UNIVERSITÉ PRIVÉE DE MARRAKECH | UP | جامعة خاصة مراكش
UNIVERSITÉ RECONNUE PAR L'ÉTAT
جامعة معترف بها من طرف الدولة

# SQL : Langage de manipulation des données (LMD)



**INFO 3**
***2020-2021***

# Plan

I. Introduction

II. Recherche de base (SELECT dans une seule table)

III. SELECT multi-tables

IV. L'insertion, la modification suppression de valeurs de tuples

# Partie I. Langage de Définition des  Données

Le Langage de Définition des Données est la partie de SQL qui permet de décrire les tables  et autres objets manipulés par les SGBD.

I. La commande CREATE
II. La commande ALTER TABLE
III. La commande DROP TABLE

# La recherche de base ou interrogation

**SELECT** …
**FROM** …
WHERE …
GROUP BY …
HAVING …
ORDER BY ...

} Obligatoires

- Ordre des clauses à respecter.

# II.1. Les opérations de projection d'une table
## *Choix des attributs*

**SELECT** nom(s) de colonne **FROM** nom_table**;**

a) SELECT * FROM STAGIAIRE;
b) SELECT Nom-St FROM STAGIAIRE;
c) SELECT Num-St**,** Nom-St FROM STAGIAIRE;
d) SELECT Nom-St AS nom FROM STAGIAIRE;
e) SELECT Nom-St nom FROM STAGIAIRE;
f) SELECT Nom-St AS 'nom de stagiaire' FROM STAGIAIRE;

- * signifie que toutes les colonnes de la table sont sélectionnées.
- Le nom complet d 'une colonne d 'une table est le nom de la table suivi d 'un point et du nom de la colonne. Ex : STAGIAIRE.Nom-St. Le nom de la table peut être omis quand il n 'y a pas d 'ambiguïté.
- L 'ordre des attributs dans la table résultat sera l 'ordre d 'apparition dans le SELECT ou l 'ordre dans la table si *
- Par défaut, les attributs de la table résultat portent le même nom que les attributs de la table d 'origine sauf si ce nom est redéfini :
- SELECT nom_colonne [AS] nom_redéfini FROM nom_table;

# II.1. Les opérations de projection d 'une table

## *Empêcher les répétitions de lignes*

| Type-Stage |
|---|
| TP |
| TP |
| TP |
| TP |
| TP |
| Cours |
| TP |
| Démonstration |
| Démonstration |

- **SELECT DISTINCT** nom(s) de colonne **FROM** nom_table**;**
- SELECT Type-Stage FROM STAGE;
- SELECT **DISTINCT** Type-Stage 'nature du stage' FROM STAGE;

| nature du stage |
|---|
| TP |
| Cours |
| Démonstration |

Remarque : Si on indique plusieurs noms de tables derrière le FROM, on obtient un produit cartésien.

# II.1. Les opérations de sélection dans une table

## *Les éléments de la clause WHERE*

•La clause WHERE permet de spécifier quelles sont les lignes à sélectionner dans une table ou dans le produit cartésien de plusieurs tables. Elle est suivie d ’un prédicat, c ’est-à- dire une expression logique prenant la valeur vrai ou faux et qui sera évalué pour chaque ligne. Les lignes pour lesquelles le prédicat est évalué à vrai sont sélectionnées.

•Les expressions logiques peuvent être composées d ’une suite de conditions combinées entre elles par les opérateurs AND, OR, NOT. L ’opérateur AND est prioritaire par rapport à l ’opérateur OR. Des parenthèses peuvent être utilisées pour imposer une priorité dans l ’évaluation du prédicat, ou simplement pour rendre plus claire l ’expression logique. L ’opérateur NOT placé devant un prédicat en inverse le sens.

•Les différentes formes d ’expressions logiques : comparaison à une valeur, comparaison à une fourchette de valeurs, comparaison à une liste de valeurs, comparaison à un filtre, test sur l ’indétermination d ’une valeur, test ‘ tous ’ ou ‘ au moins un ’, test existentiel, test d ’unicité

# II.1. Les opérations de sélection dans une table

## *La comparaison à une valeur*

```
WHERE exp1 = exp2
WHERE exp1 != exp2
WHERE exp1 < exp2
WHERE exp1 > exp2
WHERE exp1 <= exp2
WHERE exp1 >= exp2
WHERE exp1 is NULL
WHERE exp1 is not NULL
```

- Exp2 peut être une constante. Les chaînes de caractères doivent apparaître sous forme d ’une séquence de caractères entre apostrophes.
- Exp2 peut être une expression obtenue par application d ’opérateurs (+, -, x, /) sur des noms d ’attributs ou des constantes
- exp1 is NULL est vraie si exp1 a la valeur NULL. Le prédicat exp1 = NULL est toujours faux et ne permet pas de tester si l ’expression a une valeur indéterminée.

a) SELECT Num-St, Nom-St FROM STAGIAIRE WHERE Adr-St = 'Paris';
b) SELECT Num-Stage FROM STAGE WHERE Nb-jours > 4;
c) SELECT Num-Stage FROM STAGE WHERE (Prix-jour * nb-jours) > 5000;
d)SELECT Num-Stage, Libellé-Stage, Type-Stage, Num-Cat FROM STAGE WHERE Type-Stage = ’TP ’ AND Num-Cat = 1;
e) SELECT Num-Stage, Libellé-Stage, Type-Stage, Num-Cat FROM STAGE WHERE NOT(Type-Stage = ’TP ’ OR Num-Cat = 1);

# II.1. Les opérations de sélection dans une table

## *La comparaison à une fourchette ou une liste de valeurs*

**SELECT** … **FROM** … **WHERE** Cond1 [NOT] **BETWEEN** val1 **AND** val2;

SELECT Num-Stage, Libellé-Stage, Num-Cat FROM STAGE
WHERE Num-Cat BETWEEN 1 AND 3;

**SELECT** … **FROM** … **WHERE** Nom_colonne [NOT] **IN** (liste_valeurs);

SELECT Num-Stage, Libellé-Stage, Num-Cat FROM STAGE
WHERE Num-Cat IN (1,3);

•NOT IN sert à sélectionner les lignes dont l ’attribut dans la clause WHERE contient une valeur autre que celle de la liste

- MATCH est équivalent à IN
- MATCH UNIQUE rend la valeur vraie que si la valeur ne se trouve qu ’une seule fois dans la liste.

# II.1. Les opérations de sélection dans une table

## *La comparaison à un filtre*

**SELECT** … **FROM** … **WHERE** nom_colonne [NOT] LIKE ’Modèle de chaîne';

- Teste l ’égalité de deux chaînes en tenant compte des caractères jokers dans la 2ème chaîne.

« _ » remplace 1 caractère exactement, « % » remplace une chaîne de caractères de longueur quelconque.

- ’%objet%’ sélectionne sur la présence de la constante objet à n ’importe quelle place dans la chaîne
- ’%a_’ sélectionne sur la présence d ’un a minuscule à l ’avant-dernière place
- ’%@%%’ sélection sur la présence de % n ’importe où dans la chaîne.

a)SELECT Num-St, Nom-St, Prénom-St FROM STAGIAIRE
WHERE Nom-St LIKE ’Var%’;
b)SELECT Num-St, Nom-St, Prénom-St FROM STAGIAIRE
WHERE Nom-St LIKE ’PE___N’;
c)SELECT Num-St, Nom-St, Prénom-St FROM STAGIAIRE
WHERE Nom-St LIKE ’_AR%’;

# III. Le tri des tuples

- L ’ordre dans lequel les lignes d ’une table résultat apparaissent est indéterminé.
- Pour trier les lignes dans un ordre déterminé, il faut utiliser la clause ORDER BY.

**SELECT** … **FROM** … **ORDER BY num_colonne1 [DESC], num_colonne2 [DESC], ...** ;

- On peut trier selon la valeur d ’un ou de plusieurs attributs.

•Les attributs selon lesquels le tri est demandé doivent obligatoirement faire partie de la clause SELECT.

•L ’option facultative DESC donne un tri par ordre décroissant. Par défaut, l ’ordre est croissant.

•Le tri se fait d ’abord selon le premier attribut, puis les lignes ayant la même valeur pour ce premier attribut sont triées selon le 2ème attribut, etc.

- Les valeurs nulles sont toujours en tête.

a)SELECT Num-Stage, Libellé-Stage FROM STAGE ORDER BY Num-Stage;
b)SELECT Num-Stage, Libellé-Stage FROM STAGE ORDER BY 1;
c)SELECT Num-Stage, Libellé-Stage, Nom-Cat FROM STAGE WHERE NOT(Type-Stage = ’Cours’) ORDER BY Num-Cat, Num-Stage DESC;

# IV. Fonctions et expressions

## *Les différentes fonctions disponibles  sous Oracle*

- Fonctions arithmétiques
- Fonctions chaînes de caractères
- Fonctions Date
- Fonctions de conversion
- Fonctions diverses
- Fonctions de groupe

# IV. Fonctions et expressions
## *Les fonctions de groupe*

- Un groupe est un ensemble de tuples d ’une table pour lesquels la valeur d ’un attribut est constante.
- Les fonctions de groupe effectuent un calcul sur l ’ensemble des valeurs d ’un attribut pour un groupe de tuples.
- Il existe 5 fonctions de groupe (7 sous Oracle) : COUNT, SUM, AVG, MIN, MAX (+ VAR, STDEV sous Oracle)
- Une fonction peut apparaître dans une clause SELECT, elle sera affichée comme un attribut ou une expression MAIS, en l ’absence de clause GROUP BY, on ne peut mettre ensemble des fonctions de groupe et des attributs dans un même SELECT.

a) SELECT AVG(Nb-jours) Durée_moyenne FROM STAGE;
b) SELECT AVG(Poids-Art) pmoyen, MAX(PV-Art - PA-Art) Margemax FROM ARTICL

- COUNT(*) compte les lignes de la table résultat (toutes, même celles dont la valeur est NULL)
- COUNT(distinct attribut) compte le nombre de valeurs différentes prise par l ’attribut

# V. Requêtes sur les groupes
## *La clause Group by*

- Elle réarrange la table résultat en un nombre minimum de groupes, tels que, à l ’intérieur de chaque groupe, l ’attribut spécifié possède la même valeur pour chaque tuple. Cette clause n ’affecte pas l ’organisation physique de la table.
- Lorsqu ’on spécifie une clause GROUP BY, les fonctions de groupe sont calculées pour chaque groupe.
- Cette clause permet l ’affichage de l ’attribut commun aux tuples d ’un groupe, suivi de l ’affichage de la valeur des fonctions appliquées au groupe.
- On peut ajouter une clause WHERE qui sélectionne les tuples

SELECT **Nom-Cat**, AVG(Nb-jours) FROM STAGE GROUP BY **Nom-Cat**;

SQL fait des groupes d ’après la valeur de Nom-Cat, puis, pour chaque valeur de Nom-Cat, calcule la moyenne de l ’attribut Nb-jours. Il affiche les valeurs de Nom-Cat correspondant à chaque groupe et la moyenne associée.

# V. Requêtes sur les groupes
## *La clause Having*

- Cette clause est l ’équivalent du WHERE appliquée aux groupes.
- Elle ne peut exister qu ’avec une clause GROUP BY.
- Pratiquement, le critère spécifié dans le HAVING porte sur la valeur d ’une fonction calculée dans un groupe.

a)SELECT Nom-Cat, AVG(Nb-jours) FROM STAGE GROUP BY Nom-Cat
HAVING AVG(Nb-jours) > 4;
b)SELECT Nom-Cat, AVG(Nb-jours) FROM STAGE WHERE Type-Stage = ’TP’
GROUP BY Nom-Cat HAVING AVG(Nb-jours) > 4;

Attention :
1)En présence d ’un GROUP BY, tout attribut dans la clause SELECT doit figurer dans la clause GROUP BY.
2) Une fonction de groupe ne se trouver que dans les clauses SELECT ou HAVING.

# I. Jointure (syntaxe SQL-89)

Quand on précise plusieurs tables dans la clause FROM, on obtient le produit cartésien des tables. Ce qui est normalement souhaité, c 'est de joindre les informations de diverses tables, en précisant quelles relations les relient entre elles. C 'est la clause WHERE qui permet d 'obtenir ce résultat.

Ex :

```
SELECT Num-Stage, Libellé-Stage, Nom-Cat
FROM STAGE, CATEGORIE
WHERE STAGE.Num-Cat = CATEGORIE.Num-Cat;
```

```
SELECT Num-Stage, Libellé-Stage, Nom-Cat
FROM STAGE s, CATEGORIE c
WHERE s.Num-Cat = c.Num-Cat;
```

Alias

```
SELECT Num-Stage, Libellé-Stage, Nom-Cat
FROM STAGE s, CATEGORIE c
WHERE s.Num-Cat = c.Num-Cat and Num-Stage < 200;
```

Equijointures

# Jointure : syntaxe SQL-2

- Cross Join = Produit cartésien

```
SELECT * FROM STAGE CROSS JOIN CATEGORIE;
```

- Jointure naturelle : SQL choisit automatiquement les attributs possédant le même nom dans les deux tables comme attribut de liaison entre les tables.

```
SELECT Num-Stage, Libellé-Stage, Nom-Cat FROM STAGE
NATURAL JOIN CATEGORIE
WHERE Num-Stage < 200;
```

- Jointure conditionnelle : séparation des conditions de jointure des autres conditions additionnelles de filtres

```
SELECT Num-Stage, Nom-Cat FROM STAGE
JOIN CATEGORIE
ON Num-Cat
WHERE Num-Stage < 200;
```

# Jointure par non égalité

| POIDS | | |
|---|---|---|
| Nom-Poids | Poids-Min | Poids-Max |
| PLUME | 0 | 100 |
| LEGER | 101 | 500 |
| MOYEN | 501 | 2500 |
| LOURD | 2501 | 9999 |

| ARTICLES | | |
|---|---|---|
| Num-Art | Nom-Art | Poids-Art |
| A10 | CRAYON | 20.00 |
| A12 | CRAYON LUXE | 20.00 |
| A01 | AGRAFEUSE | 150.0 |
| A04 | LAMPE | 550.0 |

```
SELECT Num-Art, Nom-Art, Poids-Art, Nom-Poids
FROM ARTICLES, POIDS
WHERE Poids-Art BETWEEN Poids-Min AND Poids-Max;
```

En SQL-2 :

```
SELECT Num-Art, Nom-Art, Poids-Art, Nom-Poids
FROM ARTICLES JOIN POIDS
ON Poids-Art BETWEEN Poids-Min AND Poids-Max;
```

# Auto-jointure

Il peut être utile de rassembler des informations venant d ’une ligne d ’une table avec des informations venant d ’une autre ligne de la même table. Dans ce cas, il faut renommer au moins l ’une des deux tables en lui donnant un synonyme afin de pouvoir préfixer sans ambiguïté chaque nom de colonne.

```
SELECT a.Num-Stage, a.Libellé-Stage, a.Nb-jours, b.Num-Stage, B. Nb-jours FROM
STAGE a, STAGE b
WHERE a.Nb-jours > b.Nb-jours AND b.Num-Stage = ′471′;
```

En SQL-2 :

```
SELECT a.Num-Stage, a.Libellé-Stage, a.Nb-jours, b.Num-Stage, B. Nb-jours FROM
STAGE a JOIN STAGE b
ON a.Nb-jours > b.Nb-jours
WHERE b.Num-Stage = ′471′;
```

| Num-Stage | Libellé-Stage | Nb-jours | Num-Stage | Nb-jours |
|---|---|---|---|---|
| 153 | Windows 98 | 5 | 471 | 4 |
| 455 | Windows NT4 | 5 | 471 | 4 |
| 323 | Analyse objet | 5 | 471 | 4 |

# Jointure externe (syntaxe Oracle)

```
SELECT Num-Stage, Num-St
FROM STAGE, PARTICIPATION
WHERE STAGE.Num-Stage = PARTICIPATION.Num-Stage
```

| Num-Stage | Num-St |
|---|---|
| 350 | 106 |
| 350 | 345 |
| 471 | 457 |
| 215 | 987 |
| 467 | 342 |

Un stage qui n ’a pas de participant n ’apparaît pas

```
ELECT Num-Stage, Num-St  ROM
STAGE, PARTICIPATION
 HERE STAGE.Num-Stage = PARTICIPATION.Num-Stage (+)
```

| Num-Stage | Num-St |
|---|---|
| 152 | (NULL) |
| 153 | (NULL) |
| 215 | 987 |
| 323 | (NULL) |
| 350 | 106 |
| 350 | 345 |
| 455 | (NULL) |
| 467 | 342 |
| 471 | 457 |
| 476 | (NULL) |

Liste des différents stages , avec les participants  s ’ils en ont, sans omettre les stages sans  participants.

Le (+) ajouté après un nom de colonne peut s ’interpréter comme l ’ajout, dans la table à laquelle la colonne appartient,  d ’une ligne fictive qui réalise la correspondance avec les  lignes de l ’autre table, qui n ’ont pas de correspondant réel.

# Jointure externe - Syntaxe SQL-2

```
SELECT Num-Stage, Num-St
FROM STAGE RIGHT OUTER JOIN PARTICIPATION
ON Num-Stage;
```

De même, il existe **LEFT OUTER JOIN** qui correspond à l ’ajout d ’une ligne fictive dans la table de gauche et **FULL OUTER JOIN** pour l ’ajout de lignes fictives dans les deux tables.

Si on désire la liste des stages pour lesquels il n ’y a pas eu de participants :

```
SELECT Num-Stage, Num-St
FROM STAGE RIGHT OUTER JOIN PARTICIPATION
ON Num-Stage
WHERE Num-St IS NULL;
```

# II. Requête imbriquée

Une caractéristique puissante de SQL est la possibilité qu ’un critère de recherche employé dans une clause WHERE soit lui-même le résultat d ’un SELECT.

Exemple :
Sélection du numéro et du libellé du stage suivi par le stagiaire n°350.

```
SELECT Num-Stage, Libellé-Stage  FROM STAGE s,
PARTICIPATION p
WHERE Num-St = ′ 350 ′ and s.num-Stage = p.num-Stage ;
```

Mais on peut aussi formuler cette requête par une sous-interrogation :

```
SELECT Num-Stage, Libellé-Stage
FROM STAGE
WHERE Num-Stage =        (SELECT distinct Num-Stage
                         FROM PARTICIPATION
                         WHERE Num-St = = ′ 350 ′);
```

Le SELECT imbriquée équivaut à une valeur

Syntaxe : SELECT … WHERE exp op (SELECT …)
où op est un des opérateurs =, !=, <, >, <=, >=

# Requête imbriquée renvoyant un ensemble de valeurs

- Les opérateurs utilisables, dans ce cas, sont : IN, ALL (tous), ANY (au moins un)

```
SELECT Num-ST, Nom-ST
FROM STAGIAIRE s
WHERE ′ 350 ′ IN
(SELECT Num-Stage
FROM STAGE JOIN PARTICIPATION
ON Num-Stage
WHERE Num-St = s.Num-St);
```

Sous-interrogation synchronisée

```
WHERE exp op ANY (SELECT …)
WHERE exp op ALL (SELECT …)
WHERE exp IN (SELECT …)
WHERE exp NOT IN (SELECT …)
```

où op est =, !=, <, >, <=, >=

```
SELECT Nom-Art
FROM ARTICLES
WHERE PV-Art > ANY      (SELECT PV-Art
                         FROM ARTICLES
                         WHERE Coul-Art = 'Blanc');
```

# Requête existentielle - Test d 'unicité

La clause EXISTS est suivie d 'une sous-interrogation entre parenthèses, et prend la valeur vrai s 'il existe au moins une ligne satisfaisant les conditions de la sous-interrogation

```
SELECT Nom-Art
FROM ARTICLES a
WHERE EXISTS (
        SELECT *
        FROM ARTICLES b
        WHERE b.Coul-Art = ' Blanc ' AND PV-Art > b.PV-Art);
```

Donne les noms d 'articles plus chers qu' un article de couleur blanche

La clause UNIQUE est suivie d 'une sous-interrogation entre parenthèses, et prend la valeur vrai s 'il existe une seule ligne satisfaisant les conditions de la sous-interrogation

```
SELECT Num-St, Nom-St
FROM STAGIAIRE s
WHERE NOT UNIQUE (
        SELECT *
        FROM PARTICIPATION p
        WHERE s.Num-St = p.Num-St);
```

Donne les stagiaires qui ont participé à plus d 'un stage

# Requête imbriquée dans un HAVING

Un critère dans un HAVING peut dépendre du résultat d ’une requête imbriquée.

```
SELECT Num-Cat, SUM(Nb-jours) somme
FROM STAGE
GROUP BY Num-Cat
HAVING SUM(Nb-jours) >
        (SELECT AVG(Nb-jours)
        FROM STAGE);
```

Cette requête compare le total de jours de stage de chaque catégorie avec la  moyenne du nombre de jours d ’un stage

# III. Les opérateurs ensemblistes

- UNION

```
SELECT Num-Stage, Libellé-Stage
FROM STAGE
WHERE Num-Cat = ′1′ OR Num-Cat = ′2′
UNION
SELECT Num-Stage, Libellé-Stage
FROM STAGE
WHERE Nb-jours = 4;
```

Sélection des stages de catégorie 1 ou 2 dont la durée est de 4 jours

# III. Les opérateurs ensemblistes

- INTERSECTION

```
SELECT Num-Stage, Libellé-Stage, ′1′ as indicateur
FROM STAGE
WHERE Num-Cat = ′1′
INTERSECT
SELECT Num-Stage, Libellé-Stage, ′non ouvert as indicateur′
FROM STAGE, PARTICIPATION
WHERE STAGE. Num-Stage = PARTICIPATION.Num-Stage (+) AND Num-St IS NULL;
```

Sélection des stages de catégorie 1 sans participant

# Partie II. L'insertion, la modification ou la suppression de valeurs de tuples

I. La commande INSERT
II. La commande UPDATE
III. La commande DELETE

# La commande INSERT

## Ajout de lignes dans une table

```
INSERT INTO table (col1, …, coln)
VALUES (val1, …, valn);
```

ou

```
INSERT INTO table (col1, …, coln)
SELECT …;
```

La liste des noms de colonnes est optionnelle. Si elle est omise, Oracle prend par défaut l 'ensemble des colonnes de la table dans l 'ordre où elles ont été données lors de la création de la table. Si une liste de colonnes est spécifiée, les colonnes ne figurant pas dans la liste auront la valeur NULL.

```
INSERT INTO STAGIAIRE (Num-St, Nom-St, Prénom-St, Adr-St) VALUES (567, 'Dupont', 'Jean ', 'Paris');

INSERT INTO STAGIAIRE VALUES (567, 'Dupont', 'Jean ', 'Paris');

INSERT INTO PARIS_STAGIAIRE SELECT Num-ST, Nom-ST, Prénom-St FROM STAGIAIRE WHERE Adr-St = 'Paris';
```

# La commande UPDATE

Cette commande permet de modifier les valeurs d ’un ou de plusieurs champs, dans une ou plusieurs lignes existantes d ’une table.

```
UPDATE table
SET col1 = exp1, col2 = exp2, …
WHERE prédicat
```

col1, col2, … sont les noms de colonnes qui seront modifiés.
La clause WHERE est facultative

```
UPDATE table
SET col1, col2 , …) = (SELECT …)
WHERE prédicat
```

```
UPDATE STAGE
SET Nb-jours = 4
WHERE Type-Stage = 'TP';
```

```
UPDATE STAGE
SET Nb-jours =
SELEC MAX(Nb-jours) FROM STAGE
GROUP BY NUM-Cat);
```

```
UPDATE STAGE
SET Nb-jours = 4
WHERE Type-Stage IN ( SELECT Num-St FROM PARTICIPATION);
```

# La commande DELETE

L ’ordre DELETE permet de supprimer des lignes d ’une table

```
DELETE FROM table
WHERE prédicat
```

La clause WHERE indique quelles lignes doivent être supprimées. Cette clause est facultative. Si elle n ’est pas précisée, TOUTES les lignes de la table sont supprimées.

```
DELETE FROM STAGE  WHERE
Num-Cat = (
        SELECT Num-Cat  FROM
        STAGE
        WHERE Libellé-St = 'Programmation java' );
```