Support SQL	Création et Manipulation d'une BD

- Création de compte utilisateur et Attribution de privilèges
- Création de Bases de Données (LDD)
- Langage Manipulation d'une BD (LMD)
- Langage d'interrogation des données (LID)

A. Création de comptes utilisateurs:

Pour la gestion d'une base de données on trouve deux types d'utilisateurs l'administrateur de la base de données (DBA) et des utilisateurs simples. La responsabilité du DBA peut inclure l'installation, la mise à jour d'Oracle Server et les outils d'application, la création et gestion la structure d'une base de données, la gestion des sauvegardes, la **gestion des utilisateurs**, la gestion de la sécurité…

Le compte administrateur est créé lors de l'installation avec l'authentification suivante :

Utilisateur : **System** Mot de passe : **manager**

- **Pour créer un utilisateur :** on doit d'abord se connecter (**CONNECT**) en tant qu'administrateur (car c'est lui seul qui a se droit) en suivant cette syntaxe :

```
CREATE USER name IDENTIFIED BY password;
```

Le password ne doit pas commencer par un chiffre ou par des symboles particuliers.

- **Pour supprimer un utilisateur** : le DBA utilise la commande **DROP USER**

```
DROP USER name [CASCADE];
```

CASCADE pour supprimer tous les objets (tables, procédures, trigger, vues…) créés par cet utilisateur.

Lorsqu'un utilisateur est créé avec l'instruction CREATE USER, il **ne dispose encore d'aucun** droit car aucun privilège ne lui a encore été assigné, par ailleurs il ne peut même pas se connecter à la base. Il faut que le DBA lui assigne les privilèges nécessaires.

La clause **GRANT** permet d'attribuer des permissions à un ou plusieurs utilisateurs sur un ou plusieurs éléments de la base de données. Par contre la commande **Revoke** supprime un (ou plusieurs) privilège(s) à un (ou plusieurs) utilisateur(s).

Syntaxe :

```
GRANT system_priv1, system_priv2...
TO user1, user2...
[WITH ADMIN OPTION] ;

Revoke system_priv2...
From user1,... ;
```

*L'option **WITH ADMIN OPTION** permet de pouvoir ensuite donner ce privilège à d'autres utilisateurs.*

Exemple:

```
GRANT Connect
  TO scott
  WITH ADMIN OPTION;
```

Nom du rôle	Privilèges attribués aux Roles
Connect	Alter session, create database link, create sequence, create session, create synonym, create table, create view
Create type	Create type, execute, execute any, type

Resource	create procedure, create sequence, create table, create trigger, create type.
DBA	All system privileges with admin option.

Les rôles prédéfinis:

B. Création d'une Base de Données :

1) *Méthode 1 :*

a. Syntaxe de création d'une table :

```
Create table nom_table (
                    Attribut 1  domaine,
                ……,
                    Attribut N  domaine);
```

b. Syntaxe d'ajout de contraintes après la création de la table:

Pour assurer le maintien de la cohérence des données de la base, nous devons définir des ***contraintes d'intégrité***.

- Si on veut ajouter une **clé primaire** pour la table précédente
  ALTER TABLE nom_table **ADD CONSTRAINT** nom_contrainte **PRIMARY KEY** (Attribut1);
- Si on veut ajouter une **clé étrangère** pour cette table
  ALTER TABLE nom_table **ADD CONSTRAINT** nom_contrainte **FOREIGN KEY (AttributN) REFERENCES** tableMère(Cléprimaire_tableMère)**;**
- Si on veut ajouter une **contrainte de vérification** pour cette table
  ALTER TABLE nom_table **ADD CONSTRAINT** nom_contrainte **CHECK (condition);**

2) *Méthode 2 : ajout de contraintes lors de la création de la table:*

a. Niveau colonne: *Les contraintes sont déclarées au fur et à mesure de la déclaration des attributs :*

```
CREATE TABLE nom_table (
                    Attribut1 domaine   PRIMARY KEY,
                    Attribut2 domaine  NOT NULL,
                    Attribut3 domaine  CHECK(condition),
    ……,
                    AttributN domaine REFERENCES tableMère(Cléprimaire_tableMère);
```

b. Niveau table: Les contraintes sont déclarées après la déclaration des attributs

```
CREATE TABLE nom_table (
                    Attribut1 domaine,
                     Attribut2 domaine,
                     Attribut3 domaine,
                        ……,
                    AttributN domaine,
CONSTRAINT nom_contrainte PRIMARY KEY(attribut1),
CONSTRAINT nom_contrainte CHECK(condition),
CONSTRAINT nom_contrainte FOREIGN KEY (attributN) REFERENCES tableMère(attCP));
```

3) *Syntaxe de Suppression et Changement de nom d'une table :*

```
DROP TABLE nom_table ;
RENAME ANCIEN_nom_table to NOUVEAU_Nom ;
```

4) *Syntaxe de modification, ajout, suppression de changement de nom d'une colonne :*

```
ALTER TABLE nom_table MODIFY nom_colonne modification_a_effectuée ;
ALTER TABLE nom_table ADD nom_colonne domaine [contrainte] ;
ALTER TABLE nom_table DROP COLUMN nom_colonne ;
ALTER TABLE nom_table RENAME COLUMN ancien_nom_colonne to nouveau nom_colonne;
```

5) *Domaine :*

Il y a essentiellement trois types de données en SQL:

a. **Char(n) et Varchar2(n) :** Chaîne de n caractères
b. **Number, Number(p)** (Nombre de p chiffres) **et Number(p,f)** (Nombre de p chiffres dont f chiffres après la virgule)
c. **Date :** Une valeur de type date s'écrit sur le modèle '21/02/2012'

C. Le Langage LMD (Le Langage de Manipulation des Données)

Les commandes de mise-à-jour sont: Insertion : **insert into,** Destruction : **delete,** Modification **update**

- **L'insertion** s'effectue avec la commande **INSERT INTO** dont la syntaxe est :
  INSERT INTO Nom_table (Cl1, Cl2, ...) **VALUES** (V1,V2,…);

Cette syntaxe permet d'insérer une ligne en indiquant les valeurs (V1,V2,…) à introduire pour chaque colonne spécifiées (en respectant l'ordre)**.** Les colonnes non spécifiées seront donc à NULL (ou à la valeur par défaut).

On peut aussi insérer toute la ligne sans spécifier les colonnes : **INSERT INTO** Nom_table **VALUES** (V1,..., **Vn**); Mais dans ce cas **L'ordre des colonnes doit rester identique sinon certaines valeurs prennent le risque d'être insérées dans la mauvaise colonne**.

***A noter** : lorsque le champ à remplir est de type **Chaîne de caractères** ou **Date** il faut indiquer le texte **entre côtes**. En revanche, lorsque la colonne est un numérique il suffit juste d'indiquer le nombre.*

Exemple 1 : Soit la table Emp (*mat, nom, adresse, salaire, #sup*)

INSERT INTO Emp (*mat, nom, salaire*) **VALUES** (34099, 'Arnaud', 456) ;//***Les attributs adresse et sup auront alors des valeurs NULL.***

Exemple2 : INSERT INTO VALUES (34098, 'Gilles', '3456, Gaspe, Mtl', 456, 68735)

Bien entendu le nombre d'attributs et le type de ces derniers doivent être cohérents.

- **Suppression** s'effectue avec la clause **DELETE** dont la syntaxe est :
  DELETE FROM <table> **[WHERE** condition**] ;**

Delete permet de supprimer des lignes dans une table. En utilisant cette commande associée à **WHERE** il est possible de sélectionner les lignes concernées qui seront supprimées.

Exemple 1 : Supprimer les employés dont le nom commence par M

DELETE FROM Emp **WHERE** nom LIKE 'M%' **;**

Exemple 2 : Supprimer les employés dont les numéros des superviseurs sont : 68754, 65841

DELETE FROM Emp **WHERE** sup IN (68754, 65841)**;**

- **La modification** des données s'effectue avec la clause **UPDATE** dont la syntaxe est :

UPDATE nom_table **SET** {<nomcl> = <expression>, <nomcl> = <expression>, …} **WHERE** <condition> ;

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

Exemple 1 : **UPDATE** Emp **SET** adresse = '3456, Gaspe, Mtl' **WHERE** mat = 34098;

Exemple 2 Augmenter le salaire de l'employé n° 34098, par 1000 dinars.

UPDATE Emp **SET** Salaire = Salaire + 1000 **WHERE** mat =34098;

✔ **Toutes les mises-à-jour (*insert, update, delete*) ne deviennent définitives qu'à l'issue d'une validation par COMMIT.**
✔ **Entre-temps elles peuvent être annulées par ROLLBACK.**

D. Le Langage d'Interrogation des Données LID (Accès aux données L'ordre SELECT) :

Syntaxes: SELECT [ALL | DISTINCT] {<liste_attributs> | *} **FROM** <liste_tables> **WHERE** <condition> ;

(La Clause **WHERE** Permet de restreindre les lignes)

Un ordre SELECT permet d'extraire des informations d'une base de données dont l''utilisation offre les possibilités suivantes :

Projection : SQL permet de choisir dans une table, ***les colonnes*** que l'on souhaite ramener (afficher) au moyen d'une requête. Vous pouvez déterminer autant de colonnes que vous le souhaitez :

SELECT Cl1, Cl2,… FROM R ;

Sélection : SQL permet de choisir dans une table, ***les lignes*** que l'on souhaite ramener au moyen d'une requête. Divers critères de sélection sont disponibles à cet effet :

SELECT * FROM R WHERE C ;

Jointure : SQL permet de joindre des données stockées dans différentes tables, en créant un lien par le biais **d'une colonne commune** à chacune des tables :

SELECT * FROM R, S WHERE R.C1 = S.C1 and … R.Cl_i = S.Cl_i ; avec Ci attributs communs à R et S

Exemple 1 : soient les tables suivantes :

Emp (numE, nom, adresse, #dept,# sup) **Projet (nump, designation, budget)**

Depart (dept, #dir, appellation) **Role (#num, #nump, role_emp)**

- Donner les noms et adresses des employés : **SELECT** nom, adresse **FROM** Emp;
- Donner la liste des projets : **SELECT** * **FROM** Projet;
- Donner les noms des employés et leurs numéros départements : **SELECT** nom, dept **FROM** Emp;

Selon la condition, la clause **WHERE** peut traduire l'opération restriction ou jointure dans l'algèbre relationnelle

Exemple 2: Donner les noms et adresses des employés qui travaillent au 'DIRO'

SELECT E.nom, **E.**adresse **FROM** Emp **E,** Depart **D WHERE** D.appellation = 'DIRO' **AND E.dept=D.dept;**

Fonctions de calcul :

Ce sont des fonctions qui permettent de faire des calculs et qui s'utilisent dans les clauses : **select** et **having** : **COUNT, SUM, AVG, MAX, MIN**

Exemples :

- Donner le nombre d'employés qui travaillent au département DIRO

SELECT COUNT (*) **FROM** Emp, Depart **WHERE** appellation = 'DIRO' **AND** Emp.dept=Depart.dept;

- Donner le budget total des projets sous la responsabilité de l'employé numéro 35677

SELECT SUM (budget) **FROM** Role R, Projet P

WHERE P.nump = R.nump **AND** role_emp ='responsable' **AND** R.num = 35677;

Forme de conditions

- **IN (NOT IN) :** **Exemple :** Donner la liste des directeurs de départements :

SELECT * **FROM** Emp **WHERE** numE **IN** (**SELECT** dir **FROM** Departement);

- **ANY** ou **ALL :** **Exemple :** Donner la liste des projets qui ont un budget supérieur au budget d'au moins un des projets impliquant l'employé numéro 34888

SELECT * **FROM** Projet **WHERE** budget > **ANY** (**SELECT** budget **FROM** Role R, Projet P

WHERE R.nump = P.nump AND R.num = 34888);

- **IS (NOT) NULL :** **Exemple :** Donner la liste des employés sans superviseurs : **SELECT** * **FROM** Emp **WHERE** sup IS NULL;

- **(NOT) EXISTS :** **Exemple :** Donner les noms des employés qui ne sont impliqués dans aucun projet :

SELECT nom **FROM** Emp E **WHERE** ***NOT EXISTS*** (**SELECT** * **FROM** Projet P **WHERE** E.numE = P.nump);

- Clause **de regroupement** : la Clause est : **GROUP BY <liste_attributs>**

Exemple : Donner les noms des employés regroupés par projet

SELECT nom **FROM** Emp E, Role R **WHERE** E.numE = R.num **GROUP BY** R.nump;

- Clause : **HAVING <condition>**

Exemple : Donner les numéros de projets dans lesquels interviennent au moins dix employés autres que le responsable

SELECT nump **FROM** Role **WHERE** role_emp!= 'responsable' **GROUP BY** nump

HAVING *COUNT*(nume) > 9;

- **Ordonnancement des résultats** : la Clause est : **ORDER BY** {<attribut> [ASC | DESC]}+

Exemple Donner la liste des projets dans lesquels participe l'employé 33549 par ordre décroissant de budget **SELECT** * **FROM** Projet **WHERE** nume = 33549 **ORDER BY** budget DESC;

Syntaxes générales

- **SELECT** {<attribut$_1$>,… {<attribut$_n$>| *} **FROM** {table| table$_1$,…table$_j$}
  [**WHERE** < liste_conditions> AND <liste_jointures>] [**GROUP BY** <liste_attributs>]

  [**HAVING** <condition_groupe>] [**ORDER BY** {<attribut> [ASC | DESC]}]**;**

Si on extrait des données provenant de n tables alors on doit avoir au moins (n-1) conditions de jointures.