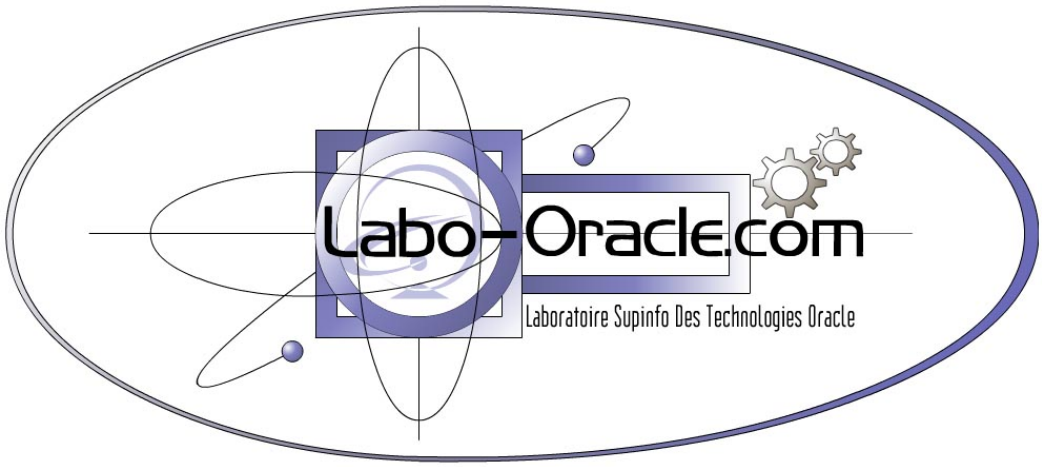


Module n°1

Ordres SELECT basique

1Z0-007

Auteur : Andrei LANGEAC
Version 1.1 – 8 octobre 2004
Nombre de pages : 46

Ecole Supérieure d'Informatique de Paris
23. rue Château Landon 75010 – PARIS
www.supinfo.com

Table des matières

1. Introduction

1.1. Les composants de ORACLE 9i

Oracle 9i est composé de deux produits : Oracle 9i Application Server et Oracle 9i Database.
Cela fournit une infrastructure simple et complète pour les applications Internet.

1.1.1. ORACLE 9i Application Server

Oracle 9i Application Server (Oracle 9i AS) permet de faire fonctionner :

- Les portails
- Les sites web
- Les applications java
- Les applications de gestion.

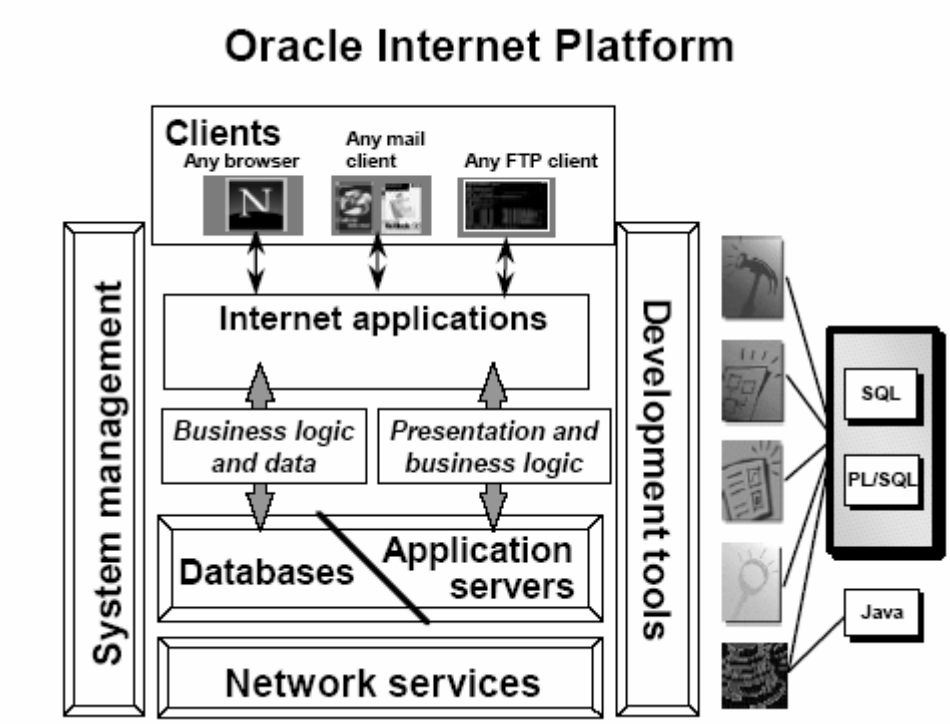
1.1.2. ORACLE 9i Database

La base de données permet de gérer toutes vos données, même les données non structurées comme les documents Word, les présentations PowerPoint, XML, etc… Les données peuvent aussi se trouver ailleurs que dans la base car Oracle 9i database possède des services qui grâce à des metadatas permettent de retrouver les informations stockées dans des fichiers.

1.2. Oracle Internet Platform

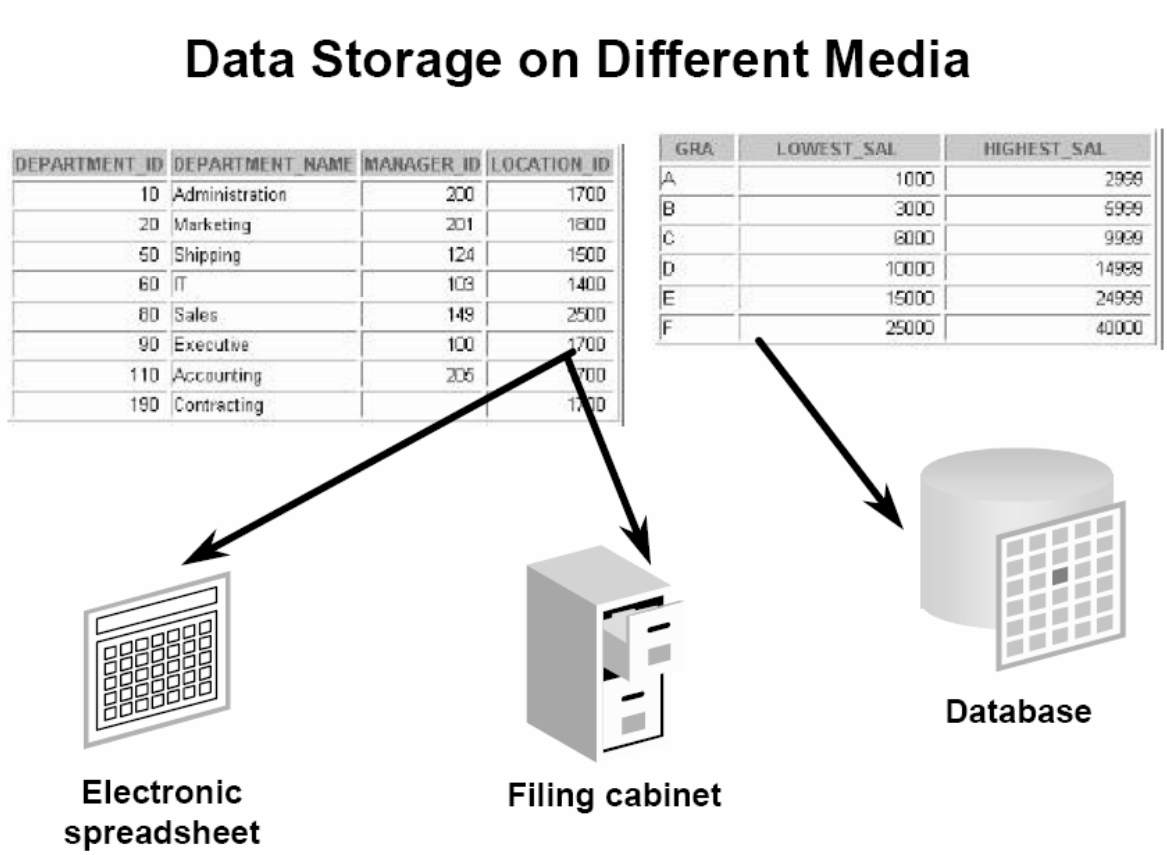
Oracle offre une très performante plateforme Internet pour les solutions d'e-commerce et le stockage des données. Cette plateforme possède trois pièces maîtresses :

- Clients Internet pour visualiser la présentation.
- Les serveurs d'applications.
- Les bases de données pour envoyer les données vers les serveurs d'application.



1.3. Stockage des données dans les différents medias

Toute organisation a besoin d'informations. Une bibliothèque contient la liste des membres, des livres Une société a besoin de sauvegarder des informations concernant leur personnel, les fiches de paye. Elles peuvent être stockées sur des supports papiers, dans les fichiers informatiques ou encore dans des bases de données.



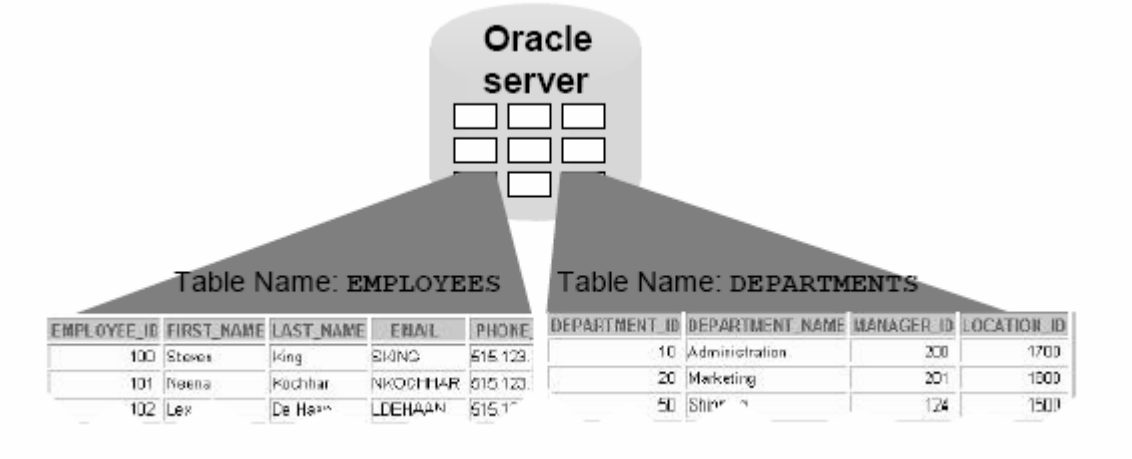
Une base de données est un ensemble d'information organisé, voici les 4 principaux types de base de données :

- Hiérarchique
- Réseau
- Relationnel
- Objet relationnel

1.4. Concept de base de données relationnel.

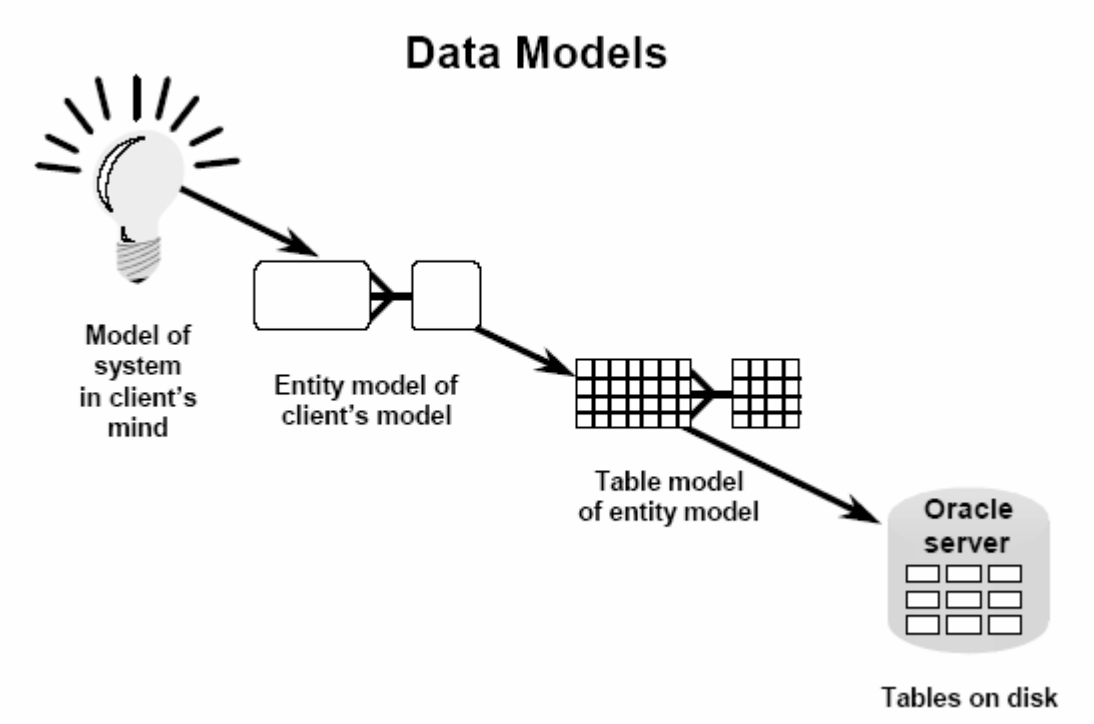
1.4.1. Définition

Une base de données relationnelle utilise des relations ou des tables à deux dimensions pour stocker les informations.



1.4.2. Modèles de données

Les modèles sont utilisés pour explorer des idées, optimiser et comprendre le design d'une base de données.



1.4.3. Modèle d'entité relationnelle

Dans un système efficace, les données sont divisées en catégories ou entités. Le modèle d'entité relationnelle est une représentation de ces différentes entités et des relations entre elles.

1.4.4. Les conventions de la modélisation

Pour les entités :

- Un rectangle aux bords arrondis sans aucune dimension
- Un nom unique et singulier
- Le nom d'entité en majuscules
- Les noms synonymes optionnels entre parenthèses

Pour les attributs :

- Des noms singuliers en minuscules
- Les attributs obligatoires suivis d'un astérisque *
- Les attributs optionnels suivis de la lettre o

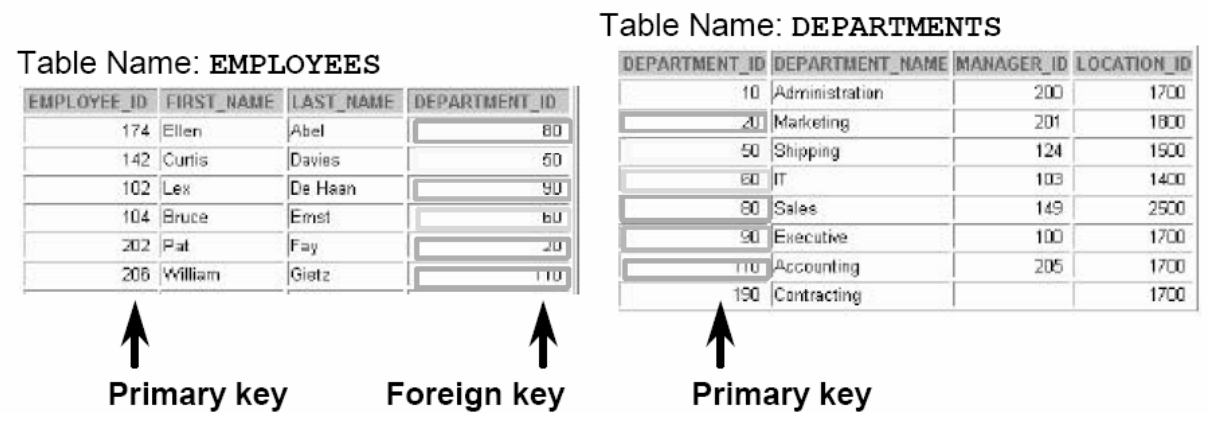
Pour les relations :

- Pointillés (----) L'élément optionnel indiquant « peut-être »
- Une ligne en gras (____) L'élément obligatoire indiquant « sûrement »
- Patte d'oie Degré indiquant « un ou plusieurs »
- Une ligne Degré indiquant « un et un seul »

Pour les identifiants uniques :

- Ils peuvent être une combinaison des attributs, des relations ou les deux
- Chaque attribut principal qui fait partie des identifiants uniques est précédé de #
- Chaque attribut secondaire qui fait partie des identifiants uniques est précédé de (#)

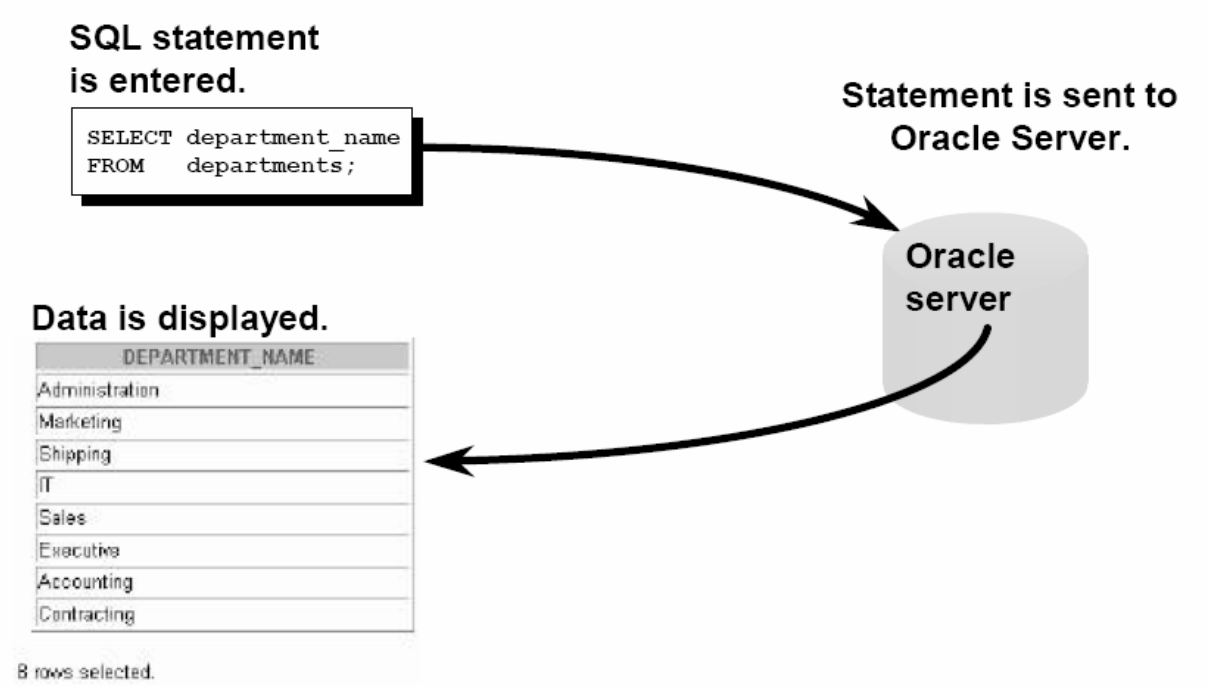
1.4.5. Relations entre plusieurs tables



Chaque table contient des données qui décrivent une seule entité. Pour répondre à des questions particulières vous serez obliger de relier plusieurs tables entre elles. Cela est possible grâce à l'utilisation des clés primaires et des clés étrangères. (CF. Module 3)

1.5. Interaction avec RDBMS en utilisant SQL

Grâce à SQL (Structured Query Langage), un langage intuitif assez proche de l'anglais, il est possible de définir, récupérer et manipuler les données d'une base. Ce langage possède des opérateurs qui permettent de partitionner et de combiner les tables.



1.6. Requêtes SQL

Il existe cinq types d'ordres SQL :

Requête	Type de langage	Définition
SELECT	**DRL Data Retrieval Language** (Langage de récupération des données)	Ensemble de commandes qui permettent de récupérer les données contenues dans une ou plusieurs tables de la base.
INSERT *UPDATE* *DELETE* *MERGE*	**DML Data Manipulation Language** (Langage de manipulation des données)	Ensemble de commandes qui permettent de modifier les données de la base.
CREATE *ALTER* *DROP* *RENAME* *TRUNCATE*	**DDL Data Definition Language** (Langage de définition des données)	Ensemble de commandes qui permettent de modifier la structure de la base.
COMMIT *ROLLBACK* *SAVEPOINT*	**TCS Transaction Control Statement** (Ordre de contrôle de transaction)	Ensemble de commandes qui permettent d'administrer les changements effectués par les commandes DML.
GRANT *REVOKE*	**DCL Data Control Language** (Langage de contrôle des données)	Ensemble de commandes qui permettent de contrôler les accès utilisateur à la base de données.

2. ECRITURE D'ORDRES SQL BASIQUES.

2.1. Ordre SELECT

2.1.1. Les capacités d'ordre SELECT

L'ordre SELECT sert à extraire les données de la base de données :

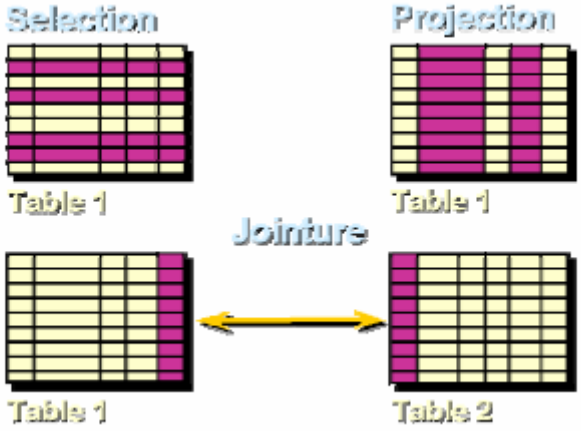
SELECT *what*
FROM *where* ;

Cet ordre est compose de deux parties :

- La clause SELECT qui spécifie les colonnes à sélectionner.
- La clause FROM qui spécifie la table où sont situées les colonnes indiquées dans la clause SELECT.

L'ordre SELECT possède trois capacités :

- Sélection : Sélection d'une ou plusieurs ligne(s).
- Projection : Sélection d'une ou plusieurs colonne(s).
- Jointure : Sélection de deux colonnes dans deux tables différentes, créant ainsi une relation entre les données des deux colonnes.



Pour exécuter une requête SQL il faut la terminer par les caractères " **;** " ou " / ". (Le caractère " / " permet aussi de re-exécuter l'ordre SQL stocké dans le buffer).

Pour faciliter la compréhension du code SQL, il est fortement conseillé de mettre chaque clause sur des lignes différentes, de noter les mots réservés en majuscule et les autres mots en minuscule.

2.1.2. Ordre SELECT basique

Un ordre SELECT sert à extraire des données stockées dans des colonnes issues de tables de la base de données :

SELECT * | *{[DISTINCT] column|expression [alias] …}*
FROM *table* ;

SELECT	Lister une ou plusieurs colonne(s)
*	Sélectionner toutes les colonnes
DISTINCT	Supprimer les doublons
Column|expression	Préciser les noms de colonnes
Alias	Attribuer les entêtes aux colonnes
FROM table	Spécifier la table qui contient les colonnes

L'ordre des colonnes spécifiées dans la clause SELECT correspond à l'ordre des colonnes affichées dans les résultats.

A l'affichage les données numériques sont automatiquement alignées à droite, et les données alpha numérique ou date automatiquement alignées à gauche.

Tous les en-têtes de colonne sont affichés, par défaut, en majuscule.

2.1.3. Sélection des colonnes

Exemple:

```
SQL>        SELECT *
  2         FROM dept;

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

Explication : On sélectionne et affiche toutes les colonnes de la table dept.

Exemple:

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

    DEPTNO LOC
---------- -------------
        10 NEW YORK
        20 DALLAS
        30 CHICAGO
        40 BOSTON
```

Explication: Seulement les colonnes DEPTNO et DNAME de la table DEPT sont sélectionnées et affichées.

2.1.4. Ecriture des requêtes SQL

Pour écrire des requêtes valides vous devez suivre ses règles suivantes :

- Les requêtes SQL ne sont pas sensibles à la casse, sauf contre indication.
- Les requêtes SQL peuvent être écrites sur une ou plusieurs lignes.
- Les mots clés ne peuvent pas être abrégés.
- Les différentes clauses sont placées sur les lignes différentes pour plus de visibilité.
- Les sauts de ligne doivent être utilisés pour rendre le code lisible.
- Les mots clés sont mis en majuscule, les autres noms tels que les noms de tables ou de colonnes sont mis en minuscule.

2.1.5. Les expressions arithmétiques

Les opérateurs arithmétiques sont utilisés pour effectuer les opérations sur les dates ou sur les nombres. Les opérateurs disponibles sont :

Opérateur	Description
+	Additionner
-	Soustraire
*	Multiplier
/	Diviser

Exemple:

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

ENAME                   SAL          SAL+300
---------------   -----------        ----------
SMITH                   800          1100
ALLEN                   1600         1900
WARD                    1250         1550
JONES                   2975         3275
MARTIN                  1250         1550
BLAKE                   2850         3150
…
14 ligne(s) sélectionnée(s).
```

Priorités des opérateurs :

Les opérateurs arithmétiques peuvent être utilisés dans toutes les clauses sauf dans la clause FROM. Les opérateurs arithmétiques ***** et / sont prioritaires par rapport aux opérateurs arithmétiques **+, –.** Les opérateurs de même priorité sont évalués de gauche à droite.

Exemple:

```
SQL>       SELECT       ename, sal, 12*sal+100
  2        FROM         emp;

ENAME             SAL 12*SAL+100
---------- ---------- ----------
SMITH              800       9700
ALLEN             1600      19300
WARD              1250      15100
JONES             2975      35800
MARTIN            1250      15100
BLAKE             2850      34300
…
14 ligne(s) sélectionnée(s).
```

Des parenthèses peuvent être utilisées pour clarifier les calculs et modifier l'ordre d'évaluation.

Exemple:

```
SQL>       SELECT       ename, sal, 12*(sal+100)
  2        FROM         emp;

ENAME             SAL 12*(SAL+100)
---------- ---------- ------------
SMITH              800        10800
ALLEN             1600        20400
WARD              1250        16200
JONES             2975        36900
MARTIN            1250        16200
BLAKE             2850        35400

17 ligne(s) sélectionnée(s).
```

Une valeur nulle est une valeur non assignée ou inconnue. Elle n'est pas équivalente à zéro ou à un espace.

Exemple:

```
SQL>       SELECT       ename, job, sal, comm
  2        FROM         emp;

ENAME      JOB              SAL       COMM
---------- --------- ---------- ----------
SMITH      CLERK             800       3000
ALLEN      SALESMAN         1600        500
WARD       SALESMAN         1250        500
JONES      MANAGER          2975
MARTIN     SALESMAN         1250       1400
BLAKE      MANAGER          2850
…
14 ligne(s) sélectionnée(s).
```

Explication : La valeur de la commission des employés BLAKE et JONES est nulle tandis que la commission des employés ALLEN et WARD est égale à 500.

Si une expression arithmétique contient une valeur nulle, alors le résultat de l'expression sera nul.

Exemple :

```
SQL>        SELECT       ename, 12*sal+comm
  2         FROM         emp
  3         WHERE        ename='KING';

ENAME      12*SAL+COMM
---------- -----------
KING
```

2.2. Personnalisation des requêtes

2.2.1. Alias des colonnes

Un alias de colonne est une chaîne de caractère qui se substitut au nom de la colonne pour simplifier l'affichage.

SELECT *column1 alias1, column2 alias2…*
FROM *table* ;

Il existe deux possibilités pour attribuer l'alias à une colonne soit l'alias est composé d'une chaîne de caractères sans espace, il suffira juste de séparer le nom de la colonne et l'alias par un espace soit d'utiliser le mot optionnel **AS**.

SELECT *column1 AS alias1, column2 AS alias2…*
FROM *table;*

Exemple:

```
SQL>        SELECT       ename name
  2         FROM         emp;

NAME
----------
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
…
14 ligne(s) sélectionnée(s).
```

La requête ci-dessus est équivalente à celle qui suit :

```
SQL>        SELECT       ename AS name
  2         FROM         emp;

NAME
----------
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
…
14 ligne(s) sélectionnée(s).
```

Dans ces deux cas, l'alias *name* apparaîtra en majuscule dans le résultat.

Pour respecter la casse de l'alias, il suffit de la placer entre guillemets.

Exemple:

```
SQL>        SELECT        ename "Name"
  2         FROM          emp;

Name
----------
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
…
14 ligne(s) sélectionnée(s).
```

Exemple:

```
SQL>        SELECT        ename "Employee's Name",
  2                       sal*12 "Annual Salary"
  3         FROM          emp;

Employee's Name   Annual Salary
---------------   --------
SMITH               9600
ALLEN              19200
WARD               15000
JONES              35700
MARTIN             15000
BLAKE              34200
…
14 ligne(s) sélectionnée(s).
```

Explication : La casse des alias est respectée.

2.2.2. Opérateur de concaténation

La combinaison de deux barres verticales " || " est utilisée pour concaténer des colonnes ou des chaînes de caractères.

Exemple:

```
SQL>        SELECT        ename || deptno "Password"
  2         FROM          emp;

Password
--------------------------------------------------
SMITH20
ALLEN30
WARD30
JONES20
MARTIN30
BLAKE30
…
14 ligne(s) sélectionnée(s).
```

2.2.3. Chaîne de caractère littérale

Des chaînes de caractères peuvent être ajoutées dans une clause SELECT (ce ne sont ni des noms de colonnes ni des alias). Ces chaînes sont constantes et s'affichent pour chaque ligne du résultat. Elles peuvent être de type caractère, nombre ou date. Les chaînes de type caractère et date doivent être placées entre simple côtes (‘ ’).

Exemple :

```
SQL> SELECT       ename, 'travaille dans le département', deptno
  2  FROM         emp;

ENAME      'TRAVAILLEDANSLEDéPARTEMENT'      DEPTNO
---------- ---------------------------- ----------
SMITH      travaille dans le département         20
ALLEN      travaille dans le département         30
WARD       travaille dans le département         30
JONES      travaille dans le département         20
MARTIN     travaille dans le département         30
BLAKE      travaille dans le département         30
…
14 ligne(s) sélectionnée(s).
```

Explication : Dans ce cas, la chaîne de caractères "travaille dans le département" possède sa propre colonne lors de l'affichage. Les champs de cette colonne ont tous la même valeur : la chaîne de caractères "travaille dans le département".

Exemple :

```
SQL> SELECT ename || ' travaille dans le département ' || deptno
  2        "Localisation"
  3  FROM   emp;

Localisation
--------------------------------------------------------------------

SMITH travaille dans le département 20
ALLEN travaille dans le département 30
WARD travaille dans le département 30
JONES travaille dans le département 20
MARTIN travaille dans le département 30
BLAKE travaille dans le département 30
…
14 ligne(s) sélectionnée(s).
```

Explication : Grâce à l’opérateur de concaténation, toutes les colonnes (y compris la chaîne de caractères) ne forment qu’une seule colonne lors de l’affichage.

2.2.4. Elimination des doublons

Un doublon est un enregistrement qui se répète plusieurs fois, grâce au mot clé DISTINCT, on peut éliminer ces doublons lors de l’affichage.

SELECT [**DISTINCT**] { * | {*column* [*alias*] | *expr, …*}}
FROM *table* ;

Exemple sans l'utilisation du mot **DISTINCT** :

```
SQL>        SELECT       deptno
  2         FROM         emp;

    DEPTNO
----------
        20
        30
        30
        20
…
14 ligne(s) sélectionnée(s).
```

Explication : Certains départements apparaissent plusieurs fois dans le résultat. La requête affiche tous les doublons.

Exemple avec le mot clé **DISTINCT** :

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

    DEPTNO
----------
        10
        20
        30

3 ligne(s) sélectionnée(s).
```

Explication : Grâce au mot-clé **DISTINCT**, seulement trois enregistrements sont retournés au lieu de quatorze. Ces trois enregistrements correspondent aux trois numéros de département.

2.3. Interaction avec iSQL*PLUS

2.3.1. SQL et iSQL*PLUS

SQL est un langage qui permet de communiquer avec le serveur Oracle à partir des différents outils ou applications.

iSQL*Plus est un outil Oracle qui reconnaît et soumet les ordres SQL au serveur Oracle pour l'exécution. SQL*Plus possède son propre langage.

Caractéristiques de SQL :

- SQL peut être utilisé par des personnes ayant peu, voir aucune expérience en programmation.
- SQL est un langage non procédural.
- SQL réduit le temps pour la création et la maintenance des systèmes.
- SQL est un langage proche de l'anglais.

Caractéristiques de iSQL*Plus :

- Accessible depuis un navigateur.
- Accepte les entrées "ad hoc" des ordres.
- Fournit un éditeur en ligne pour modifier les ordres SQL.
- Contrôle les paramètres d'environnement.
- Formate les résultats d'une requête sous la forme d'un rapport.
- Accède à des bases de données locales et distantes.

Les différences entre SQL et iSQL*Plus sont les suivantes :

SQL	iSQL*Plus
Un langage pour communiquer avec le Serveur Oracle afin d'accéder aux données	Reconnaît les ordres SQL et les envoie au serveur Oracle
Un standard ANSI	Interface propriétaire Oracle pour exécuter des ordres SQL
Les ordres SQL manipulent des données et la définition des tables dans la base de données	Ne nécessite pas la manipulation des données et la définition des tables dans la base de données
Ne possède pas de caractères de continuation	Possède un caractère de continuation : un tiret (-)
Les mots clés n'ont pas d'abréviation	Les mots clés peuvent avoir des abréviations
Utilise des fonctions pour obtenir certain formatage	Utilise des commandes pour formater les données

2.3.2. Utilisation d'iSQL*PLUS

iSQL*Plus est un environnement dans lequel on peut :

- Exécuter des ordres SQL pour récupérer, modifier, ajouter et supprimer des données de la base.
- Formater, permettre des calculs, stocker et imprimer les résultats d'une requête sous la forme d'un rapport.
- Créer des fichiers de script pour stocker des ordres SQL destinés à être souvent utilisés.

Les commandes iSQL*Plus se divisent en sept catégories :

Catégories	Explication
Environnement	Affecte le comportement général des ordres SQL pour une session
Formatage	Formate les résultats d'une requête
Manipulation des fichiers	Sauvegarde, charge et lance des fichiers script
Exécution	Envoie les ordres SQL du navigateur au serveur Oracle
Edition	Modifie les ordres SQL stockés dans le buffer
Interaction	Crée et transmet des variables à des ordres SQL, imprime les valeurs des variables et des messages à l'écran
Divers	Possède différentes commandes pour se connecter à une base de données, manipule l'environnement iSQL*Plus et affiche les définitions des colonnes

2.3.3. Commande DESCRIBE

La commande **DESCRIBE** (ou **DESC**) est une commande propre à SQL*Plus qui affiche la structure de la table passée en argument.

DESCRIBE *table_name* ;

Exemple:

```
SQL> DESCRIBE     emp;
 Nom                                       NULL ?   Type
 ----------------------------------------- -------- ----------------

 EMPNO                                     NOT NULL NUMBER(4)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(9)
 MGR                                                NUMBER(4)
 HIREDATE                                           DATE
 SAL                                                NUMBER(7,2)
 COMM                                               NUMBER(7,2)
 DEPTNO                                             NUMBER(2)
```

La commande **DESCRIBE** affiche, pour chaque colonne de la table, le nom de la colonne, l'existence ou non d'une contrainte **NOT NULL** sur la colonne et le type de données de la colonne.

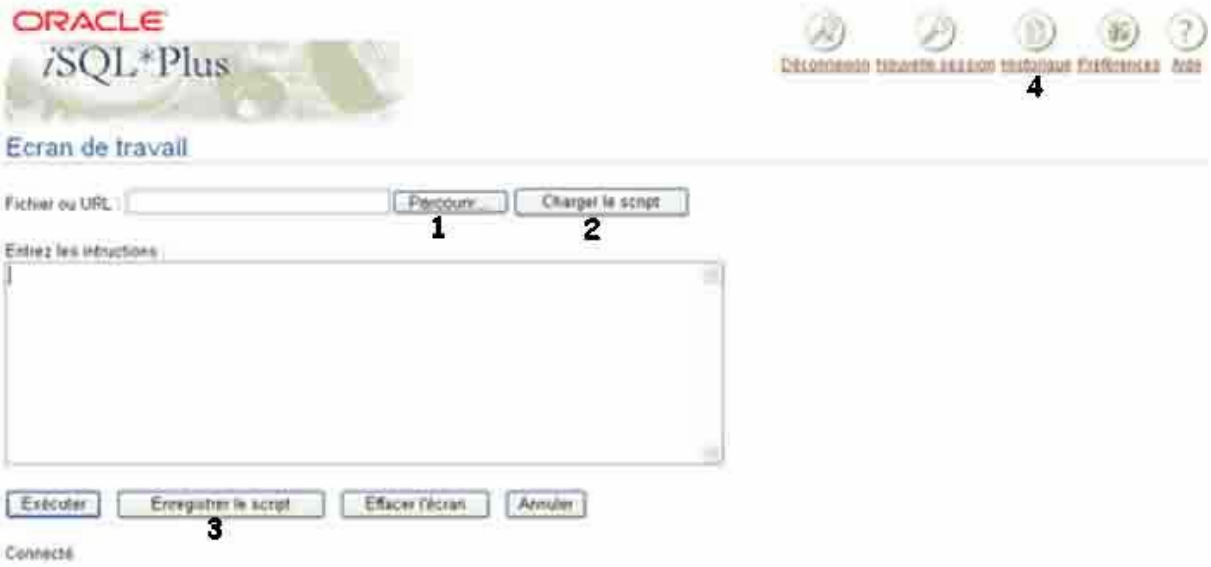
Les principaux types de données sont :

Type de données	Description
NUMBER	Chiffre
CHAR	Chaîne de caractères
VARCHAR2	Chaîne de caractères de longueur variable
DATE	La valeur de date et d'heure entre 1er Janvier 4712 avant JC et 31 Décembre 9999

2.3.4. Manipulation des fichiers dans iSQL*Plus

Pour se connecter au serveur, il suffit de saisir son nom d'utilisateur (login), son mot de passe (password) et la chaîne d'hôte (nom de service) sur la page de connexion de votre navigateur

Si la connexion réussit la page suivante apparaît :



1 – Bouton « Parcourir » permet d'aller chercher vos fichiers scripts sur le disque dur
2 – Bouton « Charger le script » permet de charger votre script dans le buffer.
3 – Bouton « Enregistrer le script » permet de sauvegarder votre script sur un disque dur
4 – Bouton « Historique » permet de retrouver vos anciens scripts déjà utilisés dans iSQL*Plus et de les charger dans le buffer.

2.3.5. Manipulation des fichiers en utilisant des commandes

Les commandes SQL*Plus manipulant les fichiers permettent de sauver, charger, exécuter des fichiers script.

Commande	Description	Exemple
SAV[**E**] *filename*[.*ext*] [**RE**[**PLACE**] | **APP**[**END**]]	Permet de sauvegarder le contenu du buffer dans un fichier. **REPLACE** spécifie que le fichier existant sera écrasé. **APPEND** spécifie que le contenu du buffer doit être ajouté au contenu d'un fichier existant.	`SQL>SAV all_emp`
GET *filename*[.*ext*]	Permet de récupérer le contenu du buffer qui a préalablement été sauvegardé dans un fichier. L'extension pas défaut est *.sql*.	SQL> GET all_emp
STA[**RT**] *filename*[.*ext*]	Permet d'exécuter le contenu d'un fichier.	SQL> STA all_emp
@*filename*[.*ext*]	Permet d'exécuter le contenu d'un fichier (équivalent à **START**).	SQL> @all_emp
ED[**IT**]	Permet d'invoquer l'éditeur et de sauvegarder le contenu du buffer dans un fichier nommé *afiedt.buf*	SQL> ED all_emp
ED[**IT**] [*filename*[.*ext*]]	Permet d'éditer le contenu d'un fichier à l'aide d'un éditeur de texte.	SQL> ED all_emp
SPO[**OL**] filename[.ext] | **OFF**|**OUT**]	Permet de stocker les résultats d'une requête dans un fichier. **OFF**	

	ferme le fichier de spool. **OUT** ferme le fichier de spool et envoie les résultats contenus dans le fichier à l'imprimante.	
EXIT	Permet de se déconnecter et de fermer SQL*Plus automatiquement après.	

2.3.6. Les commandes d'édition de SQL*Plus

Commande	Description	Exemple

A[**PPEND**] *text*	Permet de rajouter à la dernière ligne du buffer la chaîne de caractères *text*.	`SQL>SELECT ename, deptno,`
		Le buffer contient :
		SELECT ename, deptno,
		`SQL> A job`
		Le buffer contient :
		SELECT ename, deptno, job

C[**HANGE**] / *old* / *new*	Permet de remplacer la chaîne de caractères *old* de la ligne courante du buffer par la chaîne de caractères *new*.	`SQL>SELECT ename, deptno, job` Le buffer contient : SELECT ename, deptno, job

C[**HANGE**] / *old* /	Permet de supprimer la chaîne de caractères *old* de la ligne courante.	`SQL> C/job/sal`
		Le buffer contient :
		SELECT ename, deptno, sal
		`SQL> C/, sal`
		Le buffer contient :
		SELECT ename, deptno

I[**NPUT**]	Permet d'insérer une nouvelle ligne dans le buffer.	`SQL>SELECT ename, deptno, sal`
		Le buffer contient :
		SELECT ename, deptno, sal
		`SQL> I FROM emp`
		Le buffer contient :
		SELECT ename, deptno, sal
		`FROM emp`

I[**NPUT**] *text*	Permet d'insérer une nouvelle ligne contenant la chaîne de caractères *text* dans le buffer.	

L[**IST**]	Permet d'afficher le contenu du buffer.	`SQL> L`
		`1 SELECT ename, deptno, sal`
		`2 FROM emp`

L[**IST**] *n*	Permet d'afficher la ligne *n* du buffer.	

<table>
<tr><td>L[IST] m n</td><td>Permet d'afficher les lignes m à n.</td><td></td></tr>
<tr><td>R[UN]</td><td>Permet d'exécuter le contenu du buffer. La commande affiche la requête suivie du résultat de la requête.</td><td rowspan="4"><pre>SQL> 1
1 SELECT ename, deptno,
sal
SQL> 1 SELECT mgr</pre>
Le buffer contient :
SELECT mgr
FROM emp</td></tr>
<tr><td>n</td><td>Permet de spécifier la ligne courante.</td></tr>
<tr><td>n text</td><td>Permet de remplacer la ligne n avec text.</td></tr>
<tr><td>0 text</td><td>Permet d'insérer une ligne contenant text avant la ligne 1.</td></tr>
<tr><td>DEL</td><td>Permet d'effacer la ligne courante du buffer.</td><td></td></tr>
<tr><td>DEL n</td><td>Permet d'effacer la ligne n.</td><td></td></tr>
<tr><td>DEL m n</td><td>Permet d'effacer les lignes m à n.</td><td></td></tr>
<tr><td>CL[EAR BUFF[ER]</td><td>Permet d'effacer le contenu du buffer.</td><td></td></tr>
</table>

3. Restriction et tri des données.

3.1. Restreindre les résultats dans une requête SELECT

3.1.1. La clause WHERE

La clause **WHERE** restreint la requête aux enregistrements qui respectent les conditions.
La clause **WHERE** correspond à une sélection : elle restreint les enregistrements (ou lignes). La clause **WHERE** suit la clause **FROM**.
Un alias ne peut pas être utilisés dans la clause **WHERE**.

SELECT * |{[DISTINCT] column|expression [alias], …}*
FROM *table*
[**WHERE** *condition(s)];*

Exemple:

```
SQL>        SELECT       ename, job, deptno
  2         FROM         emp
  3         WHERE        job='CLERK';

ENAME      JOB           DEPTNO
---------- --------- ----------
SMITH      CLERK             20
ADAMS      CLERK             20
JAMES      CLERK             30
MILLER     CLERK             10
```

Explication : Cette requête affiche le nom, le job et le numéro de département des employés dont la fonction est "CLERK". Attention la comparaison de deux chaînes de caractères est sensible à la casse (‘CLERK’ est différent de ‘Clerk’ ou encore ‘clerk’).

3.1.2. La clause WHERE avec différents types de données

Type de données NUMBER

```
SQL>        SELECT       ename, job, deptno
  2         FROM         emp
  3         WHERE        deptno=10;

ENAME      JOB           DEPTNO
---------- --------- ----------
CLARK      MANAGER           10
KING       PRESIDENT         10
MILLER     CLERK             10
```

Explication : Cette requête affiche le nom, la fonction et le numéro de département des employés appartenant au département numéro 10.

Type de données CHAR ou VARCHAR

```
SQL>       SELECT       ename, job, deptno
  2        FROM         emp
  3        WHERE        job='ANALYST';

ENAME      JOB           DEPTNO
---------- --------- ----------
SCOTT      ANALYST           20
FORD       ANALYST           20
```

Explication : Cette requête affiche le nom, la fonction et le numéro de département des employés dont la fonction est "ANALYST".

Type de données DATE

```
SQL>       SELECT       ename, job, deptno
  2        FROM         emp
  3        WHERE        hiredate = '23-JAN-82';

ENAME      JOB           DEPTNO
---------- --------- ----------
MILLER     CLERK             10
```

Explication : Cette requête affiche le nom, la fonction et le numéro de département des employés embauchés le 1er janvier 1999. (DD-MON-YY est le format d'affichage par défaut d'une date dans la base de données).

3.2. Opérateurs de comparaison

Les opérateurs de comparaisons sont utilisés pour comparer une expression à une valeur ou à une autre expression.

…**WHERE** *expression opérateur valeur*

3.2.1. Les opérateurs arithmétiques

Les opérateurs arithmétiques sont les suivants :

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

Ces opérateurs ne tiennent pas compte des valeurs nulles (**NULL VALUE**).

Exemple:

```
SQL>        SELECT       ename, sal
  2         FROM         emp
  3         WHERE        sal > 2000;

ENAME             SAL
---------- ----------
JONES            2975
BLAKE            2850
CLARK            2450
SCOTT            3000
KING             5000
FORD             3000

6 ligne(s) sélectionnée(s).
```

Explication : Cette requête retourne la liste des employés dont le salaire est strictement supérieur à $2000 par mois.

Exemple:

```
SQL>        SELECT       ename, hiredate
  2         FROM         emp
  3         WHERE        hiredate < '01-JUL-81';

ENAME      HIREDATE
---------- --------
SMITH      17/12/80
ALLEN      20/02/81
WARD       22/02/81
JONES      02/04/81
BLAKE      01/05/81
CLARK      09/06/81

6 ligne(s) sélectionnée(s).
```

Explication : Cette requête retourne la liste des employés embauchés avant le 1er juillet 1981.

Exemple:

```
SQL>        SELECT       deptno, loc
  2         FROM         dept
  3         WHERE        loc <> 'NEW YORK';

    DEPTNO LOC
---------- -------------
        20 DALLAS
        30 CHICAGO
        40 BOSTON
3 ligne(s) sélectionnée(s).
```

Explication : Cette requête affiche les départements qui ne sont pas localisés à NEW YORK.

3.2.2. L'opérateur BETWEEN

L'opérateur **BETWEEN** permet d'afficher des enregistrements basés sur une tranche de valeurs

WHERE *column_name* **BETWEEN** *lower_limit* **AND** *higher_limit*

Les limites peuvent être des nombres, des caractères, des dates. Lorsqu'il s'agit de caractères ou de dates, les limites doivent être placées entre simples côtes.
L'opérateur **BETWEEN** est inclusif (ces limites sont incluses dans la tranche de valeurs possibles).
Le serveur Oracle traduit l'opérateur **BETWEEN...AND** comme la combinaison de deux conditions reliées par l'opérateur **AND**.

Exemple:

```
SQL>       SELECT       ename, sal
  2        FROM         emp
  3        WHERE        sal BETWEEN 1000 AND 3000

ENAME             SAL
---------- ----------
ALLEN            1600
WARD             1250
JONES            2975
MARTIN           1250
BLAKE            2850
CLARK            2450
SCOTT            3000
TURNER           1500
…
12 ligne(s) sélectionnée(s).
```

Explication : Cette requête affiche les employés dont le salaire est situé dans la tranche $1000 à $3000.

Exemple:

```
SQL>       SELECT       ename
  2        FROM         emp
  3        WHERE        ename BETWEEN 'S' AND 'W';

ENAME
----------
SMITH
SCOTT
TURNER
```

Explication: Cette requête affiche les employés dont le nom est situé dans la tranche "S" à "W".
L'employé WARD n'apparaît pas car "WA" est plus grand que "W".

Exemple:

```
SQL> SELECT       ename, hiredate
  2  FROM         emp
  3  WHERE        hiredate BETWEEN '01-JAN-81' AND '31-JUL-81';

ENAME      HIREDATE
---------- --------
ALLEN      20/02/81
WARD       22/02/81
JONES      02/04/81
BLAKE      01/05/81
CLARK      09/06/81
```

Explication : Cette requête affiche les employés embauchés entre le 1er janvier 1981 et le 31 juillet 1981.

3.2.3. L'opérateur LIKE

L'opérateur **LIKE** permet de faire des recherches de caractères spécifiques dans une chaîne de caractères données.

WHERE *column_name* **LIKE** *'value including wildcards'*

Symbole	Description
%	Représente une série de zéros ou de caractères
_	Représente un seul caractère quelconque

Exemple:

```
SQL>        SELECT       ename
  2         FROM         emp
  3         WHERE        ename LIKE '__A%'

ENAME
----------
BLAKE
CLARK
ADAMS
```

Explication : Cette requête affiche les employés dont le troisième caractère de leur nom est un 'A' suivie d'une chaîne de caractères n.
Le mot clé **ESCAPE** sert à rechercher un caractère spécial, comme ' % ' ou ' _ '. L'opérateur **LIKE** considère alors le caractère spécial comme un caractère quelconque.

WHERE *column_name* **LIKE** *'value1value2'* **ESCAPE** *'caractère d'échappement'*

Exemple:

```
SQL>        SELECT       ename
2           FROM         emp
3           WHERE        ename LIKE 'SGBD_%';

ENAME
-----------
SGBD_JAROD
SGBD_HELYOS
SGBD_YO
SGBD_ORACLE
```

Explication : Cette requête affiche les noms commençant par la chaîne de caractères "SGBD" suivis d'un caractère quelconque. Dans ce cas, le caractère "_" est interprété comme un caractère quelconque et non comme sa propre signification.
Voici la même requête avec l'utilisation du mot clé **ESCAPE** afin d'afficher les noms commençant par la chaîne de caractères "SGBD_" :

Exemple :

```
SQL>        SELECT       ename
2           FROM         emp
3           WHERE        ename LIKE ‘SGBD\_%’ ESCAPE '\' ;

ENAME
-----------
SGBD_JAROD
SGBD_HELYOS
SGBD_YO
```

3.2.4. L’opérateur IN

L'opérateur **IN** permet d'afficher des enregistrements appartenant à une liste de valeurs.

WHERE *column_name* **IN** *(value1, value2, value3…)*

Les valeurs de la liste peuvent être des nombres, des caractères, des dates. Dans le cas où il s'agirait de caractères ou de dates, les valeurs doivent être placées entre simples côtes.

Exemple:

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

     EMPNO ENAME              MGR
---------- ---------- ----------
      7369 SMITH            7902
      7788 SCOTT            7566
      7876 ADAMS            7788
      7902 FORD             7566
```

Explication : Cette requête affiche les employés dont les numéros de manager sont 7902 ou 7566 ou 7788.

3.2.5. L’opérateur IS NULL

L'opérateur **IS NULL** permet d'afficher les enregistrements dont certains champs contiennent des valeurs nulles.
Une valeur nulle signifie que la valeur n'est pas disponible, non assignée, inconnue ou inapplicable.

WHERE *column_name* **IS NULL**

Exemple:

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

ENAME              MGR
---------- ----------
KING
```

Explication : Cette requête affiche les employés qui ne possèdent pas de manager.

3.3. Les opérateurs logiques

3.3.1. L'opérateur AND

L'opérateur **AND** permet d'afficher les enregistrements qui vérifient toutes les conditions impliquées dans l'expression.

WHERE *condition1*
AND *condition2;*

Résultats d'une combinaison de deux conditions **AND** :

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Exemple :

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

     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7876 ADAMS      CLERK           1100
      7934 MILLER     CLERK           1300
```

Explication : Cette requête affiche les employés dont la fonction est CLERK et dont le salaire est supérieur ou égal à $1100.

3.3.2. L'opérateur OR

L'opérateur **OR** permet d'afficher les enregistrements qui vérifient au moins une des conditions impliquées dans l'expression.

WHERE *condition1*
OR *conditon2 ;*

Résultats d'une combinaison de deux conditions **OR** :

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Exemple :

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

     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7369 SMITH      CLERK             800
      7499 ALLEN      SALESMAN         1600
      7521 WARD       SALESMAN         1250
      7566 JONES      MANAGER          2975
      7654 MARTIN     SALESMAN         1250
      7698 BLAKE      MANAGER          2850
…
14 ligne(s) sélectionnée(s).
```

Explication : Cette requête affiche les employés dont la fonction est CLERK ou dont le salaire est supérieur ou égal à $1100.

3.3.3. L'opérateur NOT

L'opérateur **NOT** permet d'afficher les enregistrements qui ne vérifient pas la condition impliquée dans l'expression.

WHERE *column_name* **NOT** *comparison_operator value*

Résultats d'une combinaison de deux conditions **NOT** :

NOT	TRUE	FALSE	NULL
TRUE	FALSE	TRUE	NULL

Utilisation de l'opérateur **NOT**:

WHERE *column_name* **NOT IN** *list_values*
WHERE *column_name* **NOT LIKE** '*wirdcard*'
WHERE *column_name* **NOT BETWEEN** *limit* **AND** *limit*
WHERE *column_name* **IS NOT NULL**

Exemple:

```
SQL>       SELECT       ename, comm
  2        FROM         emp
  3        WHERE        comm IS NOT NULL;

ENAME            COMM
---------- ----------
SMITH            3000
ALLEN             500
WARD              500
MARTIN           1400
TURNER              0
…
10 ligne(s) sélectionnée(s).
```

Explication : Cette requête affiche la liste des employés ayant une commission.

3.3.4. Ordre d'évaluation des opérateurs

Ordre d'évaluation	Opérateur
1	Opérateurs arithmétiques
2	Opérateur de concaténation
3	Les conditions de comparaison
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Condition logique NOT
7	Condition logique AND
8	Condition logique OR

Exemple :

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

ENAME      JOB              SAL
---------- --------- ----------
ALLEN      SALESMAN        1600
KING       PRESIDENT       5000
```

3.4. Ordonner la clause SELECT

3.4.1. La clause ORDER BY

La clause **ORDER BY** permet d'afficher les enregistrements sélectionnés dans l'ordre croissant ou décroissant. L'ordre par défaut est croissant.

SELECT [**DISTINCT**] *{ * | {column [alias] | expr, …}}*
FROM *table*
[**WHERE** *condition(s)*];
[**ORDER BY** {*column* | *expr*} [**ASC** | **DESC**] ;

Exemple:

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

ENAME      JOB           DEPTNO HIREDATE
---------- --------- ---------- --------
SMITH      CLERK             20 17/12/80
ALLEN      SALESMAN          30 20/02/81
WARD       SALESMAN          30 22/02/81
JONES      MANAGER           20 02/04/81
BLAKE      MANAGER           30 01/05/81
…
14 ligne(s) sélectionnée(s).
```

Les enregistrements peuvent être triés sur une colonne qui n'a pas été sélectionnée dans la clause **SELECT**.

SELECT *column1*, *column2*
FROM *table*
ORDER BY *column3*

3.4.2. Trier en ordre décroissant

Le mot-clé **DESC** permet d'afficher les enregistrements sélectionnés dans l'ordre décroissant.

ORDER BY {*column* | *expr*} **DESC**

Les valeurs nulles sont affichées en premier lors d'un tri décroissant.

Exemple:

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

ENAME      JOB           DEPTNO HIREDATE
---------- --------- ---------- --------
ADAMS      CLERK             20 12/01/83
SCOTT      ANALYST           20 09/12/82
MILLER     CLERK             10 23/01/82
JAMES      CLERK             30 03/12/81
FORD       ANALYST           20 03/12/81
KING       PRESIDENT         10 17/11/81
MARTIN     SALESMAN          30 28/09/81

17 ligne(s) sélectionnée(s).
```

3.4.3. Trier en fonction des alias

Les enregistrements peuvent être triés grâce aux alias de colonne

SELECT *column1 alias1*, *column2 alias2*
FROM *table*
ORDER BY *alias1*

Exemple:

```
SQL>        SELECT       empno, ename, sal*12 annsal
  2         FROM         emp
  3         ORDER BY     annsal;

     EMPNO ENAME          ANNSAL
---------- ---------- ----------
      7369 SMITH            9600
      7900 JAMES           11400
      7952 PAPIER          12000
      7876 ADAMS           13200
      7521 WARD            15000
      7654 MARTIN          15000
…
14 ligne(s) sélectionnée(s).
```

3.4.4. Trier en fonction des plusieurs colonnes

Les enregistrements peuvent être triés sur plusieurs colonnes.

ORDER BY *column1*, *column2*
ORDER BY *column1* [**DESC** | **ASC**], *column2* [**DESC** | ASC**];**

Column1 et *column2* peuvent être des noms de colonnes, des expressions, des alias ou des numéros de colonnes. Vous pouvez trier en fonction de colonnes qui ne sont pas présentes dans la clause **SELECT**.

Exemple:

```
SQL>        SELECT       ename, deptno, sal
  2         FROM         emp
  3         ORDER BY     deptno, sal DESC;

ENAME          DEPTNO        SAL
---------- ---------- ----------
KING               10       5000
CLARK              10       2450
MILLER             10       1300
SCOTT              20       3000
FORD               20       3000
JONES              20       2975
ADAMS              20       1100
…
14 ligne(s) sélectionnée(s).
```

Explication : Les enregistrements sont triés dans l'ordre croissant par rapport au numéro de département et, au numéro de département identique, dans l'ordre décroissant par rapport au salaire.

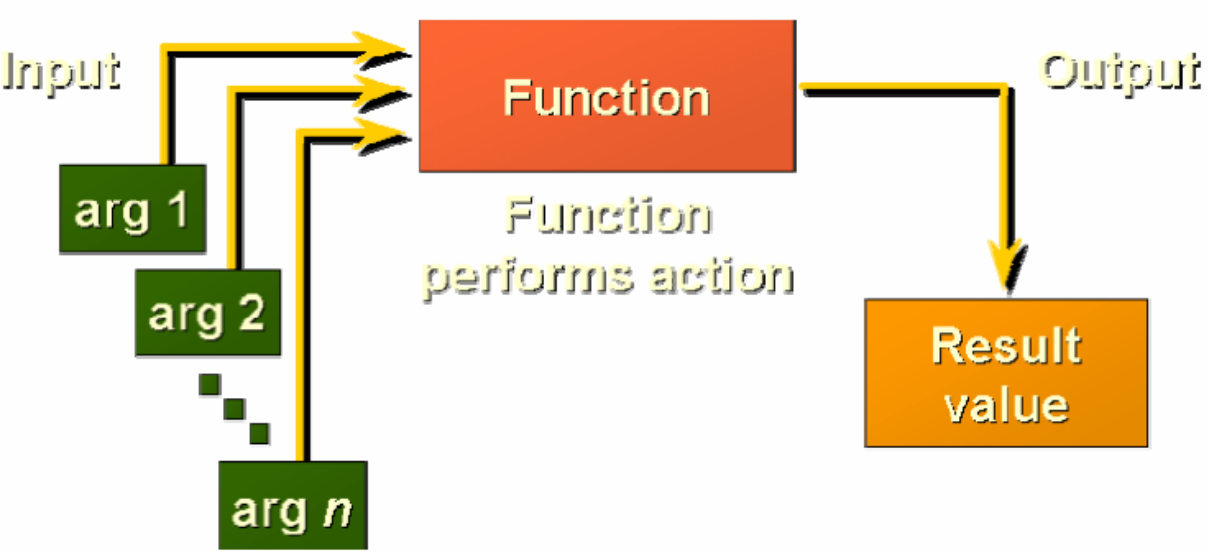
4. LES FONCTIONS SINGLE-ROW

4.1. Les fonctions SQL

Une fonction SQL est un programme qui effectue une opération sur des données.
Les fonctions SQL peuvent être utilisées pour :

- Exécuter des calculs sur des données.
- Formater des dates et des nombres pour l'affichage.
- Convertir des types de données de colonne.

Une fonction SQL peut accepter un à plusieurs arguments mais ne retourne toujours qu'une seule valeur.



4.2. Les fonctions opérant sur les caractères

4.2.1. Les fonctions de manipulation de casse

Les fonctions de conversion de la casse permettent d'afficher ou d'utiliser les données dans une casse différente de celle qu'elles possèdent dans la table.

Fonctions	Définition	Exemples	Resultat
INITCAP(column | expr)	Convertit la première lettre de chaque mot d'une chaîne de caractères en majuscule et les autres lettres en minuscule.	INITCAP('Cours de SQL')	Cours De Sql
LOWER(column | expr)	Convertit une chaîne de caractères en minuscule.	LOWER('Cours de SQL')	cours de sql
UPPER(column | expr)	Convertit une chaîne de caractères en majuscule.	UPPER ('Cours de SQL')	COURS DE SQL

Exemple :

```
SQL>        SELECT       empno, ename, deptno
  2         FROM         emp
  3         WHERE        ename = 'clark';

Aucune ligne sélectionnée
```

```
  SQL>      SELECT       empno, ename, deptno
  2         FROM         emp
  3         WHERE        ename = UPPER('clark')

     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      7782 CLARK              10
```

4.2.2. Les fonctions de manipulation de caractère

Fonction	Définition	Exemple	Résultat
LENGTH(*column* | *expr*)	Permet de récupérer le nombre de caractères d'une chaîne. LENGTH retourne une valeur de type NUMBER.	LENGTH ('Bonjour')	7
SUBSTR(*column* | *expr*, *m* [,*n*])	Permet d'extraire une chaîne de caractères de la chaîne de caractère *column* (ou issue de *expr*) sur une longueur *n* à partir de la position *m*.	SUBSTR ('Bonjour',1,3)	Bon
INSTR(*column* | *expr*, *c*)	Permet de récupérer la position de la première occurrence du caractère *c* dans la chaîne de caractères *column* ou issue de *expr*.	INSTR('Bonjour','j')	4
LPAD(*column* | *expr*, *n*, '*string*')	Permet de placer *n* caractères de type *string* à gauche de la valeur de *column* (ou *expr*).	LPAD(sal, 10,'*')	******5000
RPAD(*column* | *expr*, *n*, '*string*')	Permet de placer *n* caractères de type *string* à droite de la valeur de *column* (ou *expr*).	RPAD(sal, 10,'*')	5000******
CONCAT(column1 | expr1, Column2 | expr2)	Permet de concaténer la valeur de la première chaîne à la valeur de la seconde chaîne. (équivalent à l'opérateur "||").	CONCAT ('Bon', 'jour')	Bonjour
TRIM(*leading* | *trailing* | *both*, *trim_character* **FROM** *trim_source*)	Permet de couper les caractères *trim_character* en entête (l*eading*), en fin (*trailing*) ou les deux (*both*) d'une chaîne de caractère *trim_source*.	TRIM ('S' FROM 'SSMITH')	MITH

Exemple:

```
SQL>       SELECT       ename, CONCAT (ename, job), LENGTH(ename),
  2                     INSTR (ename, 'A')
  3        FROM         emp
  4        WHERE        SUBSTR(job,1,5) = 'SALES';

ENAME      CONCAT(ENAME,JOB)   LENGTH(ENAME) INSTR(ENAME,'A')
---------- ------------------- ------------- ----------------
ALLEN      ALLENSALESMAN                   5                1
WARD       WARDSALESMAN                    4                2
MARTIN     MARTINSALESMAN                  6                2
TURNER     TURNERSALESMAN                  6                0
PAPIER     PAPIERSALESMAN                  6                2
```

4.3. Les fonctions opérant sur les nombres

4.3.1. Fonction ROUND

ROUND(*column* | *expr* [,*n*])	Permet d'arrondir une valeur *column* ou issue de *expr* à *n* décimales près.

Si *n* est positif, l'arrondi se fera après la virgule. Si *n* est négatif l'arrondi se fera avant la virgule (à la dizaine près par exemple). Par défaut *n* vaut 0.

Oracle possède un outil très pratique : une table nommée **DUAL** contenant une seule colonne **DUMMY** et contenant un seul enregistrement ayant pour valeur **X**. Cette table peut servir à effectuer des calculs dans un ordre **SELECT** où la clause **FROM** ne contient aucune table dont on a réellement besoin. Comme la clause **FROM** est obligatoire et qu'elle doit contenir au moins une table, c'est la table **DUAL** qui sera spécifiée permettant ainsi de contourner le problème.

Exemple:

```
SQL>       SELECT       ROUND (45.923, 2), ROUND (45.923, 0),
  2                     ROUND (45.923,-1)
  3        FROM         dual;

ROUND (45.923, 2) ROUND (45.923, 0) ROUND (45.923,-1)
----------------  ----------------  ----------------
            45,92                46                50
```

4.3.2. Fonction TRUNC

TRUNC(*column* | *expr* [,*n*])	Permet de tronquer une valeur *column* ou issue de *expr* à *n* décimales près.

Si *n* est positif, la troncation se fera après la virgule. Si *n* est négatif la troncation se fera avant la virgule (à la dizaine près par exemple). Par défaut *n* vaut 0.

Exemple:

```
SQL>        SELECT       TRUNC (45.923, 2), TRUNC (45.923, 0),
  2                      TRUNC (45.923,-1)
  3         FROM         dual
SQL> /

TRUNC (45.923, 2) TRUNC (45.923, 0) TRUNC (45.923,-1)
---------------   ---------------   ----------------
          45,92              45               40
```

La fonction **TRUNC** peut être aussi utilisée avec des dates.

4.3.3. Fonction MOD

MOD(*m*,*n*)	Permet de retourner le reste de la valeur de *m* divisé par *n*.

Exemple:

```
SQL>        SELECT       ename, sal, comm, MOD(sal, comm)
  2         FROM         emp
  3         WHERE        job = 'SALESMAN';

ENAME              SAL       COMM MOD(SAL,COMM)
---------- ---------- ---------- -------------
ALLEN            1600        500           100
WARD             1250        500           250
MARTIN           1250       1400          1250
TURNER           1500          0          1500
```

4.4.Manipulation des dates

4.4.1. Fonction SYSDATE

SYSDATE	Permet de retourner la date et l'heure courante.

Le format interne à la base (Internal format) est : century, year, month, day, hour, minutes, seconds
L'affichage par défaut est DD-MON-YY soit par exemple 14-JUI-80.

La table **DUAL** peut être utilisée pour afficher la date du jour

Exemple:

```
SQL>        SELECT       SYSDATE
  2         FROM         dual;

SYSDATE
--------
04/08/04
```

Explication : Une colonne nommée **SYSDATE** s'affiche, contenant la date du jour au format par défaut.

4.4.2. Opérations arithmétiques sur les dates

Les opérateurs arithmétiques peuvent être utilisés pour effectuer des calculs arithmétiques sur les dates.
Opérations possibles sur les dates sont :

Opération	Résultat	Description
date + nombre	date	Ajoute un nombre de jours à une date
date - nombre	date	Soustrait un nombre de jours à une date
date - date	nombre de jours	Soustrait une date à une autre date
date + nombre/24	date	Ajoute un nombre d'heures à une date

Exemple:

```
SQL>       SELECT       ename, (SYSDATE-hiredate)/7 WEEKS
  2        FROM         emp
  3        WHERE        deptno = 10;

ENAME           WEEKS
---------- ----------
CLARK      1208,24441
KING       1185,24441
MILLER     1175,67298
```

4.4.3. Différentes fonctions de dates

Voici les principales fonctions opérant sur des dates :

Fonction	Définition	Exemple	Résultat
MONTHS_BETWEEN (*date1*,*date2*)	Retourne le nombre de mois séparant deux dates. Le résultat peut-être positif ou négatif. Si *date1* est inférieure à la *date2*, le résultat est positif. Si *date1* est plus récente que *date2*, le résultat est négatif.	MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS(*date*,*n*)	Ajoute *n* mois à une date. *n* doit être un entier positif ou négatif.	ADD_MONTHS ('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY(*date*, '*day of week*')	Trouve la date du prochain jour de la semaine (*day of week*) suivant *date*. La valeur de *day of week* doit être un nombre représentant le jour ou une chaîne de caractères.	NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'

LAST_DAY(*date*)	Trouve la date du dernier jour du mois qui contient *date*.	LAST_DAY('01-SEP-95')	'30-SEP-95'
ROUND(*date* [,'*format*'])	Retourne *date* arrondie à l'unité spécifiée par *format*. Si le format est omis, *date* est arrondie au jour le plus près.	ROUND('25-JUL-95','MONTH')	'01-AUG-95'
		ROUND('25-JUL-95','YEAR')	'01-JAN-96'
TRUNC(*date* [,'*format*'])	Retourne *date* tronquée à l'unité spécifiée par *format*. Si le format est omis, *date* est tronquée au jour le plus près.	TRUNC('25-JUL-95','MONTH')	'01-JUL-95'
		TRUNC('25-JUL-95','YEAR')	'01-JAN-95'

4.5. Fonctions de conversion

4.5.1. Conversion implicite

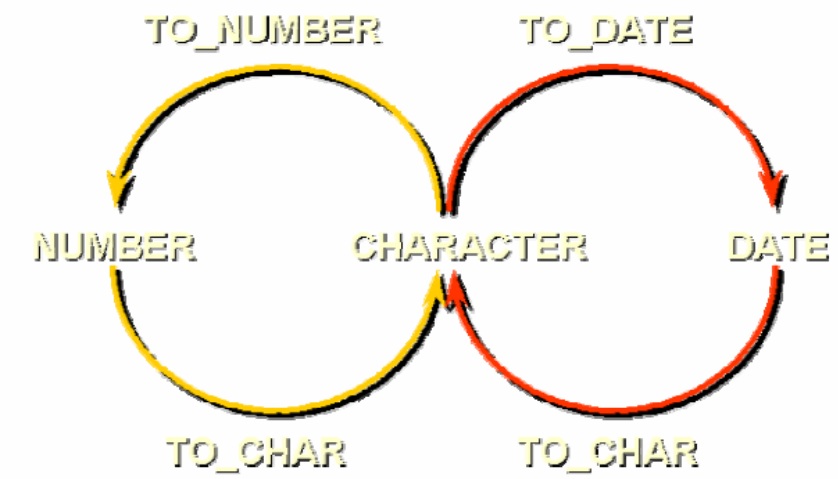
Le serveur Oracle peut automatiquement convertir les types de données suivants :

De	A
VARCHAR2 ou CHAR	NUMBER
VARCHAR2 ou CHAR	DATE
NUMBER	VARCHAR2
DATE	VARCHAR2

4.5.2. Conversion explicite

Voici les trois principales fonctions de conversion explicite de types de données :

Fonction	Définition
TO_CHAR(*number|date,[fmt],[nlsparams]*)	Convertit un nombre ou une date en une chaîne de caractères
TO_NUMBER *(char ['format'])*	Convertit une chaîne de caractères en un nombre
TO_DATE *(char [,'format'])*	Convertit une chaîne de caractères en une date



4.5.3. Utilisation de la fonction TO_CHAR avec des dates

Le format **TO_CHAR** doit être entouré de simples côtes, il est sensible à la casse. Il doit inclure des éléments de format de date valides. Les noms des jours et des mois en entrée sont automatiquement "remplis" avec des blancs. Pour éliminer les blancs ou supprimer les zéros, il faut utiliser l'élément "fill mode"

```
SQL>        SELECT       ename, TO_CHAR(hiredate,'MM/YY')
  2         FROM         emp
  3         WHERE        ename = 'KING';

ENAME      TO_CHAR
---------- -----
KING       11/81
```

4.5.4. Formatage de la date

Formats	Définition
YYYY	année sur quatre chiffres
YEAR	année écrite en toutes lettres
MM	le mois sur deux caractères
MONTH	le mois en toutes lettres
DY	le jour de la semaine en trois lettres
DAY	le jour de la semaine en toutes lettres
WW ou W	semaine de l'année ou du mois
DDD	jour de l'année
DD	jour du mois
D	jour de la semaine
J	le nombre de jour depuis le 31 décembre 4713 BC
Q	quart de l'année
MON	le mois sur trois caractères
RM	numéro romain du mois
CC	siècle
fmDAY	supprime les espaces
AM ou PM	indicateur de méridien
HH ou HH12 ou HH24	heure du jour
MI	minutes (0-59)
SS	secondes (0-59)

SSSS	secondes (0-86399)
TH	nombre ordinal
SP	nombre écrit en toutes lettres
SPTH ou THSP	nombre ordinal écrit en toutes lettres

Exemple:

```
SQL> SELECT       ename,
  2               TO_CHAR(hiredate, 'fmDD Month YYY') HIREDATE
  3  FROM         emp

ENAME      HIREDATE
---------- ----------------
SMITH      17 Décembre 980
ALLEN      20 Février 981
WARD       22 Février 981
JONES      2 Avril 981
MARTIN     28 Septembre 981
BLAKE      1 Mai 981
…
14 ligne(s) sélectionnée(s).
```

4.5.5. Utilisation de la fonction TO_CHAR avec des nombres

Voici le tableau avec les différents formats de conversion de la fonction **TO_CHAR** avec des nombres.

Elément	Description	Exemple	Résultat
9	Le nombre de 9 définie la longueur maximale à l'affichage	999999	1234
0	Force à mettre les zéros au début	099999	001234
$	Le signe dollar ($) flottant	$999999	$1234
L	Devise locale	L999999	FF1234
.	Position du point des décimales	999999.99	1234.00
,	Séparateur des milliers	999,999	1,234
MI	Ajoute signe « moins » à droite	999999MI	1234-
PR	Les nombres négatifs sont entre parenthèses	999999PR	<1234>
EEEE	Notation scientifique	99.999EEEE	$1.234^{E}+03$
V	Multiplie par multiple de 10n (n = nombre de chiffres après V)	9999V99	123400
B	Affiche les valeurs nulles comme un espace	B9999.99	1234.00

Le serveur Oracle affiche une chaîne de caractères composée de signes # à la place du résultat si la valeur fournie en paramètres excède le nombre de digits du format, le serveur Oracle arrondi la valeur décimale au nombre d'espaces décimal du format.

Exemple:

```
SQL>        SELECT       TO_CHAR (sal, '$99,999') SALARY
  2         FROM         emp
  3         WHERE        ename = 'KING'

SALARY
--------
  $5,000
```

4.5.6. Utilisation des fonctions TO_NUMBER et TO_DATE

Les fonctions **TO_NUMBER** et **TO_DATE** sont utilisées pour convertir une chaîne de caractère en nombre ou en date.

TO_NUMBER *(char[, 'format_model'])*
TO_DATE *(char[, 'format_model'])*

La fonction **TO_DATE** possède le modificateur **fx** (format exact) qui vérifie la concordance entre la chaîne de caractère et le format date.

Exemple :

Chaîne de caractère	Format date	Resultat
'15-JAN-1998'	'fxDD-MON-YYYY'	Pas d'erreur
'1-JAN-1998'	'fxDD-MON-YYYY'	Erreur

4.5.7. Le format Date RR

Pour compter une différence sur les siècles, il faut utiliser le format de date **RR** plutôt que YY pour formater l'année des dates.

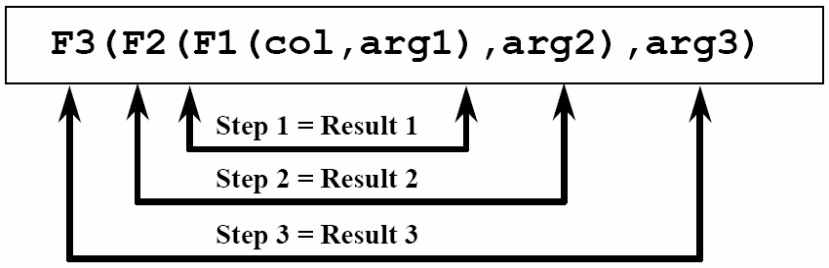
Année actuelle	Date donnée	Format RR	Format YY
1995	22-OCT-95	1995	1995
1995	22-OCT-17	2017	1917
2001	22-OCT-17	2017	2017
2001	22-OCT-95	1995	2095

Tableau d'utilisation du format RR :

		Si l'année spécifiée est :	
		0-49	50-99
Si l'année courante est :	0-49	La date retournée est du siècle courant.	La date retournée est dans le siècle précédant le siècle courant.
	50-99	La date retournée est dans le siècle précédant le siècle courant.	La date retournée est du siècle courant.

4.6. Les fonctions imbriquées

Les fonctions single-row peuvent être imbriquées sur plusieurs niveaux. Les fonctions imbriquées sont évaluées de l'intérieur vers l'extérieur.



Exemple:

```
SQL> SELECT       TO_CHAR (NEXT_DAY(ADD_MONTHS(hiredate,6),
                  'VENDREDI'),'fmDAY, Month DDth, YYYY')
  2  FROM         emp
  3  ORDER BY     hiredate;

TO_CHAR(NEXT_DAY(ADD_MONTHS(HI
------------------------------
VENDREDI, Juin 19TH, 1981
VENDREDI, Août 21ST, 1981
VENDREDI, Août 28TH, 1981
```

Explication : La requête affiche les vendredi six mois après la date d'embauche des employées.

4.7.Les fonctions générales

4.7.1. La fonction NVL

La fonction **NVL** permet de substituer (convertir) les valeurs nulles d'une colonne par une valeur choisie.

NVL *(expr1, expr2)*

Conversion de types de données variés :

Type de données	Exemple de conversion
NUMBER	NVL(number_column,9)
DATE	NVL(date_column,'01-JAN-95')
CHAR ou VARCHAR2	NVL(character_column,'unavailable')

Exemple :

```
SQL>       SELECT ename, NVL(TO_CHAR(comm),'Pas de commission') comm
  2        FROM   emp;

ENAME      COMM
---------- ----------------------------------------
SMITH      3000
ALLEN      500
WARD       500
JONES      Pas de commission
MARTIN     1400
BLAKE      Pas de commission
CLARK      Pas de commission
SCOTT      Pas de commission
KING       Pas de commission
…
14 ligne(s) sélectionnée(s).
```

Explication : La fonction **TO_CHAR** convertit le type de la colonne *comm* au type de données **CHAR**. La fonction **NVL** remplace les valeurs nulles de la colonne *comm* par la chaîne de caractères "pas de commission".

4.7.2. La fonction NVL2

NVL2 *(expr1, expr2, expr3)*

La fonction **NVL2** examine la première expression, si elle est non nulle, la fonction **NVL2** affiche la deuxième expression. Par contre si la première expression est nulle **NVL2** affiche la troisième expression.

Exemple:

```
SQL>       SELECT ename, sal, comm, NVL2(comm,'SAL+COMM', 'SAL')
  2        FROM   emp;

ENAME             SAL       COMM NVL2(COM
---------- ---------- ---------- --------
SMITH              800       3000 SAL+COMM
ALLEN             1600        500 SAL+COMM
WARD              1250        500 SAL+COMM
JONES             2975            SAL
MARTIN            1250       1400 SAL+COMM
BLAKE             2850            SAL
…
14 ligne(s) sélectionnée(s).
```

4.7.3. La fonction NULLIF

La fonction **NULLIF** compare deux expressions, si elles sont identiques la fonction retourne NULL, dans le cas contraire la fonction retourne la première expression. On ne peut pas spécifier la chaîne de caractère NULL pour la première expression.

NULLIF *(expr1, expr2)*

Exemple:

```
SQL> SELECT       ename, LENGTH(ename), job, LENGTH(job),
  2               NULLIF (LENGTH(ename),LENGTH(job)) RESULTAT
  3  FROM         emp;

ENAME      LENGTH(ENAME) JOB       LENGTH(JOB)   RESULTAT
---------- ------------- --------- ----------- ----------
SMITH                  5 CLERK               5
ALLEN                  5 SALESMAN            8          5
WARD                   4 SALESMAN            8          4
JONES                  5 MANAGER             7          5
MARTIN                 6 SALESMAN            8          6
BLAKE                  5 MANAGER             7          5
…
14 ligne(s) sélectionnée(s).
```

4.7.4. La fonction COALESCE

La fonction **COALESCE** retourne la première expression non nulle rencontré dans la liste.

COALESCE *(expr1, expr2, … exprn)*

Exemple :

```
SQL>       SELECT       ename, COALESCE(comm, sal,10)RESULTAT
  2        FROM         emp
  3        ORDER BY     comm;

ENAME                  RESULTAT
---------- --------------------
TURNER                        0
FORD                        300
ALLEN                       500
WARD                        500
MARTIN                     1400
SMITH                      3000
ADAMS                      3000
MAGNUM                       10
FUN                          10
KING                       5000
…
14 ligne(s) sélectionnée(s).
```

Explication : Le résultat pour TURNER est égal à 0 car sa commission (première valeur rencontré) est égale à 0 donc non nulle. Pour MARTIN le résultat est 3000 car sa commission est nulle donc la fonction **COALESCE** prend en compte l'expression suivante. Pour MAGNUM le résultat est 10 car il ne possède ni commission ni salaire.

4.8. Les expressions conditionnelles

4.8.1. Expression CASE

L'expression **CASE** permet d'utiliser la structure de **IF-THEN-ELSE** sans recourir à l'utilisation des procédures.

CASE expr **WHEN** *comparison_expr1* **THEN** *return_expr1*
[**WHEN** *comparison_expr2* **THEN** *return_expr2*
WHEN *comparison_expr3* **THEN** *return_expr3*
ELSE *else_expr*]
END

Les expressions (*expr*, *comparison_expr* et *return_expr*) doivent obligatoirement être du même type : **CHAR**, **VARCHAR2**, etc.…

Lors d'une simple expression **CASE**, Oracle chercher le couple **WHEN**…**THEN** pour lequel *expr* est égal à *comparison_expr* et retourne *return_expr*. Si **WHEN…THEN** ne remplit pas la bonne condition la clause **ELSE** est prévue pour ce cas. Vous ne pouvez pas spécifié la chaîne de caractère **NULL** pour *return_expr* et *else_expr*.

Exemple:

```
SQL>       SELECT       ename, job, sal,
  2                     CASE job
  3                     WHEN 'CLERK' THEN 1.10*sal
  4                     WHEN'PRESIDENT' THEN 1.20*sal
  5                     ELSE sal
  6                     END "Nouveau salaire"
  7        FROM         emp ;

ENAME      JOB              SAL Nouveau salaire
---------- --------- ---------- ---------------
SMITH      CLERK             800             880
ALLEN      SALESMAN         1600            1600
WARD       SALESMAN         1250            1250
JONES      MANAGER          2975            2975
MARTIN     SALESMAN         1250            1250
BLAKE      MANAGER          2850            2850
…
14 ligne(s) sélectionnée(s).
```

4.8.2. Fonction DECODE

La fonction **DECODE** peut faire le travail d'un ordre **IF-THEN-ELSE** ou d'un ordre **CASE**.

DECODE *(column_name | expr , search1*
[, search2, result2,]
[search3, result3,]
[…,…]
[résultat par default])

Exemple:

```
SQL>       SELECT       ename, job, sal,
  2                     DECODE (job,'CLERK',1.10*sal,
  3                     'PRESIDENT',1.20*sal,
  4                     sal)"Nouveau salaire"
  6        FROM         emp;

ENAME      JOB              SAL Nouveau salaire
---------- --------- ---------- ---------------
SMITH      CLERK             800             880
ALLEN      SALESMAN         1600            1600
WARD       SALESMAN         1250            1250
JONES      MANAGER          2975            2975
MARTIN     SALESMAN         1250            1250
BLAKE      MANAGER          2850            2850
…
14 ligne(s) sélectionnée(s).
```

Explication : Cette requête affiche le salaire de l'employé multiplié par une valeur qui dépend de sa fonction. Cette requête est équivalente à l'ordre **IF-THEN-ELSE** suivant :

```
IF job = 'CLERK' THEN nouveau_salaire = sal * 1,10 ;
ELSIF job = 'PRESIDENT' THEN nouveau_salaire = sal * 1,20 ;
ELSE nouveau_salaire = sal ;
```