Plan

1. Introduction
2. Les tables
3. Les types de données
4. Langage LDD
5. Langage LMD

4.1 Introduction

- Le langage SQL (**Structured Query Language**) est le langage standard de gestion des bases de données relationnelles. Il permet de définir, manipuler et contrôler les données. SQL a été implémenté par IBM dans les années 70.

- SQL est constitué de trois sous langages :

1. **LDD (Langage de Définition de Données) :** incluant les requêtes CREATE, ALTER, DROP, RENAME. Le LDD permet de créer, de changer et de supprimer une structure de données Oracle.

2. **LMD (langage de manipulation de données) :** incluant les requêtes SELECT, INSERT, UPDATE, DELETE. Le LMD permet de sélectionner, d'insérer, de modifier et de supprimer des lignes à partir d'une table.

3. **LCD (langage de contrôle de données) :** incluant les requêtes GRANT, REVOKE. Le LCD permet de donner ou de retirer les droits d'accès à une base de données.

4.2 Les tables

- La table est **l'objet principal** d'une base de données car c'est là ou on stocke les données.
- Elle peut avoir **une ou plusieurs** colonnes (champ, attribut, etc.). Une colonne est caractérisée par son nom et son type de données.
- La table peut avoir **zéro ou plusieurs** lignes (enregistrements, objets, tuples etc.).

Table Personne

CIN	Nom	Age
07188742	Mohamed	33
07124884	Ali	25

4.3 Les types de données

- Oracle offre les types de données suivants :
  - **CHAR(n) :** Chaîne de caractères fixe de longueur maximale n.
  - **VARCHAR(n) ,VARCHAR2(n) :** Chaîne de caractères variable de longueur maximale n.
  - **INT (ou INTEGER) :** Entier
  - **FLOAT :** Réel à virgule flottante
  - **NUMBER(n,d) :** Réel avec n le nombre de chiffres du réel, d le nombre de décimaux. Exemple : NUMBER(5,2) inclut tous les nombres de -999.99 → 999.99
  - **DATE :** Le type de données incluant la date et l'heure. Le format par défaut est le suivant : DD-MON-YY, par exemple : `04-OCT-04‘.

4.3 Les types de données

- **Remarques** :
  - Il n'y a pas de type de données booléen, ce dernier peut être simulé par NUMBER(1) ou CHAR(1).
  - Une colonne non renseignée porte la valeur NULL qui est différente de 0 pour les nombres et de la chaîne vide pour les chaînes de caractères.

4.4 Langage LDD

- Requête CREATE :
  - L 'ordre CREATE TABLE permet de créer une table en définissant le nom, le type de chacune des colonnes de la table.

CREATE TABLE nomTable(colonne1 type_donnees , colonne2 type_donnees, colonne3 type_donnees, colonne4 type_donnees, etc.)

  - ***nomTable* est le nom que l'on donne à la table.**

  - Exemple

```
CREATE TABLE Stage(
        NumStage NUMBER(3) PRIMARY KEY,
        LibelleStage VARCHAR2(15) NOT NULL,
        NbJours NUMBER(2),
        NumCat INTEGER) ;
```

4.4 Langage LDD

- Requête CREATE :
  - Sous Oracle, un nom doit :
    - commencer par une lettre
    - comporter moins de 30 caractères (lettres, chiffres, _, $, #)
    - être différent d'un mot réservé SQL

➔ Les lettres minuscules et majuscules sont équivalentes.

4.4 Langage LDD

- Requête CREATE :
  - Les contraintes : Une contrainte concerne une colonne de la table et peut être aperçue comme étant une condition que doivent respecter les valeurs de cette colonne. Elle est spécifiée lors de la création de la table et vient après le type de données de la colonne dans la requête :
    - [CONSTRAINT nom_contrainte] contrainte
  - Toute contrainte est définie par une table, une ou plusieurs colonnes, un type et un nom.
  - L'utilisateur peut affecter un nom à une contrainte, sinon Oracle lui affectera un nom par défaut. Les types de contraintes sont les suivants :

4.4 Langage LDD

- Requête CREATE :
  - Les contraintes :
    - **Clé primaire**

```
CREATE TABLE Fournisseur(
NumF number PRIMARY KEY,
NomF varchar2(20));
```

Ou

```
CREATE TABLE Fournisseur(
NumF number,
NomF varchar2(20),
constraint Fourn_PK PRIMARY KEY (NumF));
```

4.4 Langage LDD

- Requête CREATE :
  - Les contraintes :
    - **Clé primaire**

Si la clé primaire est composée par plus qu'un attribut, *il faut la définir comme une contrainte à part.* Par exemple, soit la relation :

Employe(<u>numGrade, numEparG</u>, nomE, dateNaisE, salaireE)

```
CREATE TABLE Employe(
  numGrade INT,
  numEparG INT,
  nomE VARCHAR(50),
  dateNaisE DATE,
  salaireE NUMBER,
  CONSTRAINT Employe_PK PRIMARY KEY (numGrade, numEparG));
```

4.4 Langage LDD

- Requête CREATE :
  - Les contraintes :
    - **Clé étrangère**

```
CREATE TABLE table_name (
        column_name data_type,
        ...
        CONSTRAINT fk_tbl1_tbl2
          FOREIGN KEY (this_tables_column)
          REFERENCES other_table_name (other_column_name)
        );
```

4.4 Langage LDD

Clé étrangère

Soit la relation Dept(deptno, nomDept)

```
CREATE TABLE Dept(
    deptno INT PRIMARY KEY,
    nomDept CHAR(40)
);
```

Soit la relation Emp(numE, nom, dateNaissance, #numDept)

```
CREATE TABLE Emp(
            numE INT PRIMARY KEY,
            nom VARCHAR(30),
            dateNaissance DATE,
            numDept INT,
            CONSTRAINT emp_fk_numdept FOREIGN KEY (numDept) REFERENCES Dept(deptno)
);
```

4.4 Langage LDD

- Requête CREATE :
  - Les contraintes :
    - **Non nullité :** L'attribut ne peut pas prendre de valeur nulle, il doit être toujours renseigné.

```
CREATE TABLE Fournisseur(
 NumF number PRIMARY KEY,
 NomF varchar2(20) NOT NULL);
```

4.4 Langage LDD

- Requête CREATE :
  - Les contraintes :
    - **Unicité :** Cette contrainte impose que la ou les colonnes sur lesquelles elle s'applique ne prennent pas une valeur non nulle plus qu'une fois.

```
CREATE TABLE Fournisseur(
NumF number PRIMARY KEY,
NomF varchar2(20) ,
CIN number (8) UNIQUE);
```

Ou

```
CREATE TABLE Fournisseur(
NumF number PRIMARY KEY,
NomF varchar2(20) ,
CIN number (8),
CONSTRAINT Fourn_UN UNIQUE (CIN));
```

4.4 Langage LDD

- Requête CREATE :
  - Les contraintes :
    - **Check :** parfois la colonne d'une table doit satisfaire certaines conditions posées par l'utilisateur *(un domaine de définition précis)*. Les contraintes de vérification permettent généralement de restreindre le domaine d'une colonne.

```
CREATE TABLE Pays(
NomPays varchar2(50) PRIMARY KEY,
Population number,
Continent varchar2(50) ,
CONSTRAINT Population_CK CHECK (Population >0),
CONSTRAINT Continent_CK CHECK (Continent IN ('Europe',
'Asie')));
```

4.4 Langage LDD

- Requête CREATE :
  - Application de cours :
    - Créez la table DEPT. Cette table contient une clé primaire DEPTNO de type NUMBER et DEPTNOM de type VARCHAR2(20).
    - Créez la table EMP. Cette table contient une clé primaire EMPNO de type NUMBER, un nom ENAME qui doit être unique de type VARCHAR2(20) , un poste JOB de type VARCHAR2(15), un salaire SAL de type NUMBER qui doit dépasser un salaire minimum de 600, une date d'embauche HIREDATE de type DATE qui ne peut pas être NULL, et finalement un numéro de département DEPTNO qui réfère la colonne DEPTNO de la table DEPT.

4.4 Langage LDD

- Requête CREATE :
  - Correction application de cours :

```
Create table DEPT(DEPTNO NUMBER PRIMARY KEY,  DEPTNOM VARCHAR2(20));

Create table EMP(EMPNO NUMBER PRIMARY KEY, ENAME VARCHAR2(20), JOB
VARCHAR2(15), SAL NUMBER, HIREDATE DATE NOT NULL,  DEPTNO NUMBER,

CONSTRAINT EMP_UN UNIQUE( ENAME),

CONSTRAINT EMP_CK CHECK(SAL>600),

CONSTRAINT FK_EMP FOREIGN KEY(DEPTNO) REFERENCES DEPT(DEPTNO));
```

4.4 Langage LDD

- Requête ALTER :
  - La modification d'une table consiste à ajouter, renommer, modifier ou supprimer une colonne, ou à ajouter ou supprimer une contrainte. La requête de modification d'une table est **ALTER TABLE**.
  - Il est possible de renommer une table de deux manières, via la requête RENAME ou encore la clause **RENAME** dans la requête ALTER TABLE.
    - Renommer de deux manières la table EMP pour EMPLOYE.
      - **REQ 1 :** RENAME EMP TO EMPLOYE;
      - **REQ 2 :** ALTER TABLE EMP RENAME TO EMPLOYE;

4.4 Langage LDD

- Requête ALTER : Ajouter/Renommer une colonne
  - L'ajout de colonne se fait via la clause **ADD COLUMN ou simplement ADD**. Une colonne ajoutée est mise à NULL pour toutes les lignes. Il en découle qu'on ne peut pas appliquer une contrainte sur cette colonne lui interdisant d'avoir une valeur NULL à moins qu'elle soit affectée à une valeur par défaut ou que la table soit vide.
    - Ajouter la colonne travail JOB.
      - **REQ 3 :** ALTER TABLE EMP ADD job VARCHAR2(30);
    - Ajouter les colonnes téléphone TEL et adresse ADR. La colonne TEL ne doit pas avoir la valeur NULL.
      - **REQ 4 :** ALTER TABLE EMP ADD (TEL NUMBER(8) DEFAULT '00000000' NOT NULL, ADR VARCHAR2(30)) ;
  - Il est possible de renommer une colonne via la clause **RENAME COLUMN**.
    - Renommer la colonne SAL pour SALARY dans la table EMP.
      - **REQ 5 :** ALTER TABLE EMP RENAME COLUMN SAL TO SALARY;

4.4 Langage LDD

- Requête ALTER : Modifier une colonne
  - Ça revient à modifier son type de données via la clause **MODIFY COLUMN** **ou simplement** **MODIFY**.
  - Il est possible d'augmenter la taille d'une colonne de type numérique ou chaîne de caractères.
    - Modifier la taille de la colonne JOB à 50 caractères variables.
      - **REQ 6 :** ALTER TABLE EMP MODIFY (JOB VARCHAR2(50));
  - Il n'est possible de diminuer la taille que si toutes les valeurs de la colonne le permettent.
  - Il est aussi possible de changer le type de données si la colonne en question ne contient que des valeurs NULL.

4.4 Langage LDD

- Requête ALTER : Supprimer une colonne
  - La clause **DROP COLUMN** dans la requête ALTER TABLE permet de supprimer une colonne d'une table.
  - Il est possible de supprimer des colonnes contenant des données à condition qu'elle ne soit pas l'unique colonne de la table ou référencée dans une autre table.
    - Supprimer la colonne HIREDATE de la table EMP.
      - **REQ 7 :** ALTER TABLE EMP DROP COLUMN HIREDATE;
  - Si la colonne est référencée par une colonne d'une autre table (fille), alors il faut ajouter l'option **CASCADE CONSTRAINTS** pour éliminer aussi la contrainte FOREIGN KEY de la colonne étrangère.
    - Soient les relations :

      Dept(numDept, nomDept) et

      Emp(numEmp, nomEmp, ageEmp, #deptNo) avec Emp(deptNo) référence Dept(numDept)

      On veut supprimer numDept de Dept.

      REQ 8 : ALTER TABLE Dept DROP COLUMN numDept **CASCADE CONSTRAINTS**;

4.4 Langage LDD

- Requête ALTER : Ajouter une contrainte
  - On peut ajouter une contrainte à une table existante en utilisant la requête **ALTER TABLE** et la clause **ADD CONSTRAINT**.
    - Soit la relation Client(numClt, nomClt, ageClt). On veut ajouter une contrainte Client_age_seuilmin qui impose que l'âge du client soit au moins égal à 18 ans.
      - **REQ 9 :** ALTER TABLE Client ADD CONSTRAINT Client_age_seuilmin CHECK(ageClt >= 18);
  - Cette opération ne réussit que si les données qui existent déjà dans la table respectent la contrainte qu'on veut ajouter.
    - Supposons que la table EMP(Empno, EmpName, EmpSal, EmpAdr, #Deptno) n'est soumise à aucune contrainte. Ajoutez les contraintes nécessaires sachant que Deptno référence Dept(Deptno) et on veut que EmpAdr soit un attribut obligatoire.
      - **REQ 10 :** ALTER TABLE EMP ADD (CONSTRAINT PK_Emp PRIMARY KEY (EMPNO), CONSTRAINT FK_EMP_Deptno FOREIGN KEY(DEPTNO) REFERENCES DEPT(DEPTNO)) ;
      - **REQ 11 :** ALTER TABLE EMP MODIFY EmpAdr NOT NULL;

4.4 Langage LDD

- Requête ALTER : Supprimer une contrainte
  - La clause **DROP CONSTRAINT** permet de supprimer une contrainte d'une table. Elle est donc suivie par le nom de la contrainte **(notons encore une fois l'importance du nom d'une contrainte)**.
    - Supposons qu'on a une contrainte Client_age_seuilmin qui contrôle que l'âge d'un client est au minimum égal à 18. Si on veut supprimer cette contrainte, il suffit d'écrire :
      - **REQ 12 :** ALTER TABLE Client DROP CONSTRAINT Client_age_seuilmin;
  - Si le mot clé **CASCADE** est précisé, alors la suppression d'une clé primaire d'une table mère, la suppression de la clé étrangère correspondante dans la table fille est effectuée automatiquement.
    - Soient les relations :

      Dept(numDept, nomDept) et Emp(numEmp, nomEmp, ageEmp, #deptNo) avec deptNo référence Dept(numDept)

      Si on veut supprimer la contrainte clé primaire de la table Dept tout en supprimant toute clé étrangère qui référence cette clé primaire alors :
      - **REQ 13 :** ALTER TABLE Dept DROP **PRIMARY KEY CASCADE**;

4.4 Langage LDD

- Requête DROP :
  - Cette commande permet de supprimer une table de la base de données. Les lignes de la table et la définition elle-même sont détruites. L'espace occupé par la table est libéré.

DROP TABLE nomTable;

    - Supprimer la table Stage.
      - **REQ 14 :** DROP TABLE STAGE;

  - Si votre table est une table mère, c-à-d que sa clé primaire est référencée par d'autres colonnes FOREIGN KEY qui sont dans une ou plusieurs tables filles, alors la suppression est impossible.
  - La solution est d'utiliser l'option CASCADE CONSTRAINTS ; dans ce cas là, les contraintes FOREIGN KEY posées sur les colonnes des tables filles seront à leurs tours supprimées.
    - Soient les relations :

      Dept(numDept, nomDept) et Emp(numEmp, nomEmp, ageEmp, #deptNo) avec deptNo référence Dept(numDept)

      Supprimer la table DEPT.
      - **REQ 15 :** DROP TABLE DEPT **CASCADE CONSTRAINTS**;

4.5 Langage LMD

- **INSERT INTO :** ajouter une ou plusieurs lignes dans une table
- **UPDATE :** changer des valeurs dans les lignes d'une table
- **DELETE FROM :** éliminer certaines lignes d'une table
- **SELECT :** interroger la base de données

4.5 Langage LMD

- INSERT

Syntaxe

INSERT INTO nomTable[noms_colonnes] VALUES (valeur);

Exemples : Soit la table Pays(NomPays, Population, Continent)

INSERT INTO Pays VALUES ('France', 20000, 'Europe');

INSERT INTO Pays(NomPays, Population) VALUES ('Allemagne', 15000);

NOMPAYS	POPULATION	CONTINENT
France	20000	Europe
Allemagne	15000	-

4.5 Langage LMD

UPDATE

La clause **UPDATE … SET** permet de modifier des lignes d'une table.

Syntaxe

```
UPDATE nomTable
SET nom_col1 = valeur1 [,
    nom_col2 = valeur2,
    nom_colN = valeurN]
[WHERE <condition>];
```

4.5 Langage LMD

UPDATE

Exemples

/* Augmenter de 100 les populations de tous les pays */

UPDATE Pays

SET Population = Population +100;

NOMPAYS	POPULATION	CONTINENT
France	20000	Europe
Allemagne	15000	Europe
Chine	200000	Asie
Japon	150000	Asie

NOMPAYS	POPULATION	CONTINENT
France	20100	Europe
Allemagne	15100	Europe
Chine	200100	Asie
Japon	150100	Asie

/* MAJ à 65000 la population de la France */

UPDATE Pays

SET Population = 65000

WHERE NomPays = 'France';

NOMPAYS	POPULATION	CONTINENT
France	65000	Europe
Allemagne	15100	Europe
Chine	200100	Asie
Japon	150100	Asie

4.5 Langage LMD

- DELETE
- La clause **DELETE FROM** permet de supprimer des lignes d'une table.

Syntaxe

DELETE FROM nomTable

[WHERE condition];

Exemples

/* Vider la table Fournisseur */

DELETE FROM Fournisseur ;

/* Supprimer les pays ayant plus que 170000 d'habitants */

DELETE FROM Pays

WHERE Population > 170000;

NOMPAYS	POPULATION	CONTINENT
France	20000	Europe
Allemagne	15000	Europe
Chine	200000	Asie
Japon	150000	Asie

➔

NOMPAYS	POPULATION	CONTINENT
France	20000	Europe
Allemagne	15000	Europe
Japon	150000	Asie

4.5 Langage LMD : Interrogation des données ➔ SELECT

Voiture(numV, marqueV, couleurV, prixV)

SELECT Liste_des_Informations_à_Afficher

FROM Liste_Noms_des_Tables

[**WHERE** Condition_sur_les_Lignes]

[**GROUP BY** Colonnes_du_groupement]

[**HAVING** Condition_sur_les_Groupes]

[**ORDER BY** Colonnes_de_Tri [**ASC** / **DESC**],…];

(5) **SELECT** marqueV, AVG(prixV) **AS** PrixMoyen

(1) **FROM** Voiture

(2) **WHERE** couleurV = “Rouge”

(3) **GROUP BY** marqueV

(4) **HAVING** COUNT(numV) > 3

(6) **ORDER BY** marqueV **ASC** ;

La commande SELECT est composée de 6 parties dont 4 sont optionnelles.

4.5 Langage LMD

- La requête **SELECT** nous permet d'extraire des données à partir d'une base de données.
- La clause **SELECT** est suivie **d'une ou de plusieurs colonnes**; ***ce sont les colonnes qu'on veut extraire***.
- La clause **FROM** est suivie par le nom de la table à partir de laquelle on veut afficher les données.

- Dans ce qui suit, nous considérons le schéma relationnel suivant :

DEPT(DEPTNO, DNAME, LOC)

EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

SALGRADE(GRADE, LOSAL, HISAL)

4.5 Langage LMD

- Les tables seront de la forme suivante :

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20
7900	JAMES	CLERK	7698	03-DEC-81	950	-	30
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

4.5 Langage LMD

- La requête REQ1 affiche toutes les colonnes de la table EMP :
  - **REQ1** : SELECT * FROM EMP ;
  - Le symbole **'*'** remplace toutes les colonnes de la table EMP.

- **REQ2** : SELECT EMPNO, ENAME, SAL FROM EMP ;
  - Cette opération permet de sélectionner EMPNO, ENAME et SAL de la table EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

4.5 Langage LMD

- La clause SELECT peut inclure des expressions autres que les colonnes des tables de la base de données. Elle peut contenir :
  - Des expressions arithmétiques
  - Des alias de colonnes

- Donner les noms et les salaires annuels de tous les employés.
  - **REQ3** : SELECT ENAME, SAL*12 FROM EMP ;
  - Voici ce que nous affiche cette requête :

ENAME	SAL*12
SMITH	9600
ALLEN	19200
WARD	15000
JONES	35700
MARTIN	15000
BLAKE	34200
CLARK	29400
SCOTT	36000
KING	60000
TURNER	18000
ADAMS	13200
JAMES	11400
FORD	36000
MILLER	15600

4.5 Langage LMD

- L'entête de la 2ème colonne est sal*12, on peut modifier l'entête en lui affectant un alias :
  - **REQ4** : SELECT ENAME, SAL*12 **AS ANNSAL** FROM EMP ;

- L'alias et la colonne (ou l'expression) sont séparés d'un espace ou du mot clé AS.

ENAME	ANNSAL
SMITH	9600
ALLEN	19200
⋮	⋮

4.5 Langage LMD

- **Le mot clé DISTINCT :** Ce mot clé est introduit après SELECT, il sert à éliminer les occurrences, ainsi pour afficher les codes de départements (chacun une seule fois) :
  - **REQ5** : SELECT DISTINCT DEPTNO FROM EMP ;

- **DISTINCT** ne peut être introduite qu'une seule fois, juste après le SELECT, et concerne toutes les colonnes qui la suivent. Le résultat est toute combinaison distincte de ces colonnes.
  - Afficher les jobs dans chaque département de manière distincte.
  - **REQ6** : SELECT DISTINCT JOB, DEPTNO FROM EMP ;

4.5 Langage LMD

- Le résultat retourné d'une requête n'a pas d'ordre défini.
- La clause **ORDER BY** permet de trier le résultat selon une ou plusieurs colonnes dans un ordre croissant ou décroissant.

- Afficher les employés triés par leurs DEPTNO dans l'ordre décroissant et par leurs JOB dans l'ordre croissant.
  - **REQ7 :** SELECT * FROM EMP ORDER BY DEPTNO DESC, JOB ASC ;
  - Chaque groupe de lignes ayant le même DEPTNO est lui-même trié selon le JOB dans l'ordre croissant.

4.5 Langage LMD

- **La restriction de lignes :** Supposons que nous voulons afficher les employés du département 20. La requête que nous devons formuler doit retourner chaque ligne de la table EMP dont la colonne DEPTNO a pour valeur 20. Cette opération se base sur la clause **WHERE**.

REQ8 : SELECT * FROM EMP **WHERE** DEPTNO=20 ;

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO

4.5 Langage LMD

- La clause WHERE est suivie d'une condition **simple** ou **composée** (combinaison de plusieurs conditions). Chaque ligne satisfaisant cette condition est retournée.

- Afficher les employés du département 10 dont la commission est inférieure au salaire.
  - **REQ9 :** SELECT * FROM EMP WHERE DEPTNO = 10 AND COMM < SAL ;

4.5 Langage LMD

- **Les opérateurs de comparaison :** ils sont utilisables pour comparer deux valeurs de même type (<, <=, >, >=, !=, =).
- Cependant SQL offre quatre opérateurs de comparaison qui lui sont spéciaux :
  - **col BETWEEN** v1 AND v2 ➔ col ≥ v1 AND col ≤ v2.
  - **col IN** (v1, v2, v3, ..., vn) ➔ col = v1 OR col =v2 OR ... OR col = vn.
  - **col LIKE** motif (les symboles **%** et _ sont utilisés pour spécifier un motif, le % remplace une sous chaîne y compris la chaîne vide, le _ remplace un seul caractère).
  - **col IS NULL** (l'opérateur = ne peut pas être utilisée pour comparer une colonne à NULL, on utilise cet opérateur).
- L'opérateur logique **NOT** peut être utilisé avec ces opérateurs NOT BETWEEN...., NOT IN....., NOT LIKE...., IS NOT NULL.

4.5 Langage LMD

- Donner les noms et les salaires des employés dont le salaire est entre 1000 et 2000.
- Donner les employés dont le manager est 7902, 7566 ou 7788.
- Afficher les employés dont le nom commence par 'S'.
- Afficher les employés dont le nom est composé de 4 caractères.
- Afficher les employés qui n'ont pas de manager.

4.5 Langage LMD

- Donner les noms et les salaires des employés dont le salaire est entre 1000 et 2000.
  - **REQ 10 :** SELECT ENAME, SAL FROM EMP WHERE SAL BETWEEN 1000 AND 2000 ;

- Donner les employés dont le manager est 7902, 7566 ou 7788.
  - **REQ11** : SELECT * FROM EMP WHERE MGR IN (7902, 7566, 7788) ;

- Afficher les employés dont le nom commence par 'S'.
  - **REQ 12 :** SELECT * FROM EMP WHERE ENAME LIKE 'S%' ;

- Afficher les employés dont le nom est composé de 4 caractères.
  - **REQ 13 :** SELECT * FROM EMP WHERE ENAME LIKE '_ _ _ _' ;

- Afficher les employés qui n'ont pas de manager.
  - **REQ 14 :** SELECT * FROM EMP WHERE MGR IS NULL ;

4.5 Langage LMD

- **Les opérateurs logiques :** ils sont les trois suivants **AND**, **OR** et **NOT**

- Afficher tous les employés (JOB = 'CLERK') dont le salaire est entre 1000 et 2000.
  - **REQ 15 :** SELECT * FROM EMP WHERE JOB = 'CLERK' AND (SAL BETWEEN 1000 AND 2000);

- Afficher tous les salesmans et tous les managers dont le salaire est supérieur à 1500.
  - **REQ16** : SELECT * FROM EMP WHERE SAL>1500 AND (JOB='MANAGER' OR JOB='SALESMAN');

  ou bien

  REQ16 : SELECT * FROM EMP WHERE SAL>1500 AND (JOB IN ('MANAGER', 'SALESMAN'));

4.5 Langage LMD

- Requête SELECT … GROUP BY :
  - La commande GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux (d'agrégation) sur un groupe de résultat.

```
SELECT colonne1, fonction(colonne2)
FROM table
GROUP BY colonne1;
```

  - Sur une table Achat qui contient toutes les ventes d'un magasin, il est par exemple possible de regrouper les ventes par clients identiques et d'obtenir le coût total des achats pour chaque client.

4.5 Langage LMD

- Requête SELECT …. HAVING :
  - Exemple : Soit une table « Achat » qui contient les achats de différents clients avec le coût du panier pour chaque achat.

id	client	tarif	date_achat
1	Pierre	102	2012-10-23
2	Simon	47	2012-10-27
3	Marie	18	2012-11-05
4	Marie	20	2012-11-14
5	Pierre	160	2012-12-03

4.5 Langage LMD

- Requête SELECT … GROUP BY :
  - Pour obtenir le coût total de chaque client en regroupant les commandes des mêmes clients, il faut utiliser la requête suivante :

```
SELECT client, SUM(tarif)
FROM achat
GROUP BY client;
```

  - La fonction SUM() permet d'additionner la valeur de chaque tarif pour un même client. Le résultat sera donc le suivant :

client	SUM(tarif)
Pierre	262
Simon	47
Marie	38

4.5 Langage LMD

- Il existe plusieurs fonctions qui peuvent être utilisées pour manipuler plusieurs enregistrements **(groupes d'enregistrements obtenus par GROUP BY)**. Il s'agit des fonctions d'agrégations statistiques, les principales sont les suivantes :
  - SUM() pour calculer la somme de plusieurs valeurs.
  - AVG() pour calculer la moyenne de plusieurs valeurs.
  - MAX() pour récupérer la plus grande valeur.
  - MIN() pour récupérer la plus petite valeur.
  - COUNT() pour compter le nombre de lignes concernées.

4.5 Langage LMD

- Requête SELECT …. HAVING :

  - La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer **les groupes (et non les lignes comme le fait le WHERE)** en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX().

4.5 Langage LMD

- Requête SELECT …. HAVING :
  - Exemple : Soit une table « Achat » qui contient les achats de différents clients avec le coût du panier pour chaque achat.

id	client	tarif	date_achat
1	Pierre	102	2012-10-23
2	Simon	47	2012-10-27
3	Marie	18	2012-11-05
4	Marie	20	2012-11-14
5	Pierre	160	2012-12-03

  - Si dans cette table on souhaite récupérer la liste des clients qui ont commandé plus de 40€, toutes commandes confondues, alors il est possible d'utiliser la requête suivante :

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

client	SUM(tarif)
Pierre	162
Simon	47

4.5 Langage LMD : Interrogation des données ➔ SELECT

Voiture(numV, marqueV, couleurV, prixV)

SELECT Liste_des_Informations_à_Afficher

FROM Liste_Noms_des_Tables

[**WHERE** Condition_sur_les_Lignes]

[**GROUP BY** Colonnes_du_groupement]

[**HAVING** Condition_sur_les_Groupes]

[**ORDER BY** Colonnes_de_Tri [**ASC** / **DESC**],…];

(5) **SELECT** marqueV, AVG(prixV) **AS** PrixMoyen

(1) **FROM** Voiture

(2) **WHERE** couleurV = “Rouge”

(3) **GROUP BY** marqueV

(4) **HAVING** COUNT(numV) > 3

(6) **ORDER BY** marqueV **ASC** ;

La commande SELECT est composée de 6 parties dont 4 sont optionnelles.

4.5 Langage LMD

- Dans ce qui suit, nous considérons le schéma relationnel suivant dans les exemples et applications proposés :

DEPT(DEPTNO, DNAME, LOC)

EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

SALGRADE(GRADE, LOSAL, HISAL)

4.5 Langage LMD

- Le contenu des tables est le suivant :

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20
7900	JAMES	CLERK	7698	03-DEC-81	950	-	30
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

4.5 Langage LMD : **SELECT à partir d'une seule table**

- Application : Comme susmentionné, dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Si on vous demande d'afficher le nom et le job des employés dont le salaire dépasse 1000.

```
SELECT ENAME, JOB
FROM EMP
WHERE salaire > 1000;
```

➔ **une seule table utilisée à savoir EMP**

  - Si on vous demande d'afficher les informations sur les départements dont le nom contient 'S'.

```
SELECT *
FROM DEPT
WHERE DNAME LIKE '%S%';
```

➔ **une seule table utilisée à savoir DEPT**

4.5 Langage LMD : **SELECT avec JOINTURE**

- Si on cherche le **nom** du département de **chaque employé**, cette information se trouve dans une table DEPT. Il faut donc faire le lien entre chaque employé ***(chaque ligne dans la table EMP)*** et son département ***(la ligne correspondante dans la table DEPT)*** à travers son numéro de département **DETNO**.

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	140	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-83	1100	-	20
7900	JAMES	CLERK	7698	03-DEC-81	950	-	30
7902	FORD	ANALYST	7566	03-DEC-81	3000	-	20
7934	MILLER	CLERK	7782	23-JAN-82	1300	-	10

...

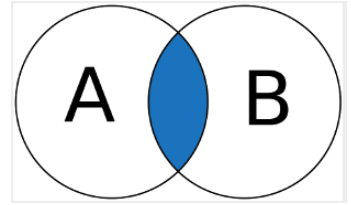
DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

La jointure nous permet de combiner des lignes issues de **deux ou plusieurs** tables en vue de retrouver des données associées.

4.5 Langage LMD

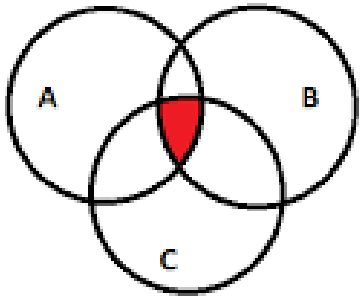
- **La jointure** nous permet de combiner des lignes issues de **deux ou plusieurs** tables en vue de retrouver des données associées.

Jointure de deux tables A et B




```
SELECT ...
FROM  Table1, Table2, ...
WHERE conditionJointure
[...];
```

Jointure de trois tables A et B et C



- Pour effectuer une jointure, il faut spécifier :
  - Les noms des tables, dans la clause FROM, séparés par des virgules
  - La condition de jointure dans la clause WHERE
  - Les noms de colonnes doivent être préfixés par les noms des tables pour éviter toute ambiguïté ***quand deux colonnes de deux tables différentes utilisées dans la condition de jointure portent le même nom***.

4.5 Langage LMD

- Application : Comme susmentionné, dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Afficher les noms des employés et les noms de leurs départements. **Il est nécessaire de lier chaque employé à son propre département moyennant une condition de jointure qui lie la clé étrangère DEPTNO de Emp à la clé primaire DEPTNO de Dept.**

```
SELECT ENAME, DNAME
FROM EMP, DEPT
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

ou bien en utilisant **les alias de tables**

```
SELECT ENAME, DNAME
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO;
```

4.5 Langage LMD

- Application : Dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Afficher les jobs des employés **dont le département se trouve à DALLAS**.

```
SELECT JOB
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO AND LOC = 'DALLAS';
```

4.5 Langage LMD

- Application : Dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Afficher les jobs des employés dont le département se trouve à DALLAS. **Le résultat doit être affiché par ordre décroissant du JOB.**

```
SELECT JOB
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO AND LOC = 'DALLAS'
ORDER BY JOB DESC;
```

4.5 Langage LMD

- Application : Dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Afficher les salaires et les commissions des employés dont le nom commence par S et dont le département ne se trouve pas à BOSTON.

```
SELECT SAL, COMM
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO AND ENAME LIKE 'S%' AND LOC != 'BOSTON';
```

4.5 Langage LMD

- Application : Dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Afficher la somme des salaires des employés ***par adresse de leurs départements*** **(regroupement suivant la colonne LOC de Dept)**. Il faut donc trouver l'adresse du département de chaque employé!

```
SELECT LOC, SUM(SAL)
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
GROUP BY LOC;
```

4.5 Langage LMD

- Application : Dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Afficher la somme des salaires des employés par adresse de leurs départements **(regroupement suivant la colonne LOC de Dept) en ne considérant que les adresses où travaillent plus de 2 employés**.

```
SELECT LOC, SUM(SAL)
FROM EMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
GROUP BY LOC
HAVING COUNT(EMPNO) > 2;  ou bien HAVING COUNT(*) > 2;
```

4.5 Langage LMD

- Application : Dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Afficher le nom de chaque employé ainsi que son grade. On rappelle que le grade d'un employé est celui dont le LOSAL et le HISAL englobent le salaire de l'employé. Par exemple, un employé dont le salaire est 1650 a le grade 3 comme 1650 est compris entre 1401 et 2000.

```
SELECT ENAME, GRADE
FROM EMP, SALGRADE
WHERE SAL BETWEEN LOSAL AND HISAL;
```

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

4.5 Langage LMD

- Application : Dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Afficher le nom de chaque employé, son job ainsi que le nom de son département et son grade.

  Il faut se rappeler que :

  - pour déterminer le nom du département d'un employé, il faut faire un lien (condition de jointure) entre chaque employé et son département :
    - FROM EMP **E**, DEPT **D** WHERE **E.DEPTNO = D.DEPTNO**
  - pour déterminer le grade d'un employé, il faut faire un lien (condition de jointure) entre chaque employé et son grade :

    FROM EMP, SALGRADE WHERE **SAL BETWEEN LOSAL AND HISAL**;

4.5 Langage LMD

- Application : Dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Afficher le nom de chaque employé, son job ainsi que le numéro et nom de son département et son grade.

```
SELECT ENAME, JOB, D.DEPTNO, DNAME, GRADE
FROM EMP E, DEPT D, SALGRADE
WHERE E.DEPTNO = D.DEPTNO AND SAL BETWEEN LOSAL AND HISAL;
```

4.5 Langage LMD

- Application : Dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Afficher le nom de chaque employé, son job ainsi que le nom de son département et son grade. **Ne considérer dans le résultat que les employés n'ayant pas de commission.**

```
SELECT ENAME, JOB, DNAME, GRADE
FROM EMP E, DEPT D, SALGRADE
WHERE E.DEPTNO = D.DEPTNO AND SAL BETWEEN LOSAL AND HISAL
      AND COMM IS NULL;
```

4.5 Langage LMD

- Application : Dans ce qui suit, nous considérons le schéma relationnel suivant :

  DEPT(DEPTNO, DNAME, LOC)

  EMP(EMPNO, ENAME, JOB, #MGR, HIREDATE, SAL, COMM, #DEPTNO)

  SALGRADE(GRADE, LOSAL, HISAL)

  - Afficher le nom de chaque employé ainsi que le nom de son manager (c'est-à-dire son chef). **Il faut faire le lien moyennant une condition de jointure entre les informations associées à un employé et les informations associées à son chef moyennant la clé étrangère MGR qui indique le numéro du chef de chaque employé.**

```
SELECT E.ENAME, Chef.ENAME
FROM EMP E, EMP Chef
WHERE E.MGR = Chef.EMPNO;
```