Collège Lionel-Groulx

Département d'informatique

Cours : 420-KB6-LG, introduction aux bases de données

Hiver 2021

Introduction aux Bases de données

SQL avec Oracle 11g

Yacoub Saliha
COLLÈGE LIONEL-GROULX

Table des matières

Historique des versions

Numéro de version	Tâches/modifications	Auteur	Date
1.0	Chapitres 1-6	Saliha Yacoub	Janvier 2019
1.1	Chapitre 7, 8	Saliha Yacoub	Février 2019
1.2	Chapitres 9,10, 11 et 12	Saliha Yacoub	Mars 2019
2.0	Ajout d'exemples CREATE TABLE l'option IDENTITY	Saliha Yacoub	Janvier 2021

Chapitre 1, introduction

Bref historique

Les données représentent une ressource organisationnelle et informationnelle cruciale que l'entreprise se doit de maîtriser. La majorité des organisations ne saurait réussir sans connaître les données exactes de leur entreprise et de leur environnement externe

Pendant très longtemps, les systèmes d'information des entreprises structuraient leurs données sous forme de fichiers, chaque application utilisait donc son propre fichier. Beaucoup de problèmes étaient connus de cette façon de gérer les données d'une entreprise : la redondance d'information, la dépendance des données des traitements, et le manque d'intégrité des données. Devant le volume d'information grandissant et la complexité des traitements, le stockage des données dans de simples fichiers n'est plus la solution pour le stockage du patrimoine informationnel de l'entreprise. Un moyen de palier aux problèmes de la gestion des fichiers est le SGBD. (Systèmes de Gestion de Bases de Données).

Vers la fin des années 60 et le début des années 70, sont apparus les premiers SGBDs hiérarchiques et réseaux. Vers le milieu des années 70, nous avons vu naître les SGBDs relationnels, qui utilisent le modèle relationnel de Edgar Frank « Ted » Codd. Aujourd'hui les SGBD relationnels sont présents dans la majorité des entreprises.

Concepts de bases

1. Définitions

Une base de données est un ensemble de données modélisant les objets d'une partie du monde réel et servant de support à une application informatique.

Un Système de gestion de base de données, SGBD, peut-être perçu comme un ensemble de logiciels système permettant aux utilisateurs d'insérer (stocker), de modifier et de rechercher efficacement des données spécifiques dans une grande masse d'information partagée par de multiples utilisateurs.

2. Objectifs des SGBDs

- Non redondance des données : permet de réduire le risque d'incohérence lors des mises à jour, de réduire les mises à jour et les saisies.

- Partage des données : ce qui permet de partager les données d'une base de données entre différentes applications et différents usagers.
- Cohérence des données : ce qui permet d'assurer que les règles auxquelles sont soumises les données sont contrôlées surtout lors de la modification des données.
- Sécurité des données : ce qui permet de contrôler les accès non autorisés ou mal intentionnés. Il existe des mécanismes adéquats pour autoriser, contrôler ou d'enlever des droits à n'importe quel usager à tout ensemble de données de la base de données.
- Indépendance physique : assurer l'indépendance des structures de stockage au structure des données du monde réel.
- Indépendance logique : possibilité de modifier un schéma externe sans modifier le conceptuel. Cette indépendance assure à chaque groupe les données comme il le souhaite à travers son schéma externe appelé encore une VUE.

SGBDS relationnels :

1. Présentation et définitions

Le modèle relationnel (mis de l'avant par Edgar Frank Codd) est le plus implémenté et le plus stable des modèles actuellement utilisés. Dans ce modèle l'unité de stockage élémentaire est la **« table ».** En général, c'est aux tables que les utilisateurs font référence pour accéder aux données.

Une base de données relationnelle est constituée de plusieurs tables reliées entre elles.

Une table est constituée de lignes et de colonnes.

Une colonne de la table définit un **attribut**. Les attributs servent à définir la structure de la table.

Une ligne de la table définit un **enregistrement.** Les enregistrements de la table définissent le contenu de celle-ci

Exemple

Voici la **table Livres** qui contient des informations par rapport à des livres de notre bibliothèque.

NumeroLivre	TitreLivre	Auteur	Langue
101	Ainsi parlait Zarathoustra	Friedrich Nietzsche	Allemand
102	La paix des profondeurs	Aldous Huxley	Anglais

103	Les misérables	Victor Hugo	Français
104	La liberté n'est pas une marque de yogourt	Pierre Falardeau	Français
105	Madame Bovary	Gustave Flaubert	Français

Explications :

Le nom de la table est Livres.
La table possède 4 attributs qui sont : NumeroLivre,TitreLivre,Auteur et Langue.
La table a 5 enregistrements ou 5 lignes qui sont les livres de notre bibliothèque.

2. Autres définitions

Termes	**Définitions et représentation**
Domaine	Ensemble de valeur caractérisé par un nom. Exemple le domaine de noms de jeux vidéo peux être Jeux {Word of Warcraft, Starcraft, Vampire, Civilization }
Table	Objet d'une base de données relationnelle. Ensemble de lignes et de colonnes.
Attribut	Sous-groupe d'information à l'intérieure d'une table Code_permanent, Nom Un attribut peut être considéré comme une colonne d'une table
Valeur d'un attribut	La valeur de l'attribut en rapport avec le domaine de valeurs. Exemple valeur de l'attribut Nom est Martin
Occurrence	Instance d'une table (ligne ou enregistrement d'une table)
Clé primaire	Identifiant dans une table. Attribut permettant d'identifier chaque occurrence(enregistrement) d'une table de marinière unique. Ou encore attribut permettant d'identifier chaque ligne d'une table d'une manière unique.
Clé étrangère	Attribut d'une table (table A) qui est clé primaire d'une autre table (table B)

3. Représentation d'une table :

Nom de la table
Liste des attributs

Etudiants
Numad Nom Prenom Adresse

4. Exemples de SGBD relationnels :

- Oracle
- MSSQL Server
- DB2
- MYSQL
- MariaDB
- PostgreSQL
- SQLite

Chapitre 2, présentation du SGBD Oracle

Architecture

Par définition un système de gestion des bases de données est un ensemble de programmes destinés à gérer les objets de la base de données.

Oracle est constitué essentiellement des couches suivantes :

1. Le noyau : ayant un rôle dans :
    a. L'optimisation dans l'exécution des requêtes
    b. La gestion des accélérateurs (index et Clusters)
    c. Le stockage des données
    d. La gestion de l'intégrité des données
    e. La gestion des connexions à la base de données
    f. L'exécution des requêtes

2. Le dictionnaire de données : le dictionnaire de données d'oracle est une métabase qui décrit d'une façon dynamique la base de données. Il permet ainsi de décrire les objets suivants :
    a. Les objets de la base de données (Tables, SYNONYMES, VUES, COLONNES, …)
    b. Les utilisateurs accédant à la base de données avec leurs privilèges (CONNECT, RESOURCE et DBA).

Ainsi toute opération qui affecte la structure de la base de données provoque automatiquement une mise à jour du dictionnaire.

3. La couche SQL : cette couche joue le rôle d'interface entre le noyau et les différents 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
4. La couche PL/SQL : cette couche est une extension de la couche SQL puisque le PL/SQL est une extension procédurale du SQL.

Le langage SQL

SQL pour Structured Query Language ou langage structuré de requêtes, est un langage qui permet de définir, de manipuler et de contrôler une base de données relationnelle. Il permet également d'extraire les données d'une base de données.
Pour définir la base de données, on utilise le langage DDL (DDL, Data Definition Language).
Pour manipuler la base de données on utilise le langage DML (DML, Data Manipulation Language).
Pour contrôler une base de données, on utilise le langage DCL (DCL, Data Control Language).

Développée chez IBM en 1970 par Donald Chamberlain et Raymond Boyce, cette première version a été conçue pour manipuler et éditer des données stockées dans la base de données relationnelles.

En 1979, Relational Software, Inc. (actuellement Oracle Corporation) présenta la première version commercialement disponible de SQL, rapidement imité par d'autres fournisseurs.

SQL a été adopté comme recommandation par l'Institut de normalisation américaine (ANSI) en 1986, puis comme norme internationale par l'ISO en 1987 sous le nom de *ISO/CEI 9075 - Technologies de l'information - Langages de base de données - SQL*.

On peut dire aussi, que SQL est un langage de commandes classées comme suit :

- Commande d'extraction des données : SELECT.
- Commandes de définition de données (DDL) : CREATE, ALTER, DROP, RENAME.
- Commandes de manipulation des données (DML) : INSERT INTO, DELETE, UPDATE. Avec ces commandes on peut utiliser les commandes COMMIT et ROLLBACK
- Commandes de control des données (DCL) : GRANT, REVOKE

Objets manipulés par Oracle

Oracle supporte plusieurs types d'objets, en voici quelques-uns :

Les tables : objets contenant les données des utilisateurs ou appartenant au système. Une table est composée de colonnes (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. Les vues sont utilisées pour ne permettre aux utilisateurs d'utiliser uniquement les données dont ils ont besoin.

User : Utilisateurs du système oracle. Chaque usager est identifié par un nom d'usager et un mot de passe.

Les séquences : générateurs de numéro unique

Synonymes : autre nom donné aux objets : Table, vue, séquence et schéma.

Les procédures et les fonctions : programme PL/SQL prêt à être exécuté.

Les déclencheurs : procédures déclenchées avant toute modification de la base de données (TRIGGER)

Conseils Généraux

- ✓ SQL n'est pas sensible à la casse, cependant il est conseillé d'utiliser les mots réservés (commandes, le type de données …) en majuscules.
- ✓ Il ne faut pas oublier le point-virgule à la fin de chaque ligne de commande.
- ✓ Utiliser les deux traits -- pour mettre une ligne en commentaire
- ✓ Utiliser /* et */ pour mettre plusieurs lignes en commentaire
- ✓ Utiliser des noms significatifs pour les objets que vous créez
- ✓ Ne pas utiliser de mots réservés comme noms d'objets (tables, vue, colonne….)
- ✓ Mettre une clé primaire pour chacune des tables que vous créez
- ✓ Si vous avez à contrôler l'intégrité référentielle, alors il faudra déterminer l'ordre dans lequel vous allez créer vos tables.

Attention : 

Le langage n'est pas sensible à la casse mais les données le sont. (Patoche différent de PATOCHE)

Convention d'écriture :

Les <> indique une obligation

Les () pourra être répété plusieurs fois, il faut juste les séparer par une virgule

Les [] indique une option.

Chapitre 3, les commandes SQL simples

La commande SELECT

La commande SELECT est la commande le plus simple à utiliser avec SQL. Cette commande n'affecte en rien la base de données et permet d'extraire des données d'une ou plusieurs tables. La syntaxe simplifiée n'utilise pas de jointure et elle se présente comme suit :

1. Syntaxe

```
SELECT <nom_de colonne1,…nom_de_colonnen>
FROM <nom_de table>
[WHERE <condition>]
[ORDER BY <nom_de_colonne>];
```

- La clause WHERE permet de cibler les enregistrements à extraire
- La clause ORDER BY spécifie le tri des données après extraction. Si l'ordre de tri n'est pas précisé alors le tri est par défaut croissant.
- Pour avoir un tri décroissant il faut ajouter l'option DESC.
- Le tri peut se faire selon plusieurs colonnes, il faut les séparer par des virgules
- Dans la commande SELECT, le joker * indique que toutes les colonnes seront sélectionnées.
- L'option AS permet de changer le nom de colonnes pour l'affichage uniquement.

2. Exemples :

```
SELECT * FROM employes;
SELECT nom, prenom FROM employes;
SELECT nom, prenom FROM employes ORDER BY nom;
SELECT num as Numéro, nom, prenom FROM employes ORDER BY nom, prenom;
```

3. Liste des opérateurs utilisés dans la clause WHERE

Cette clause permet de restreindre, en utilisant des critères, les enregistrements qui seront affectés par la requête. En d'autres mots, la requête SELECT ramène des résultats si la **condition ou** les **conditions** sont vérifiées.

Exemples :
On affiche le nom et le prénom de joueurs, s'ils sont du canadien de Montréal
On affiche le nom et le prénom des joueurs s'ils ont un salaire plus élevé que 1 000 000
On affiche on affiche le nom et le prénom des joueurs s'ils ont un salaire plus élevé que 1 000 000 ET qu'ils sont du canadien de Montréal.

Opérateurs	**Signification**	**Exemple**
=	Égalité	SELECT nom, prenom FROM employes WHERE nom ='Patoche';
<> ou != ou ^=	Inégalité ou différent	SELECT nom, prenom FROM employes WHERE nom !='Patoche';
>	Plus grand	SELECT nom, prenom FROM employes WHERE salaire >70000;
<	Plus petit	
>=	Plus grand ou égal	
<=	Plus petit ou égal	
LIKE	Si la valeur est comme une chaine de caractères. Le % est utilisé pour débuter ou compléter la chaine de caractère.	SELECT nom, prenom FROM employes WHERE nom LIKE 'Le%'; Va ramener tous les employés dont le nom commence par **Le**
NOT LIKE	Si la valeur n'est pas comme une chaine de caractères. Le % est utilisé pour débuter ou compléter la chaine de caractère.	SELECT nom, prenom FROM employes WHERE nom NOT LIKE 'Le%';
IN	Égal à une valeur dans une liste. Ramène des résultats si la valeur de	SELECT * FROM EMPLOYES where nom in('Patoche','Leroy');

	la condition est égale à au moins une des valeurs fournies par une liste.	Va ramener toutes les informations des employés Patoche et Leroy
NOT IN	N'est pas égal à une valeur dans une liste	
IS NULL	Si la valeur retournée est NULL (est vide). Attention NULL ne veut pas dire zéro.	SELECT nom, prenom FROM employes WHERE salaire is NULL; Ramène le nom et le prénom des employes qui N'ONT PAS de salaire. Si je veux ramener les employes qui ont un salaire égal à zéro alors la requête est : SELECT nom, prenom FROM employes WHERE salaire =0;
IS NOT NULL	Si la valeur retournée est n'est pas NULL	SELECT nom, prenom FROM employes WHERE salaire is NOT NULL;
BETWEEN x AND Y	Si la valeur est comprise entre x et y	SELECT * FROM employes WHERE salaire BETWEEN 9000 AND 16000 ;
NOT BETWEEN x AND y	Si la valeur n'est pas comprise entre x et y	
ANY	Si au moins une valeur répond à la comparaison	À voir avec les sous-requêtes
ALL	Si toutes les valeurs répondent à la comparaison	À voir avec les sous-requêtes
EXISTS	Si la colonne existe	À voir avec les sous-requêtes
NOT EXISTS	Si la colonne n'existe pas	À voir avec les sous-requêtes

Dans la clause WHERE, si la valeur de la condition est :

- De type caractère (CHAR ou VARCHAR2), alors elle doit être mise entre apostrophes. Si la chaine de caractère contient des apostrophes, ceux-ci doivent être doublés.
- De type numérique (NUMBER) alors la valeur est saisie en notation standard. La virgule décimale est remplacée par un point lors de la saisie
- De type date alors elle doit être mise entre apostrophes. Cependant il faudra connaître le format DATE de votre système. Pour saisir une date dans n'importe quel format, il faut s'assurer de la convertir dans le format avec la fonction TO_DATE (à voir plus loin) Il est possible
- D'utiliser une expression arithmétique dans la clause WHERE à condition que la colonne sur laquelle porte la condition soit de type numérique. (Avec des sous-requêtes)
- De combiner des opérateurs logiques dans la clause WHERE.

Exemples

```
SELECT nom, prenom from joueurs where CODEEQUIPE ='MTL' AND salaire >=1000000;
SELECT nom, prenom from joueurs where codeequipe ='MTL' OR codeequipe ='OTT';
```

La commande CREATE TABLE

Tous les objets de la base de données sont créés avec la commande CREATE. Cette commande est une commande DDL.

La commande CREATE TABLE permet de créer une table dans une base de données.
Pour créer une table nous avons besoins de connaître la liste de ses attributs et l'ensemble des contraintes sur la table. Pour chacun des attributs, nous avons besoin de connaitre le type des données et les contraintes.

1. Syntaxe simplifiée

Cette syntaxe ne sera pas la syntaxe qui sera utilisée. Une syntaxe plus complète est à la fin de ce chapitre.

```
CREATE  TABLE <nom_de_table> (<nom_de_colonne><type_de_données>);
```

Ou encore

Attention : 

La syntaxe que nous allons utiliser va permettre de définir les contraintes au niveau des colonnes et au niveau de la table.

Cette syntaxe est celle qui se rapproche de la syntaxe que nous allons utiliser.
Elle permet de définir les contraintes au niveau colonnes et au niveau table

```
CREATE TABLE nom_table
(
nom_colonne type_donnee_colonne [definition_contrainte_colonne],
nom_colonne type_donnee_colonne [definition_contrainte_colonne],
....
[definition_contrainte_table],
....
[definition_contrainte_table]
);
```

2. Types de données manipulés par Oracle

Le type de données représente la première contrainte à préciser lors de la création de table.

Pour chaque attribut ou champs de la table, on doit préciser le type de données. Les principaux types manipulés sont présentés dans le tableau suivant.

Type de données	Explications et exemple
VARCHAR2(**n**)	Chaîne de caractères de longueur variable. La taille maximale de cette chaîne est déterminée par la valeur **n** et peut atteindre 4000 caractères (bytes). La longueur minimale est 1. la précision du n est obligatoire. Exemple : NomPrograme VARCHAR2(20)
CHAR(n)	Chaine de caractères de longueur fixe allant de 1 à 2000 caractères (bytes). La chaîne est complétée par des espace si elle est plus petite que la taille déclarée Exemple CodeProgramme CHAR(3).

LONG	Données de type caractère pouvant stocker jusqu'à 2 gigabytes Exemple :Introduction LONG
NUMBER(n,d)	Pour déclarer un nombre sur maximum **n** chiffres (positions) dont **d** chiffres (positions) sont réservés à la décimale. **n** peut aller de 1 à 38.
DATE	Donnée de type date située dans une plage comprise entre le 1er janvier 4712 av JC et le 31 décembre 9999 ap JC stockant l'année, mois, jour, heures, minutes et secondes
LONG RAW	Chaîne de caractères au format binaire pouvant contenir jusqu'à 2 gigaoctet Exemple :Photo LONG RAW
BLOB	Binary Large Object : Gros objet binaire pouvant aller jusqu'à 4 gigaoctets : Exemple Image BLOB
CLOB	A character large object : Chaîne de caractère de longueur maximale allant jusqu'à 4 gigaoctet :

https://docs.oracle.com/cd/B28359_01/server.111/b28318/datatype.htm#CNCPT183

3. Les contraintes d'intégrités

Une contrainte d'intégrité est une règle sur la table qui permet d'assurer que les données stockées dans la base de données soient cohérentes par rapport à leur signification. Avec Oracle, on peut implémenter plusieurs contraintes d'intégrité.

- Au niveau de la table : Lorsque la contrainte porte sur plusieurs colonnes simultanément (clé primaire composée), il est obligatoire de déclarer la contrainte sur la table.
- Au niveau de la colonne ou attribut : se fait pour l'ensemble des contraintes à condition qu'elle porte sur une seule colonne.

Voici les explications des contraintes définies avec le CREATE TABLE plus haut :

La contrainte de PRIMARY KEY :

Rappel de définition :

Une clé primaire est attribut qui permet d'identifier chaque occurrence(enregistrement) d'une table de marinière unique. Ou encore c'est un attribut permettant d'identifier chaque ligne d'une table d'une manière unique.

Exemples :

- Le numéro d'admission d'un étudiant
- Un numéro de facture, un numéro de commande.
- Le nom d'un étudiant NE peut PAS être une clé primaire, car il est possible qu'il y ait plusieurs étudiants avec le même nom.

La contrainte de PRIMARY KEY, permet de définir une clé primaire sur la 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

Exemple 1 : Clé primaire au niveau de la colonne (attribut)

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

Remarquez :

1. Chaque définition de colonne se termine par une virgule, sauf la dernière colonne
2. 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.

Exemple2: Clé primaire au niveau de la table

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

Remarquez :

1. Chaque définition de colonne se termine par une virgule, sauf la dernière colonne.
2. La contrainte primary key n'est pas définie en même temps que la colonne NUMAD.(il y a une virgule qui termine la définition de la colonne numad)
3. La contrainte de primary key est définie comme si on définissait une colonne. **C'est une contrainte sur la table.**
4. Pour dire quel est l'attribut(colonne) qui est primary key il faudra le préciser entre parenthèses.
5. L'écriture suivante (exemple 3)est équivalente à l'écriture de **l'exemple 2**, ce qui veut dire que je peux placer la définition de la contrainte où je veux.
6. Si vous avez à définir des contraintes au niveau table (ce qui est recommandé) alors, la définition de ces contraintes doit être à la fin de la définition des colonnes (voir exemple 2).
7. Vous pouvez aussi créer les tables sans contraintes puis la modifier pout y rajouter des contraintes. (Nous allons examiner ce cas plus loin)

Exemple 3

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

Attention 

Que ce soit au niveau table ou au niveau colonne, la contrainte de PRIMARY KEY doit tout le temps avoir un nom. Elle est donc tout le temps, définie avec le mot réservé **CONSTRAINT**

La contrainte CHECK :

Indique les valeurs permises qui peuvent être saisies pour la colonne (attribut) lors de l'entrée des données ou une condition à laquelle doit répondre une valeur insérée. La condition doit impliquer le nom d'au moins une colonne. Les opérateurs arithmétiques (+,*,/,-), les opérateurs de comparaisons et les opérateurs logiques sont permis.

```
CREATE TABLE employes
(
empno NUMBER(4,0),
nom VARCHAR(30),
prenom VARCHAR(30),
salaire NUMBER(8,2) CONSTRAINT ck_salaire CHECK(salaire >25000)
);
```

La contrainte DEFAULT :

Indique la valeur par défaut que prendra l'attribut si aucune valeur n'est saisie.

La contrainte NOT NULL :

Indique que la valeur de la colonne ou de l'attribut est obligatoire. Si cette contrainte n'est pas précisée alors par défaut la valeur est NULL.

La contrainte UNIQUE :

Indique que les valeurs saisies pour les colonnes (attributs) doivent être unique Ce qui veut dire pas de Doublons.

Exemple 2, la contrainte CHECK est au niveau colonne et nous avons un DEFAULT. Le nom est NOT NULL

```
CREATE TABLE personne
(
num NUMBER(4,0) CONSTRAINT pk_personne PRIMARY KEY,
nom VARCHAR2 (15) NOT NULL,
prenom VARCHAR2 (15),
courriel VARCHAR2(40) UNIQUE,
ville VARCHAR2 (20) DEFAULT 'Montréal' CONSTRAINT ck_ville CHECK
(ville IN ('Montréal','Laval','Saint-Jérôme'))
);
```

Exemple 3 : La contrainte CHECK sur echelon est définie au niveau table

```
CREATE TABLE  employes
(
numep NUMBER(4,0)  CONSTRAINT PK_employes PRIMARY KEY,
nom VARCHAR2(30) NOT NULL,
prenom VARCHAR2(30) CONSTRAINT ck_prenom NOT NULL,
codedep CHAR(3) DEFAULT 'INF' CONSTRAINT ck_departement CHECK (codedep IN
('INF','RSH','CMP','GEM')),
salaire NUMBER (8,2) CHECK (SALAIRE > 20000),
echelon number(2,0),
constraint ck_echelon CHECK (echelon between 10 and 25)
);
```

Attention : 

1. Les tables sont des objets de la base de données. Les noms doivent être uniques. Vous ne pouvez pas avoir deux tables Employes dans votre BD
2. Les noms des tables doivent être significatifs.
3. Les noms des tables ne doivent pas être des mots réservés du SGBD. Exemple de mot réservés: Table, sequence, sysdate,create,cycle, constraint, primary etc…
4. Les colonnes des tables doivent avoir des noms significatifs et ne doivent pas être des mots réservés.
5. Les noms de colonnes sont uniques dans une table, mais pas dans la base de données.
6. Les contraintes de PRIMARY KEY et CHECK doivent être définies avec le mot réservé CONSTRAINT et doivent avoir un nom significatif.
7. La contrainte de PK et CHECK peuvent être définies sur la table ou sur la colonne.
8. Les contraintes sont des objets de la base de données, le nom doit être unique.
9. Les contraintes de NOT NULL et UNIQUE sont généralement définies sans nom.(Le système se charge de leur donner un nom)
10. Nous ne sommes pas obligés de donner un nom pour les contraintes. MAIS toutes les contraintes ont un nom dans le système. Si vous n'avez pas donné vous-même un nom à votre contrainte, le système (le SGBD) va s'en charger à votre place.
11. Mis à part les contraintes d'UNIQUE, DEFAULT et NOT NULL, vous êtes obligés de donner un nom significatif à toutes vos contraintes.

12. 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>
]
);
```

Les requêtes de manipulation de données (DML)

Après avoir créé votre table, celle-ci est vide. Pour peupler votre table nous allons utiliser une requête d'insertion de données. Une fois les données insérées, nous pouvons alors les modifier ou alors en supprimer celles qui ne sont plus utiles. Ces requêtes sont appelées de requêtes DML

1. La commande INSERT INTO

Cette commande permet d'insérer des données dans une table, une ligne à la fois. Deux syntaxes sont possibles pour cette commande.

Syntaxe1

```
INSERT INTO <nom_de_table> VALUES (<liste de valeurs>);
```

Exemple

```
INSERT INTO EmployesInfo VALUES (20,'Fafar','Patrice',40000);
```

Syntaxe 2

```
INSERT INTO <nom_de_table>(<nom_de_colonne>) VALUES
(<liste_de_valeurs>);
```

Exemple

```
INSERT INTO EmployesInfo (NumEmp, NOM) VALUES (12,'Lebeau');
```

La commande INSERT INTO est la première commande exécutée après avoir créé une table. Cette commande permet de saisir des données dans une table **une rangée à la fois.**

- Aucune insertion n'est possible si les contraintes d'intégrité ne sont pas respectées.
- Lors de l'insertion des donnés dans l'ensemble des colonnes de la tables (toutes les colonnes), il n'est pas nécessaire de préciser les noms de celles-ci. Voir syntaxe 1
- Si des valeurs dans certaines colonnes ne doivent pas être saisies (contiennent des valeurs par défaut) alors la précision des colonnes dans lesquelles la saisie doit

s'effectuer est obligatoire. Noter que les valeurs à saisir doivent être dans le même ordre de la spécification des colonnes. Voir syntaxe 2

- Une valeur de type caractère (CHAR ou VARCHAR2) doit être mise entre apostrophes. Si la chaine de caractère contient des apostrophes, ceux-ci doivent être doublés.
- Le type numérique (NUMBER) est saisi en notation standard. La virgule décimale est remplacée par un point lors de la saisie
- Le type date doit être saisie selon la norme américaine (JJ-MMM-AA pour 12 jan 99) et entre apostrophes. Pour saisir une date dans n'importe quel format, il faut s'assurer de la convertir dans le format avec la fonction TO_DATE
- Lorsque la valeur d'une colonne n'est pas connue et que celle-ci possède une contrainte de NOT NULL, alors on peut saisir le NULL entre apostrophe comme valeur pour cette colonne.
- Il est possible d'utiliser une expression arithmétique dans la commande INSERT INTO à condition que cette colonne soit de type numérique.
- Il est possible d'utiliser des insertions à partir d'une table existante (commande SELECT et une sous-requête -----à voir plus loin)
- Il est possible d'utiliser une séquence pour l'insertion automatique d'un numéro séquentiel pour une colonne (à voir plus loin)
- Il est possible de générer la clé primaire automatiquement (Oracle 12c et plus)
- Après les insertions, il est recommandé d'exécuter la commande COMMIT pour enregistrer vos données (à voir plus loin)

2. La commande UPDATE

Syntaxe simplifiée :

```
UPDATE <nom_de _table> SET <nom_de_colonne>=<nouvelle_valeur>;
```

La commande UPDATE permet d'effectuer des modifications des données sur une seule table. Cette modification peut porter sur une ou plusieurs lignes. (Enregistrements)
Lors de la modification des données, les contraintes d'intégrité doivent être respectées :

- Il est impossible de modifier une valeur de la clé primaire si cette valeur est référée par une valeur de la clé étrangère
- De plus, il faut tenir compte de des valeurs définies par les contraintes CHECK et NOT NULL

- Il est possible d'utiliser une expression arithmétique dans la commande UPDATE à condition que cette colonne soit de type numérique.
- Il est possible d'utiliser des modifications à partir d'une table existante (commande SELECT et une sous-requête -----à voir plus loin)

Syntaxe générale

UPDATE <nom_de _table> SET (<nom_de_colonne>=<nouvelle_valeur>) [WHERE <condition>];

- La clause WHERE permet de fixer la condition sur les données de mise à jour (UPDATE). Cette clause est utilisée également avec DELETE.
- Il est possible de mettre à jour plusieurs colonnes en même temps comme le montre la syntaxe.
- Les opérateurs utilisés dans le WHERE du UPDATE sont les même que ceux du WHERE dans le SELECT

Exemples :

UPDATE employesinfo SET salaire = salaire +(salaire*0.5)
WHERE nom ='Fafar'; → ajoute 1% du salaire à l'employé dont le nom est Fafar.

UPDATE employesinfo SET salaire = salaire +(salaire*0.1) → ajoute 1% du salaire à tous les employés

UPDATE employesinfo SET salaire = salaire +(salaire*0.1), commission =200 ; → on met à jour le salaire et la commission pour tous les employés

Attention : 

1. Aucune insertion n'est possible si les contraintes d'intégrités ne sont pas respectées
2. Lors des mises à jour des données UPDATE, les contraintes d'intégrité doivent être respectées sinon aucune mise à jour ne sera effectuée.

3. La commande DELETE

La commande DELETE permet de supprimer de la base de données une ou plusieurs lignes d'une table.

Pour des raisons de sécurité, exécuter cette commande jute après la commande SELECT afin d'être certain des enregistrements que l'on va supprimer

Lors de la suppression des données, les contraintes d'intégrité référentielle doivent être respectées. Il est impossible de supprimer une valeur de la clé primaire si cette valeur est référée par une valeur de la clé étrangère sauf si l'option ON DELETE CASCADE est définie ; dans ce cas TOUS les enregistrements de la table enfant (table de la clé étrangère) qui réfèrent la clé primaire supprimée seront supprimés.

Si l'option ON DELETE CASCADE n'est pas définie alors pour supprimer une clé primaire référencée il faut opter pour une des solutions suivantes :

- Supprimer d'abord les rangées de la table enfants qui référent la clé primaire, puis supprimer la clé primaire.
- Désactiver la contrainte d'intégrité référentielle (clé étrangère). Cette action est irréversible. Supprimer ensuite la clé primaire.
- Modifier la contrainte d'intégrité référentielle en ajoutant l'option ON DELETE CASCADE

Syntaxe :

Syntaxe

```
DELETE FROM <nom_de _table>
[WHERE <condition>];
```

Exemple :

DELETE FROM employes ; → supprime tous les employés

DELETE FROM employes WHERE nom ='Fafar'; → supprime tous les employés dont le nom est Fafar

4. Les transactions :

Les opérations DML, une fois exécutées doivent être confirmées pour que la sauvegarde soit effective dans la base de données.

COMMIT: elle permet d'officialiser une mise à jour (INSERT, UPDATE, DELETE) ou une transaction (série de commandes de manipulation de données effectuées depuis le dernier COMMIT) sur la base de données.

ROLLBACK : permet d'annuler une transaction avant un COMMIT ; une fois le COMMIT exécuté aucun ROLLBACK n'a d'effet sur la BD

Conclusion, les commandes DML

Il existe 3 commandes du DML (Data Manipulation Language):

- La commande INSERT INTO, qui permet d'ajouter des données dans une table une ligne à la fois. Lors des insertions les contraintes d'intégrités doivent être respectées.
- la commande UPDATE, qui permet de modifier les données d'une table. La clause WHERE est utilisée pour cibler les lignes à modifier. Lors des modifications, les contraintes d'intégrité doivent être respectées.
- La commande DELETE, qui permet de supprimer des données dans une table. La clause WHERE est utilisée pour cibler les lignes à modifier. Lors des modifications, les contraintes d'intégrité référentielles doivent être respectées

Après exécution des opérations DML, vous devez exécuter COMMIT pour officialiser les transactions.

Pour annuler une opération DML on exécute un ROLLBACK avant le COMMIT. Après un COMMIT, aucun ROLLBACK n'a d'effet.

Nouveauté à partir de la version 12c dans la commande CREATE TABLE, Insertion automatique de la clé primaire

À partir de la version 12c de la base de données, Oracle a introduit l'option IDENTITY pour incrémenter automatiquement la clé primaire. L'avantage d'avoir une telle option est que la clé primaire ne sera jamais dupliquée. On diminue les erreurs

L'option IDENTITY convient surtout pour les tables ayant comme clé primaire un numéro séquentiel : Clients, fournisseurs, joueurs, commandes, factures etc.. Le type de données pour la clé primaire est **NUMBER.**

Plusieurs syntaxes sont possibles pour le l'option IDENTITY, afin de mieux les comprendre, on va y aller avec des exemples.

Exemple 1, BY DEFAULT

```
CREATE TABLE clients
(
id_client number(4,0) GENERATED BY DEFAULT AS IDENTITY,
nom varchar2(30) not null,
prenom varchar2(30),
CONSTRAINT pk_client PRIMARY KEY(id_client)
);
```

Explications :

- Le id_client est une clé primaire qui s'insère automatiquement. Elle commence à 1 et s'incrémente de 1.
- Lorsque j'insère ces lignes, je n'ai pas d'erreurs puis que la clé s'insère automatiquement

```
insert into clients (nom, prenom) values('LeRoy','Gibbs');
insert into clients (nom, prenom) values('LeChat','Simba');
insert into clients (nom, prenom) values('LeBeau','Cheval');
```

Après l'exécution d'un select * j'obtiens :

ID_CLIENT	NOM	PRENOM
1	LeRoy	Gibbs
2	LeChat	Simba
3	LeBeau	Cheval

- À chaque insertion la clé monte de 1. Il y a une séquence
- Je peux briser la séquence en faisant une insertion manuelle de la clé primaire comme suit :

```
insert into clients(id_Client, nom, prenom)  values (10, 'Ce nouveau','Client');
```

Voir le résultat du SELECT *

ID_CLIENT	NOM	PRENOM
1	LeRoy	Gibbs
2	LeChat	Simba
3	LeBeau	Cheval
10	Ce nouveau	Client

- Lorsque je reviens à l'insertion automatique de la clé primaire, celle-ci continue à 4 (après 3 et non après 10).

```
insert into clients (nom, prenom) values('Après le nouveau','Le Client');
```

ID_CLIENT	NOM	PRENOM
1	LeRoy	Gibbs
2	LeChat	Simba
3	LeBeau	Cheval
10	Ce nouveau	Client
4	Après le nouveau	Le Client

- Lorsque, par l'insertion automatique, nous arriverons au numéro 10, il y aura une erreur.

Exemple 2, BY DEFAULT START WITH

l'exemple suivant fait exactement la même chose que l'exemple 1, sauf que nous avons la possibilité d'indiquer :

1. La séquence (le id) commence où ?
2. On incrémente de combien ?

```
CREATE TABLE clientsClg
(
id_client number(4,0) GENERATED BY DEFAULT AS IDENTITY START WITH 10
INCREMENT BY 2,
nom varchar2(30) not null,
prenom varchar2(30),
CONSTRAINT pk_clientclg primary key(id_client)
);
```

Dans cet exemple, la clé primaire commence à 10 et s'incrémente de 2

```
insert into clientsclg (nom, prenom) values('LeRoy','Des Singes');
insert into clientsclg (nom, prenom) values('Lefou','Du Village');
insert into clientsclg (nom, prenom) values('Soleil','Vert');
```

ID_CLIENT	NOM	PRENOM
10	LeRoy	Des Singes
12	Lefou	Du Village
14	Soleil	Vert

Si INCREMENT BY n'est pas précisé, par défaut il est à 1.

Exemple 3, ALWAYS

Dans l'exemple suivant, la clé primaire est toujours IDENTITY, donc aucune insertion manuelle de la clé primaire n'est possible.

```
create table fournisseurs
(id_fournisseur number(4,0) GENERATED ALWAYS AS IDENTITY,
nom varchar2(40) not null,
prenom varchar2(30) ,
constraint pk_fournisseur primary key (id_fournisseur));
```

Lorsque je fais :

```
insert into fournisseurs values (10,'Yacoub','Saliha');
```

j'obtiens l'erreur suivante :

```
Erreur à la ligne de commande: 44 Colonne: 1
Rapport d'erreur -
Erreur SQL : ORA-32795: impossible d'insérer la valeur dans une colonne d'identité avec les mots-clés GENERATED ALWAYS
32795.0000 -  "cannot insert into a generated always identity column"
*Cause:    An attempt was made to insert a value into an identity column
           created with GENERATED ALWAYS keywords.
*Action:   A generated always identity column cannot be directly inserted.
           Instead, the associated sequence generator must provide the value.
```

Étant donné que la clé a été définie avec l'option ALWAYS , l'insertion manuelle de la clé n'est plus possible.

Pour l'option ALWAYS, nous pouvons également ajouter **START WITH et INCREMENT BY**

Chapitre 4, la contrainte d'intégrité référentielle.

Définition

On parle d'intégrité référentielle lorsque les valeurs d'une ou plusieurs colonnes d'une table (exemple Joueurs) sont déterminés ou font référence à des valeurs d'une colonne d'une autre table (exemple Equipes).

Exemple
Voici le contenu de la table Equipes

	CODEEQUIPE	NOMEQUIPE	VILLE	NBCOUPES
1	MTL	LES CANADIENS DE MONTRÉAL	MONTRÉAL	24
2	TOR	LES MAPLE LEAFS	TORONTO	22
3	OTT	LES SÉNATEURS	OTTAWA	4
4	AVL	LES AVALANCHES	COLORADO	2
5	VAN	LES CANUKS	VANCOUVER	1
6	BRU	LES BRUNS DE BOSTON	BOSTON	13

Voici le contenu de la table Joueurs

	NUMJOUEUR	NOM	PRENOM	CODEEQUIPE
1	1	PRICE	CAREY	MTL
2	2	MARKOV	ANDRÉ	MTL
3	3	SUBBAN	KARL	MTL
4	4	PATIORETTY	MAX	MTL
5	10	HAMOND	ANDREW	OTT
6	6	STONE	MARC	OTT
7	9	TURIS	KYLE	OTT
8	7	GALLAGHER	BRANDON	MTL
9	8	TANGUAY	ALEX	AVL
10	11	THOMAS	BIL	AVL

On remarque que les deux tables ont toutes les deus la colonne **codeequipe**

Les valeurs de la colonne codeequipe de la table Joueurs font référence aux valeurs de la colonne codeequipe de la table Equipes

On parle d'intégrité référentielle lorsque, les seules valeurs possibles pour la colonne codeequipe de la table joueurs NE PEUVENT être que les valeurs de la table Equipes.

Dans la plupart des cas l'intégrité référentielle est implémentée par la contrainte de

FOREIGN KEY

La contrainte de FOREIGN KEY

Pour parler de contrainte de FOREIGN KEY nous avons besoins de deux tables :
Une table A (exemple Equipes) qui contient un attribut (exemple Codeequipe) de clé primaire : PRIMARY KEY et une table B (exemple Joueurs) qui va contenir un attribut de clé étrangère.

Cette contrainte indique que la valeur de l'attribut de la table B correspond à une valeur d'une clé primaire de la table spécifiée. (table A)
La clé primaire de l'autre table A (exemple table Equipes) doit être obligatoirement crée pour que cette contrainte soit acceptée.
La clé primaire la table A et l'attribut défini comme clé étrangère de la table B doivent être de même type et de même longueur. Il n'est pas nécessaire que les attributs de clé primaire et de clé étrangère aient des noms identiques.
Exemple :

```
CREATE table equipes
(
codeequipe CHAR(3) CONSTRAINT pk_codequipe PRIMARY KEY,
nomequipe VARCHAR2(50) NOT NULL,
villeVARCHAR2(40),
nbcoupes NUMBER(2,0) CONSTRAINT ck_nbcoupe CHECK (nbcoupes > =0)
) ;
---------------------------------------------------
CREATE TABLE joueurs
(
numjoueur NUMBER(3,0) CONSTRAINT pk_numjoueur PRIMARY KEY,
nom VARCHAR2(30) NOT NULL,
prenomVARCHAR2(30),

codeequipe CHAR(3),

CONSTRAINT fk_codeequie FOREIGN KEY (codeequipe) REFERENCES equipes(codeequipe)
);
```

Que faut-il comprendre de l'exemple précédent ?

1. La table equipes doit-être créée en premier. Codeequipe est la clé primaire de cette table
2. Dans la table joueurs, nous avons un attribut codeequipe de même type et de même longueur que codeequipe de la table equipe. (ils ont le même nom, mais ce n'est pas obligatoire)
3. Le codeequipe de la table joueurs fait référence à codeequipe de la table equipe.
4. Il y a une contrainte de clé étrangère sur le codeequipe de la table joueurs Attention

La contrainte de FOREIGN KEY est une contrainte sur la table et non sur un attribut.

La contrainte de FOREIGN KEY garantie l'intégrité référentielle ce qui veut dire :

1. Vous ne pouvez pas modifier (UPDATE) la valeur de l'attribut de la clé primaire s'il a une valeur de clé étrangère. Exemple dans la table EQUIPES vous ne pouvez pas modifier le codeequipe MTL pour BLA, car il existe des joueurs ayant le codeequipe MTL.
2. Vous ne pouvez pas supprimer (DELETE) une ligne de la table référencée s'il existe des enregistrements ayant une valeur de la clé étrangère égale à la valeur de la clé primaire de la ligne à supprimer. Exemple vous ne pouvez pas supprimer de la table equipes l'équipe dont le codeequipe est OTT, car il existe des joueurs ayant ce codeequipe.
3. Vous ne pouvez pas supprimer la table EQUIPES par un simple DROP TABLE. Il faudra
   - Soit supprimer la table joueurs en premier (à condition qu'elle ne soit pas référencée)
   - Soit utiliser CASCADE CONSTRAINTS.
   - Soit désactiver ou détruire la contrainte d'intégrité (la FOREIGN KEY)

On peut également préciser l'option ON DELETE CASCADE qui indique que les enregistrements soient détruits lorsque l'enregistrement correspondant à la clé primaire de la table référencée est supprimé.

```
CREATE TABLE JOUEURS
(numjoueur NUMBER(3,0) CONSTRAINT PK_NUM_JOUEUR PRIMARY KEY,
nom VARCHAR2(30) NOT NULL,
prenomVARCHAR2(30),

codeequipe CHAR(3),

CONSTRAINT fkcodeequie FOREIGN KEY (codeequipe) REFERENCES equipes(codeequipe) ON
DELETE CASCADE);
```

Attention 

Ne jamais utiliser ON DELETE CASCADE à moins que vous soyez vraiment certain.

Table avec clé primaire composée

Il arrive qu'une table ait besoin de deux (ou plus) attributs pour identifier de manière unique les enregistrements. Dans ce cas on parle de clé primaire composée. Dans la plupart des cas les attributs de la clé primaire sont des clés étrangères.

Si une table a une clé primaire composée dont les attributs sont des clés étrangères, la table est dite : **Table de relation**

Exemple :

```
CREATE TABLE cours (
code_cours  CHAR(6) CONSTRAINT pk_cours PRIMARY KEY,
titrecours VARCHAR2(20) NOT NULL
);


CREATE TABLE etudiants (
numad NUMBER(10,0) CONSTRAINT pk_etudiant PRIMARY KEY,
nom VARCHAR2(20) NOT NULL,
prenom VARCHAR2(20)
) ;
```

Création de la table RESULTATS qui est une table de lien.

```
CREATE TABLE resultats
(
num_etudiant NUMBER(10,0),
code_cours  CHAR(6),
note NUMBER(5,2),
----- definition des contraintes-----
CONSTRAINT fk_etudiants FOREIGN KEY (num_etudiant) REFERENCES etudiants(numad),
CONSTRAINT fk_cours FOREIGN KEY (code_cours  ) REFERENCES COURS(code_cours  ),
CONSTRAINT pk_resultats PRIMARY KEY(num_etudiant,code_cours)
);
```

Remarquez :

- La table resultats a deux attributs qui sont des clés étrangères.
- La clé étrangère code_cours est de même type et de même longueur que la clé primaire Code de la table Cours.
- La clé étrangère num_etudiant est de même type et de même longueur que la clé primaire numad de la table Etudiants. Ces attributs n'ont pas le même nom
- La table Resultats a trois contraintes. Deux contraintes de FOREIGN KEY et une contrainte de PRIMARY KEY.
- La table resultats est une table de relation.

Attention 

La contrainte de clé primaire composée est une contrainte sur la TABLE. (tout comme la contrainte de Foreign key

Rappel de définition :

Une base de données relationnelle est constituée de plusieurs tables reliées entre elles garantissant ainsi l'intégrité référentielle

Exemple de représentation graphique

Interprétation :

Nous avons deux tables reliées entre elles.

- Numad est clé primaire dans la table etudiants(P)
- CodeProg est clé primaire dans la table Programmes.(p)
- CodeProg est clé étrangère dans la table Etudiants(F)
- Le lien (la flèche) dit : le codeProg dans la table étudiants fait référence au codeProg de la table Programmes.

Chapitre 5, les commandes de définition de données.

La commande ALTER TABLE

Il est parfois nécessaire de modifier la structure d'une table, la commande ALTER TABLE sert à cela. Cette commande change la structure de la table mais pas son contenu. Les types de modifications acceptées sont les suivants :

- ✓ Ajout d'une nouvelle colonne avec ses contraintes à une table
- ✓ Augmente ou diminuer la largeur d'une colonne existante
- ✓ Changer la catégorie d'une colonne, d'obligation à optionnelle ou vice versa (NOT NULL à NULL ou vice versa)
- ✓ Spécification d'une valeur par défaut pour une colonne existante
- ✓ Modifier le type de données d'une colonne existante
- ✓ Spécification d'autres contraintes pour une colonne existante
- ✓ Activer ou désactiver une contrainte
- ✓ Détruire une contrainte.
- ✓ Détruire une colonne.
- ✓ Renommer une colonne.

1. L'option ADD

Cette option permet d'ajouter une colonne ou une contrainte à une table existante.

Attention !! 

- Si la table contient des valeurs, alors la colonne ajoutée doit être mise à jour.
- Si une contrainte est ajoutée à une colonne alors que celle-ci contient déjà des données qui ne correspondent pas à la contrainte, alors la modification de la structure de la table sera refusée.
- Pour ajouter une colonne c'est juste ADD et non ADD COLUMN

Exemples :

Voici la commande CREATE initiale pour la table Employes

```
CREATE TABLE Employes
(
NumEmp number(4,0), nom varchar2(15), prenom varchar2(20), ville varchar2(30)
);
```

Pour ajouter une colonne Salaire on procède comme suit :

```
ALTER TABLE Employes ADD (Salaire NUMBER (8,2));
```

Pour ajouter une colonne avec ses contraintes:

```
ALTER TABLE Employes ADD  (echelon NUMBER(2,0) NOT NULL CHECK(echelon>10))
```

Pour ajouter la contrainte de clé primaire on procède comme suit.

```
ALTER TABLE Employes ADD CONSTRAINT employe_pk PRIMARY KEY (numeemp);
```

Remarque: Dans cet exemple la contrainte de clé primaire sur la colonne Numeemp est une contrainte sur la table.

2. L'option MODIFY

Cette option permet de modifier le type de données, la valeur par défaut et la contrainte de NOT NULL sur une table déjà existante.

Attention 

Il est impossible de raccourcir la taille d'une colonne (la longueur des données) si celle-ci contient des données.

ALTER TABLE Employes MODIFY (nom NOT NULL);→on met nom à obligatoire

ALTER TABLE Employes MODIFY (nom varchar2(20)); →on augmente la taille de la colonne nom

```
ALTER TABLE employes MODIFY (ville DEFAULT 'Montréal');
```

3. L'option ENABLE /DISABLE

Cette option sert à activer ou désactiver une contrainte.

```
ALTER TABLE Employes DISABLE Primary Key;
```

```
ALTER TABLE Employes DISABLE CONSTRAINT cksalaire; → désactive la contrainte cksalaire
```

Pour activer la contrainte :

```
ALTER TABLE Employes  ENABLE CONSTRAINT cksalaire; → active la contrainte cksalaire
```

4. L'option DROP

Cette option sert à supprimer une contrainte sur une table déjà existante ou de supprimer une colonne.

```
ALTER TABLE Employes DROP Primary Key;
ALTER TABLE Employes DROP CONSTRAINT employe_pk;
ALTER TABLE Employes  DROP COLUMN nom;
```

5. L'option RENAME

Cette option permet de renommer une colonne ou une contrainte

```
ALTER TABLE joueurs RENAME CONSTRAINT   SYS_C00126194  TO  pk_joueurs;
ALTER TABLE Employes RENAME COLUMN salaire TO SalaireEmp;
```

La commande DROP TABLE

La commande DROP TABLE permet de détruire une table de la base de données.

La commande DROP Permet de supprimer un objet de la b ase de données. (DROP VIEW, DROP SEQUENCE…)

```
DROP TABLE EMPLOYES;
```

Attention 

Aucun ROLLBACK n'est possible avec la commande DROP car c'est une commande du DDL et non une commande DML.

Vous ne pouvez pas supprimer une table qui est référencée par d'autres table. Pour le faire vous devez exécuter un CASCADE CONSTRAINTS.

DROP TABLE EMPLOYES CASCADE CONSTRAINTS;

La commande RENAME

Permet de renommer une table ou un objet de la base de données

Syntaxe

```
RENAME <Ancien_nom> TO <Nouveau_nom>;
```

```
RENAME Employes TO EmployesInfo;
```

Important:

- ✓ Tous les objets de la base de données sont créés avec la commande CREATE.
- ✓ Tous les objets de base de données sont supprimés avec la commande DROP
- ✓ La commande ALTER TABLE change la structure de la table et non son contenu (contrairement à UPDATE qui change le contenu et non la structure d'une table)
- ✓ Ces deux commandes n'ont ni de COMMIT ni de ROLLBACK.

Un script SQL est un fichier SQL contenant un certain nombre de commandes SQL. (Comme vos laboratoires) TOUS les scripts SQL que vous remettrez à l'avenir vont se présenter comme suit.

- les DROP TABLE CASCADE CONSTRAINTS ;
- les CREATE TABLE
- les ALTER TABLE
- les INSERT INTO
- suite des commandes SQL

Chapitre 6, requêtes avec jointures

On utilise les jointures lorsque nous souhaitons extraire des données qui se trouvent dans différentes tables. Il existe différents types de jointures.

Contenu des tables : (syemp, sydept)

Syemp : Table du labo1

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7839	KING	PRESIDENT	(null)	17/11/81	5000	(null)	10
7698	BLAKE	MANAGER	7839	01/05/81	2850	(null)	30
7782	CLARK	MANAGER	7839	09/06/81	2450	(null)	10
7566	JONES	MANAGER	7839	02/04/81	2975	(null)	20
7902	FORD	ANALYST	7566	03/12/81	3000	(null)	20
7369	SMITH	CLERK	7902	17/12/80	800	(null)	20
7499	ALLEN	SALESMAN	7698	20/02/81	1600	300	30
7521	WARD	SALESMAN	7698	22/02/81	1250	500	30
7654	MARTIN	SALESMAN	7698	28/09/81	1250	1400	30
7844	TURNER	SALESMAN	7698	08/09/81	1500	0	30
7900	JAMES	CLERK	7698	03/12/81	950	(null)	30
7934	MILLER	CLERK	7782	23/01/82	1300	(null)	10
7876	ADAMS	CLERK	7788	23/05/87	1100	(null)	20
7788	SCOTT	ANALYST	7566	19/04/87	3000	(null)	20

Voici le contenu de la table **sydept**

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

Le produit cartésien

Le produit cartésien est une requête de sélection qui met en jeux plusieurs tables. Pour deux tables, la sélection consiste à afficher la première ligne de la première table avec toutes les

lignes de la deuxième table, puis la deuxième ligne de la première table avec toutes les lignes de la deuxième table et ainsi de suite. Ce type de sélection implique beaucoup de redondances.

Exemple

```
SELECT ename,job ,dname FROM syemp,sydept;
```

	ENAME	JOB	DNAME
1	KING	PRESIDENT	ACCOUNTING
2	BLAKE	MANAGER	ACCOUNTING
3	CLARK	MANAGER	ACCOUNTING
4	JONES	MANAGER	ACCOUNTING
5	FORD	ANALYST	ACCOUNTING
6	SMITH	CLERK	ACCOUNTING
7	ALLEN	SALESMAN	ACCOUNTING
8	WARD	SALESMAN	ACCOUNTING
9	MARTIN	SALESMAN	ACCOUNTING
10	TURNER	SALESMAN	ACCOUNTING
11	JAMES	CLERK	ACCOUNTING
12	MILLER	CLERK	ACCOUNTING
13	KING	PRESIDENT	RESEARCH
14	BLAKE	MANAGER	RESEARCH

…. (il y a une suite à la sortie de la requête). En tout il y a 48 lignes alors que nous avons 12 employés.

Dans l'exemple précédent on ne sait pas si KING est dans le département ACCOUNTING ou le département RESEARCH, ou autre ..

Jointure interne (ou jointure simple)

Une jointure simple consiste en produit cartésien avec un INNER JOIN faisant ainsi une restriction sur les lignes. La restriction est faîte sur l'égalité de la valeur de deux attributs (cas de deux tables) qui sont la valeur d'une clé primaire est égale à la valeur d'une clé étrangère.

Syntaxe simplifiée (avec deux tables)

```
SELECT colonne1, colonne2, …..
FROM  [(]nom_table1  INNER JOIN nomTable2
ON nom_table1.cleEtrangere = nomTable2.clePrimaire [)];
[WHERE <condition>]
[ORDER BY <nom_de_colonne>];
```

Exemple1

```
SELECT ename, job, dname
FROM syemp INNER JOIN sydept
ON syemp.deptno = sydept.deptno
```

L'exemple précédent ramène des enregistrements uniquement s'il y a égalité entre la valeur de deptno dans syemp et la valeur de deptno dans sydept . Seuls les employés ayant un département et les départements ayant des employés seront ramenés par la requête.

	ENAME	JOB	DNAME
1	MILLER	CLERK	ACCOUNTING
2	CLARK	MANAGER	ACCOUNTING
3	KING	PRESIDENT	ACCOUNTING
4	ADAMS	CLERK	RESEARCH
5	FORD	ANALYST	RESEARCH
6	JONES	MANAGER	RESEARCH
7	SMITH	CLERK	RESEARCH
8	SCOTT	ANALYST	RESEARCH
9	JAMES	CLERK	SALES
10	TURNER	SALESMAN	SALES
11	MARTIN	SALESMAN	SALES
12	WARD	SALESMAN	SALES
13	BLAKE	MANAGER	SALES
14	ALLEN	SALESMAN	SALES

Les employés qui n'ont pas de département ne seront pas ramenés par la requête. Les départements qui n'ont pas d'employés ne seront pas ramenés par la requête non plus. Ces cas seront traités plus loin.

Remarques :

- Lorsqu'un attribut sélectionné est présent dans plus d'une table alors il faut le précéder du nom de la table à partir de laquelle on désire l'extraire. (Voir exemple 2)
- Vous pouvez donner un alias aux noms de tables afin de faciliter la référence aux tables. Cependant si un alias est donné alors, il faudra utiliser l'alias à la place du nom de la table. (Voir exemple 3)
- Toutes les tables dont les attributs apparaissent dans la clause **SELECT** ou dans la clause **WHERE** doivent apparaître dans la clause FROM.
- Parfois, vous avez besoin de la table de relation (lien)pour faire une jointure. (Voir exemple 5)

Exemple 2,

Cette instruction va renvoyer une erreur *« nom de colonne ambiguë* »car deptno est dans les deux tables Syemp et Sysdept.

```
SELECT  ename, job, deptno, dname
FROM syemp INNER JOIN sydept ON syemp.deptno = sydept.deptno;
```

Il faut écrire :

```
SELECT  ename, job, sydept.deptno, dname
FROM (syemp INNER JOIN sydept ON syemp.deptno = sydept.deptno);
```

Cette instruction va renvoyer une erreur « nom de colonne ambiguë »car deptno est dans les deux tables Syemp et Sysdept.

```
SELECT ename, job, sal, loc
FROM (syemp INNER JOIN sydept on syemp.deptno = sydept.deptno)
WHERE deptno = 10;
```

Il faut écrire

```
SELECT  ename, job,  dname
```

```
FROM (syemp INNER JOIN sydept ON syemp.deptno = sydept.deptno)
where sydept.deptno =10;
```

Exemple 3

Les tables ont des alias.

```
SELECT  ename, job,  dname
FROM (syemp s INNER JOIN sydept D ON S.deptno = D.deptno);
```

Dès qu'un alias est donné à une table, alors il faudra toujours utiliser l'alias. L'écriture suivante va renvoyer une erreur *« sydept.deptno » identificateur invalide*

```
SELECT  ename, job,  dname, sydept.deptno
FROM (syemp s INNER JOIN sydept D ON S.deptno = D.deptno);
```

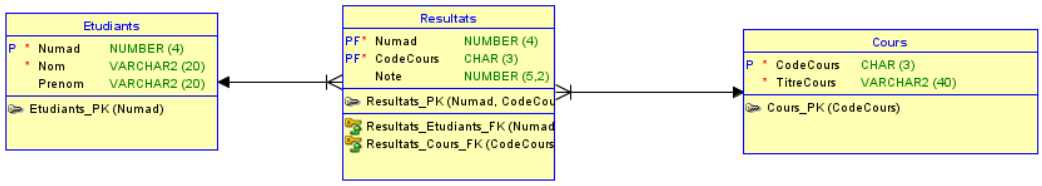
Il faut écrire

```
SELECT  ename, job,  dname, D.deptno
FROM (syemp s INNER JOIN sydept D ON S.deptno = D.deptno);
```

Exemple 4

Attention 

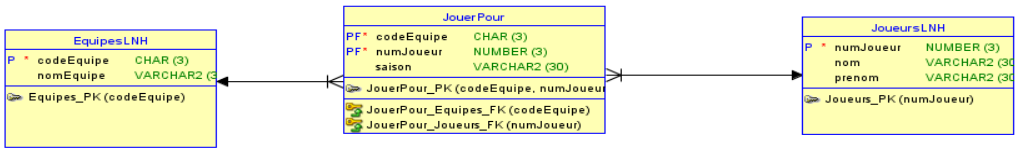
Pour pouvoir écrire correctement vos requêtes avec jointures, vous avez besoin de visualiser les liens (les relations) entre vos tables par une représentation graphique : un modèle



```
SELECT nom,prenom,  titrecours, note
FROM  ((etudiants E  INNER JOIN  Resultats R  ON E.numad = R.numad)
INNER JOIN cours C ON C.code_cours = R.code_cours);
```

Exemple 5

On souhaite chercher, le nom, le prénom et le nom de l'équipe pour afficher les équipes dans lesquelles les joueurs ont joué.
Il n'y pas de lien direct entre la Table EquipeLNH et la table JoueursLNH, mais il existe une table de lien entre ces deux tables : c'est table Jouerpour.

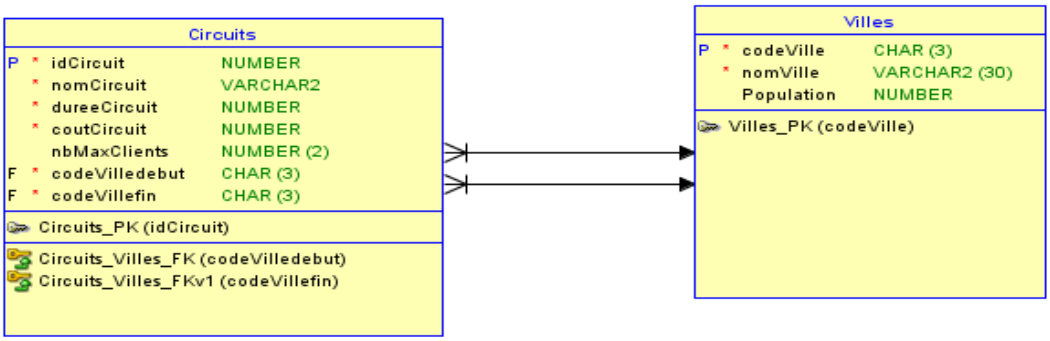


La requête sera donc :

```
SELECT nom, prenom,nomEquipe
FROM ((joueurslnh INNER JOIN jouerpour ON joueurslnh.numjoueur = jouerpour.numjoueur)
INNER JOIN equipeslnh ON equipeslnh.codeequipe=jouerpour.codeequipe);
```

Importance des Alias :

Il arrive que la clé primaire d'une table migre plus qu'une fois dans une autre table pour être clé étrangère plus qu'une fois. L'exemple de la figure suivante montre que l'attribut codeVille est clé étrangère deux fois dans la table Circuits. Une fois pour indiquer le code ville de début du circuit, une autre fois pour indiquer le code ville de la fin du circuit.



Lors de la création de la table Circuits, nous devons mettre en évidence cette réalité de la clé étrangère deux fois sur la même clé primaire. Dans ce cas, il faudra donner des noms différents aux attributs de la clé étrangère. (On se rappelle qu'il faudra cependant qu'ils soient de même type de données et de même taille que l'attribut de la clé primaire)

```
create table Villes
(
codeVille char(3) constraint pkville primary key,
nomVille varchar(40) not null,
poulation number(8,0)
);
create table Circuits
(
idCircuit number(4,0),
nomCircuit varchar2(40),
codeVilledebut char(3),
codeVillefin char(3),
coutCircuit number(6,2),
dureeCircuit number(4,0),
constraint pkcircuit primary key (idCircuit),
constraint fkvileD Foreign key (codeVilleDebut) references villes(codeVille),
constraint fkvileA Foreign key (codeVilleFin) references villes(codeVille)
);
```

On veut écrire la requête qui ramène les circuits avec le nom des villes début du circuit et les noms des villes fin du circuit.
Comme il y a deux clés étrangères dans la table Circuits sur le même attribut de la table Villes, il faudra faire deux jointures sur la table Villes. C'est comme si nous avions deux tables Villes, une qui pour les débuts de circuits et l'autre pour la fin des circuits.

Dans la première jointure – FROM-- sur la ville nous avions donné l'alias De pour la table villes (pour départ) pour chercher le nom de ville de départ. Et pour la deuxième jointure sur la ville nous avion donné l'alias Ar pour la table villes (pour Arrivée).

Dans le SELECT, l'alias sur l'attribut nomville avec le mot AS n'est pas obligatoire mais il sert à distinguer la ville de départ de la ville fin.

```
SELECT nomCircuit, De.nomville as VilleDepart ,Ar.nomVille As villeArrivee,
coutCircuit,dureeCircuit
FROM (
(Villes De INNER JOIN Circuits ON Circuits.CODEVILLEDEBUT = De.Codeville)
INNER JOIN villes Ar ON Circuits.CODEVILLEFIN = Ar.codeville);
```

Jointure externe

Jointure externe droite (RIGHT OUTER JOIN)

Dans la jointure externe droite, des enregistrements de table à droite de la jointure seront ramenés même si ceux-ci n'ont pas d'occurrences dans la première table.

Jointure externe gauche (LEFT OUTER JOIN)

Dans la jointure externe gauche des enregistrements de table à gauche de la jointure seront ramenés même si ceux-ci n'ont pas d'occurrences dans la deuxième table.

Dans le cas d'une jointure externe, il faut faire suivre la colonne pour laquelle il n'est pas obligatoire d'avoir des lignes correspondant à l'égalité par l'opérateur LEFT OUTER JOIN ou RIGHT OUTER JOIN

Exemple 6

Cette requête ramène tous les étudiants y compris ceux qui ne sont pas inscrits dans un programme

```
SELECT nom, Preno, nomprogramme
FROM Etudiants LEFT OUTER JOIN programmes  ON Etudiants. codeprg =programmes. codeprg;
```

	NOM	PRENOM	NOMPROGRAMME
1	Bouvier	Marge	Informatique
2	Simpson	bart	Informatique
3	Farar	Chantal	Informatique
4	Poirier	Juteux	Informatique
5	Patoche	Alain	Informatique
6	Saturne	Alain	Administration
7	Simpson	Christian	Administration
8	Patoche	Alain	Administration
9	Lefou	Duvillage	Santé animales
10	aaa	bbb	(null)
11	Bien	Thiery	(null)

Attention 

Notez que la table Etudiants est à GAUCHE de la jointure, donc c'est TOUS les étudiants qu'on veut ramener. Mais que les programme ayants des étudiants inscrits.

Exemple 7

La requête suivante fait une jointure entre les deux tables Etudiants et Programmes. Elle ramène également les programmes qui n'ont pas d'étudiants.

```
SELECT nom, Preno, nomprogramme
FROM Etudiants  RIGHT OUTER JOIN programmes  ON Etudiants. codeprg =programmes.
codeprg;
```

Attention 

Notez que la table Programmes est à DROITE de la jointure, donc c'est TOUS les programmes qu'on veut ramener. Mais que les étudiants ayant un programme

	NUMAD	NOM	PRENOM	NOMPROGRAMME
1	21	Bouvier	Marge	Informatique
2	12	Farar	Chantal	Informatique
3	14	Lefou	Duvillage	Santé animales
4	50	Patoche	Alain	Administration
5	10	Patoche	Alain	Informatique
6	11	Poirier	Juteux	Informatique
7	15	Saturne	Alain	Administration
8	13	Simpson	Christian	Administration
9	20	Simpson	bart	Informatique
10	(null)	(null)	(null)	AAAAAAAAAA
11	(null)	(null)	(null)	Génie industrielle
12	(null)	(null)	(null)	Hockey
13	(null)	(null)	(null)	Civilité

Si nous avons bien compris les requêtes R1 et R2 donnent le même résultat. N'Est-ce pas ?

Requête R1

```
SELECT nom, Preno, nomprogramme
FROM (Etudiants LEFT OUTER JOIN programmes ON Etudiants.codeprg =programmes. Codeprg)
;
```

Requête R2

```
SELECT nom, Preno, nomprogramme
FROM (Programmes RIGHT OUTER JOIN Etudiants ON Etudiants.codeprg =programmes.
Codeprg );
```

Le **LEFT** OUTER JOIN ou Le **RIGHT** OUTER JOIN n'a de sens que si la bonne table est placée au bon endroit.

Chapitre 7, quelques fonctions SQL : Les fonctions de groupement.

Ces fonctions sont utilisées pour traiter des groupes de rangées et d'afficher un seul résultat. Même si ce sont des fonctions de groupement, elles ne s'utilisent pas tout le temps avec la clause GROUP BY.

Les fonctions MIN et MAX

Ce sont des fonctions qui s'utilisent pour afficher la valeur MIN (ou MAX) parmi l'ensemble des valeurs de la colonne indiquée.

Syntaxes simplifiées

```
SELECT MIN (colonne) FROM nomTable [WHERE expression]
SELECT MIN (colonne) FROM nomTable [WHERE expression]
```

Exemple

```
SELECT MAX (note) FROM resultats  WHERE CODE_COURS ='KB6';
```

Les fonctions AVG et SUM

AVG s'utilise pour obtenir une valeur moyenne des valeurs de la colonne indiquée

SUM s'utilise pour obtenir une valeur totale des valeurs de la colonne indiquée

Syntaxes simplifiées

```
SELECT AVG (colonne) FROM nomTable [WHERE expression]
SELECT SUM (colonne) FROM nomTable [WHERE expression]
```

Exemple

```
SELECT AVG (NOTE) FROM RESULTATS WHERE CODE_COURS ='KB6';
```

Les fonctions VARIANCE et STDDEV

Pour calculer la variance et l'écart type sur les valeurs d'une colonne

La fonction COUNT

Cette fonction permet de compter le nombre de lignes (rangées) qui répondent à un critère.

Syntaxe générale

```
SELECT COUNT ( {*| [DISTINCT |ALL] nomcolonne}) FROM nomTable [WHERE expression]
```

D'une manière plus explicite ce sera comme décrit par les exemples suivants et en considérant la table **Joueurs** suivante :

	NUMJOUEUR	NOM	PRENOM	CODEEQUIPE	SALAIRE
1	1	PRICE	CAREY	MTL	1999999
2	2	MARKOV	ANDRÉ	MTL	1546357
3	3	SUBBAN	KARL	MTL	1654657
4	4	PATIORETTY	MAX	MTL	500000
5	10	HAMOND	ANDREW	OTT	1234565
6	6	STONE	MARC	OTT	1234567
7	9	TURIS	KYLE	OTT	870697
8	7	GALLAGHER	BRANDON	MTL	534543
9	8	TANGUAY	ALEX	AVL	1543456
10	11	PRICE	BIL	AVL	798098
11	50	(null)	Leprenom	(null)	(null)
12	52	(null)	Leprenom2	(null)	(null)

SELECT COUNT (*) FROM joueurs; Dans ce cas le résultat de la requête sera 12, puisque on ramène le nombre total des joueurs y compris ceux qui ont des colonnes avec des valeurs NULL

SELECT COUNT(nom) FROM Joueurs; Dans ce cas on calcule le nombre de joueurs selon la colonne nom. Or la colonne nom a des valeurs null. Si on utilise ce SELECT COUNT(nom) seules les valeurs non NULL seront comptées. Dans ce cas le résultat sera 10. Puisque nous avons que 10 noms.

SELECT COUNT(ALL nom) FROM Joueurs; On compte TOUS les noms sans les valeurs NULL. Donc le résultat et identique au précédent : 10

SELECT COUNT (DISTINCT nom) FROM Joueurs; Ici on compte le nombre de nom différents. Le résultat sera 09

Parfois, il est nécessaire de grouper les enregistrements avant de les compter. Par exemple dans la table joueurs précédente, comment déterminer le nombre de joueurs dans chaque équipe. ? Pour répondre à la question, il suffira de grouper les joueurs selon leurs codeequipe, puis les compter. Rien de plus simple.

Pour grouper des enregistrements on utilise la clause GROUP BY.

La clause GROUP BY : cette clause permet d'indiquer au système de regrouper des enregistrements selon des valeurs distinctes qui existent dans une table.

Exemple :

```
SELECT COUNT(*),CODEEQUIPE
FROM JOUEURS
GROUP BY CODEEQUIPE ;
```

Ce qui donne le résultat suivant :

	COUNT(*)	CODEE...
1	2	AVL
2	2	(null)
3	3	OTT
4	5	MTL

Attention 

Toutes les colonnes qui apparaissent dans le SELECT doivent apparaître dans la clause GROUP BY

Pour raffiner la requête, on pourrait utiliser un ALIAS et ordonner le résultat de la requête.

```
SELECT COUNT(*) AS nbJoueurs,CODEEQUIPE
FROM JOUEURS
GROUP BY CODEEQUIPE
ORDER BY nbJoueurs DESC;
```

	NBJOUEURS	CODEEQUIPE
1	5	MTL
2	3	OTT
3	2	AVL
4	2	(null)

Parfois, il nécessaire de cibler ou de restreindre les enregistrements spécifiés. Comme par exemple, ne sortir que les codeequipe ayant le nombre de joueurs supérieurs ou égal à 3. Dans ce cas on utilise la clause HAVING

La clause HAVING permet de mieux cibler les enregistrements spécifiés. Cette clause s'utilise à la place du WHERE une fois que le GROUP BY est réalisé.

```
SELECT COUNT(*) AS nbJoueurs,CODEEQUIPE
FROM JOUEURS
GROUP BY CODEEQUIPE
HAVING COUNT(*)>=3
ORDER BY nbJoueurs DESC;
```

Attention 

Après un GROUP BY, il n'est plus possible d'utiliser un WHERE. Utiliser à la place un GROUP BY

Les ALIAS ne sont pas utilisés dans la clause HAVING

Exemple 1 : La requête suivante va renvoyer une erreur à cause de l'alias de la colonne count(*)

```
SELECT count(*) AS totalEmpLoyes, dname
FROM (syemp s INNER JOIN sydept d ON s.deptno = d.deptno) GROUP BY d.dname
HAVING totalEmpLoyes>=5
```

Exemple2 : la requête suivante va afficher tous les départements qui ont 5 employés ou plus.

```
SELECT count(*) as totalEmpLoyes, dname
FROM (syemp s INNER JOIN sydept d ON s.deptno = d.deptno) GROUP BY d.dname
HAVING count(*)>=5;
```

IMPORTANT

Lorsque c'est possible, il est recommandé d'utiliser un WHERE à la place du HAVING,

Exemple

Requête R1

```
SELECT COUNT(*) AS nbJoueur,CODEEQUIPE
FROM JOUEURS  WHERE codeequipe ='MTL'
GROUP BY codeequipe;
```

Requête R2

```
SELECT COUNT(*) AS nbJoueur,CODEEQUIPE
FROM JOUEURS
GROUP BY CODEEQUIPE HAVING codeequipe ='MTL';
```

Les deux requêtes R1 et R2 vont retourner le même résultat qui est le suivant :

	NBJOUEUR	CODEEQUIPE
1	5	MTL

Mais, du point de vu rapidité d'exécution, pour le SGBD, la requête R1 est meilleure, car on fait le groupement d'un nombre limité d'enregistrements (à cause du WHERE)

Attention 

Les clauses GROUP BY et HAVING s'utilisent également sur les autres fonctions de groupement comme AVG, SUM, MIN, MAX

Il est possible d'utiliser les jointures avec les fonctions de groupements.

Exemple 1 :

```
SELECT COUNT(*) AS nbJoueurs,nomEquipe
FROM ( Joueurs INNER JOIN  equipes ON joueurs.codeequipe =equipes.codeequipe)
GROUP BY equipes.NOMEQUIPE
ORDER BY nbJoueurs DESC;
```

Exemple 2

```
SELECT AVG(Salaire) AS MoyenneSalaire,nomEquipe
FROM (Joueurs inner join equipes on joueurs.codeequipe =equipes.codeequipe)
GROUP BY equipes.NOMEQUIPE
HAVING AVG(salaire)>1200000;
```

Chapitre 8, les requêtes imbriquées

Définitions

Une sous requête est une requête avec la commande **SELECT** imbriquée avec les autres commandes (UPDATE, INSERT DELETE et CREATE)

Une sous-requête, peut être utilisée dans les clauses suivantes :

- La clause WHERE d'une instruction UPDATE, DELETE et SELECT
- La clause FROM de l'instruction SELECT
- La clause VALUES de l'instruction INSERT INTO
- La clause SET de l'instruction UPDATE
- L'instruction CREATE TABLE ou CREATE VIEW.

Utilisation d'une sous-requête avec la clause WHERE

Ce type de sous-requête permet de comparer une valeur de la clause WHERE avec le résultat retourné par une sous-requête, dans ce cas on utilise les opérateurs de comparaison suivant :

=, ! = , < , <= ,> ,>= , et IN.

Exemple 1 :

```
SELECT nom, prenom FROM etudiants
WHERE codeprogramme =
                    (SELECT codeprogramme FROM programmes
                      WHERE nomprogramme ='Informatique'
                    );
```

La requête en mauve, ou entre parenthèse est dite « requête interne » ou sous requêtes. Les deux requêtes forment des requêtes **imbriquées**

La requête interne retourne un seul enregistrement car le codeprogramme est unique dans la table programmes

Attention 

Lorsque la requête imbriquée est dans la clause WHERE alors la colonne sur laquelle porte la condition (colonne du WHERE) doit être dans le SELECT de la requête interne.

Remarque

Quand on regarde l'exemple 1, la requête aurait pu s'écrire à l'aide d'une jointure interne (INNER JOIN), comme suit :

```
SELECT nom, prenom
        FROM (etudiants
        INNER JOIN programmes  ON etudiants.codeprogramme=Programmes.codeProgramme)
WHERE p.nomprogramme ='Informatique';
```

Il existe des cas où des jointures peuvent s'écrire à l'aide de requêtes imbriquées. Mais les SGBDs sont conçus pour les jointures, alors il est vivement conseillé **(même obligatoire)** d'utiliser une jointure à la place de requetés imbriquées.

Attention 

Si vous pouvez écrire une requête de deux façons différentes, jointures et requêtes imbriquées alors optez pour le JOINTURE. C'est plus optimal

Exemple 2

Dans l'exemple qui suit, on cherche à ramener TOUS les étudiants qui ont des notes dans le cours de KEE.

Or, nous savons qu'il est très possible qu'il y ait beaucoup (plus que 1) étudiants (numad) qui vont répondre à ce critère de SELECTION. Le numad dans la table Resultats n'est pas unique. La requête interne renvoi plus qu'une ligne (ou plus qu'un enregistrement). L'opérateur IN est obligatoire dans ce cas.

```
SELECT nom, prenom FROM etudiants
WHERE numad IN
                (
                SELECT numad FROM resultats WHERE codecours ='KEE'
                );
```

Attention 

L'opérateur IN est utilisé lorsque la sous requête retourne plusieurs enregistrements

Exemple 3

```
SELECT nom, prenom FROM Etudiants
WHERE codeprogramme =
                    (
                    SELECT codeprogramme FROM Etudiants
                    WHERE nom ='Patoche' AND prenom =’Alain’
                    );
```

Dans cet exemple, on cherche à ramener TOUS les étudiants qui sont dans le même programme que Patoche. Cette requête **ne** peut **pas** s'écrire avec une jointure. Les deux requêtes portent sur la même table : Etudiants.

Exemple 4, les opérateurs ANY et ALL

Ces deux opérateurs s'utilisent lorsque la sous requête retourne plus qu'un enregistrement.

- On utilise l'opérateur ANY pour que la comparaison se fasse pour toutes les valeurs retournées. Le résultat est vrai si **au moins une des valeurs** répond à la comparaison
- On utilise l'opérateur ALL pour que la comparaison se fasse pour toutes les valeurs retournées. Le résultat est vrai si **toutes les valeurs** répondent à la comparaison

Voici les salaires du département no 30

```
SELECT sal FROM syemp WHERE deptno=30
```

	SAL
1	1600
2	1250
3	1250
4	2850
5	1500
6	950

Et voici les employés (avec leur salaire) du département 10

```
SELECT   ename, sal from syemp where deptno=10;
```

	ENAME	SAL
1	CLARK	2450
2	KING	5000
3	MILLER	1300

R1 : Nous souhaitons savoir, qui sont (ename) les employes du département no 10 qui ont leurs salaires plus grands que **TOUS** les salaires des employes du département 30.

La requête R1, sera :

```
SELECT ename FROM syemp
WHERE deptno=10 AND sal > ALL
                          (
                          SELECT sal FROM syemp WHERE deptno=30
                          );
```

Le résultat de cette requête sera **KING**

R2 : Si nous souhaitons savoir, qui sont (ename) les employés du département no 10 qui ont leurs salaires plus grands que **n'importe lequel** des salaires des employés du département 30.

La requête R2, sera:

```
SELECT ename FROM syemp
WHERE deptno=10 AND sal > ANY
                          (
                          SELECT sal FROM syemp WHERE deptno=30
                          );
```

Le résultat de cette requête sera

	ENAME	SAL
1	CLARK	2450
2	KING	5000
3	MILLER	1300

Analyse des résultats pour cet exemple uniquement :

Dans, le cas du **ALL,** le salaire devait être plus grand que tous les autres salaires du département 30. Il suffit qu'il soit plus grand que **le maximum** des salaires pour avoir le résultat

```
SELECT ename FROM syemp
WHERE deptno=10 AND sal >
                        (
                        SELECT MAX (sal) FROM syemp WHERE deptno=30
                        );
```

Dans, le cas du **ANY**, le salaire devait être plus grand que n'importe lequel des salaires du département 30. Il suffit qu'il soit plus grand que **le minimum** des salaires pour avoir le résultat

```
SELECT ename FROM syemp
WHERE deptno=10 AND sal >
                        (
                        SELECT MIN (sal) FROM syemp WHERE deptno=30
                        );
```

Exemple 5 , l'opérateur EXISTS

L'opérateur EXISTS teste si la requête intérieure retourne des valeurs.

Voici le contenu de votre table étudiants :

	NUMAD	NOM	PRENOM	CODEP
1	10	Patoche	Alain	420
2	11	Poirier	Juteux	420
3	12	Fafar	Chantale	420
4	13	Simpson	Christian	410
5	14	Lefou	Duvillage	430
6	15	Saturne	Alain	410
7	16	Bien	Simon	(null)
8	17	Gouin	Sébastien	(null)

La requête suivante va ramener TOUS les etudiants, car la requête interne retourne un résultat non NULL ;

```
SELECT * FROM etudiants  WHERE  EXISTS
                                        (
                                        SELECT codep FROM programmes
                                        );
```

Ne pas confondre avec IN, qui ramène des résultats lorsqu'une certaine égalité est vérifiée.

La requête suivante ne ramène que les étudiants qui ont un codep non NULL

```
SELECT * FROM etudiants  WHERE  codep IN
                                        (
                                        SELECT codep FROM programmes
                                        );
```

Les sous requêtes avec la clause SET de la commande UPDATE

Dans ce cas, la requête interne doit retourner une seule valeur.

Exemple 6

Cette requête met à jour la table etudiants. Elle met à jour le codeprogramme de l'étudiant CLARK par celui de Alain Patoche

```
UPDATE etudiants SET codeprogramme=
                                        (
                                        SELECT codep FROM etudiants
                                        WHERE nom ='Patoche'AND prenom='Alain'
                                        )
WHERE nom='CLARK';
```

Les sous requêtes avec la commande INSERT

Ce type de sous-requête permet d'insérer des données dans une table à partir d'une autre table.

La sous requête est utilisée à la place de la clause **VALUES** de la requête principale et peut retourner plusieurs résultats.

Exemple 7

```
INSERT  INTO etudiants (numad, nom)
                                    (
                                    SELECT empno,ename FROM syemp WHERE deptno=10
                                    );
```

Les sous requêtes avec la commande CREATE

Permet de créer une table (ou une vue) contenant des données issues d'une autre table.

Exemple 8 :

La requête suivante permet de créer une table etudiantsInfo de tous les étudiants en informatique.

```
CREATE TABLE etudiantsInfo  AS
                                    (
                                    SELECT * FROM etudiants WHERE codep=420
                                    );
```

Autre exemple, la table **notesKED** regroupe des informations éparpillées dans une même table.

```
CREATE TABLE notesKB6 (nomEtudiant, PrenomEtudiant,titreCours,noteKB6) AS
    SELECT nom, prenom,titrecours,note
    FROM ((etudiants E
       INNER JOIN resultats R  ON E.numad=R.numad)
       INNER JOIN cours C ON C.codecours=R.codecours)
    WHERE titrecours ='Introduction aux bases de données'
    ORDER BY note DESC;
```

Attention 

La mise à jour des données initiales (exemple note dans la table resultats) n'implique pas la mise à jour de la table créée par le CREATE TABLE …AS. (ici notesKED)

Les sous requêtes avec la clause FROM

Parfois, nous avons besoin d'extraire des informations à partir de résultats calculés, ces résultats sont ramenés par une requête SELECT. Dans ce cas nous parlons de requêtes imbriquées dans le FROM.

Exemple 9:

Nous cherchons à savoir quel est le programme ayant le maximum d'étudiants

La requête suivante nous ramène le nombre d'étudiants dans chaque programme.

```
SELECT COUNT(*), nomprogramme
FROM (programmes p INNER JOIN etudiants e ON e.CODEProgramme =p.codepramme )
GROUP BY nomprogramme
ORDER  BY COUNT(*) desc;
```

Le résultat de la requête sera le suivant :

	COUNT(*)	NOMPROGRAMME
1	4	Informatique
2	2	Administration
3	1	Santé Animale

La colonne encerclée en rouge est appelée ROWNUM

À partir de ce résultat, il suffit de faire un SELECT pour récupérer le ROWNUM =1

```
SELECT * FROM
        (
        SELECT COUNT(*), nomprogramme
        FROM (programmes p INNER JOIN etudiants e ON e.codeprogramme =p.codeprogramme)
        GROUP BY nomprogramme
        ORDER  BY COUNT(*) desc
        )
WHERE ROWNUM =1;
```

Remarque : la requête précédente pourrait s'écrire aussi de cette façon : (un peu plus long)

```
select count(*), nomprogramme
from programmes p inner join etudiants e on e.codeprogramme =p.codeprograme
 group  by nomprogramme
having count(*) =
                    (
                    select max(count(*)) from programmes p inner join etudiants e
                    on e.CODEP =p.CODEP
                    group  by p.nomprogramme
                    );
```

Chapitre 9, requêtes avec opérateurs d'ensembles.

L'opérateur INTERSECT : cet opérateur permet de ramener l'intersection des données entre deux tables, C'est-à-dire les enregistrements qui sont dans les deux tables. Les deux commandes SELECT doivent avoir le même nombre de colonnes et des colonnes de même type et dans le même ordre.

Syntaxe:

Instruction SELECT 1

INTERSECT

Instruction SELECT 2

[ORDER BY].

Attention 

- **Le nombre de colonnes renvoyées par SELECT 1 doit être le même que celui renvoyé par SELECT 2**
- **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.**

Voici le contenu de la table **Etudiants :**

	NUMAD	NOMETUDIANT	PRNETUDIANT	CODEP
1	10	Patoche	Alain	420
2	11	Poirier	Juteux	420
3	12	Fafar	Chantale	420
4	13	Simpson	Christian	410
5	14	Lefou	Duvillage	430
6	15	Saturne	Alain	410

Voici le contenu de la table Enseignants :

	NUME	NOMENSEIGNANT	PRNENSEIGNANT	CODEDEP	ADRESSE
1	1	Patoche	Alain	inf	Montréal
2	2	Leroi	Simba	rsh	Montréal
3	3	Lefou	Duvillage	cmp	Laval
4	4	Simpson	Lisa	inf	Montréal
5	5	Fafar	Chantale	rsh	Laval
6	6	Bien	Thierry	ele	Montréal
7	7	Lepage	Patrice	inf	Laval

Exemple 1 :

```
SELECT  nomEtudiant AS nom, prnEtudiant AS prenom FROM etudiants

INTERSECT

SELECT nomenseignant, prnenseignant FROM enseignants

ORDER BY 1;
```

Le résultat de cette requête est tous les étudiants qui sont également des enseignants (puis qu'ils sont dans les deux tables)

	NOM	PRENOM
1	Fafar	Chantale
2	Lefou	Duvillage
3	Patoche	Alain

Attention 

Dans les requêtes avec opérateurs d'ensembles, les noms de colonnes sont ceux de la première requête. Il est fortement recommandé de renommer ces colonnes

Attention 

Les requêtes suivantes vont renvoyer des erreurs.

R1, renvoi l'erreur « type de données incompatible », codep est de type NUMBER alors que codedep est type CHAR.

```
SELECT  nomEtudiant AS nom, prnEtudiant AS prenom, codep as code FROM etudiants
INTERSECT
SELECT nomenseignant, prnenseignant,codedep FROM enseignants
ORDER BY 1;
```

R2 , renvoi l'erreur « contient un nombre incorrect de resultat » pour dire que les deux SELECT * N'ont pas le même nombre de colonnes.

```
SELECT  * FROM etudiants
INTERSECT
SELECT * FROM enseignants
ORDER BY 1;
```

L'opérateur UNION

Cet opérateur renvoi 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

Syntaxe

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

Attention 

Les mêmes contraintes qui s'appliquent pour INTERSECT s'appliquent pour UNION

Exemple2

```
SELECT  nomEtudiant AS nom, prnEtudiant AS prenom FROM etudiants
UNION
SELECT nomenseignant, prnenseignant FROM enseignants
ORDER BY 1;
```

Le résultat de la requête est :

NOM	PRENOM
Bien	Thierry
Fafar	Chantale
Lefou	Duvillage
Lepage	Patrice
Leroi	Simba
Patoche	Alain
Poirier	Juteux
Saturne	Alain
Simpson	Christian
Simpson	Lisa

Vous remarquerez que : Patoche Alain, Fafar Chantal et Lefou Duvillage ne sont ramenées qu'une seule fois.

Attention 

L'opérateur UNION n'a pas de doublons. Si vous voulez ramener les doublons, utilisez UNION ALL

Exemple 3

```
SELECT  nomEtudiant AS nom, prnEtudiant AS prenom FROM etudiants
UNION ALL
SELECT nomenseignant, prnenseignant FROM enseignants
ORDER BY 1;
```

	NOM	PRENOM
1	Bien	Thierry
2	Fafar	Chantale
3	Fafar	Chantale
4	Lefou	Duvillage
5	Lefou	Duvillage
6	Lepage	Patrice
7	Leroi	Simba
8	Patoche	Alain
9	Patoche	Alain
10	Poirier	Juteux
11	Saturne	Alain
12	Simpson	Lisa
13	Simpson	Christian

Vous remarquerez que : Patoche Alain, Fafar Chantal et Lefou Duvillage ne sont ramenées deux fois.

L'opérateur MINUS

Cet opérateur renvoi l'ensemble des lignes de la première table MOINS les lignes de la deuxième table.

Syntaxe

Instruction SELECT1

MINUS

Instruction SELECT 2

[ORDER BY].

Attention 

Les mêmes contraintes qui s'appliquent pour INTERSECT s'appliquent pour UNION

Exemple 4 :

```
SELECT  nomEtudiant AS nom, prnEtudiant AS prenom FROM etudiants
MINUS
SELECT nomenseignant, prnenseignant FROM enseignants
ORDER BY 1;
```

Le résultat est :

NOM	PRENOM
Poirier	Juteux
Saturne	Alain
Simpson	Christian

Attention 

Si les opérateurs UNION et INTERSECT sont commutatifs, l'opérateur MINUS ne l'est pas.

Les instructions suivantes R1 et R2 donnent le même résultat.

```
R1 :   Instruction SELECT 1 UNION [ALL] Instruction SELECT 2
R2 :   Instruction SELECT 2 UNION [ALL] Instruction SELECT 1
```

Les instructions suivantes R3 et R4 **NE donnent PAS** le même résultat.

```
R3  :  Instruction SELECT 1 MINUS Instruction SELECT 2
R4 :  Instruction SELECT 2 MINUS  Instruction SELECT 1
```

Remarques

1. Les instructions SELECT1 et SELECT2 sont des instructions SQL à part entière. (Sans le ORDER BY). Elles peuvent donc contenir des jointures, des fonctions de groupement et des clause WHERE.
2. On peut avoir plusieurs requêtes SELECT dans une requête avec opérateurs d'ensemble :

Exemple :

```
Instruction SELECT 1 UNION ALL Instruction SELECT 2 UNION
Instruction SELECT 3 UNION ALL Instruction SELECT 4
```

Application :

Créer les tables COURSREG, des cours à la formation au régulier et COURSFC des cours de la formation continue comme suit, puis répondre aux questions

```
--------------COURS Au régulier
CREATE TABLE COURSREG (CODE_COURS VARCHAR2(8) CONSTRAINT
PK1COURS PRIMARY KEY,
NOMCOURS VARCHAR2(40));

INSERT INTO COURSFC VALUES('420-205','RESEAUX ET TELECOM');
INSERT INTO COURSFC VALUES('420-301','ANALYSE ET MODELISATION');
INSERT INTO COURSFC VALUES('205-301','FRANÇAIS');
INSERT INTO COURSFC VALUES('205-302','ANGLAIS LANGUE SECONDE');
INSERT INTO COURSFC VALUES('205-303','ANGLAIS ');
COMMIT;

-----------COURS À LA FORMATION CONTINUE---
CREATE TABLE COURSFC (CODECOURS VARCHAR2(8) CONSTRAINT
PKCOURS PRIMARY KEY,
TITRECOURS VARCHAR2(40));

INSERT INTO COURSREG VALUES('420-205','RESEAUX ET TELECOM');
INSERT INTO COURSREG VALUES('420-301','ANALYSE ET MODELISATION');
INSERT INTO COURSREG VALUES('420-KED','BASE DE DONNÉES');
INSERT INTO COURSREG VALUES('420-KEG','GESTION DE RESEAUX');
COMMIT;
```

1. Écrire une requête qui affiche tous les cours de la FC et ceux des cours au régulier
2. Écrire une requête qui affiche les cours de la FC et qui se donnent au régulier
3. Écrire une requête qui affiche les cours qui se donnent au régulier mais pas à la FC.

Chapitre 10, les vues pour simplifier les requêtes

Définition

Une vue c'est une table dont les données ne sont pas physiquement stockées mais se réfèrent à des données stockées dans d'autres tables. C'est une fenêtre sur la base de données permettant à chacun de voir les données comme il le souhaite.

On peut ainsi définir plusieurs vues à partir d'une seule table ou créer une vue à partir de plusieurs tables. Une vue est interprétée dynamiquement à chaque exécution d'une requête qui y fait référence.

Avantages

- Les vues permettent de protéger l'accès aux tables en fonction de chacun des utilisateurs. On utilise une vue sur une table et on interdit l'accès aux tables. C'est donc un moyen efficace de protéger les données
- Les vues permettent de simplifier la commande SELECT avec les sous requêtes complexes. On peut créer une vue pour chaque sous-requête complexe, ce qui facilite sa compréhension
- Il est possible de rassembler dans un seul objet (vue) les données éparpillées

Une vue se comporte dans la plupart des cas comme une table. On peut utiliser une vue comme source d'information dans les commandes SELECT, INSERT, UPDATE ou DELETE. Une vue est créée à l'aide d'une sous-requête comme suit :

Syntaxe

```
CREATE [OR REPLACE ][FORCE] VIEW <nom_de_la_vue> AS <sous_requête>
[WITH CHECK OPTION]
```

OR REPLACE : commande optionnelle qui indique lors de la création de la vue de modifier la définition de celle-ci ou de la créer si elle n'existe pas

FORCE : permet de créer la vue même si les sources de données n'existent pas.

WITH CHECK OPTION : cette option permet de contrôler l'accès à la vue et par conséquent à la table dont elle est issue (voire exemple plus loin)

Contraintes d'utilisation

Vous ne pouvez pas :

- Insérer dans une table à partir d'une vue, si la table contient des colonnes NOT NULL et qui n'apparaissent dans la vue il y'aura violation de contraintes d'intégrité
- Insérer ou mettre à jour dans la vue si la colonne en question est un résultat calculé.
- Insérer ou mettre à jour (INSERT, UPDATE et DELETE) si la vue contient les clauses GROUP BY ou DISTINCT.
- Utiliser une vue comme source de données pour INSERT, UPDATE et DELETE si elle définit avec :
    - Une opération d'ensemble
    - Une clause GROUP BY, CONNECT BY, DISTINCT de l'instruction SELECT
    - Une fonction de groupe (SUM, MAX…)

De manière générale, on peut dire que les commandes SQL INSERT, UPDATE et DELETE ne peuvent s'appliquer qu'a une vue utilisant une table avec restriction et sélection.

Attention 

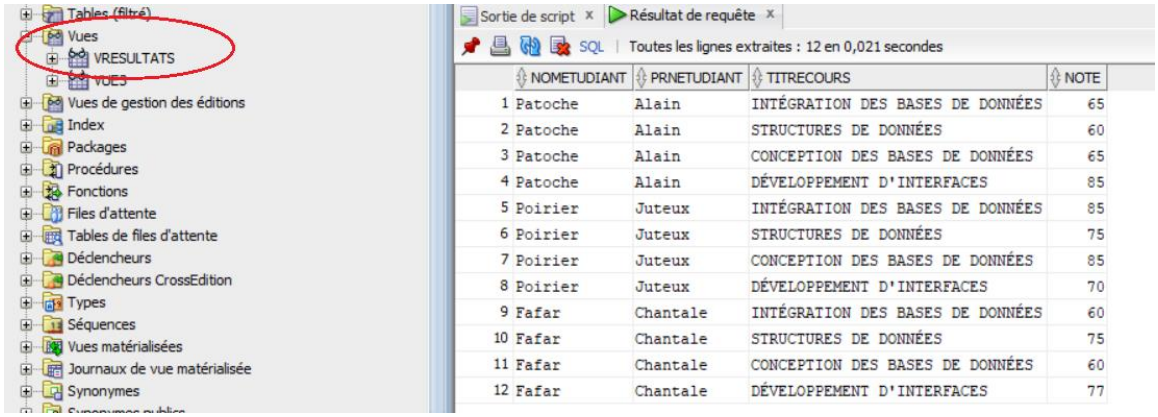
Ce qu'il faut comprendre :

1- Les vues sont dynamiques, la mise à jour (INSERT, UPDATE, DELETE) des tables entraine la mise de la view.
2- Il est possible de mettre à jour une table à partir d'une vue si la vue est issue d'une table avec sélection (SELECT simple ou SELECT AVEC JOINTURE) et restriction (WHERE)

Exemple 1

```
CREATE VIEW Vresultats as
SELECT E.NOMETUDIANT, E.PRNETUDIANT,titrecours,note
FROM ((etudiants E INNER JOIN resultats R  ON E.numad=R.numad)
INNER JOIN cours C ON C.codecours=R.codecours);
```

Si on exécute SELECT * FROM Vresultats, on a la sortie suivante.



	NOMETUDIANT	PRNETUDIANT	TITRECOURS	NOTE
1	Patoche	Alain	INTÉGRATION DES BASES DE DONNÉES	65
2	Patoche	Alain	STRUCTURES DE DONNÉES	60
3	Patoche	Alain	CONCEPTION DES BASES DE DONNÉES	65
4	Patoche	Alain	DÉVELOPPEMENT D'INTERFACES	85
5	Poirier	Juteux	INTÉGRATION DES BASES DE DONNÉES	85
6	Poirier	Juteux	STRUCTURES DE DONNÉES	75
7	Poirier	Juteux	CONCEPTION DES BASES DE DONNÉES	85
8	Poirier	Juteux	DÉVELOPPEMENT D'INTERFACES	70
9	Fafar	Chantale	INTÉGRATION DES BASES DE DONNÉES	60
10	Fafar	Chantale	STRUCTURES DE DONNÉES	75
11	Fafar	Chantale	CONCEPTION DES BASES DE DONNÉES	60
12	Fafar	Chantale	DÉVELOPPEMENT D'INTERFACES	77

À la création, les vues sont visibles à l'onglet VUES des objets de la base de données.

Si on exécute la requête suivante

```
UPDATE  resultats SET note =100 WHERE numad = 10 and codecours ='KED';
```

Puis un SELECT * FROM Vresultats ;, on aura la sortie suivante :

	NOMETUDIANT	PRNETUDIANT	TITRECOURS	NOTE
1	Patoche	Alain	INTÉGRATION DES BASES DE DONNÉES	65
2	Patoche	Alain	STRUCTURES DE DONNÉES	60
3	Patoche	Alain	CONCEPTION DES BASES DE DONNÉES	100
4	Patoche	Alain	DÉVELOPPEMENT D'INTERFACES	85
5	Poirier	Juteux	INTÉGRATION DES BASES DE DONNÉES	85
6	Poirier	Juteux	STRUCTURES DE DONNÉES	75
7	Poirier	Juteux	CONCEPTION DES BASES DE DONNÉES	85
8	Poirier	Juteux	DÉVELOPPEMENT D'INTERFACES	70
9	Fafar	Chantale	INTÉGRATION DES BASES DE DONNÉES	60
10	Fafar	Chantale	STRUCTURES DE DONNÉES	75
11	Fafar	Chantale	CONCEPTION DES BASES DE DONNÉES	60
12	Fafar	Chantale	DÉVELOPPEMENT D'INTERFACES	77

On voit bien que la note de Alain Patoche a changé. Les vues sont DYNAMIQUES

Exemple 2,

Cet exemple permet de simplifier la requête pour extraire les n meilleurs étudiants.

```
CREATE VIEWS Vresultats2 as
SELECT E.NOMETUDIANT, E.PRNETUDIANT,titrecours,note
FROM ((etudiants E INNER JOIN resultats R  ON E.numad=R.numad)
INNER JOIN cours C ON C.codecours=R.codecours) ORDER BY note desc;

SELECT * FROM Vresultats2 WHERE ROWNUM<=4;
```

Si on execute:

```
UPDATE Vresultats2 SET note =0 WHERE NOMETUDIANT='Patoche'  AND titrecours like
'%CONCE%';
```

Puis

```
SELECT * FROM Vresultat2;
```

	NOMETUDIANT	PRNETUDIANT	TITRECOURS	NOTE
1	Poirier	Juteux	CONCEPTION DES BASES DE DONNÉES	85
2	Poirier	Juteux	INTÉGRATION DES BASES DE DONNÉES	85
3	Patoche	Alain	DÉVELOPPEMENT D'INTERFACES	85
4	Fafar	Chantale	DÉVELOPPEMENT D'INTERFACES	77
5	Poirier	Juteux	STRUCTURES DE DONNÉES	75
6	Fafar	Chantale	STRUCTURES DE DONNÉES	75
7	Poirier	Juteux	DÉVELOPPEMENT D'INTERFACES	70
8	Patoche	Alain	INTÉGRATION DES BASES DE DONNÉES	65
9	Fafar	Chantale	INTÉGRATION DES BASES DE DONNÉES	60
10	Fafar	Chantale	CONCEPTION DES BASES DE DONNÉES	60
11	Patoche	Alain	STRUCTURES DE DONNÉES	60
12	Patoche	Alain	CONCEPTION DES BASES DE DONNÉES	0

Maintenant si on fait SELECT * FROM RESULTATS;

CODECOURS	NUMAD	NOTE
KED	10	0
KA5	10	85
KEE	10	60
KED	11	85
KA5	11	70
KEE	11	75
KED	12	60
KA5	12	77
KEE	12	75
KHG	10	65
KHG	11	85
KHG	12	60

Votre table RESULTATS a été mise à jour par la vue.

Exemple 3, l'option WITH CHECK OPTION.

```
CREATE VIEW ETUDIANTINFO AS
SELECT * FROM ETUDIANTS
WHERE CODEP =420
WITH CHECK OPTION;
```

La vue précédente permet d'avoir les étudiants d'informatique.

Les opérations DML suivantes sont permises et mettent à jours la table **ETUDIANTS.**

Ceci va fonctionner car l'étudiant qu'on rajoute a le code 420.

```
INSERT INTO ETUDIANTINFO VALUES(20,'Lenouveau','Etudiant',420);
```

Ceci va fonctionner car l'étudiant qu'on modifie a le code420

```
UPDATE ETUDIANTINFO SET prnEtudiant ='LenouveauPrenom' where numad =12;
```

Ceci NE VA PAS fonctionner car l'étudiant qu'on rajoute n'a PAS le code 420.

```
INSERT INTO ETUDIANTINFO VALUES(30,'autre','Etudiant',410);
```

Erreur SQL : ORA-01402: vue WITH CHECK OPTION - violation de clause WHERE

01402. 00000 - "view WITH CHECK OPTION where-clause violation"

Ceci NE VA PAS fonctionner car l'étudiant qu'on modifie n'a PAS le code 420.

```
UPDATE ETUDIANTINFO SET prnEtudiant ='LenouveauPrenom' where numad =15;
```

Attention 

L'option WITH CHECK OPTION permet de restreindre les opérations autorisées sur la base de données.

Détruire une VUE :

DROP VIEW nom_de_vue permet de supprimer la vue

Renommer une vue :

RENAME ancien_nom TO nouveau

Ce qu'il faut absolument retenir

1. Les vues sont des objets de la base de données. On utilise la commande CREATE pour créer une vue. On utilise la commande DROP pour détruire une vue.
2. Une vue se crée à l'aide d'une sous-requête.
3. Les vues sont dynamiques : ce qui veut dire que la vue se met à jour automatiquement lorsque la table (les tables) dont elle issue est mise à jour.
4. Les vues servent à simplifier l'écriture de requêtes.
5. Les vues servent à mettre ensemble des données éparpillées.
6. Les vues servent à protéger les données de la BD (cours de session 3).

Exemple :

Dans une question du laboratoire 6, on vous demandait : Qui sont les **deux** meilleurs étudiants en CONCEPTION DE BASES DE DONNÉES ?
Évidemment, la requête peut être considérée comme complexe. Mais si vos données étaient regroupées quelque part … ce serait simple.
Si vous avez répondu correctement à la question, vous avez écrit ceci :

```
SELECT * FROM (
SELECT et.numad,nom, prenom, note FROM
((etudiants et INNER JOIN resultats rs ON et.numad=rs.numad)
INNER JOIN cours cr ON rs.codecours =cr.codecours)
WHERE cr.titrecours='CONCEPTION DE BASES DE DONNÉES'
ORDER BY note DESC
)
WHERE ROWNUM <=2;
```

En examinant la requête, on pourrait se poser la question : Peut-on garder les données de la requête la plus interne (requête en jaune) quelque part ? La réponse est : OUI dans une VUE. Dans cette requête, les données éparpillées sont mises ensemble.
On va utiliser une VUE pour garder des valeurs renvoyées par une sous-requête. Ce qui veut dire que l'on va stocker des résultats intermédiaires.

Si je fais :

```
CREATE VIEW VresultatsKED AS
SELECT et.numad,nom, prenom, note FROM
((etudiants et INNER JOIN resultats rs ON et.numad=rs.numad)
INNER JOIN cours cr ON rs.codecours =cr.codecours)
WHERE cr.titrecours='CONCEPTION DES BASES DE DONNÉES'
ORDER BY note DESC ;
```

1. Exécuter le code pour créer la VUE
2. Examiner le contenu de la VUE VresultatsKED par un simple SELECT *

VresultatsKED

	NUMAD	NOM	PRENOM	NOTE
1	11	POIRIER	JUTEUX	85
2	10	PATOCHE	ALAIN	65
3	12	FAFAR	CHANTAL	60

À partir de là, vous pouvez extraire les données que vous voulez. Les deux meilleurs étudants, le meilleur étudiant, etc…
La requête est : (on va l'appeler R1)

```
SELECT * FROM  VresultatsKED WHERE ROWNUM<=2;
```

Facile. N'est ce pas ?

Question :Et si j'ajoute des données dans la table Resultats ou je modifie la note d'un etudiants en KED, la requête R1 est-elle toujours valable ?
Réponse : Aucun problème, la requête R1 est toujours valable puisque la VUE est DYNAMIQUE.

On teste ?

Étape1 : D'abord, on commence par ajouter deux étudiants en informatique (car dans la table Etudiants, il n'y que trois étudiants. Si on avait plus d'étudiants, cette étape serait innutile) Exécuter ceci :

```
INSERT INTO ETUDIANTS VALUES(20,'Saquet','Frodon',420);
INSERT INTO ETUDIANTS VALUES(21,'LeMalin','Simba',420);
```

Étape 2 : on attribut des notes en KED pour les étudiants que l'on vient d'ajouter.

1. Par curiosité, faites un SELECT * FROM Resultats (la table) juste pour voir le contenu.
2. Maintenant, exécuter les instructions suivantes :

```
INSERT INTO resultats (numad,codecours,note) VALUES(20,'KED',90);
INSERT INTO resultats (numad,codecours,note) VALUES(21,'KED',80);
COMMIT;
```

3. Puis faites un SELECT * FROM **VresultatsKED**; (on utilise la vue)

En principe vous avez ceci. On voit bien que les étudiants dont NUMAD =20 et NUMAD =21 ont été ajoutés dans la VUE. Les VUES sont DYNAMIQUES.

NUMAD	NOM	PRENOM	NOTE
20	Saquet	Frodon	90
11	POIRIER	JUTEUX	85
21	LeMalin	Simba	80
10	PATOCHE	ALAIN	65
12	FAFAR	CHANTAL	60

Évidement si vous faites un SELECT * from **Resultats**, vous aurez la même chose.

Chapitre 11, gestion de l'information hiérarchisée.

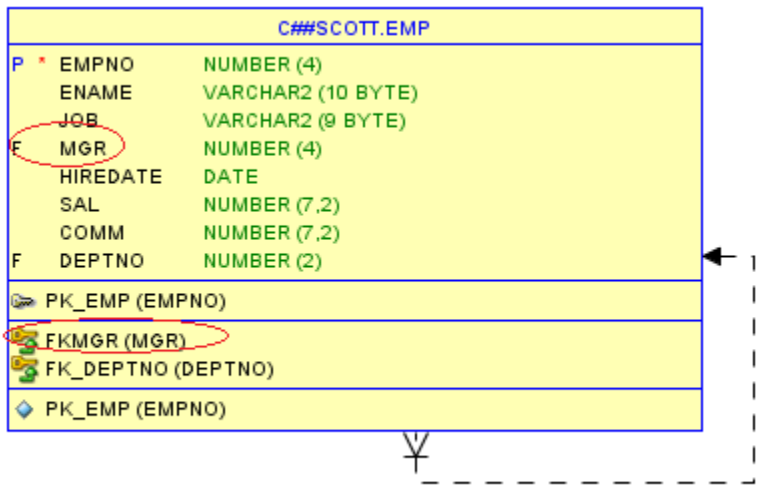
Définition

Des données hiérarchiques sont des données stockées dans une table avec une relation récursive (relation sur la même table ou entité) dans une cardinalité maximale est 1. La table contient au moins deux colonnes : une colonne de clé primaire et une colonne définissant la clé étrangère (clé enfant) et qui réfère à la clé primaire (clé parent). Un exemple de la table EMPLOYES dans laquelle on souhaite mettre en évidence la hiérarchisation entre les employés (employés avec les responsables hiérarchiques).

Diagramme référentiel :

Le diagramme référentiel sera une relation sur la même table. Une clé étrangère qui fait référence à une clé primaire de la même table (voir exemple)

Ici, la colonne MGR, a une contrainte de FK et fait référence à empno qui est la PK



Exemple : Dans la table EMPLOYES, on voit bien les employés avec leur responsable direct. Comme par exemple DUBOIS son responsable qui est ROY, et ROY a son responsable Cristophe et ainsi de suite

Pour mettre en évidence la hiérarchie dans une table lors d'une sélection, la commande SELECT est accompagnée de deux nouvelles clauses. CONNECT BY et START WITH

Syntaxe :

```
SELECT <nom_de colonne1,…nom_de_colonnen>
FROM <nom_de table>
[START WITH <condition>]
CONNECT  BY [ PRIOR] <condition>
```

NUM	NOM	NUMRES
17	DUBOIS	16
10	alibaba	(null)
12	Martin	10
11	Patoche	10
13	Cristophe	10
14	FAFAR	11
16	ROY	13

Exemple1

```
SELECT NUM, NOM,NUMRES,LEVEL
FROM EMPLOYES
START WITH NUM =10
 CONNECT BY PRIOR NUM = NUMRES;
```

Cette requête nous donne tous les employés sous l'employé dont le numéro est 10.

NUM	NOM	NUMRES	LEVEL
10	alibaba	(null)	1
11	Patoche	10	2
14	FAFAR	11	3
12	Martin	10	2
13	Cristophe	10	2
16	ROY	13	3
17	DUBOIS	16	4

Exemple 2

Cette requête va nous donner tous les employés sous l'employé Patoche

```
SELECT num, NOM,NUMRES,LEVEL
 FROM EMPLOYES
START WITH NOM ='Patoche'
 CONNECT BY PRIOR NUM = NUMRES;
```

NUM	NOM	NUMRES	LEVEL
11	Patoche	10	1
14	FAFAR	11	2

Les clauses:

CONNECT BY: cette clause est obligatoire, elle permet de connecter deux colonnes (clé primaire et clé enfant) dans une même table et indique au système comment présenter l'information (dans notre cas, si on souhaite avoir les subordonnées ou les responsables)

PRIOR : indique le sens du parcours de la hiérarchie (ou de l'arbre).

Selon qu'il soit placé à gauche de la clé parent, on parcourt l'arbre vers le bas (on extrait les subordonnés), ou à droite de la clé parent, on parcourt l'arbre vers le haut (on extrait les supérieurs)

```
SELECT num, NOM,NUMRES,LEVEL
 FROM PERSONNE
START WITH NOM='DUBOIS'
 CONNECT BY PRIOR NUM =  NUMRES;
```

```
SELECT num, nom,numres,level
 FROM PERSONNE
START WITH NOM='DUBOIS'
 CONNECT BY NUM = PRIOR NUMRES;
```

NUM	NOM	NUMRES	LEVEL
17	DUBOIS	16	1
16	ROY	13	2
13	Cristophe	10	3
10	alibaba	(null)	4

Dans le premier cas, nous avons ramené les subordonnées de DUBOIS

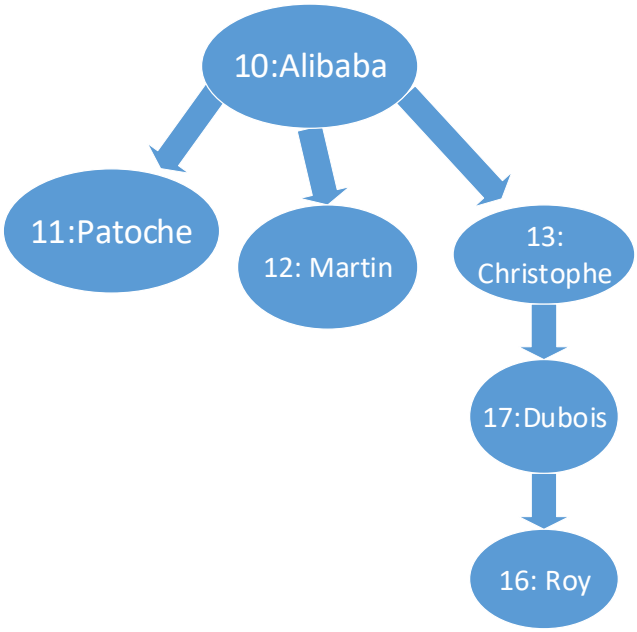
Dans le deuxième cas, nous avons ramené tous les responsables de DUBOIS.

START WITH : cette clause est optionnelle, elle indique, pour quelle occurrence (dans notre cas pour quel employé) on doit sélectionner les subordonnées ou les supérieurs

La pseudo colonne LEVEL est ajoutée pour montrer le niveau de hiérarchie entre les enregistrements.

Les données précédentes sont présentées sous forme d'un arbre à l'envers :

Alibaba n'a pas de supérieur hiérarchique. Ça prend null au sommet sinon nous sommes dans une récursion sans fin. Alibaba est la racine de l'arbre.



Attention

- Les données hiérarchiques sont représentées sous forme d'un arbre à l'envers. La racine au sommet. Ce qui veut dire qu'il y a une **FIN.**
- Pour garantir qu'il n'y ait pas de récursivité infinie, la valeur du **Parent de la racine** doit être **NULL**. Par exemple, dans l'entreprise il y aura toujours un BOSS (il n'a pas de supérieur hiérarchique)
- Le CONNECT BY joue un peu le rôle de INNER JOIN, puisqu'il lie une clé parente à une clé enfant.
- PRIOR , qu'il soit à gauche ou à droite de l'égalité, il est important de voir si c'est l'attribut PARENT ou l'attribut ENFANT qui est juste après le PRIOR. → Voir exemple plus bas

Exemple 3 : Ces deux requêtes ramènent les supérieurs hiérarchiques de JONES. Alors PRIOR est-il à gauche ou à droite ?

Ce qu'il faut remarquer c'est que dans les deux cas PRIOR suit mgr (pour cet exemple)

```
SELECT ename, mgr from syemp
WHERE ename !='JONES'
START WITH ename ='JONES'
CONNECT BY  empno = PRIOR  mgr;
```

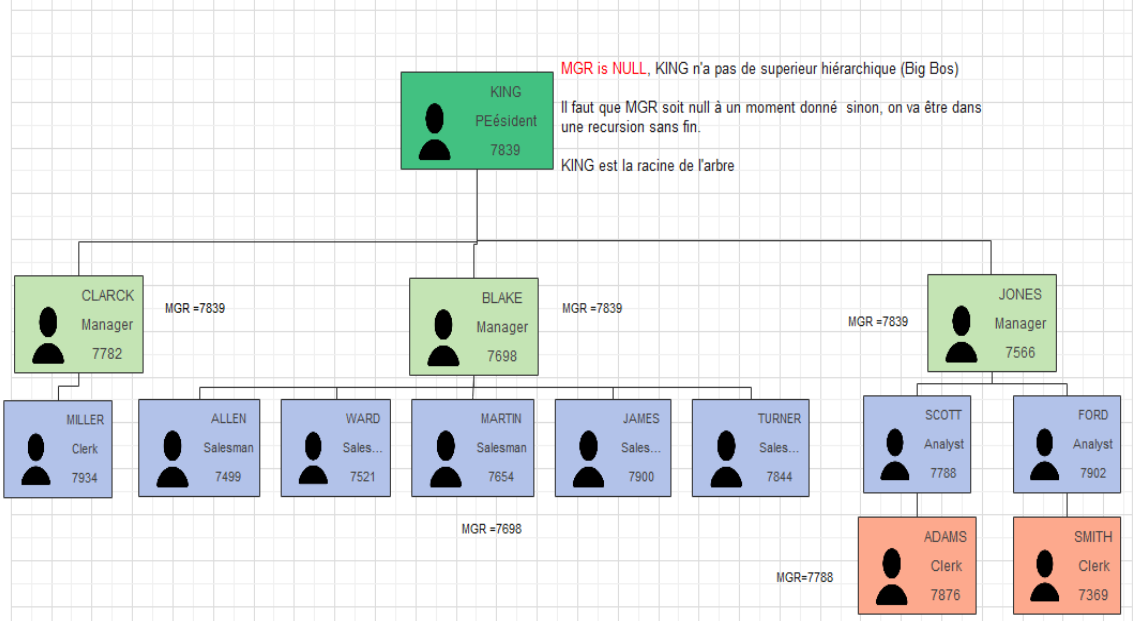
```
SELECT  ename, mgr FROM syemp
WHERE ename !='JONES'
START WITH ename ='JONES'
CONNECT BY PRIOR mgr =empno;
```

Rappelez-vous… RIGHT OUTER JOIN et LEFT OUTER JOIN…. . Est-ce un LEFT… ? Est-ce un RIGHT ? … on connait la réponse quand on sait la table qui est à gauche ou à droite. Même chose avec PRIOR.

La table syemp

EMPNO	ENAME	JOB	MGR
7839	KING	PRESIDENT	(null)
7698	BLAKE	MANAGER	7839
7782	CLARK	MANAGER	7839
7566	JONES	MANAGER	7839
7902	FORD	ANALYST	7566
7369	SMITH	CLERK	7902
7499	ALLEN	SALESMAN	7698
7521	WARD	SALESMAN	7698
7654	MARTIN	SALESMAN	7698
7844	TURNER	SALESMAN	7698
7900	JAMES	CLERK	7698
7934	MILLER	CLERK	7782
7876	ADAMS	CLERK	7788
7788	SCOTT	ANALYST	7566

Voici le diagramme hiérarchique de la table syemp



Conclusion

- Les requêtes hiérarchiques permettent d'afficher une relation hiérarchique existant entre des lignes d'une table.
- CONNECT BY est une clause obligatoire qui se compare à une jointure puisqu'elle lie la clé primaire et la clé étrangère.
- Il est possible de déterminer la direction (PRIOR) et le point de départ du parcours (START WITH).
- PRIOR, qu'il soit à gauche ou à droite de l'égalité, il est important de voir si c'est l'attribut PARENT ou l'attribut ENFANT qui est juste après le PRIOR.
- Les données hiérarchiques peuvent être vues comme un arbre à l'envers (voir page 5)
- L'arbre a toujours une racine, une valeur NULL au sommet(voir page 4)
- La ligne du START WITH a toujours le Level 1

Chapitre 12, quelques fonctions SQL

Dans ce qui suit, la table DUAL est utilisée pour afficher certaines informations système (comme SYSDATE)

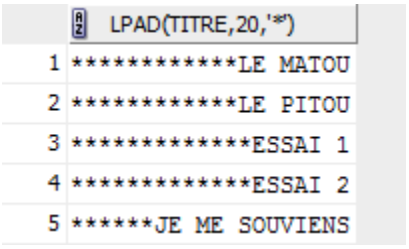
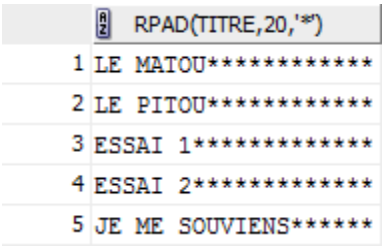
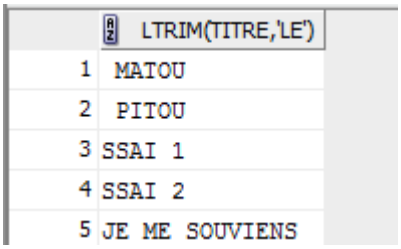
La table DUAL est une table avec une seule colonne et une seule ligne. Elle est accessible par tous les usagers en lecture seule. Elle permet d'être une source de données lorsque la source n'est pas connue. On pourrait y extraire la prochaine valeur d'une série, la date du jour ..

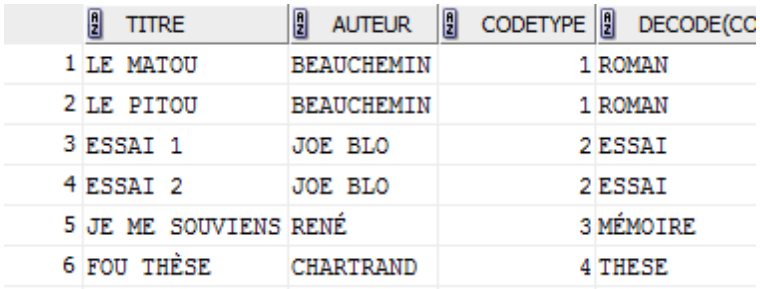
Oracle offre plusieurs fonctions pour manipuler des dates, des chaînes de caractères et des nombres. Ces fonctions seront utiles dans les situations suivantes :

- Pour convertir des chaines d'un format à un autre
- Pour générer des états
- Pour exécuter des requêtes

Les fonctions sur les chaines de caractères

onctions et syntaxes	**Rôles**	**Exemples**
LENGTH : LENGTH (colonne)	Renvoie la longueur d'une chaine de caractère	SELECT LENGTH (titre), titre FROM livres ' SELECT titre FROM livres WHERE LENGTH (titre) < 10;
UPPER UPPER (colonne)	Conversion en letter majuscule	SELECT UPPER(titre) FROM livres; SELECT * FROM LIVRES WHERE UPPER(TITRE) ='ENVOYE'
LOWER Lower (colonne)	Conversion en minuscule	SELECT LOWER (TITRE) FROM lIVRES; SELECT * FROM LIVRES WHERE LOWER(TITRE) ='envoye'
INITCAP (colonne)	Première letter en majuscule	SELECT initcap (TITRE) FROM lIVRES;
||	Concatenation	SELECT NOM || PRENOM FROM ETUDIANTS;

Colonne 1 || colonne 2		
LPAD LPAD (colonne, longueur,'chaine de remplissage)	remplir à gauche. Elle permet de générer un chaine de longueur déterminée, et dont le début la chaine passée en paramètre	SELECT LPAD(TITRE ,20,'*') FROM LIVRES; Ici la longueur totale de la chaîne est de 40 caractères. Le début de la chaîne est **********  LPAD(TITRE,20,'*') 1 **********LE MATOU 2 **********LE PITOU 3 **********ESSAI 1 4 **********ESSAI 2 5 ******JE ME SOUVIENS
RPAD	Fait la même chose que LPAD sauf que le remplissage se fait à droite	SELECT RPAD(TITRE ,20,'*') FROM LIVRES;  RPAD(TITRE,20,'*') 1 LE MATOU********** 2 LE PITOU********** 3 ESSAI 1********** 4 ESSAI 2********** 5 JE ME SOUVIENS******
LTRIM LTRIM (colonne,'chaine')	Supprime une chaine de caractères au début	SELECT LTRIM (TITRE,'LE') FROM LIVRES;  LTRIM(TITRE,'LE') 1 MATOU 2 PITOU 3 SSAI 1 4 SSAI 2 5 JE ME SOUVIENS
RTRIM	Fait la même chose que LTRIM à droit.	
DECODE DECODE (colonne, code1, chaine1,	Permet de décoder des codes au	SELECT TITRE, AUTEUR, CODETYPE, DECODE (CODETYPE,1,'ROMAN', 2,'ESSAI',3,'MÉMOIRE', 4, 'THESE')

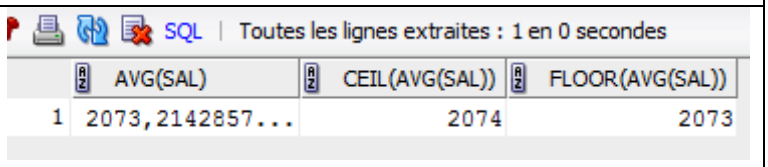
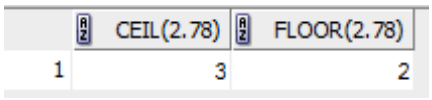
<table>
<tr><td>Code2, chaine2..)</td><td>moment de la sélection.</td><td>FROM LIVRES;
 
<table>
<tr><th></th><th>TITRE</th><th>AUTEUR</th><th>CODETYPE</th><th>DECODE(CC</th></tr>
<tr><td>1</td><td>LE MATOU</td><td>BEAUCHEMIN</td><td>1</td><td>ROMAN</td></tr>
<tr><td>2</td><td>LE PITOU</td><td>BEAUCHEMIN</td><td>1</td><td>ROMAN</td></tr>
<tr><td>3</td><td>ESSAI 1</td><td>JOE BLO</td><td>2</td><td>ESSAI</td></tr>
<tr><td>4</td><td>ESSAI 2</td><td>JOE BLO</td><td>2</td><td>ESSAI</td></tr>
<tr><td>5</td><td>JE ME SOUVIENS</td><td>RENÉ</td><td>3</td><td>MÉMOIRE</td></tr>
<tr><td>6</td><td>FOU THÈSE</td><td>CHARTRAND</td><td>4</td><td>THESE</td></tr>
</table></td></tr>
<tr><td>SUBSTR
SUBSTR (colonne,m,N) ou SUBSTR (Chaine ,m,N) ou</td><td>Permet d’extraire une portion spécifique d’une chaîne de caractères</td><td>SELECT SUBSTR (titre, 1,5)FROM livres.
À partir de la position 1, on extrait une chaîne de longueur 5.
Si le nombre m est négatif alors la recherche se fera à partir de la fin
Si le nombre N n’est pas précisé, alors tous les caractères après m seront pris</td></tr>
<tr><td>REPLACE
REPLACE (chaine1 chaine2, chaine3)</td><td>Remplace dans chaine1, lachaine2 par cahine3</td><td>SELECT titre, REPLACE (titre, 'MATOU', 'GOUROU')
FROM LIVRES;
UPDATE Fonctions SET Titre = REPLACE (Titre, ‘agent’, ‘employé’)</td></tr>
<tr><td>INSTR</td><td>Retourne la position d’une chaine dans une autre.</td><td>SELECT TITRE,COTE, INSTR (TITRE, 'ME')
FROM LIVRES WHERE COTE
LIKE 'A%' OR COTE LIKE 'C%';
<table>
<tr><td>1</td><td>LE MATOU</td><td>A001</td><td>0</td></tr>
<tr><td>2</td><td>LE PITOU</td><td>A002</td><td>0</td></tr>
<tr><td>3</td><td>JE ME SOUVIENS</td><td>C001</td><td>4</td></tr>
<tr><td>4</td><td>ENVOYE</td><td>C002</td><td>0</td></tr>
</table></td></tr>
</table>

Les fonctions sur les DATES

Fonctions et syntaxes	Rôles	Exemples
SYSDATE	Obtenir la date du jour	SELECT SYSDATE FROM DUAL;
ADD_MONTHS ADD_MONTHS (date, nb)	Ajoute un nombre nb de mois à une date. Si nb est négatif, on enlève des mois	SELECT ADD_MONTHS (DATEEMPRUNT,2) FROM EMPRUNT
NEXT_DAY	Avance la date jusqu'au jour spécifié	SELECT NEXT_DAY(SYSDATE,3)FROM DUAL;
LAST_DAY	Retourne la date du dernier jour du mois	SELECT LAST_DAY(SYSDATE)FROM DUAL;
MONTHS_BETWEEN	Retourne le nombre de mois entre deux date	SELECT MONTHS_BETWEEN (SYSDATE, DATEEMPRUNT) FROM EMPRUNT;

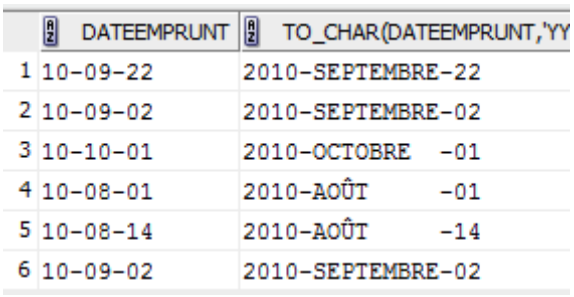
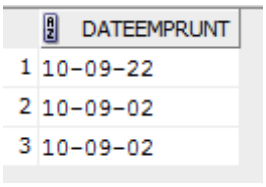
Les fonctions sur les nombres

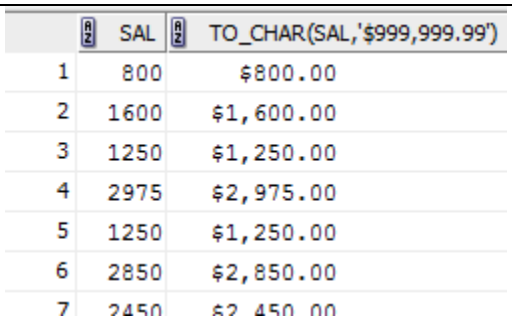
Fonctions et syntaxes	Rôles	Exemples
ROUND ROUND (nombre, m)	Reçoit deux arguments : le nombre à arrondir et m qui correspond au nbre de chiffres lors de l'arrondi	SELECT ROUND (AVG(SAL),3) FROM SYEMP; SELECT ROUND (2.78,1) FROM DUAL; A pour résultat 2.8 Si m n'est pas fourni, on arrondi au chiffre immédiatement supérieur

TRUNC (nombre, m)	Supprime la partie fractionnelle de son argument numérique	SELECT TRUNC (2.78,1) FROM DUAL; A pour résultat 2.7
CEIL	Retourne l'entier immédiatement supérieur à son argument	
FLOOR	Retourne l'entier immédiatement inférieur à son argument	SELECT CEIL (2.78), FLOOR(2.78) FROM DUAL; 
POWER POWER (nombre, n)	Élève nombre à la puissance n	
SQRT SQRT (nombre,n)	Calcule la racine carrée d'un nombre	
MOD. MOD (nombre1,nombre2)	Calcule le modulo d'un nombre	

Les fonctions de conversion

Fonctions et syntaxes	**Rôles**	**Exemples**
TO_DATE (chaine, modeledate)	Convertie une chaine de caractère en une date selon le modèle défini. Le	INSERT INTO EMPRUNT VALUES ('EX1C31',12, TO_DATE ('10-09-02', 'YY-MM-DD'));

<table>
<tr><td></td><td>modèle de date doit être dans un format valide.</td><td>SELECT * FROM Commandes WHERE SYSDATE - TO_DATE(datecommande) > 10;
Ici date commande a été creée avec le type varchar2</td></tr>
<tr><td>TO_CHAR avec les dates
TO_CHAR (valeur_date, formatDate)</td><td>Converti une date pour affiche la date selon un format prédéfini. Les fonctions sur les chaînes de caractères peuvent être utilisées</td><td>SELECT DATEEMPRUNT, TO_CHAR (DATEEMPRUNT,'YYYY-MONTH-DD') FROM EMPRUNT;
 
select dateemprunt FROM EMPRUNT
WHERE SUBSTR (to_char (dateemprunt,'YYYY-MONTH-DD'),4)
LIKE '%SEP%';
  </td></tr>
<tr><td>TO_CHAR avec les nombres
TO_CHAR (nombre, format)</td><td>Utilisée pour afficher des valeurs numérique sous un autre format..les formats numériques sont :
9 affiche une valeur numérique sans les zéro non significatifs. Autant de 9 que de chiffres.</td><td>SELECT SAL, TO_CHAR (SAL, '$999,999.99')
FROM SYEMP;</td></tr>
</table>

	0 utilisé comme le 9, sauf que les zéros non significatifs sont affichés $ affiche le symbole $ devant le nombre , affiche une virgule à la position indiquée. Séparateur de miliers. . (point) affiche un point à la position indiquée. (point décimal)	 SAL | TO_CHAR(SAL,'$999,999.99') 1 800 $800.00 2 1600 $1,600.00 3 1250 $1,250.00 4 2975 $2,975.00 5 1250 $1,250.00 6 2850 $2,850.00 7 2450 $2,450.00
TO_NUMBER	Inverse de la fonction précédente	CREATE TABLE EXEMPLE (CHAINE VARCHAR2(10)); INSERT INTO EXEMPLE VALUES ('12.78'); SELECT TO_NUMBER (CHAINE 1,'99.99') FROM EXEMPLE; affiche 12,78 (ce qui est nombre)

Les formats valides avec les dates

Formats	Exemples	Plages de valeurs
DD	TO_CHAR (DATEEMPRUNT,'DD')	Jours : 1-31 selon le mois
MM	TO_CHAR (DATEEMPRUNT,'MM-DD')	Mois 1-12
YY	TO_CHAR (DATEEMPRUNT,'YY-MM-DD')	Deux derniers chiffres de l'année
YYY	TO_CHAR (DATEEMPRUNT,'YYY-MM-DD')	Les trois derniers chiffres de l'année
YYYY	TO_CHAR (DATEEMPRUNT,'YYYY-MM-DD')	
YEAR	TO_CHAR (DATEEMPRUNT,'YEAR-MM-DD')	Année en majuscule
year		Année en minuscule
Year		Première lettre en majuscule

MON		3 première letters du mois
MONTH	TO_CHAR (DATEEMPRUNT,'YYYY-MONTH-DD')	Le mois en majuscule
month		Le mois en minuscule
DAY	TO_CHAR (DATEEMPRUNT,'YYYY-MONTH-DAY')	Jour de la semaine
day		Jour de la semaine en minuscule
DDD		Numéro du jour dans l'année : de 1 à 366 (selon l'année
D		Numéro du jour dans la semaine
WW		No de la semaine dans l'année

Suite voir http://www.techonthenet.com/oracle/functions/to_char.php

https://www.techonthenet.com/oracle/functions/index.php
ou
https://docs.oracle.com/cd/B19306_01/server.102/b14200/functions001.htm

Chapitre 13, autres objets d'Oracle : séquences, synonymes.

Les séquences

Les séquences sont des objets de la base de données. Elles sont utilisées pour générer des numéros uniques. Elles peuvent donc être utilisées pour générer une valeur de clé primaire.

```
CREATE SEQUENCE <Nom_de_sequence>
START WITH <value_de_depart>
MAXVALUE <valeur_maximale>
MINVALUE <valeur_minimale>
INCREMENT BY <intervalle>
CYCLE ou NO CYCLE
```

Exemple 1

CREATE SEQUENCE seq1 START WITH 10

MAXVALUE 100

INCREMENT BY 10;

Et on utilise le INSERT comme suit:

```
INSERT INTO EmployesInfo (numemp, nom) VALUES (seq1.nextval, 'Simpson');
```

L'employé (10, Simpson) est inséré (si c'est la première fois qu'on utilise la séquence seq1)

INSERT INTO EmployesInfo (numemp, nom) VALUES (seq1.nextval, 'Blues');

L'employé (20, Blues) est inséré (si c'est la deuxième fois qu'on utilise la séquence seq1)

Exemple 2

CREATE SEQUENCE seq2

INCREMENT BY 2

START WITH 20;

START WITH n: n indique la valeur de départ de la séquence. Dans une séquence croissante, la valeur par défaut est la valeur minimale, et dans une séquence décroissante la valeur par défaut est la valeur maximale.

INCREMENT BY n: n est le PAS. Pour préciser l'intervalle entre les nombres générés. Par défaut cette valeur est 1. Le nombre n peut être positif pour générer une séquence croissante ou négatif pour générer une séquence décroissante.

MAXVALUE n: n indique la valeur maximale de la séquence. Par défaut n=10**27 (10 puissance 27) pour une séquence positive et n=-1 pour une séquence négative.

MINVALUE n : n indique la valeur maximale de la séquence. Par défaut n=-10**27 pour une séquence décroissante et n=1 pour une séquence croissante.

CYCLE : indique que la séquence continue à générer des valeurs à partir de la valeur minimale une fois atteinte la valeur maximale pour une séquence croissante et contrairement pour une séquence décroissante. Par défaut c'est NO CYCLE

NEXTVAL : pour augmenter la séquence du PAS et obtenir une valeur.

CURRVAL : pour obtenir la valeur courante de la séquence**. Le currval ne s'exécute qu'après un Nextval**

Attention 

- ✓ **Ne jamais utiliser une séquence avec CYCLE pour générer des valeurs de clé primaire**
- ✓ Lorsqu'un enregistrement est supprimé de la table, le numéro de séquence correspondant n'est pas récupéré.
- ✓ Lors de l'insertion d'un enregistrement, s'il y a violation d'une contrainte d'intégrité (l'enregistrement n'a pas été inséré) le numéro de séquence est perdu.

Exemple :

CEREATE SEQUENCE sequence1 START WITH 20 INCREMET BY 1;

SELECT sequence1.NEXTVAL FROM dual; donne comme resulta 20;

SELECT sequence1.CURRVAL FROM dual; donne comme resulta 20;

Les synonymes :

Les synonymes est une autre désignation pour les objets (vue, tables, séquence..) de la base de données. Il est utilisé pour faciliter l'accès à un objet. Par exemple au lieu d'accéder à une table via un chemin d'accès (saliha.employes) on utilise un synonyme.

Un synonyme peut être public, ou privé. Par défaut un synonyme est privé.

Syntaxe

```
CREATE [PUBLIC] SYNONYM <nom_du_sysnonyme> FOR <nom_objet>
```

La création d'un synonyme PUBLIC ne peut se faire que **par l'administrateur.**

Exemple

CREATE PUBLIC SYNONYM syemp FOR scott.emp;

Ainsi au lieu de faire :

SELECT * FROM scott.emp on peut faire SELECT * FROM syemp;

```
DROP PUBLIC SYNONYM <nom_synonyme>
```

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

Syntaxe

```
CREATE SYNONYM <nom_du_sysnonyme> FOR <nom_objet>
```

Et la destruction d'un synonyme non public se fait par :

```
DROP SYNONYM <nom_synonyme>
```

Pour modifier un synonyme, il faut le supprimer puis le recréer.

La table DUAL

La table DUAL est une table avec une seule colonne et une seule ligne. Elle est accessible par tous les usagers en lecture seule. Elle permet d'être une source de données lorsque la source n'est pas connue. On pourrait y extraire la prochaine valeur d'une séquence, la date du jour, le résultat d'une fonction, un nombre quelconque etc….

Nous avons déjà vu cette table en allant chercher la date du jour comme suit :
SELECT SYSDATE FROM DUAL;

Exemples

SELECT SYSDATE FROM DUAL -→ donne la date du jour.

SELECT sequence1.NEXTVAL FROM dual;→ donne la prochaine valeur de la séquence

SELECT CEIL (dbms_random.value(1,10)) FROM dual; → un nombre aléatoire entre 1 et 10

SELECT * FROM joueurs WHERE numjoueur =(
SELECT CEIL(dbms_random.value(1,15))FROM DUAL;
→

Ramène un joueur de la BD de manière aléatoire. Ceci est valable que si le numéro du joueur est entre 1 et 15.
Le numéro des joueurs est séquentiel, sinon ça ne marche pas.

update joueurs set salaire = salaire+ (select ceil(dbms_random.value(100,300))from dual);→ met à jours les salaires des joueurs en ajoutant un nombre aléatoire entre 100 et 300.

Sources

https://docs.oracle.com/cd/E11882_01/appdev.112/e10767.pdf

https://docs.oracle.com/cd/E11882_01/index.htm

https://docs.oracle.com/cd/E20434_01/doc/win.112/e23174/toc.htm

https://docs.oracle.com/en/database/index.html