité	Prix
96008	A10	10	83
96008	B20	35	32
96009	A10	20	83
96010	A15	4	110
96010	B20	55	32

```
SELECT Commande, SUM(Quantité*Prix) AS Montant FROM Achat
GROUP BY Commande ;
```

Commande	Montant
96008	1950
96009	1660
96010	2200

Sélection : Agrégation

Exemple :

ID_client	ID_fournisseur	Prix
C1	F1	25
C1	F2	35
C2	F1	64
C2	F1	120
C2	F3	26

```
SELECT ID_client, ID_fournisseur, MAX(Prix) AS Maxi FROM Commandes
GROUP BY ID_client, ID_fournisseur ;
```

ID_client	ID_fournisseur	Maxi
C1	F1	25
C1	F2	35
C2	F1	120
C2	F3	26

• Conditions sur les regroupements: HAVING

- Il est aussi possible de ne sélectionner que certains des groupes, avec la clause HAVING.

```
SELECT fonction_agregation FROM nom_table
GROUP BY liste_attributs HAVING conditions ;
```

- La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX().

Sélection : Agrégation

Exemple :

id	client	tarif
1	Pierre	102
2	Simon	47
3	Marie	18
4	Marie	20
5	Pierre	160

Requête si on souhaite récupérer la liste des clients qui ont commandé plus de 40 ?

```
SELECT client, SUM(tarif) FROM Commandes
GROUP BY client HAVING SUM(tarif) > 40 ;
```

client	SUM(tarif)
Pierre	262
Simon	47

Sélection : Agrégation

Exemple :

Commande	CodeProduit	Quantité	Prix
96008	A10	10	83
96008	B20	35	32
96009	A10	20	83
96010	A15	4	110
96010	B20	55	32

Requête si on souhaite récupérer la liste des commandes qui ont un total de plus que 1700 ?

```
SELECT Commande, SUM(Quantité*Prix) AS Montant FROM Commandes
GROUP BY Commande HAVING Montant > 1700
ORDER BY Montant DESC ;
```

Commande	Montant
96010	2200
96008	1950

Langage SQL :

La Manipulation des donnés : JOINTURES

Jointures

Les jointures en SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.

En général, les jointures consistent à associer des lignes de 2 tables en associant l'égalité des valeurs d'une colonne d'une première table par rapport à la valeur d'une colonne d'une seconde table. Imaginons qu'une base de 2 données possède une table "utilisateur" et une autre table "adresse" qui contient les adresses de ces utilisateurs. Avec une jointure, il est possible d'obtenir les données de l'utilisateur et de son adresse en une seule requête.

Produit cartésien

- Le produit cartésien: construit une relation regroupant exclusivement toutes les possibilités de combinaison des occurrences de deux relations qui satisfont une expression logique E.
  - Le résultat du produit cartésien est une nouvelle relation qui a tous les attributs de R_1 et tous ceux de R_2.
  - Le nombre d'occurrences de la relation qui résulte du produit cartésien est le nombre d'occurrences de R_1 multiplié par le nombre d'occurrences de R_2

L'union

L'union est une opération portant sur deux relations ayant le même schéma et construisant une troisième constituée des n-uplets appartenant à chacune des deux relations sans doublon.

- R_1et R_2doivent avoir les mêmes attributs.
- Si une même occurrence existe dans R_1 et R_2, elle n'apparaît qu'une seule fois dans le résultat de l'union.
- Le résultat de l'union est une nouvelle relation qui a les mêmes attributs que R_1et R_2.
- Si R_1(respectivement R_2) est vide, la relation qui résulte de l'union est identique à
- R_2(respectivement R_1).

L'union

Exemple :

R_1	
Nom	**Prénom**
Durand	Caroline
Germain	Stan
Dupont	Lisa
Germain	Rose-Marie

R_2	
Nom	**Prénom**
Dupont	Lisa
Juny	Carole
Fourt	Lisa

$R = R1 \cup R2$	
Nom	**Prénom**
Durand	Caroline
Germain	Stan
Dupont	Lisa
Germain	Rose-Marie
Juny	Carole
Fourt	Lisa

L'intersection

- L'intersection est une opération portant sur deux relations ayant le même schéma et construisant une troisième dont les n-uplets sont constitués de ceux communs aux deux relations.
  - R_1et R_2 doivent avoir les mêmes attributs.
  - Le résultat de l'intersection est une nouvelle relation qui a les mêmes attributs que R_1et R_2.
  - Si R_1ou R_2ou les deux sont vides, la relation qui résulte de l'intersection est vide.

L'intersection

Exemple :

Relation R_1	
Nom	**Prénom**
Durand	Caroline
Germain	Stan
Dupont	Lisa
Germain	Rose-Marie
Juny	Carole

Relation R_2	
Nom	**Prénom**
Dupont	Lisa
Juny	Carole
Fourt	Lisa
Durand	Caroline

Relation $R = R_1 \cap R_2$	
Nom	**Prénom**
Durand	Caroline
Dupont	Lisa
Juny	Carole

La Différence

La différence est une opération portant sur deux relations ayant le même schéma et construisant une troisième relation dont les n-uplets sont constitués de ceux ne se trouvant que dans la relation R_1.

- R_1et R_2 doivent avoir les mêmes attributs.
- Le résultat de la différence est une nouvelle relation qui a les mêmes attributs que R_1et R_2.
- Si R_2 est vide, la différence est identique à R_1.

La Différence

Exemple :

Relation R_1	
Nom	**Prénom**
Durand	Caroline
Germain	Stan
Dupont	Lisa
Germain	Rose-Marie
Juny	Carole

Relation R_2	
Nom	**Prénom**
Dupont	Lisa
Juny	Carole
Fourt	Lisa
Durand	Caroline

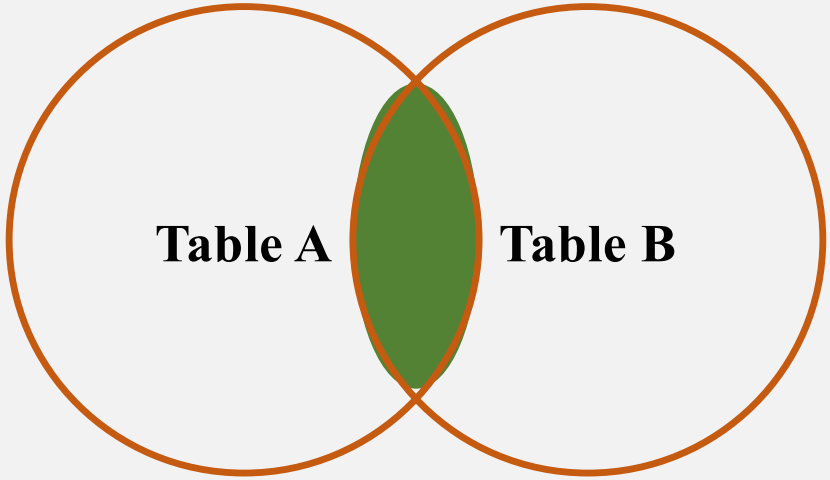
Relation $R = R_1 - R_2$	
Nom	**Prénom**
Germain	Stan
Germain	Rose-Marie

SQL INNER JOIN

Dans le langage SQL la commande INNER JOIN, aussi appelée EQUIJOIN, est un type de jointures très communes pour lier plusieurs tables entre-elles. Cette commande retourne les enregistrements lorsqu'il y a au moins une ligne dans chaque colonne qui correspond à la condition.

```
SELECT * FROM table1 INNER JOIN table2
      ON table1.id1 = table2.id2
```

- **ON**: La clause ON correspond à la condition de jointure la plus générale.



SQL INNER JOIN

Table utilisateur :

id	prenom	nom	email	ville
1	Aimée	Marechal	aime.marechal@example.com	Paris
2	Esmée	Lefort	esmee.lefort@example.com	Lyon
3	Marine	Prevost	m.prevost@example.com	Lille
4	Luc	Rolland	lucrolland@example.com	Marseille

- **Comment afficher toutes les commandes associées aux utilisateurs ?**

Table commande :

utilisateur_id	date_achat	num_facture	prix_total
1	2013-01-23	A00103	203.14
1	2013-02-14	A00104	124.00
2	2013-02-17	A00105	149.45
2	2013-02-21	A00106	235.35
5	2013-03-02	A00107	47.58

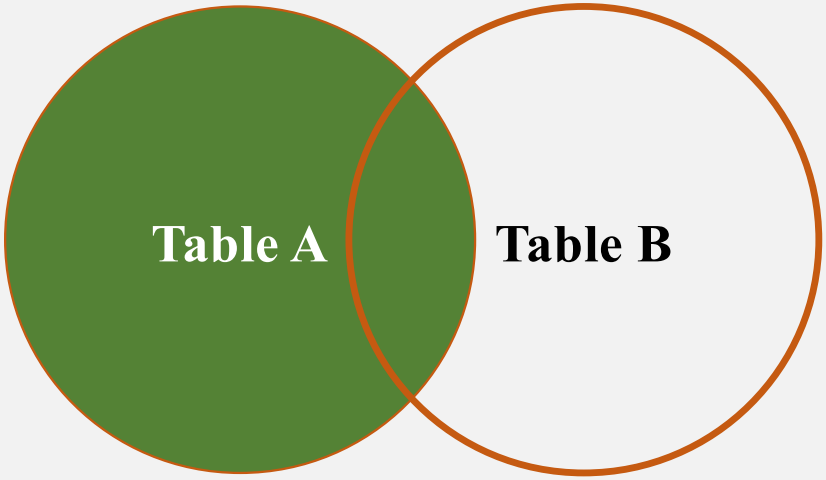
SQL INNER JOIN

```
SELECT id, prenom, nom, date_achat, num_facture, prix_total
FROM utilisateur
INNER JOIN commande
ON utilisateur.id = commande.utilisateur_id
```

id	prenom	nom	date_achat	num_facture	prix_total
1	Aimée	Marechal	2013-01-23	A00103	203.14
1	Aimée	Marechal	2013-02-14	A00104	124.00
2	Esmée	Lefort	2013-02-17	A00105	149.45
2	Esmée	Lefort	2013-02-21	A00106	235.35

SQL LEFT JOIN

La commande LEFT JOIN (aussi appelée LEFT OUTER JOIN) est un type de jointure entre 2 tables. Cela permet de lister tous les résultats de la table de gauche même s'il n'y a pas de correspondance dans la deuxième tables.



SQL LEFT JOIN

Syntaxe : Pour lister les enregistrement de table1, même s'il n'y a pas de correspondance avec table2, il convient d'effectuer une requête SQL utilisant la syntaxe suivante.

```
SELECT *
FROM table1
LEFT JOIN table2 ON table1.id = table2.fk_id
```

SQL LEFT JOIN

Table utilisateur :

id	prenom	nom	email	ville
1	Aimée	Marechal	aime.marechal@example.com	Paris
2	Esmée	Lefort	esmee.lefort@example.com	Lyon
3	Marine	Prevost	m.prevost@example.com	Lille
4	Luc	Rolland	lucrolland@example.com	Marseille

Requête Pour lister tous les utilisateurs avec leurs commandes et afficher également les utilisateurs qui n'ont pas effectuées d'achats ?

Table commande :

utilisateur_id	date_achat	num_facture	prix_total
1	2013-01-23	A00103	203.14
1	2013-02-14	A00104	124.00
2	2013-02-17	A00105	149.45
2	2013-02-21	A00106	235.35
5	2013-03-02	A00107	47.58

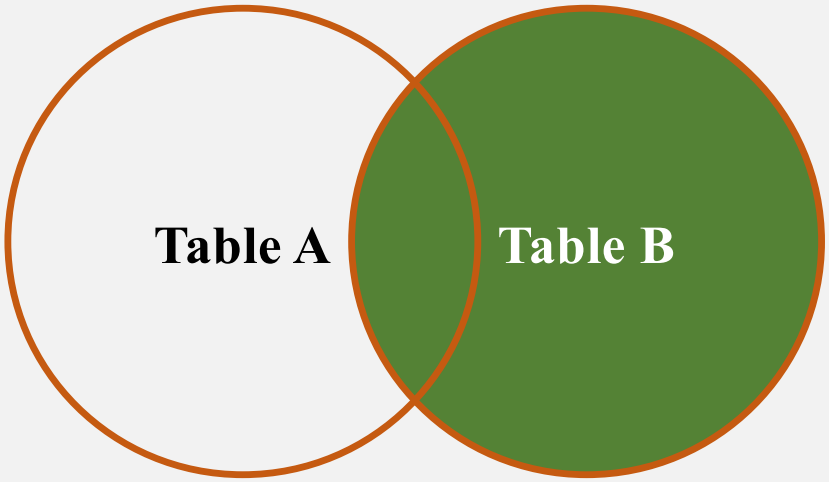
SQL LEFT JOIN

```
SELECT *
FROM utilisateur
LEFT JOIN commande ON utilisateur.id = commande.utilisateur_id
```

id	prenom	nom	date_achat	num_facture	prix_total
1	Aimée	Marechal	2013-01-23	A00103	203.14
1	Aimée	Marechal	2013-02-14	A00104	124.00
2	Esmée	Lefort	2013-02-17	A00105	149.45
2	Esmée	Lefort	2013-02-21	A00106	235.35
3	Marine	Prevost	NULL	NULL	NULL
4	Luc	Rolland	NULL	NULL	NULL

SQL RIGHT JOIN

La commande RIGHT JOIN (ou RIGHT OUTER JOIN) est un type de jointure entre 2 tables qui permet de retourner tous les enregistrements de la table de droite (right = droite) même s'il n'y a pas de correspondance avec la table de gauche. S'il y a un enregistrement de la table de droite qui ne trouve pas de correspondance dans la table de gauche, alors les colonnes de la table de gauche auront NULL pour valeur.



SQL RIGHT JOIN

Syntaxe pour lister toutes les lignes du tableau **table2** (tableau de droite) et afficher les données associées du tableau **table1** s'il y a une correspondance entre **ID** de **table1** et **FK_ID** de **table2**. S'il n'y a pas de correspondance, l'enregistrement de **table2** sera affiché et les colonnes de **table1**vaudront toutes NULL.

```
SELECT *
FROM table1
RIGHT JOIN table2 ON table1.id = table2.fk_id
```

SQL RIGHT JOIN

Table utilisateur :

id	prenom	nom	email	ville
1	Aimée	Marechal	aime.marechal@example.com	Paris
2	Esmée	Lefort	esmee.lefort@example.com	Lyon
3	Marine	Prevost	m.prevost@example.com	Lille
4	Luc	Rolland	lucrolland@example.com	Marseille

Requête Pour lister tous les achats et afficher le nom d'utilisateurs s'il existe?

utilisateur_id	date_achat	num_facture	prix_total
1	2013-01-23	A00103	203.14
1	2013-02-14	A00104	124.00
2	2013-02-17	A00105	149.45
3	2013-02-21	A00106	235.35
5	2013-03-02	A00107	47.58

SQL RIGHT JOIN

```
SELECT id, prenom, nom, utilisateur_id, date_achat, num_facture
FROM utilisateur
RIGHT JOIN commande ON utilisateur.id = commande.utilisateur_id
```

id	prenom	nom	utilisateur_id	date_achat	num_facture
1	Aimée	Marechal	1	2013-01-23	A00103
1	Aimée	Marechal	1	2013-02-14	A00104
2	Esmée	Lefort	2	2013-02-17	A00105
3	Marine	Prevost	3	2013-02-21	A00106
NULL	NULL	NULL	5	2013-03-02	A00107

SQL FULL JOIN

Dans le langage SQL, la commande FULL JOIN (ou FULL OUTER JOIN) permet de faire une jointure entre 2 tables. L'utilisation de cette commande permet de combiner les résultats des 2 tables, les associer entre eux grâce à une condition et remplir avec des valeurs NULL si la condition n'est pas respectée.

```
SELECT *
FROM table1
FULL JOIN table2 ON table1.id = table2.fk_id
```

SQL FULL JOIN

Table Commande

utilisateur_id	date_achat	num_facture	prix_total
1	2013-01-23	A00103	203.14
1	2013-02-14	A00104	124.00
2	2013-02-17	A00105	149.45
3	2013-02-21	A00106	235.35
5	2013-03-02	A00107	47.58

Table Utilisateur

id	prenom	nom	email	ville	actif
1	Aimée	Marechal	aime.marechal@example.com	Paris	1
2	Esmée	Lefort	esmee.lefort@example.com	Lyon	0
3	Marine	Prevost	m.prevost@example.com	Lille	1
4	Luc	Rolland	lucrolland@example.com	Marseille	1

Requête pour lister tous les utilisateurs ayant effectué ou non une vente, et de lister toutes les ventes qui sont associées ou non à un utilisateur.

SQL FULL JOIN

```
SELECT id, prenom, nom, utilisateur_id, date_achat, num_facture
 FROM utilisateur
FULL JOIN commande ON utilisateur.id = commande.utilisateur_id
```

id	prenom	nom	utilisateur_id	date_achat	num_facture
1	Aimée	Marechal	1	2013-01-23	A00103
1	Aimée	Marechal	1	2013-02-14	A00104
2	Esmée	Lefort	2	2013-02-17	A00105
3	Marine	Prevost	3	2013-02-21	A00106
4	Luc	Rolland	NULL	NULL	NULL
NULL	NULL	NULL	5	2013-03-02	A00107

SQL NATURAL JOIN

La commande NATURAL JOIN permet de faire une jointure naturelle entre 2 tables. Cette jointure s'effectue à la condition qu'il y ai des colonnes du **même nom et de même type** dans les 2 tables. Le résultat d'une jointure naturelle est la création d'un tableau avec autant de lignes qu'il y a de paires correspondant à l'association des colonnes de même nom.

```
SELECT *
FROM table1
NATURAL JOIN table2
```

* NATURAL JOIN c'est qu'il n'y a pas besoin d'utiliser la clause ON

SQL NATURAL JOIN

Table Utilisateurs

pays_id	user_id	user_prenom	user_ville	pays_nom
1	1	Jérémie	Paris	France
2	2	Damien	Montréal	Canada
NULL	3	Sophie	Marseille	NULL
9999	4	Yann	Lille	NULL
1	5	Léa	Paris	France

Table Pays

pays_id	pays_nom
1	France
2	Canada
3	Belgique
4	Suisse

SQL NATURAL JOIN

```
SELECT *
FROM utilisateur
NATURAL JOIN pays
```

user_id	user_prenom	user_ville	pays_id
1	Jérémie	Paris	1
2	Damien	Montréal	2
3	Sophie	Marseille	NULL
4	Yann	Lille	9999
5	Léa	Paris	1