Bases de Données Avancées 1

Abdoul Aziz Ciss
GIT / EPT

Introduction

L'objectif de ce cours est d'étudier les principes des bases de données (relationnelles) et la mise en pratique de celles-ci.

A l'issue de ce cours, l'apprenant doit être capable de :

- Concevoir et implémenter un base de données relationnelle ;
- Expliquer les opérations sur les tables d'une BD ;
- Exploiter une base de données avec le langage SQL ;
- Écrire des programmes BD efficaces avec du PL/SQL

Contenu

1ère Partie

1. Concepts Bases de données et SGBD
2. Modèle relationnel & Algèbre relationnelle
3. Langage SQL

2ème Partie

1. Bases de la programmation avec PL/SQL
2. Interactions avec le serveur Oracle
3. Curseurs explicites
4. Gestion des exceptions
5. Procédures et fonctions stockées / Packages
6. Triggers

Concepts de Bases de données et SGBD

BD : Définition

Définition : une base de données est un ensemble structuré de données (a) stockées sur des supports accessibles par l'ordinateur (b) pour satisfaire plusieurs utilisateurs simultanément(c) de manière sélective (d) en un temps opportun (e).

- (a) organisation et description des données
- (b) stockage sur disque
- (c) partage des données
- (d) autorisastion, confidentialité des données
- (e) performance, efficacité

SGBD

Définition: un système de gestion de bases de données (SGBD) est un logiciel de haut niveau qui permet de manipuler les informations stockées dans une base de données.

Un SGBD permet de créer et de maintenir une base de données

- Définition d'une base de données (modélisation, spécification des données à stocker)
- Construction de la base de données (stockage effectif des données)
- Manipulation des données (interrogation, mise à jour, recherche, etc.)

Exemples : Oracle, MySQL, PostgreSQL, Microsoft SQL Server, Redis, MongoDB, CouchDB, Cassandra, etc.

Fonctions d'un SGBD

- Définir des contraintes d'intégrité sur une BD :
  - ✓ contraintes de domaines
  - ✓ contraintes d'existence
  - ✓ contraintes d'unicité
  - ✓ etc.
- Définir des protections d'accès :
  - ✓ création de comptes, authentification
  - ✓ autorisations d'accès
- Résoudre les problèmes d'accès multiples
  - ✓ blocages
  - ✓ interblocages
- Reprise en cas de pannes
  - ✓ sauvegarde / restauration des données
  - ✓ journaux des transactions en ligne

Architecture d'un SGBD

- Les BD reposent sur la séparation effective des données et des programmes qui les exploitent.
- Un SGBD doit fournir un langage de définition des données (LDD) et un langage de manipulation des données (LMD).
- Un LDD permet de définir l'organisation des données au niveau logique (conceptuel, abstrait). L'organisation physique des données est prise en charge par le SGBD.

Modèle Relationnel

Modèles de données

Un modèle de données est un moyen de representer logiquement les données.

Exemple

- ✓ Fichiers plats : accès séquentiel
- ✓ Modèle hiérarchique : modèle en arborescence
- ✓ Modèle réseau : extension du modèle hiérarchique (liens, arborescence)
- ✓ Modèle relationnel: modèle en tables ou relations
- ✓ Modèle objet : basé sur les objets

Vocabulaire des BDR

- ✓ Dans le modèle relationnel, les données sont représentées logiquement sous forme de tables (appelées également relations).
- ✓ Un SGBD relationnel (SGBDR) est un SGBD fondé sur le modèle relationnel.
- ✓ Exemples : MySQL, Oracle, SQLServer
- ✓ Une BD relationnelle (BDR) est un ensemble de relations.
- ✓ Une relation est représentée sous forme de table (ou tableau) à deux dimensions qui respectent certaines conditions.
- ✓ Dans une BDR, chaque table est identifiée par un nom unique.

Vocabulaire des BDR

- ✓ Les lignes ou les tuples d'une table sont également appelées des enregistrements.
- ✓ Les enregistrements d'une table ont le même format et représentent des objets du monde réel.
- ✓ Le nombre total des enregistrements présents dans une table est appelé cardinalité de la table.
- ✓ Le degré d'une table est le nombre de ses attributs

Vocabulaire des BDR

Règles d'or à retenir pour une BDR:

- ✓Une table est formée de colonnes et de lignes
- ✓Chaque table a un nom unique
- ✓Chaque colonne d'une table a un nom unique
- ✓L'ordre des colonnes dans la table est sans aucune importance
- ✓Toutes les lignes d'une table ont le même format et le même nombre d'entrées.

Vocabulaire des BDR

Règles d'or à retenir pour une BDR (suite)

- ✓ Les valeurs de chaque colonne appartiennent au même domaine
- ✓ Chaque entrée d'une table doit être une valeur unique
- ✓ L'ordre des lignes est sans importance

Formalisme mathématique

Les données d'une table apparaissent comme un ensemble de lignes ou n-tuples (n-uplets), où n est le degré de la table.

Exemple

Code produit	Libellé	Prix unitaire	Qté Stock
DD120	Disque dur 120 Go	30000	23
CL102	Clavier 102	1500	58
CU1G	Clé USB 1Go	5000	79

Formalisme mathématique

Pour une relation **R** donnée, pour un attribut **A** de **R** et un tuple **t** de **R** , on note **t(A)** la valeur de l'entrée du tuple t qui se trouve à la colonne A.

Exemple: considérons le second tuple de la relation 'Produit'. Alors:

- ✓ **t(Code Produit) = CL102;**
- ✓ **t(Libellé) = Clavier 102;**
- ✓ **t(Prix unitaire) = 1500 ;**
- ✓ **t(Quantité) =58 ;**

Formalisme mathématique

Un schéma relationnel **R** est un ensemble fini de noms d'attributs $A_1, A_2, \ldots, A_n$. On a $R = \{A_1, A_2, \ldots, A_n\}$

A chaque attribut A_i est associé un ensemble de valeurs appelées domaine de l'attribut A_i et noté $D_i = Dom(A_i)$

Soit **D** le produit cartésien des ensembles D_i,

ie $D = D_1 \times D_2 \times \ldots \times D_n$.

Une relation **r** sur le schéma relationnel **R** est un ensemble fini de correspondances $r = \{t_1, t_2, \ldots, t_m\}$ de **R** sur **D**.

Formalisme mathématique

- Une correspondance individuelle t_k est appelée tuple (ou n-uplet).

- Pour tout tuple **t** appartenant à la relation, on note $t(A_i)$ la valeur de l'attribut A_i pour le tuple t.

- Pour tout élément **t** de **r**, on a $t(A_i) \in D_i$

- La cardinalité de **r** est **card(r) = m**.

- Le degré de **r** est **deg(r) = n**

Notion de clé primaire

- ✓ La notion de clé primaire est un concept fondamental du modèle relationnel. Elle fournit le mécanisme de base pour récupérer des tuples dans une table d'une BD.
- ✓ La clé primaire d'une relation est le plus petit sous-ensemble des attributs qui permet d'identifier chaque ligne de manière unique.
- ✓ Un SGBD n'autorise qu'une seule clé primaire par table
- ✓ La clé primaire peut être unique si elle comporte un seul attribut ou composée si elle contient au moins deux attributs.
- ✓ Les attributs de la clé primaire sont soulignés pour les distinguer des autres attributs ne faisant pas partie de la clé.

Notion de clé primaire

- Les clés primaires sont définies au moyen d'instruction LDD et sont automatiquement imposées par le SGBDR.
- Les clés primaires sont généralement définies au moment de la création des tables.
- Les clés primaires peuvent être naturelles ou artificielles.
- Dans une table contenant des données descriptives de cours, le nom des modules serait une clé primaire naturelle.

Notion de clé primaire

- Les clés sont des éléments très importants dans la conception d'une BD relationnelle, puisqu'elles forment la base pour représenter les relations entre les tables.

- Les clés sont les éléments qui lient les tables entre elles.

La valeur NULL

- ✓ Dans une relation, la valeur **NULL** représente des données manquantes, inconnues ou des données inapplicables.
- ✓ La valeur **NULL** peut correspondre à une entrée non renseignée (non saisie, non obligatoire).
- ✓ Contrainte d'intégrité de la clé primaire : aucun des attributs de la clé primaire ne doit être **NULL**

La valeur NULL

- La règle précédente signifie que la saisie des valeurs pour les attributs de la clé primaire est obligatoire.
- Un attribut n'appartenant pas à la clé primaire peut avoir la valeur **`NULL`**.
- Attention: la valeur **`NULL`** n'est pas égale à 0 et ne représente aucune valeur particulière pour l'ordinateur.

Exercice

On souhaite informatiser des commandes et l'édition des additions d'un restaurant en utilisant une base de données.

- ✓ Le restaurant dispose de plusieurs tables, chacune identifiée par un numéro et le nombre de convives qu'elle peut recevoir. Plusieurs serveurs travaillent dans le restaurant. Une table est toujours servie par un et un seul serveur.
- ✓ Chaque serveur est identifié par un numéro, a un nom et un grade. Un serveur est affecté par journées entières à une ou plusieurs tables.
- ✓ Les commandes, identifiées par un numéro, correspondent toujours à une table unique. On connait l'heure et la date d'encaissement, le montant total, ainsi que le moyen de paiement utilisé.
- ✓ Chaque commande porte sur un ou plusieurs plats. Les plats sont identifiés par un numéro, ont un nom, un type (entrée, dessert, …). Dans une commande, chaque ligne correspond à un plat commandé en une certaine quantité.

Définir les différents schémas relationnels et indiquer le domaine de chaque attribut

Présentation du LDD SQL

- ✓ Le SQL (Structured Query Language) est le langage informatique standard pour la communication avec les SGBDR.
- ✓ SQL est essentiellement un langage pour effectuer des requêtes sur une base de données pour en extraire ou modifier des données.
- ✓ Le SQL est également utilisé comme LDD pour définir la représentation logique des données d'une BDR.

Présentation du langage SQL

- ✓ C'est un langage déclaratif et non procédural : on explicite ce que l'on veut et non pas la manière de l'avoir.
- ✓ Il offre un LDD pour définir et créer des objets de BDR (tables, vues, etc.)
- ✓ Il offre un LMD pour manipuler les objets d'une BDR.
- ✓ C'est un langage facile à comprendre et à utiliser (syntaxe très proche du langage naturel, pas d'instructions de contrôle, ni de structures de données).

Présentation du langage SQL

Utilisation du SQL :

- ✓ En mode interactif : l'utilisateur écrit textuellement une commande SQL et récupère le résultat immédiatement.
- ✓ En mode intégré : les requêtes SQL sont intégrées dans un programme écrit avec un langage de haut niveau tel que C, C++, Java ou Python.

Types de données SQL

Les entiers :

- **`BIT(M)`** : chaine de bits de longueur M, avec max(M) = 64
- **`TINYINT`** : entier de 8 bits
- **`BOOL`** ou **`BOOLEAN`** : valeurs booléennes. Equivaut à TINYINT(1)
- **`SMALLINT`** : entier de 16 bits
- **`MEDIUMINT`** : entier de 24 bits
- **`INT`** : entier de 32 bits
- **`BIGINT`** : entier de 64 bits

Types de données SQL

Les réels :

- **`DECIMAL(M, D)`** : nombre décimal de **`M`** chiffre avec **`D`** chiffres après la virgule.
- **`FLOAT(M, D)`** : nombre décimal de **`M`** chiffre avec **`D`** chiffres après la virgule. Valeurs par défaut : 10, 2
- **`DOUBLE(M, D)`** : nombre décimal de **`M`** chiffre avec **`D`** chiffres après la virgule. Valeurs par défaut : 16, 4

Types de données SQL

Les dates :

- **`DATE`** : date au format **`'aaaa-mm-jj'`** (entre '1000-01-01' et '9999-12-31')
- **`DATETIME`** : date au frormat **`'aaaa-mm-jj hh-mn-ss'`** (entre '1000-01-01 00-00-00' et '9999-12-31 23-59-59')
- **`TIMESTAMP`** : Date au format **`'aaaammjjhhmnss'`** (entre le 1er Janvier 1970 et le 31 Décembre 2037)
- **`TIME`** : heure au format **`hh-mn-ss`**
- **`YEAR(M)`** : année au format **`aa`** si **`M = 2`** (année entre 1970 et 2069) et **`aaaa`** si **`M = 4`** (entre 1901 et 2155). Valeur par défaut : **`M = 4`**

Types de données SQL

Les chaines de caractères :

- **`CHAR(M)`** : chaine de caractères de longueur fixe **`M`**, avec **`M`** compris entre 1 et 255
- **`VARCHAR(M)`** : chaine de caractères dynamique de longueur **`M`**, avec **`M`** compris entre 1 et 255
- **`TEXT / BLOB`** : Fichier texte/binaire de longueur maximale 65535 caractères
- **`LONGTEXTE / LONGBLOB`** : Fichier texte de longueur maximale 4Go
- **`ENUM`** : liste prédéfinie de valeurs

Création d'une BD

Cette opération donne comme résultat une table vide (ne contenant aucun enregistrement).

Il faut préciser :

- ✓ Le nom de la table ;
- ✓ La description de ses colonnes : nom, type de données ;
- ✓ Les contraintes sur les attributs

Création de table

Quelques propriétés/contraintes des attributs

- ✓ **`PRIMARY KEY`** : clé primaire.
- ✓ **`FOREIGN KEY`** : clé étrangère.
- ✓ **`NULL / NOT NULL`** : valeurs non obligatoires / obligatoires.
- ✓ **`DEFAULT`** : pour définir une valeur par défaut
- ✓ **`AUTO_INCREMENT`** : valeur numérique assignée directement par auto incrémentation
- ✓ **`INDEX`** : index sur des colonnes
- ✓ **`UNIQUE`** : valeurs uniques pour la colonne

Création de table

```
CREATE TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)

CREATE TABLE [IF NOT EXISTS] tbl_name
      AS query_expression

CREATE TABLE [IF NOT EXISTS] tbl_name
    LIKE old_tbl_name
```

Création de table

```
CREATE TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)


create_definition:

    col_name column_definition
  | [CONSTRAINT [symbol] PRIMARY KEY (col_name,...)
  | {INDEX} index_name(index_col_name,...)
  | [CONSTRAINT [symbol] FOREIGN KEY (index_col_name,...)
          reference_definition|


column_definition:
    data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE] [PRIMARY KEY]
```

Création de table

Exemple 1 :

```
CREATE  TABLE client(
id          INTEGER AUTO_INCREMENT PRIMARY KEY,
nom               VARCHAR(15) NOT NULL,
prenom VARCHAR(15) NOT NULL,
date_nce DATE NOT NULL,
genre     ENUM('M', 'F'),
tel               INTEGER,
email  VARCHAR(50)
);
```

Modification de la structure d'une table

Sur une table de la BD, on peut :

- ✓Ajouter une ou plusieurs colonnes ;
- ✓Supprimer une ou plusieurs colonnes ;
- ✓Modifier les propriétés d'une ou de plusieurs colonnes ;
- ✓Ajouter une contrainte ;
- ✓Supprimer une contrainte.

Les commandes SQL relatives à ces actions :

`DROP, ADD et MODIFY`

Modification de la structure d'une table

Pour modifier la structure d'une table, on utilise la commande suivante :

```
ALTER TABLE tbl_name
| ADD [COLUMN] (col_name column_definition,...)
| ADD {INDEX} [index_name] (index_col_name,...)
| ADD [CONSTRAINT [symbol]] PRIMARY KEY  (index_col_name,...)
| ADD [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (index_col_name,
   …)  reference_definition
| DROP [COLUMN] col_name
| DROP INDEX index_name
| DROP PRIMARY KEY
| DROP FOREIGN KEY fk_symbol
| MODIFY  col_name column_definition
```

Modification de la structure d'une table

Exemples

```
CREATE TABLE t1 (a INTEGER, b CHAR(10));

ALTER TABLE t1 RENAME t2;    (<==> RENAME TABLE t1 TO t2;)

ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);

ALTER TABLE t2 ADD d TIMESTAMP;

ALTER TABLE t2 ADD INDEX (d), ADD UNIQUE (a);

ALTER TABLE t2 DROP COLUMN c;

ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
  ADD PRIMARY KEY (c);
```

L'algèbre relationnelle

L'algèbre relationnelle consiste en un ensemble d'opérations qui permettent de manipuler des relations, considérées comme des ensembles de tuples : on peut ainsi faire l'union ou la différence de deux relations, sélectionner une partie de la relation, effectuer des produits cartésiens ou des projections, etc.

Une propriété fondamentale de chaque opération est qu'elle prend une ou deux relations en entrée et produit une relation en sortie. Cette propriété permet de composer des operations : on peut, par exemple, appliquer une sélection au résultat d'un produit cartésien, puis une projection au résultat d'une sélection et ainsi de suite.

L'algèbre relationnelle

Une **requête** est une expression algébrique qui s'applique à un ensemble de relations (la base de données) et produit une relation finale (le résultat de la requête). On peut voir l'algèbre relationnelle comme un langage de programmation très simple qui permet d'exprimer des requêtes sur une base de données relationnelle.

Opérateur de sélection

- Appliquée à une relation **r**, la **sélection** produit une autre relation dont les lignes sont un sous-ensemble des lignes de **r** qui ont une valeur particulière pour un attribut spécifique. La relation résultante et **r** comportent les mêmes attributs.

- Si **r** est une relation sur le schéma **R**, **e** une expression booléenne, alors la sélection sur **r** relativement à **e** est l'ensemble des tuples **t** de **r** tels que **e** est vérifié.

Opérateur de sélection

- ✓ Cela signifie que toutes les lignes de la nouvelle relation vérifient **e.**
- ✓ La sélection se note $\sigma_e(r)$.

- ✓ Mathématiquement, la sélection se traduit par la relation suivante:

$$\sigma_e(r) = \{ t \in r / e = vrai\}.$$

- ✓ Le schéma de la nouvelle relation $\sigma_e(r)$ est le même que celui de la relation **r**.

Opérateur de projection

- Cette opérateur choisit un ensemble de colonnes.

- La projection de la relation **r** sur un ensemble **X** des attributs, notée par $\pi_X(r)$, est une relation obtenue à partir de **r** en éliminant d'abord les colonnes de **r** non spécifiées dans **X**, puis en supprimant tout tuple redondant (les doublons).

- La projection s'exprime mathématiquement par la relation :

$$\pi_X(r) = \{ t(X) / t \in r\}$$

Opérateur de jointure

- C'est une relation binaire qui permet d'associer deux relations d'une même BD.

- En général, cet opérateur associe deux relations sur tous leurs attributs communs.

- La jointure est constituée de tous les tuples résultant de la concaténation des tuples de la première relation avec ceux de la seconde qui ont des valeurs identiques pour un ensemble commun d'attributs X.

Opérateur de jointure

Par attributs communs, on entend des attributs qui, bien qu'ils puissent ne pas avoir le même nom, doivent avoir le même domaine et la même signification sous-jacente.

- Soit **r** une relation avec un ensemble d'attributs **R** et **s** une relation ave un ensemble d'attributs **S.**

- On suppose de plus que **R** et **S** aient des attributs communs, et soit **X** cet ensemble d'attributs communs.

Opérateur de jointure

- La jointure de **r** et **s**, notée **r ⋈ s**, est une nouvelle relation dont les attributs sont les éléments de **R ∪ S**

- En outre, pour chaque tuple **t** de cette nouvelle relation, les trois conditions suivantes doivent être vérifiées:
  - ✓ **t(R) = t_r** pour un tuple **t_r** de la relation **r**.
  - ✓ **t(S) = t_s** pour un tuple **t_s** de la relation **s**.
  - ✓ **t_r(X) = t_s(X)**.

Opérateurs ensemblistes

La réunion :

- La réunion de deux relations **r** et **s** est la relation formée par les tuples qui sont présents dans **r** ou dans **s**.
- Elle se note par **r** ∪ **s**.

L'intersection :

- L'intersection de deux relations **r** et **s** est la relation formée par les tuples qui sont présents simultanément dans **r** et **s**.
- Elle se note par **r** ∩ **s**.

Opérateurs ensemblistes

La différence :

- ✓ La différence de deux relations **r** et **s** est la relation formée par les tuples qui sont présents dans **r** et non présents dans **s**.
- ✓ Elle se note par **r - s**.

- ➢ Pour effectuer l'une de ces trois opérations, il faut que les deux relations r et s soient **compatibles**.

- ➢ Deux relations **r** et **s** sont dites compatibles si elles ont le même degré et leurs attributs ont les mêmes domaines.

Opérateurs ensemblistes

Le produit cartésien :

- Le produit cartésien de deux relations **r** et **s** est la relation formée en concaténant tous les tuples de **r** avec tous les tuples de **s**.
- Elle se note par **r ⊗ s**.

- Remarques :
  - ✓ **Degré(r ⊗ s) = Degré(r) + Degré(s).**
  - ✓ le produit cartésien des relations n'est pas commutatif.

Le langage SQL

Introduction

SQL est un langage qui permet d'interroger une BD sans se soucier de la représentation interne (physique) des données, de leur localisation, des chemins d'accès ou des algorithmes nécessaires.

On peut l'utiliser de manière interactive, mais également en association avec des interfaces graphiques, ou très généralement des langages de programmation.

Requêtes de Sélection

```
SELECT
    [ALL | DISTINCT ]
     select_expr [, select_expr ...]
    [FROM table_references]
        [WHERE where_condition]
    [GROUP BY {col_name | expr}
      [ASC | DESC], ...
     [HAVING where_condition]
     [ORDER BY {col_name | expr }
      [ASC | DESC], ...]
    [INTO OUTFILE 'file_name'
       export_options
```

Jointures

La jointure est une des opérations les plus utiles (et donc une des plus courantes) puisqu'elle permet d'exprimer des requêtes portant sur des données réparties dans plusieurs tables. La syntaxe pour exprimer des jointures avec SQL est une extension directe de celle étudiée précédemment dans le cas des sélections simples.

Jointures simples

Prenons l'exemple de la requête suivante : donner le nom des employés avec le nom des villes où ils travaillent.

```
SELECT nom, ville
FROM Personnel, Departement
WHERE  dept = libelle
```

Jointures

On peut remarquer qu'il n'y a pas dans ce cas d'ambiguité sur les noms des attributs : nom et dept viennent de la table Personnel, tandis que ville et nom viennent de la table Département. Il peut arriver (il arrive de fait fréquemment) qu'un même nom d'attribut soit partagé par plusieurs tables impliquées dans une jointure. Dans ce cas on résout l'ambiguité en préfixant l'attribut par le nom de la table.

Jointures complexes

```
SELECT
    select_expr [, select_expr ...]
   FROM table_reference
  [INNER ] JOIN {table_reference | (table_references)  [join_condition]
  | NATURAL JOIN table_reference
  | {LEFT|RIGHT} JOIN table_reference  join_condition
  | CROSS JOIN table_reference  [join_condition]
```

```
join_condition:
    ON conditional_expr
  | USING (column_list)
```

Jointures complexes

On considère ici deux tables A et B

- ✓ [INNER] JOIN : fonctionne exactement comme une jointure simple. Il offre cependant plus de lisibilité. Équivaut au produit cartésien en l'absence de USING ou ON.
- ✓ NATURAL JOIN : Jointure sur tous les attributs en communs (identiques) des deux tables. Renvoie un produit cartésien si les deux tables n'ont pas d'attributs communs.
- ✓ CROSS JOIN : Produit cartésien. Equivaut à [INNER] JOIN lorsqu'on ajoute une clause USING ou ON.
- ✓ LEFT JOIN : S'il y a une ligne dans A qui répond à la clause WHERE, mais qu'il n'y a aucune ligne dans B qui répond à la condition du LEFT JOIN, alors une ligne supplémentaire de B est générée avec toutes les colonnes mises à NULL.
- ✓ RIGHT JOIN : s'il n'existe pas de lignes dans la table A qui répondent à la jointure, une ligne supplémentaire de A est générée avec toutes les colonnes mises à NULL.

[INNER] JOIN

Exemple 1 : Lieux de travail des employés

```
SELECT nom, ville
FROM Personnel
JOIN Department
ON   (Personnel.dept = Departement.libelle);
```

Équivaut à (avec des alias)

```
SELECT nom, ville
FROM Personnel AS p
JOIN Department AS d
ON   (p.dept = d.libelle);
```

NATURAL JOIN

Exemple : Serveur Diop et les numéros de tables qui lui sont affectées.

```
SELECT nom, grade, num_tab
FROM serveur
NATURAL JOIN tab
WHERE   nom = 'Diop'
```

Équivalent à :

```
SELECT nom, grade, num_tab
FROM serveur
JOIN tab
USING (numero)
WHERE   nom = 'Diop'
```

Union, Intersection

L'expression de ces deux opérations ensemblistes en SQL est très proche de l'algèbre relationnelle. On construit deux requêtes dont les résultats ont même arité (même nombre de colonnes et mêmes types d'attributs), et on les relie par un des mot-clé **`UNION`**, **`INTERSECT`**.

Illustrons par les 2 exemples suivants:

Union, Intersection

- Tous les développeurs

```
SELECT * FROM Dev_Python
UNION
SELECT * FROM Dev_Javascript
```

- Développeurs Python et Javascript à la fois

```
SELECT * FROM Dev_Python
INTERSECT
SELECT * FROM Dev_Javascript
```

Agrégation

Les fonctionnalités d'agrégation de SQL permettent d'exprimer des conditions sur des groupes de tuples, et de constituer le résultat par agrégation de valeurs au sein de chaque groupe.

La syntaxe SQL fournit donc le moyen de partitionner une relation en groupes selon certains critères, le moyen d'exprimer des conditions sur ces groupes, des fonctions d'agrégation.

Fonctions d'agrégation

Ces fonctions s'appliquent à une colonne, en général de type numérique. Ce sont :

- **COUNT** qui compte le nombre de valeurs non nulles.
- **MAX** et **MIN**.
- **AVG** qui calcule la moyenne des valeurs de la colonne.
- **SUM** qui effectue le cumul.

La clause GROUP BY

Dans les requêtes précédentes, on appliquait la fonction d'agrégation à l'ensemble du résultat d'une requête (donc éventuellement à l'ensemble de la table elle-même). Une fonctionnalité complémentaire consiste à partitionner ce résultat en groupes, et à appliquer la ou les fonction(s) à chaque groupe.

On construit les groupes en associant les tuples partageant la même valeur pour une ou plusieurs colonnes

La clause GROUP BY

Exemple:

Afficher le salaire avec le nombre d'employés correspondant

```
SELECT Salaire , COUNT(Code)
FROM Employe
GROUP BY  Salaire
```

Mise à jour: Insertion

L'insertion s'effectue avec la commande **`INSERT`**

```
INSERT  INTO tbl_name  [(col_name [, col_name] ...)]
    {VALUES | VALUE} (value_list) [, (value_list)] ...


INSERT  INTO  tbl_name
   [(col_name [, col_name] ...)]
    SELECT ...

value:
    {expr | DEFAULT}

value_list:
    value [, value] ...
```

Mise à jour: Insertion

Exemple:

```
INSERT INTO Employe
VALUES (1008, 'Kane', 'Saliou', 350000,31);
```

```
INSERT INTO Employe (code, nom,prenom)
VALUES (1009, 'Fall', 'Doudou');
```

Mise à jour : Suppression

La suppression s'effectue avec la clause **DELETE** dont la syntaxe est :

```
DELETE FROM tbl_name
   [WHERE where_condition]
   [ORDER BY ...]
   [LIMIT row_count]
```

Exemple:

```
DELETE FROM Employe
WHERE Age > 65 ;
```

Mise à jour : Modification

La modification s'effectue avec la clause `UPDATE`.

```
UPDATE  table_reference
    SET assignment_list
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]

value:
    {expr | DEFAULT}

assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```