Administration des bases de données

Gestion de données

Mariam BENLLARCH

Prérequis et objectifs

- *Prérequis*
  - Modèle relationnel (structure, contraintes, SQL)

- *Objectifs*
  - Gérer les données d'une base de données Oracle:
    - Définition de données
    - Manipulation de données

Partie 1:

Définition de données

Définition de données

- **Description des instructions SQL qui constituent l'aspect LDD (langage de définition des données) de SQL. A cet effet, nous verrons notamment comment déclarer une table, ses éventuels contraintes et index.**

  - ✓ *Une table est créée en SQL par l'instruction CREATE TABLE, modifiée au niveau de sa structure par l'instruction ALTER TABLE et supprimée par la commande DROP TABLE.*

Création d'une table (CREATE TABLE)

- Pour pouvoir créer une table dans votre schéma, il faut avoir le privilège **CREATE TABLE**.
- Si vous avez le privilège **CREATE ANY TABLE**, vous pouvez créer des tables dans tout schéma.
- La syntaxe SQL simplifiée est la suivante :

CREATE TABLE [schéma.] nomTable

(**colonne1** type1 [DEFAULT valeur1] [NOT NULL]

[, **colonne2** type2 [DEFAULT valeur2] [NOT NULL]]

[**CONSTRAINT** nomContrainte1 typeContrainte1]…) ;

Création d'une table (CREATE TABLE)

- **NomTable :**
  - Peut comporter :
    - des lettres majuscules ou minuscules (accentuées ou pas),
    - des chiffres et les symboles, par exemple : _, $ et #.

➢ Oracle est insensible à la casse et convertira au niveau du dictionnaire de données les noms de tables et de colonnes en majuscules.

Création d'une table (CREATE TABLE)

- **Colonnes:**
  - **Nom d'une colonne** (mêmes caractéristiques que pour les noms des tables)
  - **Son type** (NUMBER, CHAR, DATE…).
  - **La directive DEFAULT** fixe une valeur par défaut.
  - **La directive NOT NULL** interdit que la valeur de la colonne soit nulle.

➢ NULL représente une valeur qu'on peut considérer comme non disponible, non affectée, inconnue. Elle est différente d'un espace pour un caractère ou d'un zéro pour un nombre.

Création d'une table (CREATE TABLE)

- **Colonnes:**

Type	**Description**	**Commentaires**
CHAR(*n* [BYTE | CHAR])	Chaîne fixe de *n* caractères ou octets.	Taille fixe (complétée par des blancs si nécessaire). Maximum de 2000 octets ou caractères.
VARCHAR2(*n* [BYTE|CHAR])	Chaîne variable de *n* caractères ou octets.	Taille variable. Maximum de 4000 octets ou caractères.
NCHAR(*n*)	Chaîne fixe de *n* caractères	Taille fixe (complétée par des blancs si nécessaire).
NVARCHAR2(*n*)	Chaîne variable de *n* caractères	Taille variable. Mêmes caractéristiques que NCHAR sauf pour la taille maximale qui est ici de 4000 octets.
NUMBER[(t,d)]	Valeur numérique de *t* chiffres dont *d* décimales.	Maximum pour *t* + *d* : 38. Espace maximum utilisé : 21 octets.
DATE	Permet de stocker des moments ponctuels, la précision est composée du siècle,de l'année, du mois, du jour, de l'heure, des minutes et des secondes.	

Création d'une table (CREATE TABLE)

- **Colonnes:**

Type	**Description**
LONG	Texte de longueur variable pouvant stocker jusqu'à 2 gigas. On ne pourra mettre qu'une colonne de type LONG par table.
RAW (taille)	Equivalent à VARCHAR2, mais il permet de stocker des données binaires qui ne sont pas interprétées par Oracle. La taille maximum est de 2000.
LONGRAW	Equivalent à LONG mais pour des données de type binaire non interprétées par Oracle.
CLOB	Permet de stocker un pointeur vers un fichier de données composé de caractère et pouvant contenir jusqu'à 4 gigas.
BLOB	Permet de stocker un pointeur vers un fichier composé de données binaire et pouvant contenir jusqu'à 4 gigas.
BFILE	Permet de stocker les données binaires d'un fichier externe pouvant contenir jusqu'à 4 gigas.

Création d'une table (CREATE TABLE)

- **Colonnes (exemple):**
  - Table : Fournisseur (num_four, nom_four, ville, classe)
  - L'instruction SQL (de création de la table fournisseur):

```
CREATE TABLE Fournisseur
(num_four NUMBER(3),
Nom_four CHAR(15) NOT NULL ,
Ville_four CHAR(15) DEFAULT 'Marrakech',
classe CHAR(2));
```

Création d'une table (CREATE TABLE)

- **Contraintes:**
  - Les contraintes ont pour but de programmer des règles de gestion au niveau des colonnes des tables.
  - Elles peuvent alléger un développement côté client.
  - ex. si on déclare qu'une note doit être comprise entre 0 et 20, les programmes de saisie n'ont plus à tester les valeurs en entrée mais seulement le code retour après connexion à la base ; on déporte les contraintes côté serveur)..

Création d'une table (CREATE TABLE)

- **Contraintes:**
  - Quatre types de contraintes sont possibles :
    - **UNIQUE** (colonne1 [,colonne2]…)
    - **PRIMARY KEY** (colonne1 [,colonne2]…)
    - **FOREIGN KEY** (colonne1 [,colonne2]…)
    - [**ON DELETE** { CASCADE | SET NULL }]
    - **CHECK** (condition)

Création d'une table (CREATE TABLE)

- **Contraintes:**
  - **La contrainte UNIQUE**
    - Impose une valeur distincte au niveau de la table (les valeurs nulles font exception à moins que NOT NULL soit aussi appliquée sur les colonnes).

Création d'une table (CREATE TABLE)

- **Contraintes:**
  - **La contrainte PRIMARY KEY**
    - Déclare la clé primaire de la table.
    - Un index est généré automatiquement sur la ou les colonnes concernées.
    - Les colonnes clés primaires ne peuvent être ni nulles ni identiques.

Création d'une table (CREATE TABLE)

- **Contraintes:**
  - **La contrainte FOREIGN KEY et ON DELETE**
    - Déclare une clé étrangère entre une table enfant (child) et une table père (parent).
    - La directive **ON DELETE** dispose de deux options : **CASCADE** propagera la suppression de tous les enregistrements fils rattachés à l'enregistrement père supprimé,
    - La directive **SET NULL** positionnera seulement leur clé étrangère à NULL

Création d'une table (CREATE TABLE)

- **Contraintes:**
  - **La contrainte CHECK**
    - Impose un domaine de valeurs ou une condition simple ou complexe entre colonnes

      (exemple : CHECK (note BETWEEN 0 AND 20)).

Création d'une table (CREATE TABLE)

- **Contraintes:**
  - **exemple 1:**

```
CREATE TABLE Fournisseur
(num_four NUMBER(3),
nom_four CHAR(15) NOT NULL ,
ville_four CHAR(15) DEFAULT 'Marrakech',
classe_four CHAR(2),
CONSTRAINT pk_Fournisseur PRIMARY KEY(num_four));
```

Création d'une table (CREATE TABLE)

- **Contraintes:**
  - **exemple 2:**

```
CREATE TABLE lpi.Fournisseur
(num_four NUMBER(3),
Nom_four CHAR(15) NOT NULL ,
Ville_four CHAR(15) DEFAULT 'Marrakech',
classe_four CHAR(2),
retard_four NUMBER(2),
CONSTRAINT pk_Fournisseur PRIMARY KEY(num_four),
CONSTRAINT un_nom_four UNIQUE (nom_fou),
CONSTRAINT ck_ retard_four CHECK (retard_four BETWEEN 0 AND 7));
```

Création d’une table (CREATE TABLE)

- **Vérifier la création d’une table:**
  - Pour vérifier que la table à bien été créée, il est possible d'utiliser la commande **DESCRIBE** qui permet d'afficher les colonnes d'une table avec leur type, leur taille et leur contrainte.

Modifier une table

- **Supprimer une table:**
  - **DROP TABLE** table_name [CASCADE CONSTRAINTS];
  - **Exemple**

    SQL> **DROP TABLE** employee

    CASCADE CONSTRAINTS;

- ***CASCADE CONSTRAINTS*** *permet de supprimer de manière automatiquement toutes les contraintes d'intégrité de cette table.*
- *Seul le propriétaire ou un utilisateur ayant le privilège DROP ANY TABLE pourra supprimer une table.*
- *Après la suppression de la table toutes les vues et synonymes qui y faisaient références restent en place et sont invalidés.*

Modifier une table

- **Renommer une table**

  RENAME ancien_nom_table

  TO nouveau_nom_table;

  - **Exemple**

    SQL> **RENAME** employee **TO** emp_table;

➢ *Seul le propriétaire de la table pourra la renommer.*

Modifier une table

- **Ajout des colonnes**

  - La directive **ADD** de l'instruction **ALTER TABLE** permet d'ajouter une nouvelle colonne à une table.

  - **Exemple:**

    ALTER TABLE Employe **ADD** (prenom_emp VARCHAR2(4), ville VARCHAR2(30) DEFAULT 'Marrakech');

Modifier une table

- **Renommer des colonnes**

  - La directive **RENAME COLUMN** de l'instruction **ALTER TABLE** permet de renommer une colonne existante.

  - **Exemple**

    ALTER TABLE Employe **RENAME COLUMN** ville **TO** adresse**;**

Modifier une table

- **Modifier le type des colonnes**
  - La directive **MODIFY** de l'instruction **ALTER TABLE** modifie le type d'une colonne existante.
  - **Exemple:**

ALTER TABLE Employe **MODIFY** ville VARCHAR(10) NOT NULL;

Modifier une table

- **Supprimer des colonnes**

  - La directive **DROP COLUMN** de l'instruction **ALTER TABLE** permet de supprimer une colonne.

  - **Exemple:**

    ALTER TABLE Employe **DROP COLUMN** adresse;

- *Il n'est pas possible de supprimer avec cette instruction :*
  - *des clés primaires référencées par des clés étrangères ;*
  - *des colonnes à partir desquelles un index a été construit ;*

Modifier une table

- **Colonnes UNUSED**

- la directive **SET UNUSED COLUMN** de l'instruction **ALTER TABLE** permet de marquer des colonnes à l'effacement (sans les enlever de la table).

  - **Exemple:**

    ALTER TABLE Employe **SET UNUSED COLUMN** n_dept_attach;

Modifier une table

- **Supprimer une contrainte**

```
ALTER TABLE nom_table
DROP CONSTRAINT nom_contrainte
[CASCADE];
```

- **Exemple:**

```
SQL> ALTER TABLE employee
DROP CONSTRAINT employee_ide_pk
CASCADE;
```

Modifier une table

- **Ajouter des contraintes**

  - La directive ADD CONSTRAINT de l'instruction ALTER TABLE permet d'ajouter une contrainte à une table. La syntaxe générale est la suivante :

ALTER TABLE [*schéma*.]*nomTable*
ADD [**CONSTRAINT** *nomContrainte*] *typeContrainte*;

Modifier une table

- **Vider le contenu d'une table**

- La commande **DELETE** permet de vider une table de toutes ses données tout en conservant sa structure :

DELETE nom_table;

- **Exemple:**

SQL> **DELETE** employee;

- *Seul le propriétaire ou un utilisateur ayant le privilège **DELETE ANY TABLE** ou **DELETE** pourra vider une table.*

Index

- Un index est un objet utilisé pour augmenter la recherche des éléments dans une table à l'aide d'un pointeur.
- Les index sont transparents pour l'utilisateur. Ils sont créés de manière automatique ou manuelle.
- Leur but principal est de réduire le nombre d'entrées/ sorties.

  - *Quand une recherche sur une colonne, qui n'a pas été indexée, est effectuée, la requête fait une recherche sur chaque ligne et vérifie si les critères de recherche lui sont applicables, si la colonne a été indexée, le serveur localise directement les lignes qui correspondent, d'où un gain de temps.*

Index

- **Types d’index**

  Il existe deux types d'index:

  - Les index uniques: Créés automatiquement lors de la définition d'une colonne comme étant une colonne PK ou UNIQUE.
    - L'index prendra le nom de la contrainte à laquelle il est associé.

  - Les index non unique: Créés manuellement. On pourra s'en servir pour indexer les colonnes FK afin d'accélérer la recherche des données.
    - De plus ils pourront contenir des valeurs doubles (colonne non UNIQUE).

Index

- **Créer un index**

```
CREATE INDEX nom_index
ON table(colonne [,colonne] …);
```

- **Exemple**

```
SQL> CREATE INDEX emp_deptno_idx
ON emp (deptno);
```

Index

- **Supprimer un index**

DROP INDEX nom_index ;

- **Exemple**

SQL> **DROP INDEX** emp_deptno_idx;

Séquences

- Une séquence est un objet de la base de données qui génère des numéros uniques qui pourront être insérés dans une colonne (comme par exemple une Primary Key).
    - La SEQUENCE permet d'incrémenter un numéro d'une manière séquentielle.

- Les séquences ne sont pas stockées avec les tables et peuvent donc être utilisées par tous les utilisateurs. Une séquence incrémente ou décrémente un compteur .

- Il est possible de stocker les valeurs dans un cache, ce qui accélèrera l'accès au données de la séquence. Pour créer des séquences il est nécessaire de posséder le privilège **CREATE SEQUENCE**

Séquence

- **Créer une séquence**
  - Voici la syntaxe de la commande qui permet de créer une séquence :

CREATE SEQUENCE nom_sequence
[INCREMENT BY n]
[START WITH n]
[MAXVALUE n | NOMAXVALUE]
[MINVALUE n | NOMINVALUE]
[CYCLE|NOCYCLE]
[CACHE n | NOCACHE];

Séquence

- **Créer une séquence**

INCREMENT BY	Incrémentation de la colonne (valeur par défaut : 1)
START WITH	Départ du compteur (valeur par défaut : 1)
MAXVALUE | NOMAXVALUE	Valeur maximale du compteur ou pas de valeur maximale. La valeur maximale sera de 10^27 pour une incrémentation et de –1 pour une décrémentation.
MINVALUE|NOMINVALUE	Valeur minimale du compteur ou pas de valeur minimale. La valeur minimale sera de 1 pour une séquence ascendante et de – (10^26) pour une séquence descendante.
CYCLE|NOCYCLE	Autorise la séquence à reprendre le compteur à partir de START WITH une fois qu'elle a atteint MAXVALUE ou MINVALUE. A ne pas utiliser dans le cas d'une utilisation pour une colonne PK.
CACHE n | NOCACHE	Spécifie le nombre de valeurs de la séquence que la base de données pré-alloue et conserve en mémoire pour un accès plus rapide. Cette valeur entière peut comporter 28 chiffres ou moins. La valeur minimale de ce paramètre est 2.

Séquence

- **Créer une séquence**
  - Exemple:

```
SQL> CREATE SEQUENCE num_emp
     INCREMENT BY 1
     START WITH 1
     MAXVALUE 9999
     MINVALUE 0
     NOCYCLE
     CACHE 25;
```

Séquence

- **Modifier une séquence**
  - Voici la syntaxe de la commande qui permet de modifier une séquence :

ALTER SEQUENCE nom_sequence
[INCREMENT BY n]
[START WITH n]
[MAXVALUE n | NOMAXVALUE]
[MINVALUE n | NOMINVALUE]
[CYCLE|NOCYCLE]
[CACHE n | NOCACHE];

Séquence

- **Modifier une séquence**
  - Exemple:

```
SQL> ALTER SEQUENCE emp_empno
     INCREMENT BY 2
     MAXVALUE 7777
     CYCLE
     CACHE 30;
```

Séquence

- **Supprimer une séquence**
  - Pour pouvoir supprimer une séquence de son propre schéma ou de n'importe quel schéma, il faut posséder les privilèges DROP SEQUENCE ou DROP ANY SEQUENCE:

    DROP SEQUENCE nom_sequence;

  - Exemple:

    SQL> **DROP SEQUENCE** emp_empno;

Partie 2:

Manipulation de données

Manipulation de données

- Description des instructions SQL qui constituent l'aspect LMD (langage de manipulation des données) de SQL. A cet effet, nous verrons notamment comment insérer des enregistrements, les modifier et les supprimer.

  - *Un enregistrement est créé en SQL par l'instruction **INSERT**, modifiée par l'instruction **UPDATE** et supprimée par l'instruction **DELETE**.*

L'Ordre INSERT

- L'ordre INSERT permet d'ajouter de nouvelles lignes dans une table.
- Cette syntaxe n'insère qu'une seule ligne à la fois.

```
INSERT INTO      table [(column [, column...])]
VALUES           (value [, value...]);
```

Insertion de Nouvelles Lignes

- Insérez une nouvelle ligne en précisant une valeur pour chaque colonne.
- Eventuellement, énumérez les colonnes dans la clause INSERT.
- Indiquez les valeurs dans l'ordre par défaut des colonnes dans la table.
- Placez les valeurs de type caractère et date entre simples quotes.

```
SQL> INSERT INTO    dept (deptno, dname, loc)
     VALUES(50, 'DEVELOPMENT', 'DETROIT');
1 row created.
```

Insertion de Lignes Contenant des Valeurs NULL

- Méthode implicite : ne spécifiez pas la colonne dans la liste.

```
SQL> INSERT INTO     dept (deptno, dname )
     VALUES(60, 'MIS');
1 row created.
```

  - Méthode explicite : spécifiez le mot-clé NULL.

```
SQL> INSERT INTO     dept
    VALUES           (70, 'FINANCE', NULL);
1 row created.
```

Insertion de Valeurs Spéciales

- La fonction SYSDATE renvoie la date et l'heure courantes.

```
SQL> INSERT INTO     emp (empno, ename, job,
                     mgr, hiredate, sal, comm,
                     deptno)
    VALUES(7196, 'GREEN', 'SALESMAN',
                     7782, SYSDATE, 2000, NULL,
                     10);
1 row created.
```

Copie de Lignes d'une Autre Table

- Ecrivez votre ordre INSERT en spécifiant une sous-interrogation.

```
SQL> INSERT INTO managers(id, name, salary, hiredate)
                    SELECT empno, ename, sal, hiredate
                    FROM   emp
                    WHERE  job = 'MANAGER';
3 rows created.
```

- N'utilisez pas la clause VALUES.
- Le nombre de colonnes de la clause INSERT doit correspondre à celui de la sous-interrogation.

L'Ordre UPDATE

- Utilisez l'ordre UPDATE pour modifier des lignes existantes.

- Si nécessaire, vous pouvez modifier plusieurs lignes à la fois.

```
UPDATE      table
SET         column = value [, column = value]
[WHERE      condition];
```

Modification de Lignes d'une Table

- La clause WHERE permet de modifier une ou plusieurs lignes spécifiques.
- Si vous omettez la clause WHERE, toutes les lignes sont modifiées.

```
SQL> UPDATE   emp
     SET      deptno = 20
     WHERE    empno = 61;
1 row updated.
```

```
SQL> UPDATE   employee
     SET      deptno = 20;
14 rows updated.
```

Modification avec une Sous-Interrogation Multi-colonne

- Modifier le poste et le n° de département de l'employé 55 à l'identique de l'employé 61.

```
SQL> UPDATE   emp
     SET      (job, deptno) =
                (SELECT job, deptno
                FROM    emp
               WHERE    empno = 61)
     WHERE    empno = 55;
1 row updated.
```

Modification de Lignes en Fonction d'une Autre Table

- Utilisez des sous-interrogations dans l'ordre UPDATE pour modifier des lignes d'une table à l'aide de valeurs d'une autre table.

```
SQL> UPDATE   employee
     SET      deptno =   (SELECT    deptno
                          FROM      emp
                          WHERE     empno = 54)
     WHERE    job    =   (SELECT    job
                          FROM      emp
                          WHERE     empno = 42);
2 rows updated.
```

L'Ordre DELETE

- Vous pouvez supprimer des lignes d'une table au moyen de l'ordre DELETE.

```
DELETE [FROM]   table
[WHERE          condition];
```

Suppression de Lignes d'une Table

- La clause WHERE permet de supprimer une ou plusieurs lignes spécifiques.

```
SQL> DELETE FROM    department
     WHERE    dname = 'DEVELOPMENT';
1 row deleted.
```

- Si vous omettez la clause WHERE, toutes les lignes sont supprimées.

```
SQL> DELETE FROM department;
4 rows deleted.
```

Suppression de Lignes en Faisant Référence à une Autre Table

- Utilisez des sous-interrogations dans l'ordre DELETE pour supprimer des lignes dont certaines valeurs correspondent à celles d'une autre table.

```
SQL> DELETE FROM     employee
     WHERE    deptno =
                          (SELECT    deptno
                           FROM      dept
                           WHERE     dname ='SALES');

6 rows deleted.
```

Suppression de Lignes : Erreur de Contrainte d'Intégrité

```
SQL> DELETE FROM    dept
     WHERE deptno = 10;
```

```
DELETE FROM dept
            *
ERROR at line 1:
ORA-02292: integrity constraint (USR.EMP_DEPTNO_FK)
violated - child record found
```

- Vous ne pouvez pas supprimer une ligne qui contient une clé primaire utilisée comme clé étrangère dans une autre table.

Transactions de Base de Données

- Une transaction se compose des éléments suivants :
  - Ensemble d'ordres du LMD effectuant une modification cohérente des données.
  - Un ordre du LDD.
  - Un ordre du LCD.

Transactions de Base de Données

Une transaction :

- Commence à l'exécution du premier ordre SQL
- Se termine par l'un des événements suivants :
  - COMMIT ou ROLLBACK
  - Exécution d'un ordre LDD ou LCD (validation automatique)
  - Fin de session utilisateur
  - Panne du système

Etat des Données Avant COMMIT ou ROLLBACK

- Il est possible de restaurer l'état précédent des données.
- L'utilisateur courant peut afficher le résultat des opérations du LMD au moyen de l'ordre SELECT.
- Les résultats des ordres du LMD exécutés par l'utilisateur courant *ne peuvent pas* être affichés par d'autres utilisateurs.
- Les lignes concernées sont *verrouillées*. Aucun autre utilisateur ne peut les modifier.

Etat des Données Après COMMIT

- Les modifications des données dans la base sont définitives.
- L'état précédent des données est irrémédiablement perdu.
- Tous les utilisateurs peuvent voir le résultat des modifications.
- Les lignes verrouillées sont libérées et peuvent de nouveau être manipulées par d'autres utilisateurs.
- Tous les savepoints sont effacés.

Validation de Données

- Effectuez les modifications.

```
SQL> UPDATE   emp
     SET      deptno = 10
     WHERE    empno = 63;
1 row updated.
```

- Validez les modifications.

```
SQL> COMMIT;
Commit complete.
```

Etat des Données Après ROLLBACK

- L'ordre ROLLBACK rejette toutes les modifications de données en instance.
  - Les modifications sont annulées.
  - L'état précédent des données est restauré.
  - Les lignes verrouillées sont libérées.

```
SQL> DELETE FROM    employee;
14 rows deleted.
SQL> ROLLBACK;
Rollback complete.
```

Résumé

Ordre	Description
INSERT	Ajoute une nouvelle ligne dans une table
UPDATE	Modifie des lignes dans une table
DELETE	Supprime des lignes d'une table
COMMIT	Valide toutes les modifications de données en instance
ROLLBACK	Annule toutes les modifications de données en instance

Ordre SELECT Elémentaire

```
SELECT    [DISTINCT] {*, column [alias],...}
FROM      table;
```

- SELECT indique *quelles* colonnes rapporter
- FROM indique dans *quelle* table rechercher

Ordre SELECT Elémentaire

- Sélection de Toutes les Colonnes

```
SQL> SELECT *
  2  FROM   dept;
```

- Sélection d'Une ou Plusieurs Colonnes Spécifiques

```
SQL> SELECT deptno, loc
  2  FROM   dept;
```

Expressions Arithmétiques

- Possibilité de créer des expressions avec des données de type NUMBER et DATE au moyen d'opérateurs arithmétiques

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division

Utilisation des Opérateurs Arithmétiques

```
SQL> SELECT ename, sal, sal+300
     FROM   emp;
```

L’Alias de Colonne

- Renomme un en-tête de colonne
- Est utile dans les calculs
- Suit immédiatement le nom de la colonne ; le mot-clé AS placé entre le nom et l’alias est optionnel
- Doit obligatoirement être inclus entre guillemets s’il contient des espaces, des caractères spéciaux ou si les majuscules/minuscules doivent être différenciées

Utilisation des Alias de Colonnes

```
SQL> SELECT ename AS name, sal salary
     FROM   emp;
```

```
SQL> SELECT ename "Name",
            sal*12 "Annual Salary"
     FROM   emp;
```

L’Opérateur de Concaténation

- Concatène des colonnes ou chaînes de caractères avec d’autres colonnes
- Est représenté par deux barres verticales (||)
- La colonne résultante est une expression caractère

```
SQL> SELECT ename || job AS "Employees"
     FROM emp;
```

Doublons

- Par défaut, le résultat d'une requête affiche toutes les lignes, y compris les doublons.

```
SQL> SELECT deptno
     FROM emp;
```

➡ Elimination des Doublons

Pour éliminer les doublons il faut ajouter le mot-clé DISTINCT à la clause SELECT.

```
SQL> SELECT DISTINCT deptno
     FROM emp;
```

Chaînes de Caractères et Dates

- Les constantes chaînes de caractères et dates doivent être placées entre simples quotes.
- La recherche tient compte des majuscules et minuscules (pour les chaînes de caractère) et du format (pour les dates.)
- Le format de date par défaut est 'DD-MON-YY'.

```
SQL> SELECT ename, job, deptno
     FROM emp
     WHERE ename = 'JAMES';
```

Opérateurs de Comparaison

Opérateur	Signification
=	Egal à
>	Supérieur à
>=	Supérieur ou égal à
<	Inférieur à
<=	Inférieur ou égal à
<>	Différent de

Utilisation des Opérateurs de Comparaison

```
SQL> SELECT ename, sal, comm
     FROM   emp
     WHERE  sal<=comm;
```

```
ENAME           SAL       COMM
---------- --------- ---------
MARTIN         1250 <--> 1400
```

Autres Opérateurs de Comparaison

Opérateur	Signification
BETWEEN ...AND...	Compris entre ... et ... (bornes comprises)
IN (liste)	Correspond à une valeur de la liste
LIKE	Ressemblance partielle de chaînes de caractères
IS NULL	Correspond à une valeur NULL

Utilisation de l'Opérateur BETWEEN

- BETWEEN permet de tester l'appartenance à une fourchette de valeurs.

```
SQL> SELECT ename, sal
     FROM emp
     WHERE sal BETWEEN 1000 AND 1500;
```

Utilisation de l'Opérateur IN

- IN permet de comparer une expression avec une liste de valeurs.

```
SQL> SELECT empno, ename, sal
     FROM emp
     WHERE sal IN (7902, 7566, 7788);
```

Utilisation de l'Opérateur LIKE

LIKE permet de rechercher des chaînes de caractères à l'aide de caractères génériques

Les conditions de recherche peuvent contenir des caractères ou des nombres littéraux.

(%) représente zéro ou plusieurs caractères

```
SQL> SELECT   ename
     FROM emp
     WHERE    ename LIKE 'S%';
```

La Valeur NULL

- NULL représente une valeur non disponible, non affectée, inconnue ou inapplicable.
- La valeur NULL est différente du zéro ou de l'espace.

  - Recherche de valeurs NULL avec l'opérateur IS NULL

```
SQL> SELECT   ename, mgr
     FROM     emp
     WHERE    mgr IS NULL;
```

Opérateurs Logiques

Opérateur	Signification
AND	Retourne TRUE si *les deux* conditions sont VRAIES
OR	Retourne TRUE si *l'une au moins* des conditions est VRAIE
NOT	Ramène la valeur TRUE si la condition qui suit l'opérateur est FAUSSE

Utilisation de l'Opérateur AND

- Avec AND, les deux conditions doivent être VRAIES.

```
SQL> SELECT empno, ename, job, sal
     FROM   emp
     WHERE  sal>=1100
     AND    job='CLERK';
```

Utilisation de l'Opérateur OR

- Avec OR, l'une ou l'autre des deux conditions doit être VRAIE.

```
SQL> SELECT empno, ename, job, sal
     FROM   emp
     WHERE  sal>=1100
     OR     job='CLERK';
```

Utilisation de l'Opérateur NOT

```
SQL> SELECT ename, job
     FROM   emp
     WHERE  job NOT IN ('CLERK','MANAGER','ANALYST');
```

```
... WHERE   sal   NOT   BETWEEN   1000 AND   1500
... WHERE   ename NOT LIKE '%A%'
... WHERE   comm   IS   NOT   NULL
```

Règles de Priorité

Ordre de priorité	Opérateur
1	Tous les opérateurs de comparaison
2	NOT
3	AND
4	OR

- Les parenthèses permettent de modifier les règles de priorité

Règles de Priorité

```
SQL> SELECT ename, job, sal
     FROM   emp
     WHERE  job='SALESMAN'
     OR     job='PRESIDENT'
     AND    sal>1500;
```

- Utilisation de parenthèses pour forcer la priorité.

```
SQL> SELECT   ename, job, sal
     FROM     emp
     WHERE    (job='SALESMAN'
     OR       job='PRESIDENT')
     AND      sal>1500;
```

Clause ORDER BY

- Tri des lignes avec la clause ORDER BY
  - ASC : ordre croissant (par défaut)
  - DESC : ordre décroissant
- La clause ORDER BY se place à la fin de l'ordre SELECT

```
SQL> SELECT ename, job, deptno, hiredate
     FROM emp
     ORDER BY hiredate;
```

- Tri par Ordre Décroissant

```
SQL> SELECT ename, job, deptno, hiredate
     FROM emp
     ORDER BY hiredate DESC;
```