Département d'informatique

420 –KED-LG, conception de bases de données

SQL d'ORACLE

Table des matières

Présentation du SGBD Oracle.

Architecture :

Par définition un système de gestion des bases de données est un ensemble de programmes destinés à gérer les fichiers

Oracle est constitué essentiellement des couches suivantes :

1. Le noyau : dont le rôle est de
    a. Optimisation dans l'exécution des requêtes
    b. Gestion des accélérateurs (index et Clusters)
    c. Stockage des données
    d. Gestion de l'intégrité des données
    e. Gestion des connexions à la base de données
    f. 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 puis que le PL/SQL est une extension procédurale du SQL.

Les commandes SQL

Il existe principalement trois types de commandes SQL

1. les commandes de définition des données (LDD, langage de définition des données)

   Ces commandes permettent de créer ou de modifier un objet de la base de données : CREATE, ALTER, DROP, MODIFY,

2. les commandes de manipulation des données (LMD,Langage de Manipulation des Données)

   Ces commandes permettent de faire la mise à jour ou la consultation des données dans une table de la base de données.

   SELECT, UPDATE, INSERT, DELETE

3. les commandes de control de données (LCD) :

   Exemple : GRANT et REVOKE

4. Les commandes de transaction (TCL), qui permettent de gérer les modifications apportées aux données de la base de données.

   Exemples : COMMIT, et ROLLBACK

Bref historique

SQL pour Structured Query Language est un langage de définition de données (DDL, Data Definition Language), de manipulation de données (LMD, Data Manipulation Language) et de contrôle de données (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*.

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 (Champs ou attributs) et de lignes (enregistrements ou occurrences)

Les vues : Une vue est une table virtuelle issue du modèle externe contenant une partie d'une ou plusieurs tables. 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 : générateur de numéro unique

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

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

Les déclencheurs : procédure déclenchée avant toute modification de la base de données (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.

Définition de données

Création de tables : La commande <CREATE TABLE>

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.

Syntaxe simplifiée :

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

Syntaxe générale

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

Définitions et exemples :

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 : 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 plus haut :

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

```
CREATE TABLE Etudiants
(
numad NUMBER(10) CONSTRAINT pk1 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 pk1 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 définirait 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 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 doivent être à la fin de la définition des colonnes (voir exemple 2).

Exemple 3

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

La contrainte de FOREIGN KEY: cette contrainte indique que la valeur de l'attribut correspond à une valeur d'une clé primaire de la table spécifiée.
La clé primaire de l'autre table doit être obligatoirement crée pour que cette contrainte soit acceptée. La clé primaire de l'autre table et l'attribut défini comme clé étrangère doivent être de même type et de même longueur

On peut également préciser l'option ON DELETE CASCADE qui indique que les enregistrements (occurrences) soient détruits lorsque l'enregistrement correspondant à la clé primaire de la table référencée est supprimé. Si cette option n'est pas précisée alors aucun enregistrement ne sera supprimé de la table qui contient la clé primaire

Exemple

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

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

La contrainte CHECK :
Indique les valeurs permises qui peuvent être saisies pour la colonne (champ ou 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.

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 saisie pour les colonnes (champs ou attributs) doivent être unique Ce qui veut dire pas de Doublons.

Exemple 1

```
CREATE TABLE personne (num NUMBER CONSTRAINT pk5 PRIMARY KEY,
nom VARCHAR2 (15), prenom VARCHAR2 (15),
ville VARCHAR2 (20) DEFAULT 'Montréal'
CONSTRAINT ck1 CHECK
(ville IN ('Montréal','Laval','Saint-Jérôme')));
```

Exemple 2

Exemple 2 la contrainte est sur la table

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

Remarquez :

1. 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.

2. Mis à part les contraintes d'UNIQUE, DEFAULT et NOT NULL, moi, je vous oblige à donner un nom significatif à toutes vos contraintes.
3. La contrainte CHECK peut être définie sur la table ou sur la colonne.

Types de données manipulés :

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 gigaoctet : Exemple Image BLOB
CLOB	A character large object : Chaîne de caractère de longueur maximale allant jusqu'à 4 gigaoctet :

Pour plus d'information consulter :

http://download.oracle.com/docs/cd/E11882_01/server.112/e10592/sql_elements001.htm#i45441

Modification de la définition d'une table :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 à la table avec ses contraintes
- ✓ 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
- ✓ Changer le type de données d'une une colonne existante
- ✓ Spécification d'autres contraintes pour une colonne existante
- ✓ Activer ou désactiver une contrainte
- ✓ Détruire une contrainte.

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.
Exemples :
Voici la commande CREATE initiale pour la table Employes

```
CREATE TABLE Employes(NumEmp number, nom varchar2(15), prenom
varchar2(20));
```

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

Permet d'ajouter la colonne Salaire à la table Employes

ALTER TABLE Employes ADD CONSTRAINT emppk PRIMARY KEY (NumEmp);

Permet d'ajouter une contrainte de clé primaire sur la colonne Numemp de la table Employes

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

```
ALTER TABLE Employes MODIFY (nom NOT NULL);
```

L'option ENABLE /DISABLE

Cette option sert à activer ou désactiver une contrainte.

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

L'option DROP

Cette option sert à supprimer une contrainte sur une table déjà existante. Lorsque

```
ALTER TABLE Employes DROP Primary Key;
```

Ou

```
ALTER TABLE Employes DROP CONSTRAINT emppk;
```

```
ALTER TABLE Employes  DROP COLUMN nom;
```

```
ALTER TABLE Employes  RENAME COLUMN Salaire  TO SalaireEmp;
```

Supprimer une table : la commande <DROP TABLE>

La commande DROP permet de supprimer un objet de la b ase de données. (Table, indexe, synonyme..)

DROP TABLE personne;

Renommer une table : 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;

Création d'un index : La commande <CREATE INDEX>

Un index est un objet permettant d'accélérer l'accès aux données. La création d'u n index sur une clé primaire se fait automatiquement par le système. Pour créer un index sur un champ autre que la clé primaire on utilise la commande CREATE INDEX.

```
CREATE [UNIQUE] INDEX Nom_Index ON Nom_Table(nomClonne) ;
```

Exemple
CREATE INDEX index1 ON ETUDIANTS(nom), crée un index sur la colonne nom de la table ETUDIANTS.

CREATE INDEX index2 ON ETUDIANTS(nom, prenom) crée un index sur les colonnes nom et prénom de la table ETUDIANTS.

Remarque :

- Un index ne peut être créé que sur une table (pas une vue).
- Les index UNIQUE applique les contraintes d'unicité
- Ne jamais créer trop d'index
- Créer des index sur une colonne ayant une petite plage de valeurs inutiles
- Un index se créer sur maximum 16 colonnes.

Manipulation de données

La commande INSERT INTO

Syntaxe 1 : la syntaxe qui suit permet d'insérer des valeurs pour toute une rangée (une ligne ou un enregistrement) dans une table

```
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 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.**

- Lors de l'insertion des donné 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 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 des séquences pour l'insertion automatique d'un numéro séquentiel pour une colonne
- Après les insertions, il est recommandée d'exécuter la commande COMMIT (à voir plus loin)

Création de séquence pour Insertion :

```
CREATE SEQUENCE <Nom_de_sequence> INCREMENT BY
<intervalle>
START WITH <value_de_depart>
MAXVALUE <valeur_maximale>
MINVALUE <valeur_minimale>
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 WITN 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 nombre 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 pour une séquence négative 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.

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

CURRVAL : pour obtenir la valeur courante de la séquence.

ATTETION !!!:

- ✓ 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.

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. (Enregistrement)
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 s valeur définie 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)

Utilisation de la clause WHERE

La cluse WHERE permet de fixer la condition sur les données de mise à jour (UPDATE). Cette clause est utilisée également avec DELETE et SELECT.

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

Liste des opérateurs utilisés dans la condition :

Opérateurs	Signification	Exemple
=	Égalité	UPDATE employesinfo SET salaire = salaire +(salaire*0.5) WHERE nom ='Fafar';
<> ou != ou ^=	Inégalité ou différent	
>	Plus grand	
<	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.	UPDATE employesinfo SET salaire = salaire +(salaire*0.5) WHERE nom like 'Faf%';
NOT LIKE	Si la valeur n'est pas comme une chaine de caractères. Le % est utilisé pour débuter ou	

	compléter la chaine de caractère.	
IN	Égal à une valeur dans une liste	UPDATE employesinfo SET salaire = salaire +(salaire*0.5) WHERE nom IN ('Fafar', 'Simpson','Lebeau');
NOT IN	N'est pas égal à une valeur dans une liste	
IS NULL	Si la valeur retournée est NULL	UPDATE employesinfo SET salaire =10000.99 WHERE salaire is NULL;
IS NOT NULL	Si la valeur retournée est n'est pas NULL	
BETWEEN x AND Y	Si la valeur est comprise entre x et y	UPDATE employesinfo set salaire = salaire +(salaire*0.5) 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	
ALL	Si toutes les valeurs répondent à la comparaison	
EXISTS	Si la colonne existe	
NOT EXISTS	Si la colonne n'existe pas	

```
UPDATE employes SET NOMEMP ='BIDON', SALAIRE = 44000 WHERE
NUMEMP =10;
```

La commande DELETE

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

Pour des raisons de sécurité, exécuter cette commande jute après la commande SELECT afin d'être certains des enregistrements que l'on va supprimer
Lors de la suppression des données, les contraintes d'intégrité doivent être répétées. .
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 supprimer.
Si l'option ON DELETE CASCADE n'est pas définie alors pour supprimer une clé primaire référencée il faut opter pour une des solutions suivante :

- Supprimer d'abord les rangées de la table enfants qui réfèrent 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

```
DELETE FROM <nom_de _table>;
```

Cette syntaxe permet de détruire TOUS les enregistrements d'une table

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

Cette syntaxe permet de détruire les enregistrements d'une table répondant à la condition spécifiée dans la clause WHERE
Exemples :

```
DELETE FROM employesinfo WHERE NUMEMP =30;
DELETE FROM employesinfo WHERE NOM IN ('Blues','Simpson');
```

Avant toute opération **COMMIT**, faire un SELECT afin de vérifier que nous n'avons pas fait des suppressions par erreur. Utiliser un **ROLLBACK** dans le cas d'une suppression par erreur.

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 :

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

- ✓ 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.

Exemples

```
SELECT * FROM employes;
```

```
SELECT empno, ename, job
FROM EMPLOYES;
```

```
SELECT empno, ename, job
FROM EMPLOYES ORDER BY ename, job;
```

```
SELECT empno, ename, job
FROM EMPLOYES ORDER BY ename DESC, job;

SELECT * FROM EMPLOYES ORDER BY 1,2;
```
Le tri se fait selon la colonne 1, puis la colonne 2

```
SELECT empno, ename, job
FROM EMPLOYES
WHERE SAL >2500
ORDER BY ename ;

SELECT * FROM  etudiants
WHERE nom LIKE 'P%';
```
Ramène tous les étudiants dont le nom commence par P.

Quelques fonctions SQL:

Les fonctions agissant sur les groupes

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.
Exemple

```
SELECT MAX (NOTE) FROM RESULTATS WHERE CODE_COURS ='KED';
```

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
Exemple

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

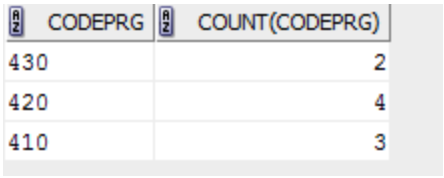
Les fonctions VARIANCE et STDDEV: Pour calculer la variance et l'écart type sur les valeurs d'une colonne

La fonction COUNT: cette fonction permet de compter le nombre de lignes (rangées) qui répondent à un critère. La clause GROUP BY peut être utilisée ou non

Si une colonne est présente dans la clause SELECT alors elle doit être présente dans la clause GROUP BY
La clause GROUP BY: cette clause permet d'indiquer au système de regrouper des enregistrements selon des valeurs distincts qui existent pour les colonne spécifiées. La clause HAVING permet de mieux cibler les enregistrements spécifiés.

Exemples

```
SELECT CODEPRG, COUNT(CODEPRG)
FROM ETUDIANTS
GROUP BY CODEPRG;
```



CODEPRG	COUNT(CODEPRG)
430	2
420	4
410	3

```
SELECT CODEPRG, COUNT(CODEPRG)
FROM ETUDIANTS
GROUP BY CODEPRG
HAVING CODEPRG ='420';
```

Cette requête calcule le nombre total d'étudiants.

```
SELECT COUNT(*)
FROM ETUDIANTS;
```

Les requêtes avancées (jointures, sous requêtes et gestion des informations hiérarchisée)

Requêtes avec jointure

Une jointure est une opération relationnelle qui sert à chercher des lignes ou des enregistrements à partir de deux ou plusieurs tables disposant d'un ensemble de valeur communes, en général les clés primaires.

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 NOM, PRENOM, NOMPROG
FROM ETUDIANTS,PROGRAMME;
```

NOM	PRENOM	NOMPROG
PATOCHE	ALIAN	INFORMATIQUE
PATOCHE	ALIAN	ADMINISTRATION
PATOCHE	ALIAN	ELECTRONIQUE
FAFAR	CHANTALE	INFORMATIQUE
FAFAR	CHANTALE	ADMINISTRATION
FAFAR	CHANTALE	ELECTRONIQUE
MARTIN	RACHA	INFORMATIQUE
MARTIN	RACHA	ADMINISTRATION
MARTIN	RACHA	ELECTRONIQUE

…. (il y a une suite à la sortie de la requête)

Jointure simple:

Une jointure simple consiste est un 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 champs (cas de deux tables) qui sont généralement les clés primaires.

Exemple

```
SELECT NOM, PRENOM, NOMPROG
FROM ETUDIANTS E INNER JOIN PROGRAMME P
ON E.CODEPRG =P.CODEPRG;
```

NOM	PRENOM	NOMPROG
PATOCHE	ALIAN	INFORMATIQUE
FAFAR	CHANTALE	INFORMATIQUE
MARTIN	RACHA	ADMINISTRATION
PLOUFFE	STEVEN	ELECTRONIQUE
ALLARD	MATHIEU	ADMINISTRATION
VIENS	NATHALIE	ADMINISTRATION
DANIS	SAMUEL	ELECTRONIQUE
FAVRE	NICOLAS	INFORMATIQUE
JACOB	YANICK	INFORMATIQUE

Autres exemples :

```
select ename, job, sal, dname
from (syemp inner join sydept on syemp.deptno = sydept.deptno);
```

```
select ename, job, sal, loc
from (syemp inner join sydept on syemp.deptno = sydept.deptno)
where sydept.deptno = 10;
```

```
SELECT nom,prenom,  description, note
FROM  ((etudiant E  INNER JOIN  NOTE  ON E.numad = R.numad)
INNER JOIN cours C ON C.code_cours = R.code_cours);
```

Jointure externe

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

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

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

```
SELECT NOM, PRENOM, NOMPROG
FROM ETUDIANTS E LEFT OUTER JOIN PROGRAMME P ON
E.CODEPRG=P.CODEPRG;
```

RESPONSABLE	EMPLOYE
alibaba	Martin
alibaba	Patoche
alibaba	Cristophe
Patoche	FAFAR
Cristophe	ROY
Patoche	SIMPSON

IMPORTANT:

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

Lorsqu'un champ 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.
Exemple

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.

Requêtes imbriquées (sous requêtes)

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.

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.

Cette requête ramène les numéro d'admission (NUMAD)des étudiants dont la note en KED est plus petite que celle de l'étudiant dont le numéro d'admission est 100 pour le même cours (CODE_COURS)

```
SELECT NUMAD FROM RESULTATS
WHERE  CODE_COURS='KED' AND  NOTE <
   (SELECT NOTE FROM RESULTATS
   WHERE NUMAD=100 AND CODE_COURS='KED');
```

Quelle est-la requête qui ramène les noms des étudiants et non leur numéro ?

Réponse

```
SELECT E.NUMAD, E.NOM,R.NOTE FROM RESULTATS R, ETUDIANTS E
WHERE  E.NUMAD= R.NUMAD
AND CODE_COURS='KED'
AND  NOTE <
   (SELECT NOTE FROM RESULTATS
   WHERE NUMAD=100 AND CODE_COURS='KED');
```

- ✓ On utilise l'operateur IN lorsque la sous requête retourne plus qu'une rangée (plus qu'un enregistrement)

Exemple1 : la requête suivante ramène tous les étudiants qui n'ont pas de notes

```
SELECT NUMAD,NOM, PRENOM
FROM ETUDIANTS
WHERE NUMAD NOT IN
     (SELECT NUMAD FROM RESULTATS);
```

- ✓ On utilise l'operateur ANY pour que la comparaison se fasse pour toutes les valeurs retournée. Le résultat est vrai si au moins une des valeurs répond à la comparaison
- ✓ On utilise l'operateur ALL pour que la comparaison se fasse pour toutes les valeurs retournée. Le résultat est vrai si toutes les valeurs répondent à la comparaison
- ✓ On utilise l'opérateur EXISTS est similaire à l'opérateur IN pour déterminer l'existence ou non de lignes. EXISTS teste si la requête intérieure retourne des valeurs.

Exemple

```
SELECT  * FROM clients WHERE EXISTS (SELECT numclient FROM commande);
```

Pour la table résultat suivante, les sorties des requêtes avec ANY et ALL sont les suivant :

CODE_COURS	NUMAD	NOTE
KED	100	78
KED	101	65
KEG	100	88
KEG	101	78
KED	109	80
KED	107	78
KEG	107	55
KED	102	90

```
SELECT NUMAD FROM RESULTATS
WHERE NOTE >ALL
(SELECT NOTE FROM RESULTATS WHERE  CODE_COURS ='KEG') ;
```

Donne le résultat

NUMAD
102

```
SELECT NUMAD FROM RESULTATS
WHERE NOTE >ANY
(SELECT NOTE FROM RESULTATS WHERE  CODE_COURS ='KEG') ;
```

NUMAD
102
100
109
100
101
107
101

Utilisation d'une sous-requête avec l'instruction 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.

```
INSERT INTO COURS_DU_SOIR
(SELECT * FROM COURS
WHERE CODE_COURS ='KEG');
```

Ou pour insérer toutes les lignes de la table COURS dans COURS_DU_SOIR

```
INSERT INTO COURS_DU_SOIR
(SELECT * FROM COURS);
```

Utilisation d'une sous-requête pour la création de table

Ce type de sous requête permet de créer une table à partir d'une autre table. La nouvelle table contient les valeurs de la sous-requête

EXEMPLE

La requête suivante permet de créer la table des notes du cours KED

```
CREATE TABLE NOTEKED (NUMADMISSION,NOTEKED)
AS SELECT NUMAD,NOTE FROM RESULTATS WHERE CODE_COURS='KED';
```

OU

```
CREATE TABLE NOTEKED (NUMADMISSION PRIMARY KEY ,NOTEKED)
AS SELECT NUMAD,NOTE FROM RESULTATS WHERE CODE_COURS='KED';
```

QUESTION

Quelle-est la requête qui permet DE créer une table NOTESKED qui va avoir comme attributs NOMETUDIANT, PRENOMETUDIANT,TITRECOURS,NOTEKED

REPONSE

```
CREATE TABLE NOTESKED (NOMETUDIANT,
PRENOMETUDIANT,TITRECOURS,NOTEKED)AS
SELECT NOM, PRENOM,TITRE,NOTE
FROM ETUDIANTS E INNER JOIN RESULTAT R ON E.NUMAD=R.NUMAD
INNER JOIN COURS CR ON CR.CODE_COURS=R.CODE_COURS WHERE
R.CODE_COURS='KED' ;
```

Remarque : Parfois, il est utile de créer une table ayants les mêmes attributs que la table principale mais sans avoir les valeurs de celle-ci. Dans ce cas il suffit de mettre une clause WHERE impossible dans la sélection.

Utilisation d'une sous-requête avec la clause SET

Dans ce cas le résultat retourné par la sous-requête doit contenir une seule rangée.

```
UPDATE RESULTATS SET NOTE =
    (SELECT AVG(NOTE) FROM RESULTATS WHERE CODE_COURS ='KEG')
WHERE NOTE <60 AND CODE_COURS ='KEG';
```

Requêtes corrélées

Question : comment avoir tous les étudiants qui sont dans le même programme que l'étudiant dont le nom est PATOCHE

```
SELECT * FROM ETUDIANTS
WHERE CODEPRG = (SELECT CODEPRG FROM ETUDIANTS WHERE
NOM='PATOCHE');
```

NUMAD	NOM	PRENOM	CODEPRG
100	PATOCHE	ALIAN	420
101	FAFAR	CHANTALE	420
109	FAVRE	NICOLAS	420
107	JACOB	YANICK	420

Question

Comment peut-on avoir les étudiants (nom et prénoms) qui ont la même note que l'étudiant numéro 100 et dans le cours KED

Réponse

```
SELECT *  FROM ETUDIANTS
WHERE NUMAD IN
        (SELECT NUMAD FROM RESULTATS WHERE CODE_COURS ='KED' AND
NOTE =
                (SELECT NOTE FROM RESULTATS WHERE CODE_COURS='KED' AND
NUMAD =100));
```

NUMAD	NOM	PRENOM	CODEPRG
100	PATOCHE	ALIAN	420
107	JACOB	YANICK	420

Requêtes SELECT avec les opérateurs d'ensembles

L'opérateur INTERSECT : cet opérateur permet de ramener l'intersection des données entre deux tables. Les deux commandes SELECT doivent avoir le même nombre de champs, et des champs de même type et dans le même ordre.
Syntaxe:

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

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

Exemple

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

Permet de ramener tous les étudiants qui sont en même temps des enseignants.

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

Syntaxe

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

SELECT NOM, PRENOM FROM ETUDIANTS
UNION
SELECT NOM, PRENOM FROM ENSEIGNANTS
ORDER BY 1
Permet de ramener tous les étudiants et tous les enseignants. Les enseignants qui sont en même temps des étudiants sont ramenés une seule fois.

L'opérateur MINUS

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

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

Syntaxe

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

Gestion de l'information hiérarchisée

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).

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

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

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

```
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

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

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, on parcourt l'arbre vers le bas (on extrait les subordonnés), ou à droite, 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
La pseudo colonne LEVEL est ajoutée pour montrer le niveau de hiérarchie entre les enregistrements.

Les fonctions SQL

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

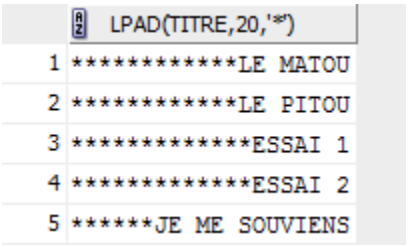
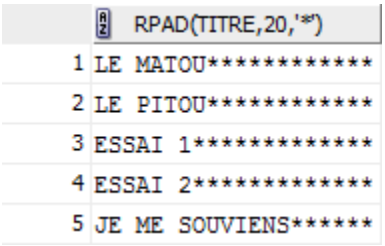
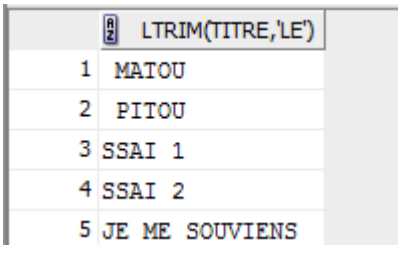
La table DUAL est une table avec une seule colonne et une seule ligne. Elle est accessible par tous les usagers en lecture seule. Elle permet d'être une source de données lorsque la source n'est pas connue. On pourrait y extraire la prochaine valeur d'une série, la date du jour ..

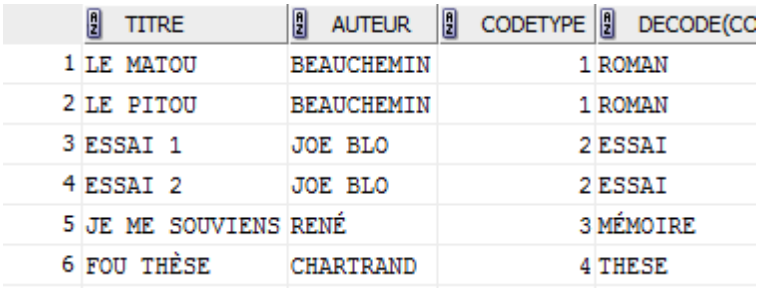
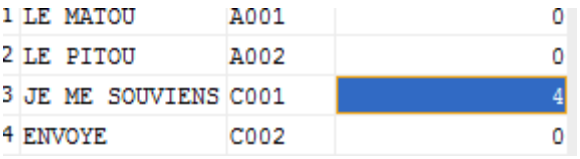
Oracle offre plusieurs fonctions pour manipuler des dates, des chaînes de caractères et des nombres. Ces fonctions seront utiles dans les situations suivantes :

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

Manipulation des chaînes de caractères :

Fonctions 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;
|| Colonne 1 || colonne 2	Concatenation	SELECT NOM || PRENOM FROM ETUDIANTS;
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	

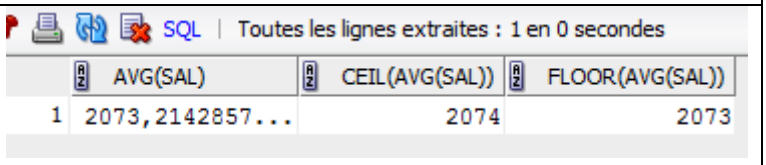
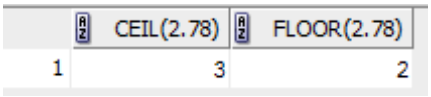
<table>
<tr><td></td><td>LTRIM à droit.</td><td></td></tr>
<tr><td>DECODE
DECODE (colonne, code1, chaine1,
Code2, chaine2..)</td><td>Permet de décoder des codes au moment de la sélection.</td><td>SELECT TITRE, AUTEUR, CODETYPE,
DECODE (CODETYPE,1,'ROMAN',
2,'ESSAI',3,'MÉMOIRE', 4, 'THESE')
FROM LIVRES;
 
</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%';
 
</td></tr>
</table>

Les fonctions sur les dates

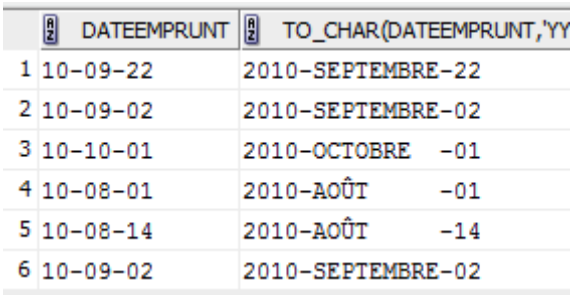
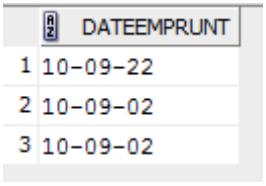
Fonctions et syntaxes	Rôles	Exemples
SYSDATE	Obtenir la date du jour	SELECT SYSDATE FROM DUAL;
ADD_MONTHS ADD_MONTHS (date, nb)	Ajoute un nombre nb de mois à une date. Si nb est négatif, on enlève des mois	SELECT ADD_MONTHS (DATEEMPRUNT,2) FROM EMPRUNT
NEXT_DAY	Avance la date jusqu'au jour spécifié	SELECT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

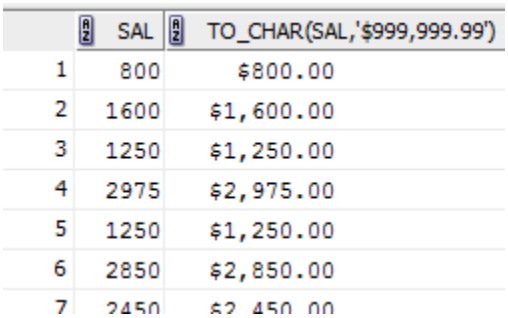
TRUNC (nombre, m)	Supprime la partie fractionnelle de son argument numérique	SELECT TRUNC (2.78,1) FROM DUAL; A pour résultat 2.7
CEIL	Retourne l'entier immédiatement supérieur à son argument	
FLOOR	Retourne l'entier immédiatement inférieur à son argument	SELECT CEIL (2.78), FLOOR(2.78) FROM DUAL; 
POWER 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	

Fonctions de conversion

Fonctions et syntaxes	Rôles	Exemples
TO_DATE (chaine, modelededdate)	Convertie une chaine de caractère en une date selon le modèle défini. Le modèle de date doit être dans un format valide.	INSERT INTO EMPRUNT VALUES ('EX1C31',12, TO_DATE ('10-09-02', 'YY-MM-DD')); SELECT * FROM Commandes WHERE SYSDATE - TO_DATE(datecommande) > 10; Ici date commande a été creée avec le type varchar2
TO_CHAR avec les dates TO_CHAR (valeur_date, formatDate)	Converti une date pour affiche la date selon un format prédéfini. Les fonctions sur les chaînes de caractères peuvent être utilisées	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%'; 
TO_CHAR avec les nombres TO_CHAR (nombre, format)	Utilisée pour afficher des valeurs numérique sous un autre format..les formats numériques sont : 9 affiche une valeur	SELECT SAL, TO_CHAR (SAL, '$999,999.99') FROM SYEMP;

	DATEEMPRUNT	TO_CHAR(DATEEMPRUNT,'YY
1	10-09-22	2010-SEPTEMBRE-22
2	10-09-02	2010-SEPTEMBRE-02
3	10-10-01	2010-OCTOBRE -01
4	10-08-01	2010-AOÛT -01
5	10-08-14	2010-AOÛT -14
6	10-09-02	2010-SEPTEMBRE-02

	DATEEMPRUNT
1	10-09-22
2	10-09-02
3	10-09-02

	numérique sans les zéro non significatif. Autant de 9 que de chiffres. 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)	
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-	Les trois derniers

	MM-DD')	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

Les vues

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 vue 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 plus part 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.

Syntaxe

```
CREATE [OR REPLACE ][FORCE] VIEW <nom_de_la_vue> AS <sou_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.

Contraintes d'utilisation

Vous ne pouvez pas :

- Insérer dans une table à partir d'une vue, si la table contient des champs 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 définie avec :
    - Une jointure
    - Une opération d'ensemble
    - Une clause GROUP BY, CONNECT BY, DISTINCT de l'ordre 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.

Exemple s

```
CREATE VIEW Eleves as
select numad, nom, prenom
from etudiants;

insert into Eleves values (100,'nom1','prn1');

CREATE VIEW ElevesDeMonteral as
select numad, nom, prenom,ville
from etudiant1 where Ville ='MONTREAL'
WITH CHECK OPTION;

insert into eleves values (21,'nom3','prn3','dallas');
```

L'instruction INSERT va renvoyer une erreur à cause du WITH CHECK OPTION

Détruire une VUE :

DROP VIEW nom_de_vue permet de supprimer la vue

RENAME ancien_nom TO nouveau_NOM renomme une vue.

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.

Les privilèges :

La commande GRANT permet d'attribuer des droits à des utilisateurs sur vos objets.
La commande REVOKE permet de supprimer des privilèges.

La commande GRANT

Syntaxe

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

Les privilèges sont : SELECT, INSERT, UPDATE, DELETE, ALTER, INDEX, REFERENCES ou (tout ce la et donc ALL)

- On peut attribuer plusieurs privilèges, mais il faut les séparer par une virgule
- On peut préciser plusieurs colonnes, en autant qu'elles soient séparées par une virgule
- WITH GRANT OPTION, permet à l'usager à qui on vient d'attribuer des droits d'attribuer les mêmes droits à d'autres usagers sur le même objet.
- PUBLIC permet d'attribuer les droits à TOUS.

Exemple :

GRANT SELECT,INSERT ON etudiant TO mathieu

GRANT ALL on employe TO PUBLIC

GRANT SELECT ON department TO martin WITH GRANT OPTION.

La commande REVOKE

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

EXEMPLE

REVOKE INSERT ON etudiant FROM Mathieu

REVOKE ALL ON employe TO PUBLIC.

Les rôles :

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

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

Création d'un ROLE

```
CREATE ROLE <nom_du_role>
```

Exemple

CREATE ROLE role1

Attribuer des privilèges à un ROLE

```
GRANT <privileges> ON <nom_objet>TO <nom_role>
```

Exemple

GRANT SELECT, UPDATE, INSERT ON etudiants TO role1

Attribuer un ROLE à un utilisateur.

```
GTRANT <nom_role> TO <nom_usager>
```

GRANT role1 TO scott, martin, mathieu.

Détruire un ROLE par DROP <nom_du role>

Les transactions

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 derniers COMMIT) sur la base de données.

ROLLBACK [TO nom_save_point]: permet d'annuler une transaction (un COMMIT).

SAVEPOINT : permet de fixer des points de sauvegarde.

Syntaxe : SAVEPOINT <nom_savepoint>.

Sources :

http://download.oracle.com/docs/cd/E11882_01/server.112/e10592/statements_6015.htm#i2067093

http://www.oracle.com/pls/db112/search

http://download.oracle.com/docs/cd/E11882_01/server.112/e10713.pdf (chapitre 7)

SQL (introduction au SQL et SQL*Plus avec Oracle) manuel SQL de l'enseignant DENIS BRUNET

SQL, manuel pour le cours 420-551-93 de l'enseignante Saliha Yacoub