



Université IBN TOFAIL

Faculté des sciences

Département d'Informatique

BASE DE DONNEES II

SMI – S6

Année universitaire : 2021/2022

Pr. My EL HASSAN CHARAF

PLAN

- I- **Principes de fonctionnement des (SGBD) (Rappels)**
 - Introduction aux bases de données
 - Modèles de base de données
 - Système de gestion de base de données (SGBD)
 - Création de tables, contraintes et relations
 - Création et gestion des vues, synonymes, priviléges
- II- **PL/SQL : Extension au standard SQL**
 - Extensions au standard SQL (Délimiteur, contrôle flux, boucles)
 - Structures de contrôles (conditionnelles, répétitives)
 - Tableaux et Structures
 - Sous programmes : Procédures et Fonctions
 - Transactions et validation des données
- III- **PL/SQL : Les Exceptions**
 - Les Exceptions : Prédéfinies
 - Les Exceptions : utilisateurs
- IV- **PL/SQL : Curseurs et Déclencheurs**
 - Les curseurs : Mécanisme, procédure d'utilisation
 - Les déclencheurs : Simple, MAJ, attributs : when, :old, :new.
- V- **JDBC [Connexion JAVA-Oracle]**
 - Problématique
 - JDBC : Fonctionnement
 - Exemples
- VI- **TD & TP**
 - TD 1, TD2
 - TP

ANNEXE

BIBLIOGRAPHIE

I- Principes de fonctionnement des (SGBD) (Rappels)

1. Introduction aux bases de données

Bases de Données : Collection de données stockées dans des fichiers accessibles pour plusieurs utilisateurs. C'est une collection d'informations organisées afin d'être facilement consultables, gérables et mises à jour.

Ces données représentent des informations servant aux activités et au management d'une entreprise. Elles contiennent généralement des agrégats d'enregistrements ou de fichiers de données, contenant des informations sur les transactions de vente ou les interactions avec des clients spécifiques.

2. Modèles de base de données

L'histoire des bases de données remonte aux années 1960, avec l'apparition des bases de données réseau et des bases de données hiérarchiques :

Modèles de Bases de Données : Abstraction mathématique selon laquelle l'utilisateur voit les données. Exemples :

- ❖ **Modèle hiérarchique**
- ❖ **Modèle réseau**
- ❖ **Modèle relationnel**
- ❖ **Modèle Objet**

a. **Modèle hiérarchique** : Une base de données hiérarchique est une forme qui lie des enregistrements dans une structure arborescente de façon à ce que chaque enregistrement n'ait qu'un seul possesseur. Cependant, à cause de leurs limitations internes, elles ne peuvent pas souvent être utilisées pour décrire des structures existantes dans le monde réel.
→ Si le principe de relation « 1 vers N » n'est pas respecté (par ex., un malade peut avoir plusieurs médecins et un médecin a, *a priori*, plusieurs patients): la hiérarchie se transforme en un réseau.

b. **Modèle réseau** : Le modèle réseau est en mesure de lever de nombreuses difficultés du modèle hiérarchique grâce à la possibilité d'établir des liaisons de type n-n. Les liens entre objets pouvant exister sans restriction. Pour retrouver une donnée dans une telle modélisation, il faut connaître le chemin d'accès

c. **Modèle relationnel** : Une base de données relationnelle est une base de données structurée suivant les principes de l'algèbre relationnelle.

Edgar Frank Codd (Fondateur) : Mathématicien de formation, Chercheur chez IBM à la fin des années 1960, il était persuadé qu'il pourrait utiliser la théorie des ensembles et la logique des prédictats du premier ordre pour résoudre des difficultés telles que la redondance des données, l'intégrité des données ou l'indépendance de la structure de la base de données avec sa mise en œuvre physique. Un premier prototype de Système de gestion de bases de données relationnelles (SGBDR) a été construit dans les laboratoires d'IBM. Depuis les années 80, cette technologie a mûri et a été adoptée par l'industrie.

d. **Modèle Objet** : La notion de *bases de données objet* est plus récente et encore en phase de recherche et de développement. Elle sera très probablement ajoutée au modèle relationnel.

3. Système de gestion de base de données (SGBD)

a. **Définition d'un SGBD** : Ensemble de programmes qui permettent à des utilisateurs de créer et maintenir une base de données

b. **Rôle d'un SGBD** : Un SGBD est en général, multi utilisateurs, multitâches.

- Il permet l'accès à la base à plusieurs utilisateurs simultanément
- Il traite en les optimisant les requêtes utilisateurs
- Il gère l'exécution cohérente de plusieurs programmes simultanés (accès concurrents)
- Il assure l'intégrité la sécurité, et la protection des données
- Il offre des moyens d'interaction, langages et interfaces, faciles à utiliser

c. **Niveaux d'un SGBD** : La plupart des SGBD suivent l'architecture standard ANSI/Sparc qui permet d'isoler les différents niveaux d'abstraction nécessaires pour un SGBD.

- **Niveau interne ou physique** : Décrit le modèle de stockage des données et les fonctions d'accès.
- **Niveau conceptuel ou logique** : Décrit la structure de la base de données globalement.
 - Le schéma conceptuel est produit par une analyse de l'application à modéliser et par intégration des différentes vues utilisateurs.
 - Ce schéma décrit la structure de la base indépendamment de son implantation.
- **Niveau externe** : Correspond aux différentes vues des utilisateurs. Chaque schéma externe donne une vue sur le schéma conceptuel à une classe d'utilisateurs.

Le SGBD doit être capable de faire des transformations entre chaque niveau, de manière à transformer une requête exprimée en terme du niveau externe en requête du niveau conceptuel puis du niveau physique.

d. **EXEMPLE : Présentation du SGBD Oracle**

Les couches Oracle : Par définition, un système de gestion des bases de données est un ensemble de programmes destinés à gérer les fichiers. Oracle est constitué essentiellement des couches :

- ❖ **Le noyau**
- ❖ **Le dictionnaire de données**
- ❖ **La couche SQL**
- ❖ **La couche PL/SQL**

Le noyau dont le rôle est l'optimisation dans l'exécution des requêtes, la gestion des accélérateurs (index et Clusters), le stockage des données, la gestion de l'intégrité des données la gestion des connexions à la base de données et l'exécution des requêtes.

Le dictionnaire de données est une métabase qui décrit d'une façon dynamique la base de données. Il permet de décrire les objets suivants : Les objets de la base de données (Tables, SYNONYMES, VUES, COLONNES, ...), les utilisateurs accédant à la base de données avec leurs priviléges (CONNECT, RESOURCE et DBA). Ainsi toute opération qui affecte la structure de la base de données provoque automatiquement une mise à jour du dictionnaire.

La couche SQL. Elle joue le rôle d'interface entre le noyau et les outils d'oracle. Ainsi, tout accès à la base de données est exprimé en langage SQL. Le rôle de cette couche est d'interpréter les commandes SQL de faire la vérification syntaxique et sémantique et de les soumettre au noyau pour exécution.

La couche PL/SQL : Cette couche est une extension de la couche SQL puisque le PL/SQL est une extension procédurale du SQL.

e. Les commandes SQL : Il existe principalement quatre types de commandes SQL

- **Langage de définition des données (LDD)** : Ces commandes permettent de créer ou de modifier un objet de la base de données. Exemples : CREATE, ALTER, DROP, MODIFY
- **Langage de manipulation des données (LMD)** : Ces commandes permettent de faire la mise à jour ou la consultation des données dans une table de la base de données. Exemples : SELECT, UPDATE, INSERT, DELETE
- **Langage de contrôle de données (LCD)** : Exemples : GRANT et REVOKE
- **Langage de transaction (TCL)** : Ces commandes permettent de gérer les modifications apportées aux données de la base de données. Exemples : COMMIT, et ROLLBACK

f. Les objets manipulés par Oracle. Oracle supporte plusieurs types d'objets, voici quelques-uns :

- **Les tables** : objets contenant les données des utilisateurs ou appartenant au système. Une table est composée de colonnes (Champs ou attributs) et de lignes (enregistrements ou occurrences).
- **Les vues** : Une vue est une table virtuelle issue du modèle externe contenant une partie d'une ou plusieurs tables.
- **User** : Utilisateurs du système oracle. Chaque usager est identifié par un nom d'usager et un mot de passe.
- **Les séquences** : Générateur de numéro unique
- **Synonymes** : Autre nom donné aux objets Table, vue, séquence et schéma.
- **Les procédures** : Programme PL/SQL prêt à être exécuté.
- **Les déclencheurs** : Procédure déclenchée avant toute modification de la base de données.

4. Création des tables, contraintes et relations

a. Création des Tables

- ❖ Pour créer une table, on utilise la commande : CREATE TABLE. Dans cette commande, il est nécessaire d'indiquer, au minimum, le nom de la table, les noms des colonnes et le type de données. Par exemple :

```
CREATE TABLE ma_premiere_table  
    ( premiere_colonne text,  
      deuxième_colonne integer );
```

- ❖ Pour supprimer une table, on utilise la commande : DROP TABLE.

```
DROP TABLE ma_premiere_table ;
```

Convention d'écriture (ORACLE) : (Les <> indique une obligation, les () pourra être répétées plusieurs fois (séparées par une virgule) et les [] indique une option).

Syntaxe simplifiée : `CREATE TABLE <nom_de_table> (<nom_de_colonne><type_de_données>);`

Syntaxe générale : `CREATE TABLE <nom_de_table> (<nom_de_colonne> <type_de_données>`

```
[DEFAULT <valeur>]  
[ [CONSTRAINT <nom_de_contrainte>]  
NULL Ou  
NOT NULL OU  
UNIQUE OU  
PRIMARY KEY OU  
FOREIGN KEY OU  
REFERENCES <Nom_de_Table><nom_de_colonne> OU  
[ON DELETE CASCADE] OU  
CHECK <nom_de_condition> ] );
```

Création des Tablespaces sous Oracle : L'architecture Oracle ne comprend qu'une seule base par serveur, dans laquelle peuvent se trouver plusieurs tablespaces, équivalents des objets bases de données d'autres SGBD comme MySQL et MS-SQL, contenant des tables et procédures stockées.

Exemple :

```
CREATE TABLESPACE My_table_Sp
DATAFILE 'C:\oraclexe\app\oracle\oradata\my_table_sp.dbf' size 10M reuse
DEFAULT STORAGE (INITIAL 10K NEXT 50K MINEXTENTS 1 MAXEXTENTS 999)
ONLINE;
```

Pour lister les tables spaces qui existent :

```
Select tablespace_name, file_name, bytes from dba_data_files;
```

Il y a plusieurs Tablespaces par défaut sous Oracle :

- ❖ **SYSTEM** : les objets du système.
- ❖ **SYSAUX** : depuis la 10g, tablespace auxiliaire pour certains objets du système
- ❖ **TEMP** : tables temporaires pour les tris.
- ❖ **UNDO** : depuis la 9i, utilisé pour les transactions (commit, rollback)
- ❖ **USERS** : c'est le tablespace où sont créés les objets des utilisateurs par défaut.

b. **Création des Contraintes**

Les types de données sont un moyen de restreindre la nature des données qui peuvent être stockées dans une table. Toutefois, les contraintes fournies sont trop grossières.

Exemple : Une colonne du prix d'un produit ne doit accepter que des valeurs positives. Mais il n'y a pas de type de données qui n'accepte que des valeurs positives. Un autre problème peut provenir de la volonté de contraindre les données d'une colonne par rapport aux autres colonnes ou lignes. D'où l'importance de définir des contraintes sur les colonnes et les tables.

b.1. **Contrainte : NOT NULL**

Une contrainte NOT NULL indique simplement qu'une colonne ne peut pas prendre la valeur NULL. Si cette contrainte n'est pas précisée alors par défaut la valeur est NULL. Exemple :

```
CREATE TABLE produits (
    no_produit integer NOT NULL,
    nom text NOT NULL,
    prix numeric);
```

b.2. **Contrainte : PRIMARY KEY**

Une contrainte de type clé primaire (**PRIMARY KEY**) indique qu'une colonne, ou un groupe de colonnes, peut être utilisée comme un identifiant unique de ligne pour cette table. Lorsque la clé primaire est une clé primaire composée, la contrainte doit être définie au niveau de la table et non au niveau de la colonne. Les attributs faisant partie de la clé doivent être entre parenthèse. La contrainte de **PRIMARY KEY** assure également les contraintes de NOT NULL et UNIQUE. Exemples:

```
CREATE TABLE Etudiants (
    numad NUMBER (10) CONSTRAINT pk1 PRIMARY KEY, Nom VARCHAR2(20) NOT NULL,
    Prenom VARCHAR2 (20)
);
```

La contrainte **PRIMARY KEY** est définie par le mot réservé **CONSTRAINT** suivi du nom de la contrainte. Dans cet exemple la contrainte est définie en même temps que la colonne NUMAD, donc c'est une contrainte sur la colonne

```
CREATE TABLE Etudiants (
    numad NUMBER (10),
    Nom VARCHAR2(20) NOT NULL,
    Prenom VARCHAR2 (20),
    CONSTRAINT pk1 PRIMARY KEY (numad));
```

Dans cet exemple, la contrainte de PRIMARY KEY est définie comme si on définirait une colonne.
C'est une contrainte sur la table.

b.3. Contrainte : FOREIGN KEY

Une contrainte de type clé étrangère (FOREIGN KEY) stipule que les valeurs d'une colonne (ou d'un groupe de colonnes) doivent correspondre aux valeurs qui apparaissent dans les lignes d'une autre table. Cette contrainte indique que la valeur de l'attribut correspond à une valeur d'une clé primaire de la table spécifiée. Dans ce cas, la clé primaire de l'autre table doit être obligatoirement créée pour que cette contrainte soit acceptée. D'autant plus, la clé primaire de l'autre table et l'attribut défini comme clé étrangère doivent être de même type et de même longueur.

Remarque : On peut également préciser l'option **ON DELETE CASCADE** (Voir : **Création des tables-syntaxe générale**) qui indique que les enregistrements soient détruits lorsque l'enregistrement correspondant à la clé primaire de la table référencée est supprimé. Exemple:

```
CREATE TABLE programme (
    codePrg VARCHAR2(3) CONSTRAINT pk3 PRIMARY KEY,
    nomProg VARCHAR2(20));
```

```
CREATE TABLE etudiants (
    NumAd NUMBER CONSTRAINT pk4 PRIMARY KEY,
    Nom VARCHAR2(20),
    Prenom VARCHAR2(20),
    codePrg VARCHAR2(3),
    CONSTRAINT fk1 FOREIGN KEY(codePrg) REFERENCES programme (codePrg);
```

b.4. Contrainte de vérification : CHECK

Pour obliger par exemple les prix des produits à être positifs, on peut utiliser :

```
CREATE TABLE produits (
    no_produit integer,
    nom text,
    prix numeric CHECK (prix > 0));
```

Une contrainte de vérification peut aussi faire référence à plusieurs colonnes.

Exemple : Dans le cas d'un produit, on peut vouloir stocker le prix normal et un prix réduit en s'assurant que le prix réduit soit bien inférieur au prix normal :

```
CREATE TABLE produits (
    no_produit integer,
    nom text,
    prix numeric CHECK (prix > 0),
    prix_promo numeric CHECK (prix_promo > 0),
    CHECK (prix > prix_promo));
```

b.5. Contrainte : DEFAULT

La contrainte DEFAULT : indique la valeur par défaut que prendra l'attribut si aucune valeur n'est saisie.

b.6. Contrainte d'unicité : UNIQUE

Les contraintes d'unicité garantissent l'unicité des données contenues dans une colonne ou un groupe de colonnes par rapport à toutes les lignes de la table. La syntaxe est :

Contrainte colonne : *CREATE TABLE produits (*

```
    no_produit integer UNIQUE,  
    nom text,  
    prix numeric);
```

Contrainte table : *CREATE TABLE produits (*

```
    no_produit integer,  
    nom text,  
    prix numeric,  
    UNIQUE (no_produit));
```

c. Création des Relations (Jointures)

Les relations (Jointures) permettent d'exploiter pleinement le modèle relationnel des tables d'une base de données. Elles sont faites pour mettre en relation deux (ou plus) tables concourant à rechercher la réponse à des interrogations. Une jointure permet donc de combiner les colonnes de plusieurs tables. Les jointures normalisées s'expriment à l'aide du mot clef JOIN dans la clause FROM. Suivant la nature de la jointure, on devra préciser sur quels critères se base la jointure :

c.1. Jointures internes

Il s'agit de la plus commune des jointures. C'est celle qui s'exerce par défaut si on ne précise pas le type de jointure. Après le mot-clé ON, nous devons préciser le critère de jointure. Exemple :

```
SELECT CLI_NOM, TEL_NUMERO  
FROM T_CLIENT INNER JOIN T_TELEPHONE  
ON T_CLIENT.CLI_ID = T_TELEPHONE.CLI_ID
```

c.2. Jointures externes

Les jointures externes sont extrêmement pratiques pour rapatrier le maximum d'informations disponibles, même si des lignes de table ne sont pas renseignées entre les différentes tables jointes. La syntaxe de la jointure externe est la suivante :

```
SELECT ... FROM <table gauche>  
LEFT | RIGHT | FULL OUTER JOIN <table droite 1>  
ON <condition de jointure>  
[LEFT | RIGHT | FULL OUTER JOIN <table droite 2>  
ON <condition de jointure 2>] ...
```

Les mots clefs LEFT, RIGHT et FULL indiquent la manière dont le moteur de requête doit effectuer la jointure externe. Ils font référence à la table située à gauche (LEFT) du mot clef JOIN ou à la table située à droite (RIGHT) de ce même mot clef. Le mot FULL indique que la jointure externe est bilatérale.

c.3. Jointures croisées

La jointure croisée CROSS JOIN n'est autre que le produit cartésien de deux tables. Rappelons que le produit cartésien de deux ensembles n'est autre que la multiplication généralisée. Dans le cas des tables, c'est le fait d'associer à chacune des lignes de la première table, toutes les lignes de la seconde. La syntaxe de la jointure croisée est la suivante :

```
SELECT colonnes  
FROM T_1 CROSS JOIN T_2
```

c.4. Jointures UNION

La jointure d'union **UNION JOIN** permet de faire l'union de deux tables de structures quelconque. En fait, c'est comme si l'on avait listé la première table, puis la seconde en évitant toute colonne commune et compléter les espaces vides des valeurs NULL. Syntaxe :

```
SELECT colonnes  
FROM T_1 UNION JOIN T_2
```

d. Requêtes SELECT avec les opérateurs d'ensembles

d.1. L'opérateur INTERSECT

Il permet de ramener l'intersection des données entre deux tables. Les deux commandes SELECT doivent avoir le même nombre de champs, et des champs de même type et dans le même ordre.

```
Instruction SELECT1  
INTERSECT  
Instruction SELECT 2  
[ORDER BY]
```

Le nombre de colonnes renvoyées par SELECT 1 doit être le même que celui renvoyé par SELECT 2 et le type de données SELECT 1 doit être le même que celui de SELECT 2. La clause optionnelle ORDER BY doit se faire selon un numéro de colonne et non selon le nom. SELECT 1 et SELECT 2 ne peuvent contenir des clauses ORDER BY.

Exemple : La requête permet de ramener tous les étudiants qui sont en même temps des enseignants.

```
SELECT NOM, PRENOM FROM ETUDIANTS  
INTERSECT  
SELECT NOM, PRENOM FROM ENSEIGNANTS  
ORDER BY 1
```

d.2. L'opérateur UNION

Cet opérateur permet de renvoyer l'ensemble des lignes des deux tables. Si des lignes sont redondantes elles sont renvoyées une seule fois. Pour renvoyer toutes les lignes, utiliser l'option ALL. Les mêmes contraintes qui s'appliquent pour INTERSECT s'appliquent pour UNION. Syntaxe :

```
Instruction SELECT1  
UNION [ALL]  
Instruction SELECT 2  
[ORDER BY]
```

Exemple : La requête suivante permet de ramener tous les étudiants et tous les enseignants. Les enseignants qui sont en même temps des étudiants sont ramenés une seule fois.

```
SELECT NOM, PRENOM FROM ETUDIANTS  
UNION  
SELECT NOM, PRENOM FROM ENSEIGNANTS  
ORDER BY 1
```

d.3. L'opérateur MINUS

Cet opérateur renvoie l'ensemble des lignes de la première table MOINS les lignes de la deuxième table. Les mêmes contraintes qui s'appliquent pour INTERSECT s'appliquent pour MINUS.

```
Instruction SELECT1  
MINUS  
Instruction SELECT 2  
[ORDER BY]
```

5. Création et gestion des vues, synonymes, priviléges

a. Création et gestion des vues

Une vue est essentiellement une requête nommée à laquelle on donne un nom et qui s'utilise comme une table. **SYNTAXE :**

**CREATE OR REPLACE VIEW ma_vue AS SELECT colonnes FROM T_1
Where condition**

C'est la définition de la vue qu'on stocke dans le SGBD, pas le résultat de l'évaluation de la requête.
→ **DROP VIEW ma_vue** : ne supprime que la définition.

Exemple:

➤ **CREATE OR REPLACE VIEW V_EMP_DEP AS
SELECT * FROM EMP WHERE dep = 1;
SELECT * FROM V_EMP_DEP;**

Pour des raisons de performances, on peut avoir intérêt à volontairement enregistrer le résultat de la vue, on parle alors de vue matérialisée :

CREATE MATERIALIZED VIEW...

Attention à la taille volumineuse des vues matérialisées (en présence de jointure)

b. Création des Synonymes

Les synonymes est une autre désignation pour les objets (vue, tables, séquences...) de la base de données. Un synonyme est utilisé pour faciliter l'accès à un objet. Par exemple au lieu d'accéder à une table via un chemin d'accès (User.employes) on utilise un synonyme. Un synonyme peut être public, ou privé. Par défaut un synonyme est privé. La création d'un synonyme PUBLIC ne peut se faire que par l'administrateur.

SYNTAXE : **CREATE [PUBLIC] SYNONYM <nom_du_synonyme> FOR <nom_objet>**

Exemple: **CREATE PUBLIC SYNONYM syemp FOR scott.emp;**

Ainsi au lieu de faire : **SELECT * FROM scott.emp;** → On peut faire : **SELECT * FROM syemp;**

La création d'un synonyme non public se fait comme suit :

CREATE SYNONYM <nom_du_synonyme> FOR <nom_objet>

c. Création des priviléges

Les priviléges : La commande **GRANT** permet d'attribuer des droits à des utilisateurs sur vos objets. La commande **REVOKE** permet de supprimer des priviléges. **SYNTAXE :**

**GRANT <privilege>[ou ALL] ON <no_objet>
TO <nom_usager> [ou PUBLIC]
[WITH GRANT OPTION]**

- Les priviléges sont : SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX, REFERENCES ou (ALL).
- On peut attribuer plusieurs priviléges, mais il faut les séparer par une virgule.
- On peut préciser plusieurs colonnes, séparées par une virgule

WITH GRANT OPTION, permet à l'usager à qui on vient d'attribuer des droits d'attribuer les mêmes droits à d'autres usagers sur le même objet.

PUBLIC permet d'attribuer les droits à TOUS.

EXEMPLES : GRANT SELECT, INSERT ON etudiant TO mathieu
GRANT ALL ON employe TO PUBLIC
GRANT SELECT ON department TO martin WITH GRANT OPTION.

La commande REVOKE permet de supprimer des privilèges. **SYNTAXE :**

REVOKE <privilege> [ou ALL] ON <no_objet> FROM <nom_usager> [ou PUBLIC]

EXEMPLES : REVOKE INSERT ON etudiant FROM Mathieu
REVOKE ALL ON employe TO PUBLIC.

d. Création des rôles

Un rôle représente un ou plusieurs privilèges. On crée des rôles lorsqu'un ensemble de privilèges vont être attribués à plusieurs usagers.

SYNTAXE : CREATE ROLE <nom_du_role>

EXAMPLE : CREATE ROLE role1

Attribuer des privilèges à un ROLE

SYNTAXE : GRANT <privileges> ON <nom_objet> TO <nom_role>

EXAMPLE : GRANT SELECT, UPDATE, INSERT ON etudiants TO role1

Attribuer un ROLE à un utilisateur.

SYNTAXE : GRANT <nom_role> TO <nom_usager>

EXAMPLE : GRANT role1 TO scott, martin, mathieu.

Détruire un ROLE : DROP <nom_du_role>

Il existe des rôles déjà prédéfinis.

- Le rôle **CONNECT** permet à un usager de se connecter à la base de données. Il n'a de sens que si d'autres privilèges lui sont accordés.
- Le rôle **RESOURCES** permet à un utilisateur de créer ses propres objets.
- Le rôle **DBA** permet d'attribuer à un usager des rôles d'administration.