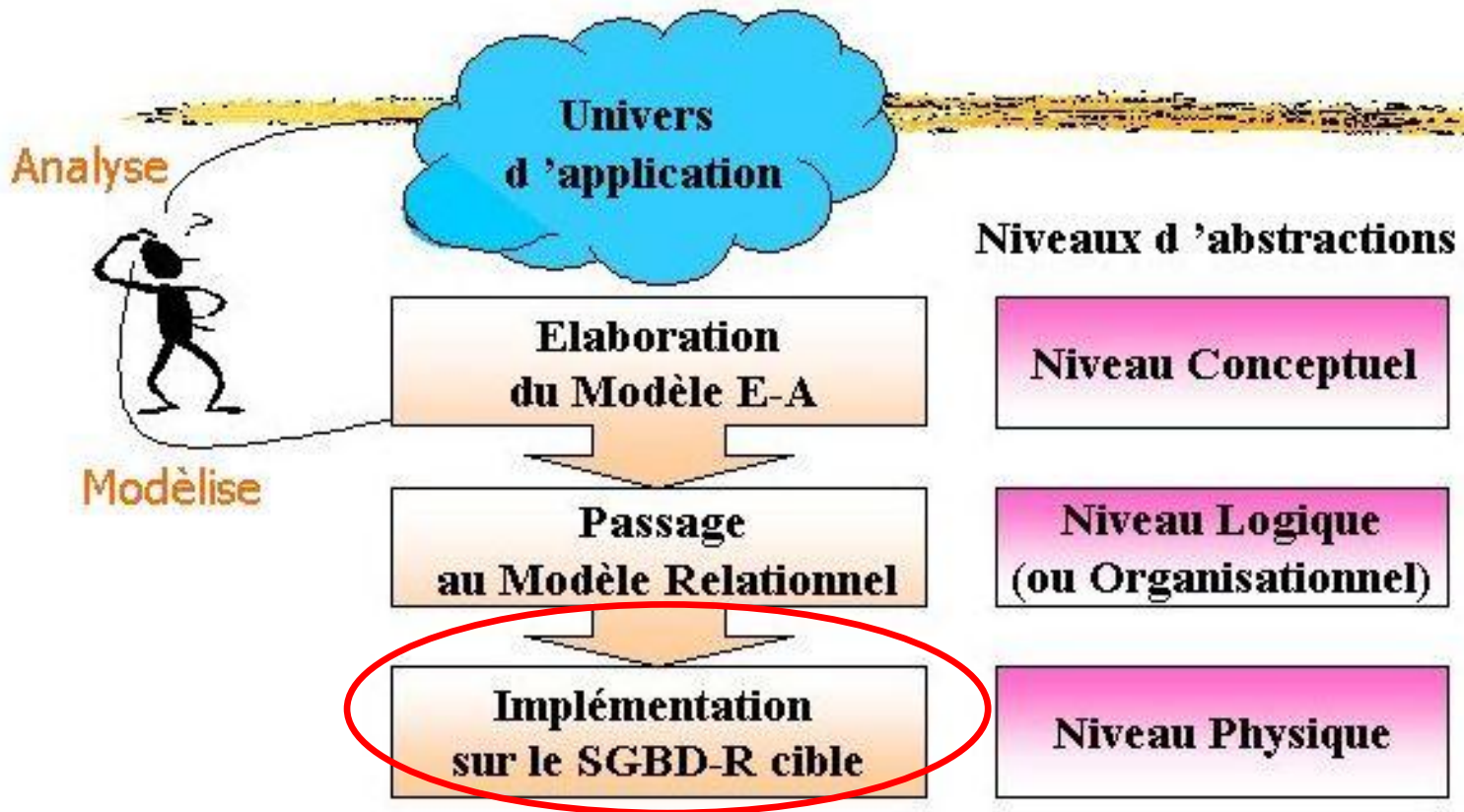


# COURS SYSTÈMES DE GESTION DES BASES DE DONNÉES - PARTIE I SQL AVANCE

**Réalisé par : Mme YEKKEN Sabine**

**Version : 2012 -2013**

# Introduction générale.



- Pour interroger une BD, il faut utiliser un langage .
- SQL est le langage le plus complet pour interagir une BD.

# CHAPITRE 1

## Introduction SQL Langage de définition de données

# Objectifs du cours

(1/2)

## Introduction du langage SQL

### 1<sup>ème</sup> Partie : Présentation du langage LDD

1. Définir « LDD »
2. Les types de données

### 2<sup>ème</sup> Partie : Création d'une table/des contraintes d'intégrités

1. Créer une table/ table à partir d'une autre
2. Ajouter une contrainte clé primaire, clé étrangère
3. Ajouter une contrainte (On delete cascade, on delete set null)
4. Autres contraintes ( not null, check, unique)
5. Activer / désactiver une contrainte

# Objectifs du cours

(2/2)

## 3<sup>ème</sup> Partie : Manipulation de la structure de la table

1. Ajouter, modifier, supprimer une colonne/contrainte
2. Supprimer une table
3. Renommer une table

## 4<sup>ème</sup> Partie : Autres Objets

1. Définition et création de :
  - i. vues
  - ii. Synonymes
  - iii. Index
  - iv. Séquences

# Introduction du langage SQL

## **Objectifs de la partie :**

Introduire et définir le langage SQL.

# Définition de l'SQL.

---

**SQL** (**S**tructured **Q**uery **L**anguage ), en français langage de requête structurée, est composé d'un ensemble de sous langages:

- un langage de définition de données (**LDD**),
- un langage de manipulation de données (**LMD**),
- un langage d'interrogation de données (**LID**),
- un langage de contrôle de données (**LCD**),

pour les bases de données relationnelles.

# 1<sup>ère</sup> Partie :

## Présentation du langage LDD

### Objectifs de la partie :

Définition et présentation du langage de définition de données ainsi que les différents types de données.



# LDD.

---

Le langage de définition de données, c'est un langage qui permet de **définir** et **manipuler** les structures de données et non pas les données.

Structure de données : tout objet de la BD destiné à contenir des données : Exemple : TABLE.

# Les types de données.

Types de données	Description
INT	Entier
CHAR [(size [BYTE   CHAR])]	Taille fixe comprise entre 1 et 2000
VARCHAR2 (size)	Taille Variable comprise entre 1 et 4000
NUMBER [(t, d)]	Réel de t chiffres dont d sont décimales.
LONG	Données caractères ayant une taille $\leq 2GO$
DATE	Date comprise entre 1/1/4712 AJC et 31/12/999 APJC
TIMESTAMP	Année, mois, jour, heure, minute et seconde, fraction de seconde
INTERVAL YEAR(year_precision) TO MONTH	Stocke une période de temps en année et mois, où year_precision est compris entre 0 et 9. La valeur par défaut est 2

# 2ème et 3ème Partie :

Création d'une table/contraintes d'intégrités -  
Manipulation de la structure d'une table

## Objectifs de la partie :

Comment créer une table et définir les contraintes d'intégrités.

Comment ajouter, modifier ou supprimer des colonnes ou des contraintes...

# Créer une table. Syntaxe

---

## Syntaxe :

```
CREATE TABLE <nom_de_la_table>  
  
    ( <nom_colonne1> <type_colonne1>,  
      <nom_colonne2> <type_colonne2>,    ...  
      <nom_colonne'> <type_colonne'>  
  
    );
```

**Remarque :** Pour créer une table il faut avoir :

- Le privilège CREATE TABLE
- Un espace de stockage

# Créer une table. Exemple

- **CREATE TABLE** Etudiants  
( Netudiant number,  
Nom varchar2(10),  
Prenom varchar2(10)  
);
- **DESCRIBE** (ou **DESC**) Etudiants ; --pour voir la description de la table

Type d'objet <b>TABLE</b> Objet <b>ETUDIANTS</b>									
Table	Column	Type De Données	Longueur	Précision	Echelle	Clé Primaire	Valeur Nullable	Valeur Par Défaut	Commentaire
<u>ETUDIANTS</u>	<u>NETUDIANT</u>	Number	-	-	-	-	✓	-	-
	<u>NOM</u>	Varchar2	10	-	-	-	✓	-	-
	<u>PRENOM</u>	Varchar2	10	-	-	-	✓	-	-
									1 - 3

# Créer une table. Table à partir d'une autre table

---

- **CREATE TABLE** Etud **AS** SELECT \* From Etudiants ;

la table Etud aura la même structure de la table Etudiants avec tous les enregistrements de la table.

→ la table Etud aura la même structure de la table Etudiants **avec tous** les enregistrements de la table.

- **CREATE TABLE** Etud **AS** SELECT \* From Etudiants WHERE (0=1);

→ la table Etud aura la même structure de la table Etudiants **sans** les enregistrements de la table.

- **DESCRIBE** (ou **DESC**) Etud ; --pour voir la description de la table

# Ajouter une contrainte clé primaire - Syntaxe (1/3)

---

## 1<sup>er</sup> cas : clé primaire simple

**Syntaxe 1:** lors de la création de la table:

### 1<sup>ère</sup> Méthode : Contrainte niveau COLONNE

**CREATE TABLE** <nom\_de\_la\_table>

( <nom\_col1 > <type\_col1> **CONSTRAINT** <nom\_contrainte> **PRIMARY KEY**,  
    <nom\_col2 > <type\_col2>, ...  
    <nom\_coln > <type\_coln>  
);

# Ajouter une contrainte clé primaire - Syntaxe (2/3)

---

**Syntaxe 1:** lors de la création de la table:

2<sup>ème</sup> Méthode : Contrainte niveau TABLE

**CREATE TABLE** <nom\_de\_la\_table>

( <nom\_col1 > <type\_col1> ,

<nom\_col2 > <type\_col2> , ...

<nom\_coln > <type\_coln> ,

**CONSTRAINT** <nom\_contrainte> **PRIMARY KEY**( colonne(s) ),  
);



# Ajouter une contrainte clé primaire - Syntaxe (3/3)

## 2ème cas : clé primaire composée

**Syntaxe 1:** lors de la création de la table:

```
CREATE TABLE <nom_de_la_table>  
  
( <nom_col1 > <type_col1>,  
  <nom_col2> <type_col2>,  
  ...  
  <nom_coln> <type_coln>,  
  Constraint <nom_contrainte> Primary key (nomCol1, nomCol2...) );
```

→ Contrainte niveau Table

**Syntaxe 2:** après la création de la table :

```
ALTER TABLE <nom_de_la_table> ADD constraint <nom_contr> Primary key  
(nomCol1,nomcol2...)
```

# Ajouter une contrainte clé primaire - Exemple

## Syntaxe :

**ALTER TABLE** <nom\_de\_la\_table> **ADD Constraint** <nom\_contrainte>  
**PRIMARY KEY** (nom\_colonne(s))

**ALTER TABLE** Etudiants **ADD Constraint** pk\_nom\_prenom **PRIMARY KEY**  
 (nom,prenom) ;

Type d'objet TABLE Objet ETUDIANTS

Table	Column	Type De Données	Longueur	Précision	Echelle	Clé Primaire	Valeur Nullable	Valeur Par Défaut	Commentaire
ETUDIANTS	NETUDIANT	Number	-	-	-	-	✓	-	-
	NOM	Varchar2	10	-	-	1	-	-	-
	PRENOM	Varchar2	10	-	-	2	-	-	-

Type d'objet	TABLE	Objet	ETUDIANTS						
Table	Column	Type De Données	Longueur	Précision	Echelle	Clé Primaire	Valeur Nullable	Valeur Par Défaut	Commentaire
ETUDIANTS	NETUDIANT	Number	-	-	-	-	✓	-	-
	NOM	Varchar2	10	-	-	-	✓	-	-
	PRENOM	Varchar2	10	-	-	-	✓	-	-
									1 - 3

# Ajouter une contrainte clé étrangère.

## Syntaxe :

```
ALTER TABLE <nom_de_la_table> ADD constraint  
<nom_contrainte> FOREIGN KEY (<nom_cols>) references  
<table_référencée> (nom_cols) ;
```

Create table Etud as select \* from Etudiants;

Alter table Etud ADD constraint pk\_nom\_prenom PRIMARY KEY (nom,  
prenom) ;

```
ALTER TABLE Etud ADD constraint fk_etud_etudiants FOREIGN KEY  
(nom,prenom) references Etudiants (nom,prenom) ;
```

# Ajouter une contrainte clé étrangère. Options

**Constraint** <nom\_contrainte> **Foreign Key** (<colonne(s)>)

**References** <TableRéféréncée> (<colonne(s)>)

**ON DELETE CASCADE | SET NULL**

- La clause **ON DELETE CASCADE** permet de supprimer automatiquement les valeurs d'une clé étrangère dépendantes d'une clé primaire si une valeur de cette dernière vient d'être supprimée.
- La clause **ON DELETE SET NULL** permet de mettre à Zéro automatiquement les valeurs d'une clé étrangère dépendantes d'une clé primaire si une valeur de cette dernière vient d'être supprimée.

# Ajouter une contrainte <NOT NULL>.

Cette contrainte est définie :

- Soit lors de la création de la table, syntaxe :

```
<nom_colonne> <type_colonne> constraint <nom_contrainte> NOT NULL
```

- Soit après la création de la table, syntaxe :

```
Alter table <nom_tab> modify <nom_col> constraint < nom_contrainte > NOT NULL
```

**La contrainte NOT NULL ne peut être définie qu'au niveau de la colonne, pas au niveau de la table**

# Ajouter une contrainte <UNIQUE> - Définition

---

Une **contrainte d'intégrité** de type clé **unique** exige que chaque valeur dans une colonne ou dans un ensemble de colonnes constituant une clé soit unique.

**Remarque** : Une contrainte unique autorise la valeur NULL à moins que vous définissiez des contraintes NOT NULL

# Ajouter une contrainte <UNIQUE> - Syntaxe

---

Cette contrainte est définie :

- Soit lors de la création de la table, syntaxe :

```
<nom_colonne> <type_colonne> constraint <nom_de_contrainte> UNIQUE
```

- Soit après la création de la table, syntaxe :

```
Alter table <nom_table> add constraint <nom_de_contrainte> UNIQUE  
(<nom_colonne>)
```



# Ajouter une contrainte <CHECK>.

---

La contrainte Check définit une condition que chaque ligne doit vérifier

Cette contrainte est définie :

- Soit lors de la création de la table, syntaxe :

```
<nom_colonne> <type_col> CONSTRAINT <nom_de_contr> CHECK (condition)
```

- Soit après la création de la table, syntaxe :

```
Alter table <nom_table> ADD CONSTRAINT <nom_de_contr> CHECK (condition)
```

**ALTER TABLE** <nom\_table>

- + ADD CONSTRAINT **<Def\_Contrainte>**
- + DROP CONSTRAINT
- + ENABLE |  
DISABLE CONSTRAINT

**<Nom contrainte>**

- + ADD
- + MODIFY
- + DROP

**<Def\_Colonne>**

**<nom colonne>**

# Ajouter une/plusieurs colonne(s). Exemple

## Ajouter 2 colonnes à la table « Etudiants »

```
ALTER TABLE Etudiants ADD (telephone number ,email varchar2(50) );
```

Type d'objet TABLE Objet ETUDIANTS

Table	Column	Type De Données	Longueur	Précision	Echelle	Clé Primaire
<u>ETUDIANTS</u>	<u>NETUDIANT</u>	Number	-	-	-	-
	<u>NOM</u>	Varchar2	10	-	-	1
	<u>PRENOM</u>	Varchar2	10	-	-	2
	<u>TELEPHONE</u>	Number	-	-	-	-
	<u>EMAIL</u>	Varchar2	50	-	-	-

# Modifier une colonne. Exemple

## Syntaxe :

**ALTER TABLE** <nom\_de\_la\_table> **MODIFY** <nouvelle définition de la colonne>

**ALTER TABLE** Etudiants **MODIFY** ( email varchar2(100) );

Type d'objet TABLE Objet ETUDIANTS

Table	Column	Type De Données	Longueur	Précision	Echelle	Clé Primaire
<u>ETUDIANTS</u>	<u>NETUDIANT</u>	Number	-	-	-	-
	<u>NOM</u>	Varchar2	10	-	-	1
	<u>PRENOM</u>	Varchar2	10	-	-	2
	<u>TELEPHONE</u>	Number	-	-	-	-
	<u>EMAIL</u>	Varchar2	100	-	-	-

# Supprimer une colonne. Exemple

## Syntaxe :

**ALTER TABLE** <nom\_de\_la\_table> **DROP** <colonnes>

**ALTER TABLE** Etudiants **DROP** (email, telephone);

Type d'objet	TABLE	Objet	ETUDIANTS						
Table	Column	Type De Données	Longueur	Précision	Echelle	Clé Primaire	Valeur Nullable	Valeur Par Défaut	Commentaire
ETUDIANTS	NETUDIANT	Number	-	-	-	-	✓	-	-
	NOM	Varchar2	10	-	-	-	✓	-	-
	PRENOM	Varchar2	10	-	-	-	✓	-	-
									1 - 3

# Activer/Désactiver une contrainte.

---

## Syntaxe :

```
ALTER TABLE <nom_de_la_table>  
ENABLE | DISABLE CONSTRAINT <nom_de_la_contrainte>;
```

# Supprimer une table. DROP / TRUNCATE

---

## Syntaxe :

```
DROP TABLE <nom_de_la_table> ;
```

## Remarque:

On a aussi la commande « TRUNCATE » qui permet de vider la table

```
TRUNCATE TABLE <nom_de_la_table> ;
```

# Renommer une table.

---

## Syntaxe :

```
RENAME <ancien _nom> TO <nouveau_nom> ;
```

## Exemple:

```
RENAME Etudiants TO Etudiants1;
```



# 4ème Partie :

## Autres Objets

### Objectifs de la partie :

Présenter les vues, Les synonymes, les indexes, et les séquences

# Qu'est-ce qu'une VUE ?.

Une vue est une table **virtuelle**, dans laquelle il est possible de rassembler des informations provenant de plusieurs tables.

On parle de "vue" car il s'agit simplement d'une **représentation des données** dans le but d'une **exploitation visuelle**.

Les données présentes dans une vue sont définies grâce à une clause **SELECT**.

## Remarque:

Les vues existent en tant qu'objets dans la base de données. Les changements apportés à la structure de la table sous-jacente ne sont pas répercutés dans la vue. Pour inclure des colonnes supplémentaires dans une vue, vous devez redéfinir la vue.

# L' Intérêt des VUES.

La vue est une sorte **d'intermédiaire** entre la base de données et l'utilisateur.

Une vue permet ainsi :

- Une **sélection des données** à afficher,
- Une **restriction d'accès** à la table pour l'utilisateur, donc une sécurité des données
- Une **présentation des données** sous plusieurs formes

# Types de VUE.

---

## Vue Simple :

- utilise une seule table
- ne contient ni fonction ni groupe de données
- permet d'exécuter des opérations du LMD

## Vue Complexe :

- utilise plusieurs tables
- contient des fonctions ou des groupes de données
- ne permet pas toujours des opérations du LMD

# Création des VUES.

**CREATE VIEW** Vue

(<colonneA>, <colonneB>, <colonneC>, <colonneD>...)

**AS SELECT** (<colonne1>, <colonne2>, <colonne3>, <colonne4>)

**FROM** <Nom\_table(s)>

**WHERE** ( <condition(s)> )

**WITH READ ONLY | WITH CHECK OPTION**

- L'option WITH READ ONLY indique qu'il est impossible de réaliser des mises à jour sur la vue.
- L'option WITH CHECK OPTION empêche que l'utilisateur d'ajouter ou modifier dans une vue des lignes non conformes à la définition de la vue.

# Suppression des VUES.

---

Une vue peut être détruite par la commande :

```
DROP VIEW <nom de la vue>
```

Lister les vues :

```
SELECT * FROM USER_VIEWS
```

# Exemples de VUES.

---

Créez des vues qui permettent d'afficher :

1. La liste des employés du département 50,
2. Les employés qui ont été embauché avant l'an 2000,
3. Le salaire des employés du département 10

```
1: CREATE OR REPLACE VIEW vue1 (nom, prenom) AS  
SELECT last_name, first_name from employees  
WHERE department_id = 50
```

# Qu'est-ce qu'un SYNONYM ?

---

- C'est un **Alias** sur un objet de la base de données, soit un raccourcis.
- L'objet peut être une Table, une Vue, une Séquence, une Procédure, une Fonction, un Package.
- Le SYNONYM peut être **Public** ou **Privé**.
  - **Public** : il sera accessible à partir de tous les schémas ou utilisateurs.
  - **Privé** : il sera accessible uniquement à partir du schéma dans lequel il a été créé.



# Intérêts d'un SYNONYM.

- Masquer le vrai nom des objets et leur localisations.
- Simplifier les noms des objets.
- Éviter le pré-fixage dans les requêtes avec le nom de son propriétaire.  
( exemple : HR. Employees )

## **Exemples:**

Synonyme	Vue correspondante
cols	User_tab_columns
dict	Dictionnary
ind	User_indexes
obj	User_objects
seq	User_sequences
syn	User_synonyms
tabs	User_tables

# Création d'un SYNONYM.

---

Tout d'abord vous devez avoir le privilège

- Soit **CREATE SYNONYM** pour créer un SYNONYM Privé dans votre schéma,
- Soit **CREATE ANY SYNONYM** pour créer un SYNONYM Privé dans n'importe quel schéma,
- Soit **CREATE PUBLIC SYNONYM** pour créer un SYNONYM Public.

```
CREATE [PUBLIC | ANY] SYNONYM <synonym> FOR <object>  
DROP SYNONYM <synonym>
```

# Qu'est-ce qu'un INDEX ?

---

Un index est un objet de la base de données qui permet **d'accélérer** la recherche des lignes.

Un index peut être créé automatiquement ou manuellement.

- Un index unique est créé automatiquement lors de la définition d'une contrainte Clé Primaire

→ La vue **USER\_INDEXES** contient les noms d'index ET

La vue **USER\_IND\_COLUMNS** contient les noms d'index , des tables et des colonnes

# Manipulation d'un INDEX.

---

## Création d'un index :

**CREATE [UNIQUE] INDEX** <Nom\_index> **ON** table (col1, col2, ...)

### Exemple:

```
CREATE INDEX idx_employees_lastname ON employees  
(last_name)
```

## Suppression d'un index :

**DROP INDEX** <Nom\_index>

### Exemple:

```
DROP INDEX idx_employees_lastname
```

# Qu'est-ce qu'une SEQUENCE ?

---

- Une séquence est un objet de base de données Oracle, au même titre qu'une table, une vue, etc...
- Une séquence permet de **Générer** automatiquement des numéros uniques
- Elle Est partageable entre plusieurs utilisateurs et entre plusieurs tables

→ **Elle permet de créer une valeur de clé primaire**

# Création d'une SEQUENCE.

## CREATE SEQUENCE

```
[ INCREMENT BY  n ]  
[ START WITH  n ]  
[ {MAXVALUE n | NOMAXVALUE} ]  
[ {MINVALUE n | NOMINVALUE} ]  
[ {CYCLE | NOCYCLE} ] [ {CACHE n | NOCACHE} ]
```

- **INCREMENT BY n** : définit l'intervalle entre les numéros
- **START WITH n** : premier numéro de la séquence
- **{MAXVALUE n | NOMAXVALUE}** : valeur maximale
- **{MINVALUE n | NOMINVALUE}** : valeur minimale
- **{CYCLE | NOCYCLE}** : la séquence peut continuer à générer ou non des valeurs
- **{CACHE n | NOCACHE}** : nombre de valeurs pré-allouées et conservées en mémoire.

# Interrogation d'une SEQUENCE.

---

L'interrogation d'une séquence se fait par l'utilisation des "pseudo-colonnes" CURRVAL et NEXTVAL. On parle de **pseudo-colonne** car cela se manipule un peu comme une colonne de table, mais ce n'est pas une colonne de table.

- La pseudo-colonne **CURRVAL** retourne la valeur courante de la séquence.
- La pseudo-colonne **NEXTVAL** incrémente la séquence et retourne la nouvelle valeur.
- On peut modifier ou supprimer une séquence :

**ALTER SEQUENCE** <nom\_sequence> ...

**DROP SEQUENCE** <nom\_sequence>

# CHAPITRE 2

## Le Langage de Manipulation de données



# Objectifs du cours

---

1. Définir le langage « LMD »
2. Insertion des données
3. Mise à jour des données
4. Suppression des données

# Définir le langage LMD

---

**LMD** : **L**angage de **M**anipulation des **D**onnées (DML, *Data Manipulation Language*)

- i. Il Permet la manipulation des tables, soit l'insertion, la mise à jour et la suppression des données.
- ii. Il est composé de 3 Requêtes :

**UPDATE, INSERT, DELETE.**

# Insertion des données - Syntaxe IMPLICITE

(1/3)

**1<sup>er</sup> cas:** Insertion dans toutes les colonnes

**INSERT INTO** <nom\_de\_la\_table> **VALUES**

( <valeur\_colonne1>,  
 <valeur\_colonne2>, ...  
 <valeur\_colonne'>  
);

**Remarque:** le nombre de valeurs de colonnes doit être égal au nombre de colonne de la table et conformément à leurs types

# Insertion des données - Syntaxe EXPLICITE

(2/3)

**2ème cas:** Insertion dans quelques colonnes

**INSERT INTO** <nom\_de\_la\_table>

( <colonne1>,  
 <colonne2>,  
 <colonne3> )

**VALUES**

( <valeur\_colonne1>,  
 <valeur\_colonne2>,  
 <valeur\_colonne3> );

# Insertion des données - Syntaxe avec sous requête (3/3)

## 3<sup>ème</sup> cas: Insertion à partir d'une autre table

**INSERT INTO** <nom\_de\_la\_table>

( <colonne1>,  
<colonne2>,  
<colonne3>... )

**( SELECT**

<colonne1>,  
<colonne2>,  
<colonne3> ...

**FROM** <nom de la table2> )

# Insertion des données - Exercice - Enoncé (1/2)

---

Soit le schéma relationnel suivant :

CLIENTS (CodeClient, NomClient, AdrClient, TelClient)

COMMANDES (NumCommande, Date, CodeClient#)

## Questions:

1. Créer les 2 tables avec les contraintes d'intégrité (clés primaires et étrangères)

# Insertion des données - Exercice - Enoncé (2/2)

## 2. Insérer ses lignes:

### CLIENTS

CodeClient	NomClient	AdrClient	TelClient
Co1	Bensalem Ali	3 rue tunis 2080 tunis	71123800
Co2	Toumi salma	-	71129870

### COMMANDES

NumCommande	Date	CodeClient
110	03/04/2011	Co1
111	09/07/2011	Co2

# Mise à jour des données

---

## Syntaxe:

```
UPDATE <nom_table> SET <colonne_a_modifié> = <nouvelle_valeur>  
WHERE ( <condition_de_mise_à_jour> );
```

## Exemple:

Le client Bensalem Ali vient de changer d'adresse et habite avec le client Toumi salma à « 15 rue Basra Bizerte 7000 », veuillez mettre à jour les deux adresses.



# Suppression des données

---

## Syntaxe:

```
DELETE FROM <nom_table> WHERE ( <condition> );
```

## Exemple:

1. Supprimer les commandes du client « Toumi salma »
2. Vider la table client

# CHAPITRE 3

## Le Langage d'interrogation de données

# Objectifs du cours.

---

1. Définir le LID
2. Savoir formuler des Requêtes simples « SELECT »
3. Distinguer les fonctions mono lignes/multi lignes
4. Définir les Jointures, les Opérateurs ensemblistes et les sous-interrogations

# 1ère Partie :

## Les Requêtes simples « la clause SELECT »

### Objectifs de la partie :

Définir «LID», savoir sélectionner des colonnes avec ou sans doublons, avec ou sans des conditions connaître les Expressions arithmétiques, la Valeur « NULL », les Alias de colonne, les opérateurs logique et ceux de comparaison.

# LID.

(1/2)

Le langage LID permet, comme son nom l'indique, d'interroger une BD. Il sert à **rechercher**, **extraire**, **trier**, mettre en forme des données et **calculer** d'autres données à partir des données existantes.

La structure de base d'une interrogation est formée des 3 clauses suivantes :

```
SELECT *|<liste champ(s)> FROM <nom_table>  
WHERE <condition(s)> ;
```

# LID.



(2/2)

- La clause SELECT : indique les colonnes à récupérer. (ou encore des champs projetés).
- La clause FROM indique le nom de la ou des table(s) impliquée(s) dans l'interrogation.
- La clause WHERE correspond aux conditions de sélection des champs.
- Chaque nom de champ ou de table est séparé par une virgule.

# Sélection des colonnes. Toutes les colonnes

**SELECT**  Affiche tous les champs de la table  
**FROM** employees;

## Résultat:

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID
100	Steven	King	SKING	 515.123.4567 	17/06/87	AD_PRES
101	Neena	Kochhar	NKOCHHAR	 515.123.4568 	21/09/89	AD_VP
102	Lex	De Haan	LDEHAAN	 515.123.4569 	13/01/93	AD_VP
103	Alexander	Hunold	AHUNOLD	590.423.4567	03/01/90	IT_PROG
104	Bruce	Ernst	BERNST	590.423.4568	21/05/91	IT_PROG
105	David	Austin	DAUSTIN	590.423.4569	25/06/97	IT_PROG

# Sélection des colonnes. Une/plusieurs colonnes

```
SELECT first_name, last_name  
FROM employees;
```

Affiche les champs :  
« last\_name » et  
« first\_name »

## Résultat:

FIRST_NAME	LAST_NAME
Ellen	Abel
Sundar	Ande
Mozhe	Atkinson
David	Austin
Hermann	Baer
Shelli	Baida
Amit	Banda
Elizabeth	Bates



# Expressions arithmétiques. Définition

- i. Une expression contient des données de type **NUMBER**, **DATE** et des **opérateurs arithmétiques**.

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division

# Expressions arithmétiques. Exemple 1

```
SELECT    min_salary + max_salary , min_salary - max_salary,
           min_salary * max_salary, min_salary / max_salary
FROM     jobs;
```

## Résultat:

[illegible]

# Expressions arithmétiques. Exemple 2

Ecrire la requête qui permet d'afficher le nom, prénom de tous les employés ainsi que leur salaire annuel avec une augmentation de 50 dinars par mois.

FIRST_NAME	LAST_NAME	Salaire Annuel
Steven	King	288600
Neena	Kochhar	204600
Lex	De Haan	204600
Alexander	Hunold	108600
Bruce	Ernst	72600
David	Austin	58200
Valli	Pataballa	58200
Diana	Lorentz	51000

# Valeur «NULL». Définition

- NULL est une valeur qui n'est pas **disponible, non affectée ou inconnue**.
- NULL est différent de **zéro, espace ou chaîne vide**

## Exemple 1:

```
SELECT employee_id, commission_pct  
FROM employees ;
```

## Résultat:

EMPLOYEE_ID	COMMISSION_PCT
100	
101	-
102	-
103	-
104	-
105	

# Valeur «NULL». Dans les expressions arithmétique.

## Exemple 2 :

```
SELECT Employee_id, (1+commission_pct)*salary  
FROM employees;
```

## Résultat:

EMPLOYEE_ID	(1+COMMISSION_PCT)*SALARY
100	-
101	-
102	-
103	-
104	-
105	-

# Alias de colonne.

---

(1/2)

- Renomme un **en-tête** de colonne
- Suit le nom de colonne
- Peut utiliser le mot clé « **as** »
- Doit **obligatoirement** être inclus entre **guillemets** s'il contient des espaces, des caractères spéciaux ou s'il y a des majuscules et des minuscules

# Alias de colonne. Exemple

(2/2)

```
SELECT first_name as nom, last_name prenom  
FROM employees;
```

## Résultat:

NOM	PRENOM
Ellen	Abel
Sundar	Ande
Mozhe	Atkinson
David	Austin
Hermann	Baer

```
SELECT first_name as "nom de l'employee", last_name "prenom de l'employee"  
FROM employees;
```

## Résultat:

Nom De L'employee	Prénom De L'employee
Ellen	Abel
Sundar	Ande
Mozhe	Atkinson
David	Austin
Hermann	Baer

# Opérateur de concaténation.

- Concatène des colonnes **et/ou** des chaînes de caractères
- Est représenté par le symbole **||**

## Exemple:

```
SELECT ' le nom est ' || first_name || ', le prénom est ' || last_name as "nom et  
    prenom de l'employe"  
FROM employees;
```

## Résultat:

Nom Et Prénom De L'employé
le nom est Ellen, le prénom est Abel
le nom est Sundar, le prénom est Ande
le nom est Mozhe, le prénom est Atkinson
le nom est David, le prénom est Austin



# Éliminer les doublons

- le mot-clé **DISTINCT** élimine les doublons.

## Exemple:

```
SELECT DISTINCT(commission_pct)  
  
FROM employees;
```

## Résultat:

COMMISSION_PCT
-
,15
,35
,4
,3
,2
,25
,1

# Restriction avec la clause « WHERE »

## Syntaxe :

```
SELECT <nom_des_colonnes> FROM <nom_de_la_table >  
WHERE <conditions>;
```

**Exemple:** liste des employés du département 20

```
SELECT last_name , first_name FROM employees  
WHERE department_id=20;
```

## Résultat:

LAST_NAME	FIRST_NAME
Hartstein	Michael
Fay	Pat

2 lignes renvoyées en 0,05 secondes

# Opérateurs de comparaison.

(1/2)

OPERATEUR	DESCRIPTION
=	Egal à
<	Inférieur à
<=	Inférieur à ou égal
>	Supérieur à
>=	Supérieur à ou égale à
<> Ou !=	Différent
BETWEEN val1 AND val2	val1 <= val <= val2
In (liste de valeurs)	Valeur de la liste
LIKE ( _ :un caractère, %=plusieurs caractères)	Comme
IS NULL	Correspond à une valeur NULL

# Opérateurs logiques.

(2/2)

OPERATEUR	DESCRIPTION
AND	Retourne TRUE si les deux conditions sont VRAIES
OR	Retourne TRUE si au moins une des conditions est VRAIE
NOT	Inverse la valeur de la condition TRUE si la condition est FAUSSE FALSE si la condition est VRAIE

# Trier avec « ORDER by ». Définition

---

Triez les lignes selon un ordre bien précis avec la clause ORDER BY:

- **ASC** : ordre croissant (par défaut)
- **DESC** : ordre décroissant

**Remarque:** La clause ORDER BY vient en dernier dans l'instruction SELECT

# Trier avec « ORDER by ». Exemple

```
SELECT last_name, job_id, department_id, hire_date  
FROM employees  
ORDER BY hire_date DESC;
```

## Résultat:

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
Kumar	SA_REP	80	21/04/00
Banda	SA_REP	80	21/04/00
Ande	SA_REP	80	24/03/00
Markle	ST_CLERK	50	08/03/00
Lee	SA_REP	80	23/02/00
Philtanker	ST_CLERK	50	06/02/00

# Exercices d'applications.

(1/2)

( nous allons utilisé la table « employees » sous le schéma HR )

- Afficher la liste de tous les employés.
- Afficher le nom, le prénom, la date d'embauche de tous les employés et renommer les colonnes comme suit: « nom de l'employe », « prenom de l'employe » et « date dembauche »
- Afficher le nom et le prénom des employés triés par date d'embauche.
- Afficher les employés qui ont été embauché au cours de l'année 1990.
- Afficher la liste des employés dont le salaire est > 2800.
- Afficher la liste des employés des départements 10, 30 et 50.

# Exercices d'applications.

(2/2)

- Afficher la liste des employés dont le salaire entre 4000 et 6000 du département 20
- Afficher la liste des employés dont la commission\_pct est NULL.
- Afficher la liste des employés dont le nom commence par la lettre 'A'
- Afficher la liste des employés dont le prénom comporte la lettre 's'
- Afficher la liste des employés dont la deuxième lettre du nom est 'i'
- Afficher la liste des employés embauchés pendant les années entre 1986 et 1990
- Afficher la liste des employés dont le job est différent de 'CLERK' ou 'MANAGER'



# 2ème Partie : Les Fonctions Mono lignes

## Objectifs de la partie:

Définir et savoir utiliser les :

- i. Fonctions de caractères
- ii. Fonctions de manipulation des caractères
- iii. Fonctions de manipulation des dates
- iv. Fonctions numériques
- v. Fonctions de conversions
- vi. Autres fonctions

# Fonctions de caractères. Conversion Minuscule/Majuscule(1/2)

---

**Il existe trois fonctions principales:**

- **LOWER** : convertir les caractères « M » en « m »
- **UPPER** : convertir les caractères « m » en « M »
- **INITCAP** : convertir l'initiale de chaque mot en « M » et les caractères suivants en « m »
- **Remarque:** Une fonction **mono ligne** est une fonction qui s'applique enregistrement par enregistrement.

# Fonctions de caractères. Conversion Minuscule/Majuscule(2/2)

---

SELECT LOWER (last\_name) as "prénom" FROM employees;

Prénom
abel
ande

SELECT UPPER (last\_name) as "prénom" FROM employees;

Prénom
ABEL
ANDE

SELECT INITCAP (last\_name) as "prénom" FROM employees;

Prénom
Abel
Ande

# Fonctions de manipulation des caractères. Définitions (1/2)

---

- **CONCAT**<sub>(chaîne1,chaîne2)</sub> : concatène 2 chaînes = ||
- **SUBSTR**<sub>(chaîne, pos, taille)</sub> : extrait une sous chaîne d'une autre chaîne
- **LENGTH**<sub>(ch)</sub> : taille d'une chaîne en caractères
- **INSTR**<sub>(ch,sch)</sub> : position d'une chaîne de caractères dans une autre chaîne
- **TRIM**<sub>(ch)</sub> : élimine les espaces à gauche et à droite
- **LTRIM**<sub>(ch)</sub> : élimine les espaces à gauche
- **RTRIM**<sub>(ch)</sub> : élimine les espaces à droite

# Fonctions de manipulation des caractères. Définitions (2/2)

---

- **LPAD** (chaîne, nbr, caract) : complète une chaîne de caractères sur la gauche avec une autre chaîne pour avoir n caractères.
- **RPAD** (chaîne, nbr, caract) : complète une chaîne de caractères sur la droite avec une autre chaîne pour avoir n caractères.
- **ASCII** (chaîne) : retourne le code ascii du premier caractère de la chaîne.
- **CHR** (nbr) : retourne le caractère (inverse de ascii).

# Fonctions de manipulation des caractères. Exemples

```
SELECT  'station' ch1, 'agil' ch2,  
        Concat ('station 1','agil') "la station",  
        Substr ('station 1',9,1) "numéro station",  
        Length ('agil') "longueur ch1",  
        Instr ('agil','i') "position de i",  
        Length (Trim(' agil ')) "trim",  
        Length (Rtrim(' agil ')) "Rtrim",  
        Length (Ltrim(' agil ')) "Ltrim",  
        Rpad ('agil',8,'*') "RPAD", Lpad ('agil',8,'-') "LPAD",  
        ASCII ('test') " code ascii " , CHR(75)  
FROM    Dual;
```

# Fonctions numériques. Présentation

---

- **ROUND** (x, n) : Arrondir la valeur de x à la précision spécifiée n
- **TRUNC** (x, n) : Tronquer la valeur de x à la précision spécifiée n
- **FLOOR** (x) : si  $n < x < n+1$  alors  $\text{FLOOR}(x) = n$
- **CEIL** (x) : si  $n < x < n+1$  alors  $\text{CEIL}(x) = n+1$
- **MOD** (x, y) : reste de la division de x sur y
- **REMAINDER** (m, n) : reste d'une division calculé comme suit :  
 $m - (n * q)$  avec q égale à la partie entière du quotient

# Fonctions numériques. Round

(1/2)

```
SELECT round(123.646, 0) "0 chiffres" , round(123.646, 1) "1 chiffres",  
        round(123.646, 2) "2 chiffres", round(123.646, 3) "3 chiffres"  
FROM Dual;
```

## Résultat:

0 Chiffres	1 Chiffres	2 Chiffres	3 Chiffres
124	123.6	123.65	123.646



# Fonctions numériques. Round

(2/2)

```
SELECT round(123.646, -1) "1 chiffres", round(123.646, -2) "2 chiffres",  
       round(123.646, -3) "3 chiffres"  
FROM Dual;
```

## Résultat:

1 Chiffres	2 Chiffres	3 Chiffres
120	100	0

# Fonctions numériques. TRUNC

```
SELECT      TRUNC (123.646, 0) "n=0", TRUNC (123.646, 1) "n=1",
            TRUNC (123.646, 2) "n=2", TRUNC (123.646, 3) "n=3", TRUNC (123.636, -1)
            "n=-1", TRUNC (123.646, -2) "n=-2", TRUNC (123.646, -3) "n=-3"
FROM Dual;
```

## Résultat:

N=0	N=1	N=2	N=3	N=-1	N=-2	N=-3
123	123.6	123.64	123.646	120	100	0

# Fonctions numériques. FLOOR/ CEIL

```
SELECT  floor(-2.56), floor(-2.02), floor(-2.8),  
        ceil(-2.56), ceil(-2.02), ceil(-2.8)  
FROM dual;
```

## Résultat:

FLOOR(-2.56)	FLOOR(-2.02)	FLOOR(-2.8)	CEIL(-2.56)	CEIL(-2.02)	CEIL(-2.8)
-3	-3	-3	-2	-2	-2

```
SELECT  floor(2.56), floor(2.02), floor(2.8),  
        ceil(2.56), ceil(2.02), ceil(2.8)  
FROM dual;
```

## Résultat:

FLOOR(2.56)	FLOOR(2.02)	FLOOR(2.8)	CEIL(2.56)	CEIL(2.02)	CEIL(2.8)
2	2	2	3	3	3

# Fonctions numériques. MOD/ REMAINDER

```
SELECT  MOD (12, 4), MOD (16, 6)
FROM Dual;
```

**Résultat:**

MOD(12,4)	MOD(16,6)
0	4

```
SELECT  remainder (15, 6), remainder (15, 5),
        remainder (15, 4), remainder (-15,4)
FROM Dual;
```

$$15-(2*6) = \mathbf{3} \quad / \quad 15-(3*5) = \mathbf{0} \quad / \quad 15-(4*4) = \mathbf{-1} \quad / \quad -15-(-4*4) = \mathbf{1}$$

**Résultat:**

REMAINDER(15,6)	REMAINDER(15,5)	REMAINDER(15,4)	REMAINDER(-15,4)
3	0	-1	1

# Fonctions de manipulation des dates.

---

(1/3)

- **MONTHS\_BETWEEN (date1,date2):** nombre de mois entre deux dates
- **ADD\_MONTHS (date, nb\_mois):** Ajoute des mois calendaires à une date
- **NEXT\_DAY (date, jour) :** le jour suivant
- **LAST\_DAY (date) :** le dernier jour du mois
- **ROUND (date, précision) :** arrondi une date
- **TRUNC (date, précision) :** tronque une date
- **EXTRACT (day/month/year/hour/minute/seconde from date):** extraction du jour, mois, année, heure, minute et seconde

# Fonctions de manipulation des dates.

(2/3)

```
SELECT  sysdate "date du jour",  
        sysdate+2 "Date jr plus 2jrs",  
        to_date ('10/09/2012','dd/mm/yyyy')-10 "date du jour moins 10" ,  
        Trunc (sysdate - to_date('01/05/2012','dd/mm/yyyy'),0)  
        "tronquer une date"  
FROM dual ;
```

## Résultat:

Date Du Jour	Date Jr Plus 2jrs	Date Du Jour Moins 10	Tronquer Une Date
09-MAY-12	11-MAY-12	31-AUG-12	8

# Fonctions de manipulation des dates.

(3/3)

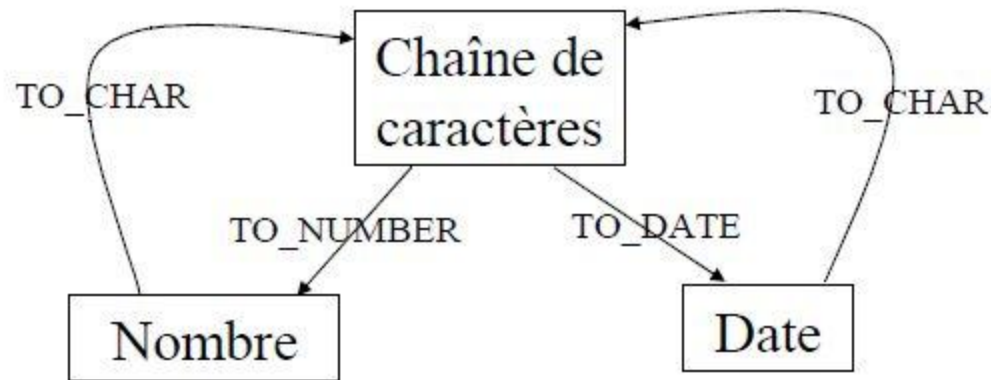
```
SELECT
Trunc (months_between (to_date ('2012/01/01', 'yyyy/mm/dd'), to_date
('2012/03/15', 'yyyy/mm/dd') ),2) "nbr de mois",
next_day(to_date ('2012/03/01', 'yyyy/mm/dd'), 'Monday') "lundi suivant",
last_day(to_date ('2012/03/01', 'yyyy/mm/dd')) "dernier jr du mois mars12" ,
add_months(to_date ('2012/03/01', 'yyyy/mm/dd'),3) "ajouter 3 mois"
FROM Dual;
```

## Résultat:

Nbr De Mois	Lundi Suivant	Dernier Jr Du Mois Mars12	Ajouter 3 Mois
-2.45	05-MAR-12	31-MAR-12	01-JUN-12

# Fonctions de conversions.

---



- **TO\_DATE :** Convertir une Chaîne en format Date
- **TO\_NUMBER :** Convertir une Chaîne en format Numérique
- **TO\_CHAR :** Convertir une Date /un nombre en une Chaîne



# Fonctions de conversions.

Options avec To\_char

OPTION	DESCRIPTION
yyyy	l'année (quatre chiffres)
month	nom complet du mois en minuscule
mon	abréviation du nom du mois en minuscule (trois caractères)
day	nom complet du jour en minuscule
DD	jour du mois (01-31)
D	jour de la semaine (de 1 à 7, dimanche étant le 1)
dy	abréviation du nom du jour en minuscule (3 caractères)
ddd	jour de l'année (001-366)
q	trimestre
w	numéro de semaine du mois (de 1 à 5) (la première semaine commence le premier jour du mois.)
ww	numéro de semaine dans l'année (de 1 à 53) (la première semaine commence le premier jour de l'année.)

# Autres Fonctions.

(1/5). définition1

- **NVL (nom\_col, valeur)** : remplace une valeur nulle
- **NVL2 (expr, val1, val2)** : si expr n'est pas nulle alors elle est remplacée par val1 sinon par val2.
- **NULLIF (val1, val2)** : si val1= val2 la valeur NULL est retournée sinon val1
- **Case** : évalue une liste de conditions et retourne un résultat parmi les cas possibles

# Autres Fonctions.

(2/5). Exercices1

- Ecrire les requêtes **SELECT** qui permettent de:
  - Remplacer « `commission_pct` » par 0 au niveau de la table « `employees` » si elle est null
  - Remplacer « `commission_pct` » par 0 au niveau de la table « `employees` » si elle est null sinon par 1
  - Afficher la liste des employés en ajoutant une colonne « `nom_departement` » qui affiche :
    - Si `department_id=10` , `nom_departement='depart_10'`
    - Si `department_id=20` , `nom_departement='depart_20'`
    - Si `department_id=30` , `nom_departement='depart_30'`
    - ...

# Autres Fonctions.

(3/5). Définitions2

- **ROW\_NUMBER** : retourne le numéro séquentiel d'une ligne d'une partition d'un ensemble de résultats, en commençant à 1 pour la première ligne de chaque partition. **Ne prend pas en considération les doublons**
- **RANK**: retourne le rang de chaque ligne au sein de la partition d'un ensemble de résultats. **Compte les doublons mais laisse des trous**
- **DENSE\_RANK** : retourne le rang des lignes à l'intérieur de la partition d'un ensemble de résultats, sans aucun vide dans le classement.  
**Compte les doublons mais ne laisse pas des trous**

# Autres Fonctions.

(4/5). Syntaxe

```
SELECT
```

```
    Row_number() over (partition by <col x> order by <col_y>
```

```
    DESC/ASC), Col1,...colN
```

```
FROM <nom_table> ;
```

## A retenir:

- La clause **<order by>** est obligatoire.
- La clause **<partition by>** est facultative, est utilisée pour faire un ordre par ensemble de lignes selon **<colx>**.
- **Rank** et **dense\_rank** ont la même syntaxe.

# Autres Fonctions.

(5/5). Exercices2

Ecrire les requêtes SELECT qui permettent de :

- Afficher la liste des employees numérotés par « department\_id ».
- Afficher la liste des employees numérotés par département et selon le salaire **décroissant** :
  - Utiliser « row\_number »
  - Utiliser « rank »
  - Utiliser « dense\_rank »

# 3ème Partie :

## Les Fonctions Multi lignes ( ou fonctions de groupe )

### Objectif du cours :

Définir la clause « GROUP BY »,  
la clause « HAVING »

# Fonctions Multi lignes. Définitions

---

(1/3)

Une fonction **mono ligne** est une fonction qui s'applique enregistrement par enregistrement.

Une fonction **multi ligne** ou fonction de groupe s'applique sur un groupe d'enregistrements et donne un résultat par groupe.

- Clause **GROUP BY** : Définit le critère de groupement pour la fonction
- Clause **HAVING** : Permet de mettre des conditions sur les groupes d'enregistrements



# Fonctions Multi lignes. Définitions

---

(2/3)

- **COUNT** : Nombre de lignes,
- **SUM** : Somme des valeurs,
- **AVG** : Moyenne des valeurs,
- **MIN** : Minimum,
- **MAX** : Maximum,
- **STDDEV** : Ecart type,
- **VARIANCE** : Variance

# Fonctions Multi lignes. Définitions

---

(3/3)

- **COUNT** : Nombre de lignes,
- **SUM** : Somme des valeurs,
- **AVG** : Moyenne des valeurs,
- **MIN** : Minimum,
- **MAX** : Maximum,
- **STDDEV** : Ecart type,
- **VARIANCE** : Variance

# Fonctions Multi lignes. Exemple1

```
SELECT      COUNT (salary) as count,  
            Trunc (SUM (salary),2) as sum,  
            MIN (salary) as min,  
            MAX (salary) as max,  
            Trunc (AVG (salary),2) as avg,  
            Trunc (VARIANCE (salary),2) as variance,  
            Trunc (STDDEV(salary),2) as ecart  
FROM employees;
```

## Résultat:

COUNT	SUM	MIN	MAX	AVG	VARIANCE	ECART
107	691400	2100	24000	6461,682	15283140,539	3909,365

# Fonctions Multi lignes. Syntaxe de la clause GROUP BY

---

SELECT <colonnes>, <fonction de groupe>

FROM table

WHERE <conditions>

**GROUP BY** <col> | <expr>

**HAVING** <conditions>

ORDER BY <col> | <expr>

# Fonctions Multi lignes. Exemple2

```
SELECT department_id,  
COUNT(employee_id) "nbr employees"  
FROM employees  
GROUP BY department_id;
```

DEPARTMENT_ID	Nbr Employees
100	6
30	6
-	1
90	3
20	2
70	1
110	2
50	45
80	34
40	1
60	5
10	1

12  
Départements

```
SELECT department_id, count(employee_id) "nbr employees"  
FROM employees  
GROUP BY department_id  
HAVING department_id < 40;
```

DEPARTMENT_ID	Nbr Employees
30	6
20	2
10	1

# 4<sup>ème</sup> Partie : Les Jointures

## Objectifs de cette partie :

- i. Equijointure et jointure interne, Jointure naturelle
- ii. Jointure externe
- iii. Non équijointure
- iv. Produit cartésien

# Les Jointures. Définition.

---

- Une jointure permet **d'extraire** des données de plusieurs tables à la fois.
- La condition de jointure peut être exprimée dans la clause « WHERE » ou « ON ».
- **Précédez** le nom de la colonne par le nom de la table lorsque celui-ci figure dans plusieurs tables

# Types de Jointures.

---

- Equijointure ou jointure interne (=)
- Jointure naturelle
- Jointure externe (droite, gauche ou complète)
- Non équijointure
- Produit cartésien



# Equijointure. Définition

---

- Une jointure **interne** ou **équijointure** est une jointure avec une condition de jointure contenant **un opérateur d'égalité**.
- C'est la plus répandue, elle combine les lignes qui ont des valeurs **équivalentes** pour les colonnes de la jointure.
- Généralement dans ce type de jointure, ce sont les **Primary Key** et **Foreign Key** qui sont utilisées.

# Equijointure. Syntaxe

(1/2)

```
SELECT table1.column, table2.column
```

```
FROM table1
```

```
INNER JOIN table2
```

```
ON (table1.column_name = table2.column_name) ;
```

# Equijointure. Syntaxe

(2/2)

```
SELECT      T1.Colonne1, ...T1.ColonneN,  
            T2.Colonne1, ...T2.ColonneN  
FROM table1 T1 INNER JOIN table2 T2  
ON T1.C1=T2.C1  
WHERE <Condition(s)>
```

- ✓ **T1** : Alias de la table : table1
- ✓ **T2** : Alias de la table : table2
- ✓ **C1** : colonne permettant la jointure entre les 2 tables  
( primary key + foreign key)

## Équivalent à :

```
SELECT      T1.Colonne1, ...T1.ColonneN,  
            T2.Colonne1, ...T2.ColonneN  
FROM T1 ,T2  
WHERE T1.C1=T2.C1 AND <Condition(s)>
```

# Equijointure. Exemple

## Exemple1:

```
SELECT nomE, e.numdept, nomdept  
FROM emp e INNER JOIN dept d  
ON e.numdept=d.numdept;
```

## Exemple2:

```
SELECT nomE, e.numdept, nomdept  
FROM emp e INNER JOIN dept d  
ON e.numdept=d.numdept  
WHERE e.numdept=11;
```

# Jointure naturelle. Définition

---

- La jointure naturelle est une **équijointure**.
- La clause **NATURAL JOIN** est basée sur toutes les colonnes des deux tables portant **le même nom**.
- Elle sélectionne les lignes des deux tables dont les valeurs sont **identiques** dans toutes les colonnes qui correspondent.
- Si les colonnes portant le même nom présentent des types de données **différents**, une erreur est renvoyée.

# Jointure naturelle. **Syntaxe**

---

```
SELECT colonne1, colonne2...  
FROM Table1  
NATURAL JOIN Table2  
WHERE Condition(s) ;
```



Conditions autres que la condition  
de jointure

# Jointure naturelle. Exemple

---

## Exemple 1:

```
SELECT nomE, numdept,nomdept  
FROM emp NATURAL JOIN dept;
```

## Exemple 2:

```
SELECT nomE, numdept,nomdept  
FROM emp NATURAL JOIN dept  
WHERE numdept=11;
```

# Non équijointure. Définition

- Il s'agit là d'utiliser n'importe quelle condition de jointure entre deux tables, exceptée la stricte égalité. Ce peuvent être les conditions suivantes :

>	supérieur
>=	supérieur ou égal
<	inférieur
<=	inférieur ou égal
<>	différent de
IN	dans un ensemble
LIKE	correspondance partielle
BETWEEN ... AND ...	entre deux valeurs
EXISTS	dans une table



# Jointure externe. Définition

---

- JOINTURES EXTERNES = **OUTER JOINS**.
- Une jointure externe **élargie** le résultat d'une jointure interne (INNER JOINS) et permet d'extraire des enregistrements qui ne répondent pas aux critères de jointure.
- Une jointure externe renvoie **toutes les lignes** qui satisfont la condition de jointure et retourne également une partie de ces lignes de la table pour laquelle aucune des lignes de l'autre ne satisfait pas la condition de jointure.

# Jointure externe. Types de jointures externes

```
SELECT t1.column, t2.column..  
FROM table1 t1  
LEFT | RIGHT | FULL OUTER JOIN table2 t2 ON  
(t1.column_name = t2.column_name) ;
```

- Le sens de la jointure externe **LEFT** ou **RIGHT** de la clause **OUTER JOIN** désigne la table **dominante**.
- **FULL = INNER + LEFT + RIGHT**

# Jointure externe. Gauche

## Exemple:

```
SELECT nomE, e.numdept, nomdept
FROM dept d LEFT JOIN emp e
ON e.numdept=d.numdept
```

NOME	NUMDEPT	NOMDEPT
afia tarek	11	informatique
benflen salah	11	informatique
bensalah salma	11	informatique
hamid ali	11	informatique
benafia ahmed	11	informatique
benflen ali	21	télécommunication
benafia tarek	21	télécommunication
benafia salma	31	électromécanique
-	-	genie logiciel

# Jointure externe. Droite

## Exemple:

```
SELECT nomE, e.numdept,nomdept  
FROM dept d RIGHT JOIN emp e  
ON e.numdept=d.numdept
```

NOME	NUMDEPT	NOMDEPT
test test	-	-
benafia ahmed	11	informatique
hamid ali	11	informatique
bensalah salma	11	informatique
afia tarek	11	informatique
benflen salah	11	informatique
benflen ali	21	télécommunication
benafia salma	31	électromécanique
benafia tarek	21	télécommunication

# Jointure externe. complète

## Exemple:

```
SELECT nomE, e.numdept,nomdept  
FROM dept d FULL JOIN emp e  
ON e.numdept=d.numdept
```

NOME	NUMDEPT	NOMDEPT
afia terek	11	informatique
benflen salah	11	informatique
bensalah salma	11	informatique
hamid ali	11	informatique
benafia ahmed	11	informatique
benflen ali	21	télécommunication
benafia terek	21	télécommunication
benafia salma	31	électromécanique
-	-	genie logiciel
test test	-	-

# Produit Cartésien. Définition

---

On obtient un produit cartésien lorsque :

- Une condition de jointure est omise
- Une condition de jointure est incorrecte
- A chaque ligne de la table 1 sont jointes toutes les lignes de la table 2.
- Le nombre de lignes renvoyés est égal  $n1 * n2$  où
  - ✓  $n1$  est le nombre de lignes de la table 1
  - ✓  $n2$  est le nombre de lignes de la table 2

# Produit Cartésien. Syntaxe

```
SELECT « nom_colonne1 », « nom_colonne2 »...  
FROM table1 t1  
CROSS JOIN table2 t2
```

## Exemple:

Select \* from employees CROSS JOIN departments

→ On va avoir **107\*27** lignes = 2889 lignes

# Produit Cartésien. Exemple

---

SELECT \* From emp e

**INNER JOIN** dept d

**ON** e.numdept=e.numdept

▪ OU

SELECT \* From emp, dept

→ On va avoir **9\*4** lignes



# Jointure. Récapitulatif

Jointure interne	<pre>SELECT ... FROM   &lt;table gauche&gt;        [INNER]JOIN &lt;table droite&gt;        ON &lt;condition de jointure&gt;</pre>
Jointure externe	<pre>SELECT ... FROM   &lt;table gauche&gt;        LEFT   RIGHT   FULL OUTER JOIN &lt;table droite&gt;        ON condition de jointure</pre>
Jointure naturelle	<pre>SELECT ... FROM   &lt;table gauche&gt;        NATURAL JOIN &lt;table droite&gt;        [USING &lt;noms de colonnes&gt;]</pre>
Jointure croisée	<pre>SELECT ... FROM   &lt;table gauche&gt;        CROSS JOIN &lt;table droite&gt;</pre>

# 5<sup>ème</sup> Partie : Les Opérateurs ensemblistes

- i. UNION
- ii. UNION ALL
- iii. INTERSECT
- iv. MINUS

# Les opérateurs ensemblistes.

---

OPERATEUR	DESCRIPTION
INTERSECT	Ramène toutes les lignes communes aux deux requêtes
UNION	Toutes les lignes distinctes ramenées par les deux requêtes
UNION ALL	Toutes les lignes ramenées par les deux requêtes y compris les doublons
MINUS	Toutes les lignes ramenées par la première requête sauf les lignes ramenées par la seconde requête

# Union(ALL). Exemple

---

- Créer une table d'archivage qui contient les informations des employés qui ont été embauchés après l'année 2009.
- Ecrire une requête qui va afficher tous les employés:

```
SELECT * from emp
```

```
Union (ALL)
```

```
SELECT * from emp2
```

# Intersect/Minus. Exemple

---

Ecrire les requêtes permettant de :

- Afficher les employés qui **ont été archivés**.
- Afficher les employés qui **n'ont pas été archivés**.

```
SELECT * from emp
```

**Intersect/minus**

```
SELECT * from emp2 ;
```

# Exercice d'application.

---

- Ecrire une requête qui permet d'afficher ce résultat à partir de la table 'dept' :

NOMDEPT	NUMDEPT
électromécanique	31
genie logiciel	41
informatique	11
télécommunication	21
TOTAL DES DEPARTEMENTS	4

# 6<sup>ème</sup> Partie :

## Les Sous interrogations

- i. Sous interrogations mono lignes
- ii. Sous interrogations multi lignes

# Sous interrogations. Définition

---

- La sous-interrogation (**requête interne**) est exécutée une **seule** fois avant la requête principale
- Le résultat de la sous-interrogation est utilisé par la requête principale (**requête externe**)
- Une sous-interrogation est utilisée dans les clauses suivantes :
  - **WHERE**
  - **HAVING**
  - **FROM**



# Sous interrogations mono lignes. Définition

---

Une sous-interrogation peut ramener **un seul résultat** on l'appelle sous interrogation mono ligne.

Les opérateurs de comparaison **mono ligne** sont :

=, !=, <, >, <=, >=

# Sous interrogations mono lignes. Exemple1

## Exemple de Mise à jour avec une sous interrogation :

l'employé numéro **112** est affecté au même poste et même département que l'employé numéro **111**

```
UPDATE employee_id SET (job_id, department_id)
= ( SELECT job_id, department_id
FROM employees
WHERE employee_id=111 )
WHERE employee_id=112 ;
```

# Sous interrogations mono lignes. Exemple2

## Exemple de Suppression avec une sous interrogation :

Supprimer tous les employés du département 'SALES'

```
DELETE FROM employees
WHERE Department_id =
( SELECT department_id
  FROM departments
 WHERE department_name='SALES' );
```

# Sous interrogations mono lignes. Exemples

---

Ecrire les requêtes permettant de:

- Afficher le nom du département de l'employé 'Benafia tarek'.
- Afficher la liste des employés qui touchent un salaire supérieur à l'employé 'Benafia tarek'.
- Afficher le nom et la date d'embauche des employés qui n'ont pas été embauchés la même année que l'employé 'Benafia tarek'.

# Sous interrogations multi lignes. Définition (1/3)

---

- Une sous-interrogation peut ramener **plusieurs** lignes à condition que **l'opérateur de comparaison** admette à sa droite un **ensemble** de valeurs.
- Les opérateurs permettant de comparer une valeur à un ensemble de valeurs sont :
  - l'opérateur **IN** et
  - les opérateurs obtenus en ajoutant **ANY** ou **ALL** à la suite des opérateurs de comparaison classique :  
**=, !=, <, >, <=, >=.**

# Sous interrogations multi lignes. Définition (2/3)

---

- **ANY** : la comparaison sera vraie si elle est vraie pour au moins un élément de l'ensemble (elle est donc fausse si l'ensemble est vide).
- **ALL** : la comparaison sera vraie si elle est vraie pour tous les éléments de l'ensemble (elle est vraie si l'ensemble est vide).
- L'opérateur **IN** est équivalent à **= ANY**,
- L'opérateur **NOT IN** est équivalent à **!= ALL**.

# Sous interrogations multi lignes. Définition (3/3)

Opérateur de comparaison	Résultat
WHERE salaire <b>IN</b> (1200,1600,2000,2900)	SALAIRE doit être égale à une des valeurs dans la liste
WHERE salaire <b>NOT IN</b> (1200,1600,2000,2900)	SALAIRE ne doit pas être égale à une des valeurs dans la liste
WHERE salaire <b>&gt;ANY</b> (1200,1600,2000,2900)	SALAIRE doit être supérieur à au moins une des valeurs. Donc plus que le minimum (+ de 1200).
WHERE salaire <b>&lt;ANY</b> (1200,1600,2000,2900)	SALAIRE doit être inférieur à au moins une des valeurs. Donc moins que le maximum (- de 2900).
WHERE salaire <b>&gt;ALL</b> (1200,1600,2000,2900)	SALAIRE doit être supérieur au maximum de toutes les valeurs. Donc plus que le maximum (+ de 2900).
WHERE salaire <b>&lt;ALL</b> (1200,1600,2000,2900)	SALAIRE doit être inférieur au minimum de toutes les valeurs. Donc moins que le minimum (- de 1200).

# Sous interrogations multi lignes. Exemples

---

Ecrire les requêtes permettant de:

- Afficher la liste des employés qui touchent un salaire supérieur à tous les employés du département 11.
- Afficher le nom et la date d'embauche des employés qui n'ont pas été embauchés la même année que l'employé 'Benflen ali' et l'employé 'Hamid ali'