Module n°1
Ordres SELECT :
les bases

ORACLE

Table des matières

1 CONCEPTS D'UNE BASE DE DONNEE RELATIONNELLE

1.1 Principes Relationnels

1.1.1 Concepts d'une base de données relationnelle

Toute organisation a besoin d'informations.

Exemples :

- Une bibliothèque maintient une liste de membres, de livres...
- Une société maintient des informations sur ses salariés, ses départements…

Ces informations sont appelées des données.

Elles peuvent être stockées sur différents types de support et format comme : des fiches papiers, des tableaux Excel...ou des bases de données.

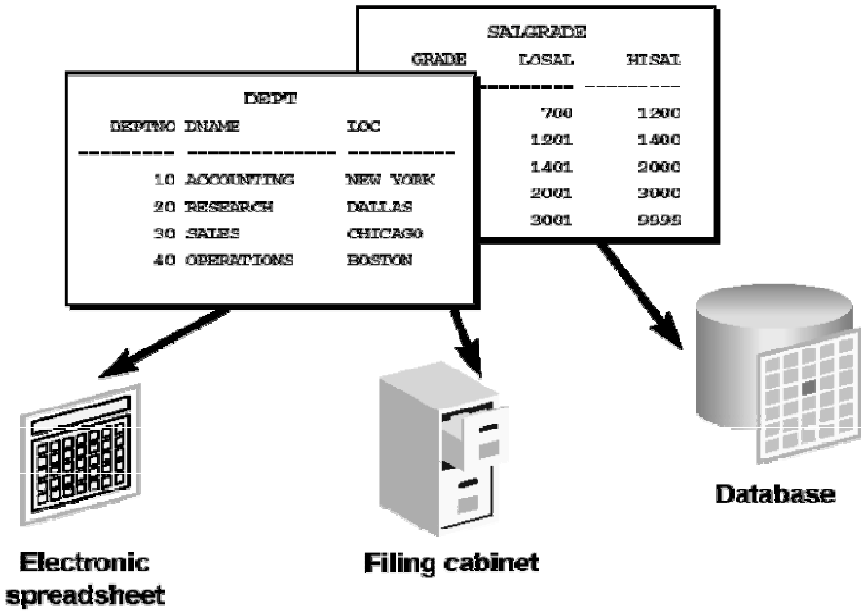


Figure 1 : Stocker des données sur différents supports

Une base de données est une collection organisée d'informations. Elle permet un accès et une administration plus facile des données.
Pour administrer efficacement une base de données, il est nécessaire de posséder un système de management de base de données (DBMS : **D**ata Base **M**anagement **S**ystem). Un DBMS permet de stocker, modifier, supprimer et retrouver les données dans une base de données tout en assurant leur cohérence.
Les objectifs d'un DBMS sont les suivants :

- Indépendance physique
- Indépendance logique
- Manipulation des données par des non informaticiens
- Efficacité des données
- Administration cohérente des données
- Partageabilité des données
- Sécurité des données

Il existe quatre types de bases de données :

- hiérarchique
- en réseau
- relationnelle
- objet relationnelle (la plus récente)

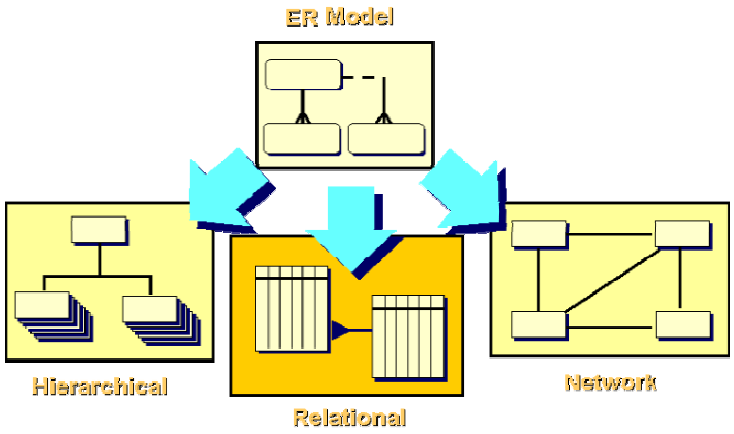


Figure 2 : Les quatre types de bases de données

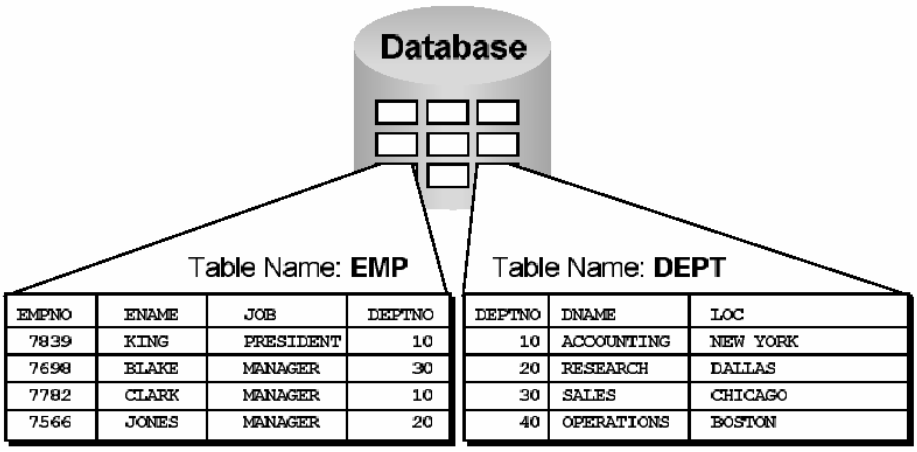
Ce cours portera essentiellement sur les bases de données relationnelle et objet-relationnelle.
Une base de données relationnelle (RDB relational database) est une collection de relations ou de tables à deux dimensions dans lesquelles sont stockées les informations.

Le modèle relationnel a été proposé par le Dr E.F Codd en juin 1970 dans un article appelé " A relational model of data for large shared data banks".
Le modèle relationnel est constitué de trois composants :

- une collection d'objets ou de relations (appelées aussi tables),
- un groupe d'opérateurs pour agir sur les tables,
- des règles d'intégrité des données.

Une contrainte d'intégrité est une assertion logique devant être vérifiée en permanence pour maintenir la base de données dans un état fiable.

Un RDBMS (Relational Data Base Management System) est un DBMS gérant une collection de tables (appelées aussi relations) dans lesquelles sont stockées et organisées les données.



EMPNO	ENAME	JOB	DEPTNO
7839	KING	PRESIDENT	10
7698	BLAKE	MANAGER	30
7782	CLARK	MANAGER	10
7566	JONES	MANAGER	20

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

Figure 3 : Base de données relationnelle

Le SQL (**S**tructured **Q**uery **L**anguage) est un langage de programmation Standard ANSI utilisé pour récupérer et manipuler les données dans une base de données relationnelle.

Trois concepts importants :

- Le stockage des données se fait dans des tables de la base de données relationnelle.
- L'accès aux données est possible grâce à des requêtes SQL qui interrogeront la base de données au sujet des informations que l'utilisateur a demandé.
- La récupération des données ne nécessite pas une connaissance de l'organisation physique des données pour l'utilisateur.

1.1.2 Terminologie d'une base de données relationnelle

Une base de données relationnelle contient une ou plusieurs tables (ou relations) bidimensionnelles qui correspondent au moyen de stockage le plus basique. Une table contiendra des informations qui pourront être retrouvées à tout moment.

Structure générale d'une table :

Un ligne, un enregistrement ou un tuple (row) :
Ensemble de caractéristiques définissant une occurrence de l'objet table. (Correspond dans table EMP aux informations relatives à un employé). L'ordre des lignes stockées dans la table est sans importance car un ordre de tri peut être spécifié lors d'une requête SQL.

Une colonne (column) :
Ensemble de données relatives à une information caractéristique (dans la table EMP, tous les employés ont un nom, un salaire...). Une colonne peut contenir une clé primaire, une clé étrangère ou des valeurs simples.

Une clé primaire (Primary Key PK) :
Sous-ensemble minimal d'attributs d'une table permettant d'identifier un enregistrement de manière unique. Cette valeur ne peut donc pas être nulle ou double. (En règle générale, cette valeur n'est pas modifiée par l'utilisateur).

Une clé étrangère (Foreign Key FK) :
Sous-ensemble d'attributs référençant la clé d'une autre table. Une clé étrangère traduit une relation entre deux tables.

Un champ (field) :
Intersection d'une ligne avec une colonne. Un champ ne peut contenir qu'une seule valeur . Sa valeur peut être nulle (elle ne contiendra aucune valeur).

Les valeurs d'une clé étrangère correspondent aux valeurs d'une clé primaire dans une autre colonne d'une table.

Les colonnes qui ne sont ni clé primaire ni clé étrangère contiennent des valeurs qui ne font pas référence à des valeurs d'une autre colonne d'une table.

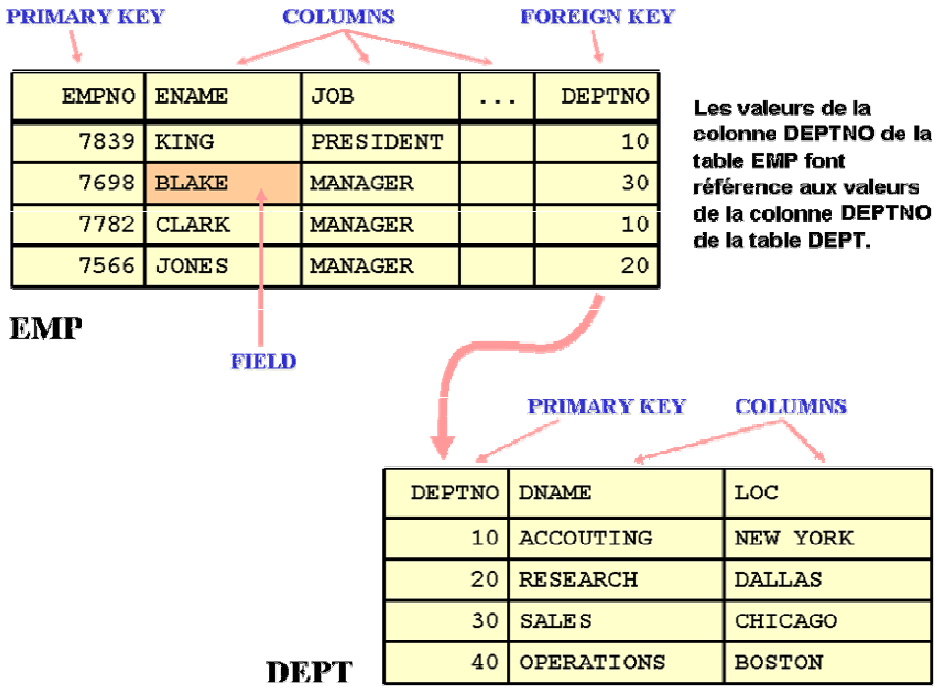


Figure 4 : Structure générale des tables

1.2 Conception d'une base de données

1.2.1 Cycle de développement d'un système d'information

Du concept initial à la production, une base de données peut être développée en se basant sur les étapes de développement d'un système d'information. Cette approche systématique permet de transformer (ou retranscrire) les informations nécessaires d'une entreprise en une base de donnée opérationnelle.

Le développement d'un système d'information compte 5 étapes :

- La stratégie et l'analyse :
  Il s'agit d'étudier et d'analyser les besoins de l'entreprise. Les managers et les utilisateurs doivent être interrogés afin de connaître les informations nécessaires et de pouvoir ainsi prévoir les spécifications futures du système. Le modèle du système est créé en transformant les informations et les règles de l'entreprise en une représentation graphique des besoins de l'entreprise.

- La conception (design) :
  Les informations de l'étape précédente serviront à modéliser la base de données. La modélisation finale est un diagramme relationnel représentant les détails de la structure de la base de données.

- La construction et la documentation :
  Il s'agit de la construction d'un prototype du système (création d'une base de test et des objets requis), de la rédaction d'une documentation d'aide explicative destinée aux utilisateurs finaux et d'un manuel des opérations effectuées sur le système.

- La transition :
  Le prototype est affiné afin qu'il corresponde au plus près aux exigences des utilisateurs. La conversion des données existantes est réalisée ainsi que toutes les modifications nécessaires sur le prototype.

- La production :
  Le système est fournit aux utilisateurs, et une analyse de ses performances sera effectuée afin de l'optimiser.

1.2.2 Les composants du modèle Entité-Relation

La représentation visuelle contribue dans une large mesure à l'établissement d'un dialogue constructif entre tous les partenaires qui collaborent pour concevoir ensemble un système d'information.

Le modèle ER est une description du système d'information utilisant un formalisme de représentation précis, simple et rigoureux, pour la description des données. Ce formalisme est normalisé au plan international par l'ISO (International Standard Organisation) sous le nom de : modèle « Entité Relation ».

Les modèles sont utilisés pour communiquer, classer par catégories, décrire, spécifier, enquêter, élaborer, analyser, imiter. Le but est de produire un modèle qui s'adapte à toutes ces utilisations, qui peut être compris par les utilisateurs finaux et qui contient suffisamment de détails pour permettre au développeur de construire la base de données.

Le modèle entité-relation (Entity Relationship ER) est utilisé couramment par les concepteurs de base de données pendant la phase de stratégie et analyse.

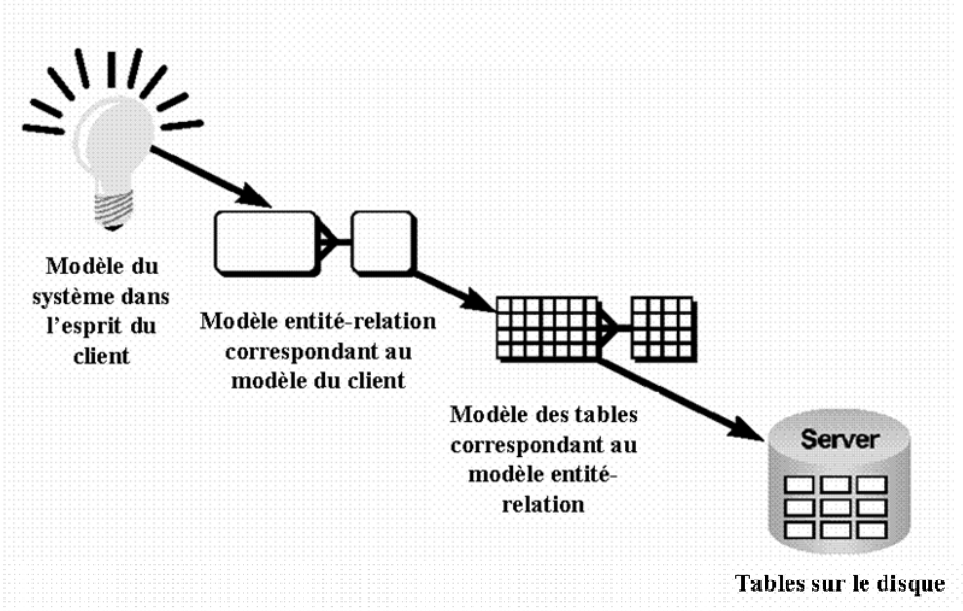


Figure 5 : Les modèles de données

Le modèle ER est constitué de trois composantes :

- Les entités :
  Une entité (ou objet) est un objet pourvu d'une existence propre et conforme aux choix de gestion. Exemple : employé, département, ordres.

- Les attributs :
  Un attribut (ou propriété) est une donnée élémentaire décrivant ou qualifiant une entité ou une relation entre objets. Un attribut peut être obligatoire ou optionnel. Cette caractéristique est appelée "optionalité de l'attribut" (optionnality). Exemple : nom de l'employé, salaire de l'employé, commission de l'employé.

- Les relations :

Une relation est une association nommée entre deux ou plusieurs entités représentant l'optionalité et le degrés de la relation. Il y a 3 types de degrés :

- 1-1 (un à un)
- 1-n (un à plusieurs)
- n-n (plusieurs à plusieurs)

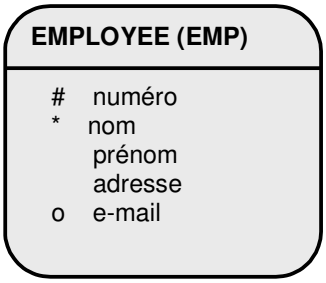
La combinaison de ces composants dans le modèle ER aboutie à une carte graphique claire et précise de la base de données permettant de définir l'éventail des informations requises.

1.2.3 Les conventions du modèle Entité-Relation

Une entité est représentée par un rectangle avec les bords arrondis. Le nom de cette entité doit être singulier, unique et noté en majuscule. Les synonymes ou raccourcis du nom de l'entité apparaîtront en majuscule dans des parenthèses.

Un attribut est représenté par un nom singulier en minuscule. Un attribut obligatoire sera précédé du caractère " ***** " et ne pourra pas contenir une valeur nulle. Un attribut optionnel sera précédé du caractère " **o** ". Un attribut unique (unique identifier UID) est un attribut qui distingue de manière unique chaque occurrence de l'entité. On désigne ce UID en le faisant précédé du caractère " **#** ". Si on doit rajouter un second UID le fera précédé des caractères " **(#)** ".

Exemple :



Une relation est représentée par une ligne reliant les deux entités. Chaque extrémité de la relation possède un nom, une optionalité et un degré.

Le nom de la relation décris l'association entre les deux entités. Le nom doit être un verbe au participe passé.

Le type de la ligne de la relation représente le caractère optionnel de la relation. Une ligne pleine signifie que la relation est obligatoire (exemple : un employé **doit être assigné à** un département), alors qu'une ligne pointillée signifie que la relation est optionnelle (exemple : un département **peut être assigné à** un employé). Une ligne peut être à la fois pleine et pointillée, dans ce cas la relation est obligatoire pour une entité et optionnelle pour l'autre.

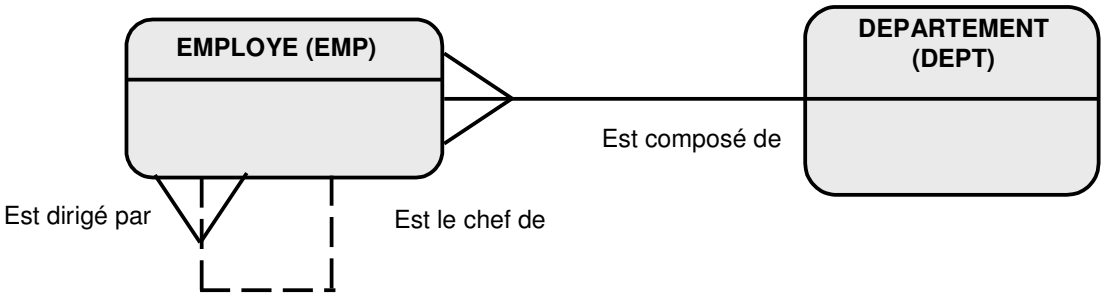
Le degré de la relation est représenté par :

- 1-1 : une ligne simple.
- 1-n : une ligne simple finissant à l'extrémité n par le symbole composé de trois lignes (three line symbole, crow's foot).
- n-n : une ligne simple finissant à chaque extrémité par le symbole composé de trois lignes.

Exemple :

Un employé doit **être assigné à un** département, ce qui signifie que la relation « est assigné à » entre les entités EMPLOYE et DEPARTEMENT est obligatoire et de degré 1 pour l'entité EMPLOYE.
Un département **peut être composé d'un ou plusieurs** employé, ce qui signifie que la relation « est composé de » entre les entités EMPLOYE et DEPARTEMENT est optionnelle et de degré n pour l'entité DEPARTEMENT.

Un employé **peut être dirigé par** un autre employé, ce qui signifie que la relation réflexive « est dirigé par » sur l'entité EMPLOYE est optionnelle et de degré n du côté de l'entité EMPLOYE considéré comme l'employé dirigé.
Un employé **peut être le manager** d'un ou plusieurs employés, ce qui signifie que la relation réflexive « est le chef de » sur l'entité EMPLOYE est optionnelle et de degré n du coté de l'entité EMPLOYE considéré comme le manager.



On lit le modèle en passant d'une entité à l'autre avec la syntaxe suivante :

Entité source { peut | doit } nom de la relation {un et un seul | un à plusieurs} entité destination

1.3 ORDBMS (Objet Relationship Database Management System)

1.3.1 Définition d'un ORDBMS

Un ORDBMS est un système de gestion pour les bases de données relationnelle-objet.

Différence entre RDBMS et ORDBMS :
Un ORDBMS permet de représenter l'information sous forme objet et permet aux objets de stocker des données et des processus.

Points communs entre DBDMS et ORDBMS :
Ils administrent le stockage et la définition des données. Ils contrôlent l'accès aux données. Ils fournissent un ensemble de tables, vues et autres objets de la base de données qui permettent une lecture seule sur la base de données.

Oracle 8I est une base de données relationnelle objet, elle possède donc un ORDBMS.

1.3.2 Interactions avec le serveur Oracle

Interaction avec SQL:

SQL (**S**tructured **Q**uery **L**angage) est un langage proche de l'anglais qui permet à des utilisateurs sans notions de développement de récupérer les données dont ils ont besoin.
Le SQL est utilisé pour définir, retrouver et manipuler les données sur un serveur. Le SQL contient de nombreux opérateurs qui permettent de partitionner et de combiner les tables.

Intéraction avec PL/SQL

Le PL/SQL (**P**rocedural **L**anguage / **S**tructured **Q**uery **L**anguage) est un langage procédural qui permet d'étendre les fonctions du SQL.

Intéraction avec SQL*Plus

SQL*Plus est un environnement qui reconnaît et envoie des requêtes à exécuter au serveur.
SQL*Plus possède son propre langage. Il ne permet pas de modifier les données mais seulement de modifier l'environnement de SQL*Plus et le formatage des données pour l'affichage.

1.3.3 Les types d'ordres SQL

Il existe cinq types d'ordre SQL :

- **D**ata **R**etrieval **L**angage (DRL) :
  Ensemble de commandes qui permettent de récupérer les données contenues dans une ou plusieurs table de la base. (Exemple : l'ordre SELECT)

- **D**ata **M**anipulation **L**angage (DML) :
  Ensemble de commandes qui permettent de modifier les données de la base. (Exemple : les ordres INSERT, DELETE, UPDATE)

- **D**ata **D**efinition **L**angage (DDL) :
  Ensemble de commandes qui permettent de modifier la structure de la base. (Exemple : les ordres CREATE, DROP, ALTER, RENAME)

- **T**ransaction **C**ontrol **S**tatement (TCS) :
  Est un ensemble de commandes qui permettent d'administrer les changement effectués par les commandes DML. (Exemple : les commandes COMMIT, ROLLBACK, SAVEPOINT)

- **D**ata **C**ontrol **L**anguage (DCL) :
  Est un ensemble de commandes qui permettent de contrôler les accès utilisateur à la base de données. (Exemple : les ordres GRANT, REVOKE)

2 ECRITURE D'ORDRES SQL BASIQUES

2.1 Ordres SELECT

2.1.1 Les capacités d'un ordre SELECT

L'ordre **SELECT** sert à extraire des données de la base de données.

```
SELECT      what
FROM        where ;
```

Un ordre SELECT est composé de deux clauses :

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

L'ordre SELECT possède trois capacités :

- Sélection : Sélection de une ou plusieurs lignes
- Projection : Sélection de une ou plusieurs colonnes
- Jointure : Sélection de deux colonnes dans deux tables différentes, créant ainsi une relation entre les données des deux colonnes.

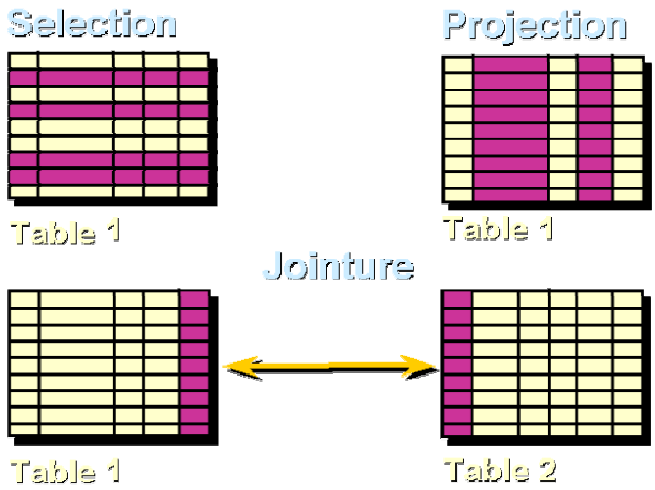


Figure 5 : Sélection, projection et jointure

On exécute une requête SQL en la terminant par les caractères " **;** " ou " / ". (Le caractère " / " permet aussi de re-exécuter l'ordre SQL stocké dans le buffer).

Pour faciliter la relecture 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.

Remarque : Le serveur ne tient pas compte de la casse des mots.

2.1.2 Ecriture d'un ordre SELECT

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

```
SELECT        { * | column, …}
FROM          table ;
```

Le caractère ' * ' placé dans la clause SELECT signifie que toutes les colonnes de toutes les tables de la clause FROM sont sélectionnées.

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

Les données de type numérique sont automatiquement alignées à droite, et les données de type alpha numérique ou date automatiquement alignées à gauche.

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

2.1.3 Sélectionner des colonnes et des enregistrements

Exemple :

```
SQL> SELECT    *
  2  FROM      dept;

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

Æ Toutes les colonnes de la table DEPT sont sélectionnées et affichées.

Exemple :

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

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

4 rows selected.
```

Æ Les colonnes DEPTNO et DNAME de la table DEPT sont sélectionnées et affichées.

2.1.4 Les expressions arithmétiques

Les opérateurs arithmétiques peuvent être utilisés pour effectuer des opérations sur les dates ou les nombres. Les opérateurs arithmétiques disponibles sous Oracle sont les suivants :

Opérateur	Description
+	Additionner
-	Soustraire
*	Multiplier
/	Diviser

Exemple :

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

ENAME            SAL   SAL+300
---------- --------- ---------
KING            5000      5300
BLAKE           2850      3150
CLARK           2450      2750
JONES           2975      3275
MARTIN          1250      1550
ALLEN           1600      1900
...
14 rows selected.
```

Les opérateurs arithmétiques peuvent être utilisés dans toutes les clauses sauf 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
---------- --------- ----------
KING            5000      60100
BLAKE           2850      34300
CLARK           2450      29500
JONES           2975      35800
MARTIN          1250      15100
ALLEN           1600      19300
...
14 rows selected.
```

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)
---------- --------- -----------
KING            5000       61200
BLAKE           2850       35400
CLARK           2450       30600
JONES           2975       36900
MARTIN          1250       16200
...
14 rows selected.
```

Une valeur nulle est une valeur non assignée, inconnue ou inapplicable. 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
---------- --------- -------- --------
KING       PRESIDENT     5000
BLAKE      MANAGER       2850
...
TURNER     SALESMAN      1500         0
```

```
...
14 rows selected.
```

Æ La valeur de la commission des employés KING et BLAKE est nulle tandis que la commission de l'employé TURNER est de 0.

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

1 row selected.
```

Personnaliser les requêtes

2.2.1 Alias de colonne

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

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

L'utilisation des alias de colonne clarifiera et allègera les requêtes SQL lorsque ces dernières se compliqueront. Il est fortement conseillé de les utiliser.

Les alias servent également à "renommer" les colonnes lors de l'affichage.

Si 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 ou bien d'utiliser le mot optionnel **AS**.

SELECT *column1* **AS** *alias1*, *column2* **AS** *alais2*…
FROM *table*;

Exemple :

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

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

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

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

```
NAME
-------------
...
```

Pour respecter la casse de l'alias lorsque celui-ci ne compte qu'un seul mot, il suffit de la placer entre guillemets.

Exemple :

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

Æ Dans ce cas, la casse de l'alias est respectée et il s'affichera comme suit :

Si l'alias est composé de plusieurs mots, il doit être placé entre guillemets.

Exemple :

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

Æ La casse des l'alias est respectée et il s'affichera comme suit :

```
Employee's Name   Annual Salary
----------------- --------------
...
```

2.2.2 Opérateur de concaténation

La combinaison de caractères " || " est utilisée pour concaténer des colonnes ou des chaînes de caractères à d'autres colonnes.

Exemple :

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

Password
---------------
SMITH20
ALLEN30
WARD30
JONES20
MARTIN30
BLAKE30
CLARK10
...
14 rows selected
```

2.2.3 Chaîne de caractères littérale

Des chaînes de caractères peuvent être ajoutées dans une clause SELECT.
Ces chaînes sont constantes et s'affichent pour chaque ligne du résultat. Elles peuvent être du type caractère, nombre ou date. Les chaînes de type caractères et date doivent être inclues entre simple côtes.
Une chaîne de caractère placée dans un ordre SELECT doit être mise entre simples côtes.

Exemple 1 :

```
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
CLARK          travaille dans le département             10
...
14 rows selected
```

Æ Dans ce cas, la chaîne de caractère "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 caractère "travaille dans le département".

Exemple 2 :

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

               Localisation
----------------------------------------
SMITH travaille au département 20
ALLEN travaille au département 30
WARD travaille au département 30
JONES travaille au département 20
MARTIN travaille au département 30
BLAKE travaille au département 30
CLARK travaille au département 10
...
14 rows selected
```

Æ 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

Le mot-clé **DISCTINCT** élimine les doublons dans le résultat de la requête lors de l'affichage.
Un doublon est un enregistrement qui se répète plusieurs fois.
Le mot-clé DISTINCT est utilisé dans la clause SELECT comme suit :

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

Exemple sans l'utilisation du mot clé DISTINCT :

```
SQL> SELECT    deptno
  2  FROM      emp;

   DEPTNO
---------
       10
       30
       10
       20
...
14 rows selected
```

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

Exemple avec l'utilisation du mot-clé DISTINCT :

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

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

```
3 rows selected .
```

Æ 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 SQL*Plus*

2.3.1 Caractéristiques de SQL et SQL*Plus

SQL est un langage qui permet de communiquer avec le serveur Oracle depuis des outils ou des applications. Le SQL Oracle contient des extensions. Quand un ordre SQL est entré, il est stocké dans une partie de la mémoire appelée SQL buffer et y reste jusqu'à l'entrée d'un nouvel ordre.

SQL*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 utilisateurs ayant peu voir aucune expérience en programmation.
- SQL est un langage non procédural.
- SQL réduit le temps pour créer et maintenir les systèmes.
- SQL est un langage proche de l'anglais.

Caractéristiques de SQL*Plus :

- accepte les entrée "ad hoc" des ordres.
- accepte du code SQL provenant de fichiers.
- fournit un éditeur de lignes pour modifier les ordres SQL.
- Contrôle les caractéristiques de l'environnement.
- Formate les résultats d'une requête sous la forme d'un rapport.
- Accède à des bases de données locales et lointaines.

Les différences entre SQL*PLUS et SQL :

SQL	**SQL*Plus**
- un langage pour communiquer avec le Serveur Oracle pour accéder aux données	- reconnaît les ordres SQL et les envoie au serveur Oracle
- un standard ANSI	- une propriété Oracle pour interfacer l'exécution d'ordres SQL
- Les mots clés n'ont pas d'abbréviation	- Les mots clés peuvent avoir des abbréviations
- Les ordres SQL manipulent des données et la définition des tables dans la base de données	- Les commandes SQL*Plus gèrent l'environnement SQL*Plus
- Les ordres SQL sont placés dans le buffer	- Les commandes SQL*Plus ne sont pas placées dans le buffer
- nécessite un caractère de terminaison pour exécuter immédiatement un ordre	- ne nécessite pas de caractère de terminaison pour exécuter une commande immédiatement

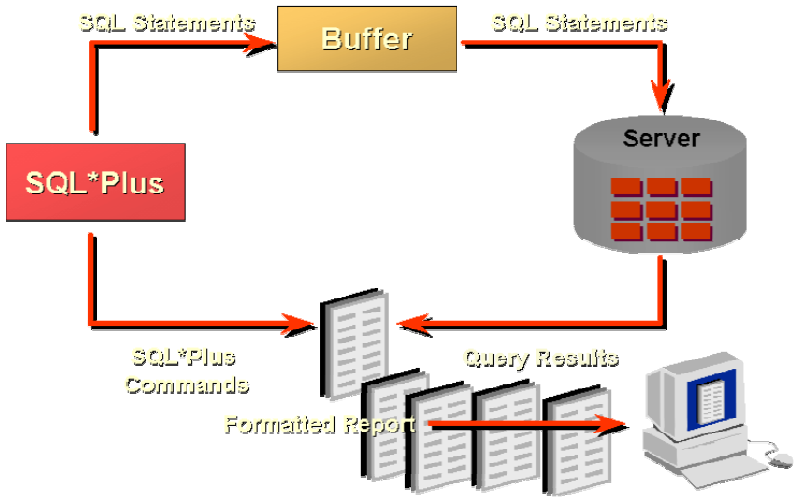


Figure 6 : interaction avec SQL et SQL*Plus

SQL*Plus est un environnement dans lequel on peut :

- Exécuter des ordres SQL pour récupérer, modifier, ajouter et supprimer des données de la base de données.
- 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 SQL*Plus se divise en sept catégories :

- Environnement : affecte le comportement général des ordres SQL pour une session.
- Formatage : formate les résultats d'une requête.
- Manipulation des fichiers : sauve, charge et lance des fichiers script.
- Exécution : envoie les ordres SQL du buffer SQL au serveur Oracle.
- Edition : modifie les ordres SQL stockés dans le buffer.
- Interaction : permet de créer et de transmettre des variables à des ordres SQL, d'imprimer les valeurs des variables et des messages à l'écran.

Le module 1 du cours SQLP traite les commandes d'édition et les commandes manipulant les fichiers, tandis que le module 2 du cours SQLP traite des commandes de formatage, d'interaction et d'environnement.

2.3.2 Se connecter à SQL*Plus

Pour se logger au serveur, il suffit de saisir son nom d'utilisateur (loggin), son mot de passe (password) et la chaîne d'hôte (nom de service) dans la boite de dialogue qui s'ouvre automatiquement au lancement de SQL*PLUS.

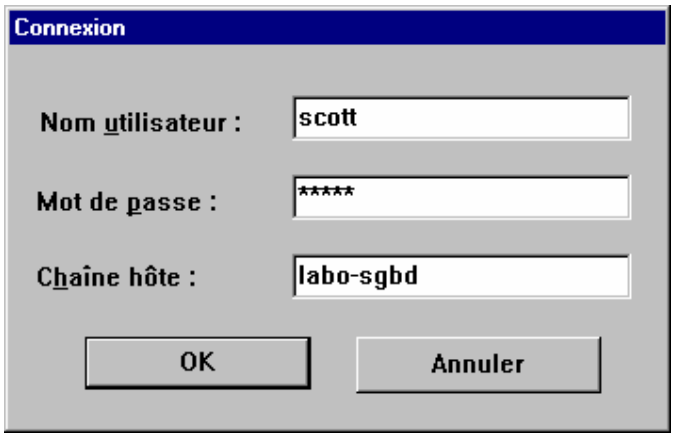


Figure 7 : Boîte de dialogue de connexion à la base de données

Pour ouvrir une session SQL*Plus en ligne de commande sous DOS :
sqlplus *loggin*/*password*[@*host_string*]

Pour ouvrir une session SQL*Plus en ligne de commande sous SQL*Plus :
CONNECT *loggin*/*password*[@*host_string*]

2.3.3 La commande DESCRIBE de SQL*Plus

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  dept

Name              Null?    Type
----------------- -------- ------------
DEPTNO            NOT NULL NUMBER(2)
DNAME                      VARCHAR2(14)
LOC                        VARCHAR2(13)
```

La commande DESCRIBE affiche, pour chaque colonne de la table, 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 :
- NUMBER : chiffre
- CHAR : chaîne de caractères
- VARCHAR2 : chaîne de caractères de longueur variable
- DATE

2.3.4 Les commandes d'édition de SQL*Plus

Les commandes d'édition de SQL*Plus permettent de manipuler le contenu du buffer.

A[**PPEND**] *text*	Permet de rajouter à la dernière ligne du buffer la chaîne de caractères *text*.

Exemple :

```
SQL> SELECT    ename, deptno,
```

Æ Le buffer contient : SELECT ename, deptno,

```
SQL> A job
```

Æ 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*.
C[**HANGE**] / *old* /	Permet de supprimer la chaîne de caractères *old* de la ligne courante.

Exemple :

```
SQL> SELECT    ename, deptno, job
```

Æ Le buffer contient : SELECT ename, deptno, job

```
SQL> C/job/sal
```

Æ Le buffer contient : SELECT ename, deptno, sal

```
SQL> C/, sal
```

Æ Le buffer contient : SELECT ename, deptno

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

Exemple :

```
SQL> SELECT    ename, deptno, sal
```

Æ Le buffer contient : SELECT ename, deptno, sal

```
SQL> I FROM emp
```

Æ Le buffer contient : SELECT ename, deptno, sal
FROM emp

L[**IST**]	Permet d'afficher le contenu du buffer.
L[**IST**] *n*	Permet d'afficher la ligne *n* du buffer.
L[**IST**] *m n*	Permet d'afficher les lignes *m* à *n*.

Exemple :

```
SQL> L
```

```
  1  SELECT    ename, deptno, sal
  2* FROM      emp
```

R[**UN**]	Permet d'exécuter le contenu du buffer. La commande affiche la requête suivie du résultat de la requête.

n	Permet de spécifier la ligne courante.
n text	Permet de remplacer la ligne *n* avec *text.*
0 text	Permet d'insérer une ligne contenant text avant la ligne 1.

Exemple :

```
SQL> 1
  1  SELECT    ename, deptno, sal
```

```
SQL> 1 SELECT  mgr
```

Æ Le buffer contient : SELECT mgr
FROM emp

DEL	Permet d'effacer la ligne courante du buffer.
DEL *n*	Permet d'effacer la ligne *n*.
DEL *m n*	Permet d'effacer les lignes *m* à *n*.

CL[**EAR BUFF**[**ER**]	Permet d'effacer le contenu du buffer.

2.3.5 Les commandes SQL*Plus manipulant les fichiers

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

SAV[**E**] *filename*[*.ext*] [**RE**[**PLACE**] | **APP**[**END**]]	Permet de sauvegarder le contenu du buffer dans un fichier. APPEND spécifie que le contenu du buffer doit être ajouté au contenu d'un fichier existant. REPLACE spécifie que le fichier existant sera écrasé.

Exemple :

```
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*.

Exemple :

```
SQL> GET all_emp
```

STA[**RT**] *filename*[*.ext*]	Permet d'exécuter le contenu d'un fichier.

@*filename*[.*ext*]	Permet d'exécuter le contenu d'un fichier (équivalent à START).

Exemple :

```
SQL> STA all_emp

SQL> @all_emp
```

ED[**IT**]	Permet d'invoquer l'éditeur et sauver le contenu du buffer dans un fichier nommé afiedt.buf
ED[**IT**] [*filename*[.*ext*]]	Permet d'éditer le contenu d'un fichier à l'aide d'un éditeur de texte.

Exemple :

```
SQL> ED all_emp
```

Æ cet commande édite le fichier 'all_emp' dans le Bloc Note.

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.

3 RESTREINDRE ET EXTRAIRE DES DONNEES

3.1 Restreindre des enregistrements

3.1.1 La clause WHERE

La clause WHERE restreint la requête aux enregistrements qui respectent sa ou ses conditions.
La clause WHERE correspond à une sélection : elle restreint les enregistrements (ou lignes).

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

value peut être une colonne, une constante ou une liste de valeurs.

Exemple :

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

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

4 rows selected
```

Æ 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 Les règles de conduite de la clause WHERE

Les chaînes de caractères sont sensibles à la casse.
Les dates sont sensibles au format.
Les chaînes de caractères et les dates doivent être placés entre simples côtes.

Exemples d'utilisation de la clause WHERE avec différents types de données :

Colonne de type NUMBER :

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

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

Colonne de type CHAR ou VARCHAR :

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

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

Colonne de type DATE :

```
SQL> SELECT    ename, job, deptno
  2  FROM      emp
  3  WHERE     hiredate = '01-JAN-99';
```

Æ 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 Les opérateurs de comparaison

3.2.1 Expressions de comparaison

Les opérateurs de comparaison disponibles sous Oracle sont les suivants :

Opérateur	Significations
=	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 1 :

```
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 rows selected
```

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

Exemple 2 :

```
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 rows selected
```

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

Exemple 3 :

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

  DEPTNO  LOC
--------  ------------
      20  DALLAS
      30  CHICAGO
      40  BOSTON

3 rows selected
```

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

3.2.2 L'opérateur BETWEEN

L'opérateur **BETWEEN..AND** 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. Dans le cas où il s'agirait 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 : (a >= lower_limit) AND (a <= higher_limit)

L'utilisation de l'opérateur BETWEEN..AND n'apporte aucun bénéfices de performance.

Exemple 1 :

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

ENAME                SAL
------------  ----------
WARD               1250
MARTIN             1250
TURNER             1500
ADAMS              1100
MILLER             1300

5 rows selected
```

Æ Cette requête affiche les employés dont le salaire est situé dans la tranche $1000 à $1500.

Exemple 2 :

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

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

3 rows selected
```

Æ 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 3 :

```
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

5 rows selected.
```

Æ Cette requête affiche les employés embauchés entre le 1er janvier 1981 et le 31 juillet 1981. La date peut être écrite sous différent format dans l'expression de l'opérateur BETWEEN (par exemple sous la forme 01-01-81).

3.2.3 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

5 rows selected
```

Æ Cette requête affiche les employés dont le numéro de manager est 7902 ou 7566 ou 7788.

3.2.4 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*' ;

Le symbole '_' représente un seul caractère quelconque.
Le symbole '%' représente une série de zéros ou de caractères..

L'opérateur LIKE peut être utilisé comme un raccourci de plusieurs conditions BETWEEN..AND

Exemple :

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

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

3 rows selected
```

Æ 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*' ;

Exemples :

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

ENAME
-----------
SGBD_LILI
SGBD_HELYOS
SGBD_WEAN
SGBD-ORACLE

4 rows selected
```

Æ 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_" :

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

ENAME
-----------
SGBD_LILI
SGBD_HELYOS
SGBD_WEAN

3 row selected
```

3.2.5 L'opérateur IS NULL

L'opérateur **IS NULL** permet d'afficher les enregistrements dont certains champs contiennent des valeurs nulle. 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

1 row selected
```

Æ 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

L'opérateur AND retourne TRUE si les deux conditions de l'expression logique sont toutes les deux TRUE.

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

2 rows selected.
```

Æ 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

L'opérateur OR retourne TRUE si au moins une des deux conditions de l'expression logique est TRUE.

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
...
14 rows selected.
```

Æ 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

L'opérateur NOT retourne TRUE si la condition de l'expression logique est FALSE.

L'opérateur NOT ne peut être utilisé qu'avec les opérateurs de comparaison IN, LIKE, BETWEEN et IS NULL.

Placement :

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
--------------------  ---------
ALLEN                       300
WARD                        500
MARTIN                     1400
TURNER                        0

4 rows selected.
```

Æ Cette requête affiche la liste des employés touchant une commission.

3.3.4 Ordres d'évaluation des opérateurs

Ordre d'évaluation des opérateurs :
Evaluer en premier : opérateurs de comparaison
Evaluer en second : opérateurs logiques (dans cet ordre NOT, AND, OR)

Les conditions entre parenthèses sont évaluées en premier.

Exemple :

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

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

2 rows selected.
```

3.4 Ordonner les enregistrements

3.4.1 La Clause ORDER BY

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

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

L'ordre par défaut est croissant.

Exemple :

```
SQL> SELECT    ename, job, deptno, hiredate
```

```
  2  FROM      emp
  3  ORDER BY  hiredate;

ENAME      JOB          DEPTNO HIREDATE
---------- --------- --------- ---------
SMITH      CLERK            20 17-DEC-80
ALLEN      SALESMAN         30 20-FEB-81
...
14 rows selected.
```

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 dans l'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-JAN-83
SCOTT      ANALYST          20 09-DEC-82
MILLER     CLERK            10 23-JAN-82
JAMES      CLERK            30 03-DEC-81
FORD       ANALYST          20 03-DEC-81
KING       PRESIDENT        10 17-NOV-81
MARTIN     SALESMAN         30 28-SEP-81
...
```

3.4.3 Trier sur un alias de colonne ou une expression

Les enregistrements peuvent être triés sur un alias de colonne ou une expression.

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

SELECT *column1*, *column2*, *expression*
FROM *table*
ORDER BY *expression* ;

Exemple1 :

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


    EMPNO ENAME         ANNSAL
--------- ---------- ---------
```

```
      7369 SMITH           9600
      7900 JAMES          11400
      7876 ADAMS          13200
      7654 MARTIN         15000
...
14 rows selected.
```

Exemple 2 :

```
SQL> SELECT    ename "Employee's Name", sal
  2  FROM      emp
  3  ORDER BY  "Employee's Name" ;

Employee's Name        SAL
--------------- ----------
ADAMS                 1100
BLAKE                 2850
CLARK                 2450
FORD                  3000
JAMES                  950
...
14 rows selected.
```

Exemple 3 :

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

ENAME             SAL     SAL*12
---------- ---------- ----------
SMITH              800       9600
JAMES              950      11400
ADAMS             1100      13200
WARD              1250      15000
MARTIN            1250      15000
MILLER            1300      15600
...
14 rows selected.
```

Les enregistrements peuvent être triés sur un numéro de colonne.

Exemple :

```
SQL> SELECT    ename "Employee's Name", sal
  2  FROM      emp
  3  ORDER BY  1 ;
```

Æ Cette requête retourne le même résultat que l'exemple 2 : les enregistrements sont triés dans l'ordre croissant sur le nom des l'employés.

3.4.4 Trier sur 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.

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
FORD              20      3000
...
14 rows selected.
```

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

4 LES FONCTIONS SINGLE-ROW

4.1 Les fonctions SQL

4.1.1 Les types de fonctions SQL

Une fonction SQL est un programme qui effectue une opération sur des données.

Les fonctions SQL peuvent être utilisées pour formater des dates et des nombres pour l'affichage, et pour convertir des types de données de colonne.

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

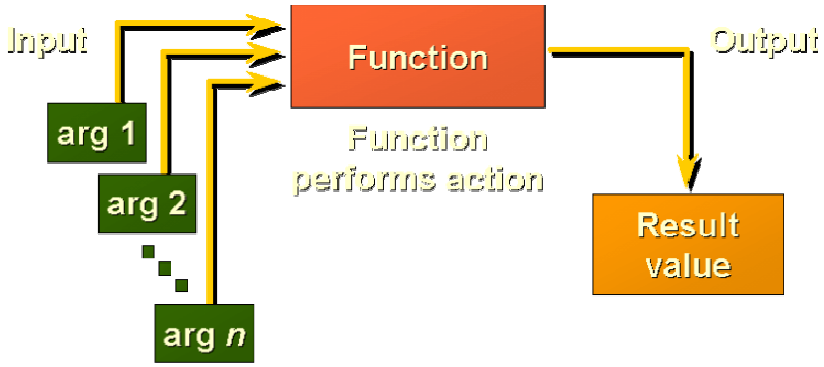


Figure 8 : Les fonctions SQL

Il existe deux types de fonction SQL :

- Single-row functions :
  Elles opèrent sur des enregistrements seuls et produisent un résultat par enregistrement.
- Multiple-row functions :
  Elles opèrent sur un groupe d'enregistrements et produisent un résultat par groupe d'enregistrements. Elles sont aussi appelées fonctions de groupe.

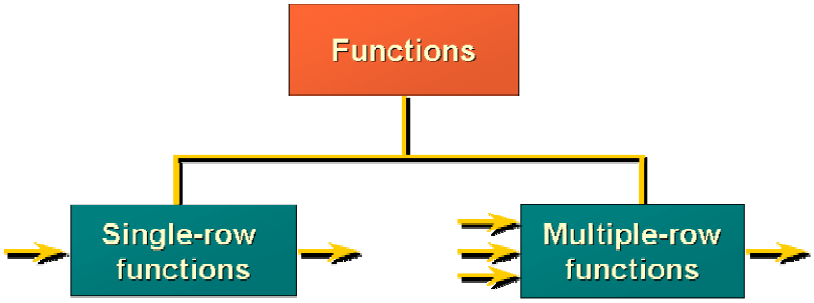


Figure 9 : Les types de fonctions SQL

4.1.2 Les fonctions SQL single-row

Les fonctions single-row sont utilisées pour manipuler des données.

Function_name(column | expr, [arg1,arg2 …])

Un argument d'une fonction single-row peut-être : une constante, une variable défini par l'utilisateur, une colonne ou une expression.

Elles peuvent :
- retourner un type différent de son ou ses argument(s),
- être utilisées dans les clauses SELECT, WHERE et ORDER BY,
- s'imbriquer.

Les fonctions single-row sont classées en cinq types (type de données en entrée Æ type(s) de données en sortie) :

- character function
  character Æ character / number

- number function
  number Æ number

- date function
  date Æ date / number / character

- data type conversion function
  number Æ varchar2
  varchar2 Æ number / date
  date Æ varchar2

- general function
  Substitution des valeurs nulles d'une colonne par une valeur choisie (NVL).
  Exécution de requêtes basées sur une condition IF-THEN-ELSE (DECODE).

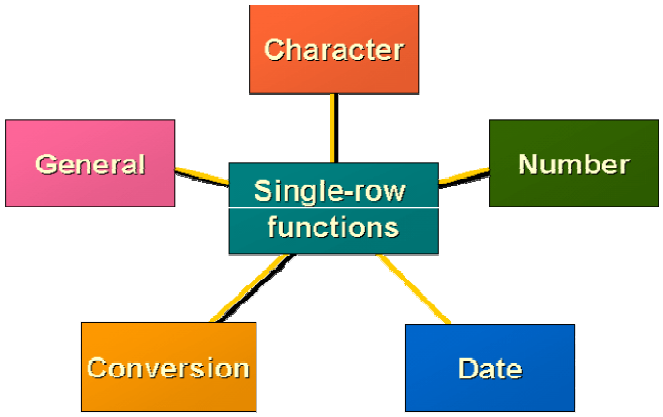


Figure 10 : Classement des fonctions single-row

4.2 Les fonctions opérant sur les caractères

Les fonctions opérant sur des caractères sont divisées en deux groupes :
- Les fonctions de conversion de la casse
- Les fonctions de manipulation des données

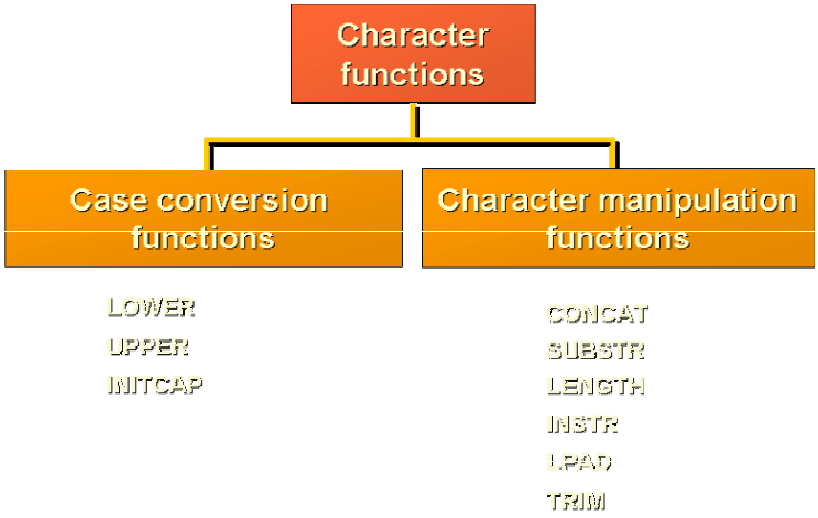


Figure 11 : Les fonctions opérant sur les caractères

4.2.1 Les fonctions de conversion 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.

INITCAP(column | expr)	Convertit la première lettre de chaque mot d'une chaîne de caractères en majuscule et les autres lettres en minuscule.
LOWER(column | expr)	Convertit une chaîne de caractères en minuscule.
UPPER(column | expr)	Convertit une chaîne de caractères en majuscule.

Exemple 1 :

Fonctions	**Résultats**
INITCAP('Cours de SQL')	Cours De Sql
LOWER('Cours de SQL')	cours de sql
UPPER ('Cours de SQL')	COURS DE SQL

Exemple 2 :

```
SQL> SELECT    empno, ename, deptno
  2  FROM      emp
  3  WHERE     ename = 'blake';
no rows selected.

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

EMPNO ENAME         DEPTNO
--------- ---------- ---------
     7698 BLAKE             30

1 row selected.
```

4.2.2 Les fonctions de manipulation de caractères

LENGTH(*column* | *expr*)	Permet de récupérer le nombre de caractères d'une chaîne. LENGTH retourne une valeur de type NUMBER.
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*.
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*.
LPAD(*column* | *expr*, *n*, '*string*')	Permet de placer *n* caractères de type *string* à gauche de la valeur de *column* (ou *expr*).
RPAD(*column* | *expr*, *n*, '*string*')	Permet de placer *n* caractères de type *string* à droite de la valeur de *column* (ou *expr*).
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 "||").
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*.

La fonction TRIM est une nouvelle fonction intégrée dans Oracle8i qui fait le travail de LTRIM et RTRIM à la fois.

4.2.3 Utilisation des fonctions SQL manipulant les caractères

Exemple 1 :

Fonctions	Résultats
CONCAT ('Bon', 'jour')	Bonjour
SUBSTR ('Bonjour',1,3)	Bon
LENGTH ('Bonjour')	7
INSTR('Bonjour','j')	4
LPAD(sal, 10,'*')	******5000
RPAD(sal, 10,'*')	5000******
TRIM ('S' FROM 'SSMITH')	MITH

Exemple 2 :

```
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')
---------- ------------------- ------------- -----------------
MARTIN     MARTINSALESMAN                  6                 2
ALLEN      ALLENSALESMAN                   5                 1
TURNER     TURNERSALESMAN                  6                 0
WARD       WARDSALESMAN                    4                 2
```

4.3 Les fonctions opérant sur les nombres

4.3.1 La 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 a effectuer des calculs dans un ordre SELECT où la clause FROM ne contient aucune table dont on ai 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

1 row selected.
```

La fonction ROUND peut-être utilisé avec des dates.

4.3.2 La 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),
  2            TRUNC(45.923,-1)
  3  FROM      dual;

TRUNC(45.923,2) TRUNC(45.923) TRUNC(45.923,-1)
--------------- ------------- ---------------
          45.92            45              40

1 row selected.
```

La fonction TRUNC peut être utilisée avec des dates.

4.3.3 La 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)
---------- --------- --------- -------------
MARTIN          1250      1400          1250
ALLEN           1600       300           100
TURNER          1500         0          1500
WARD            1250       500           250

4 rows selected.
```

4.4 Les fonctions opérant sur les dates

4.4.1 La 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 :

```
SQL> SELECT    SYSDATE
  2  FROM      dual ;
```

Æ 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 :

Opération	Résultat	Description
date + number	date	ajoute un nombre de jours à une date
date - number	date	soustrait un nombre de jours à une date
date - date	nombre de jours	soustrait une date à une autre date
date + number/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
---------- ---------
KING       830.93709
CLARK      853.93709
MILLER     821.36566

3 rows selected.
```

4.4.3 Les fonctions opérant sur les dates

Voici les principales fonctions opérant sur des dates :

MONTHS_BETWEEN(*date1*,*date2*)	Retourne le nombre de mois séparant deux dates. Le résultat peut-être positif ou négatif. Si *date1* est plus vieille que *date2*, le résultat est positif. Si *date1* est plus récente que *date2*, le résultat est négatif.
ADD_MONTHS(*date*,*n*)	Ajoute *n* mois à une date. *n* doit être un entier positif ou négatif.
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.
LAST_DAY(*date*)	Trouve la date du dernier jour du mois qui contient *date*.
ROUND(*date* [,'*format*'])	Retourne *date* arrondie à l'unité spécifié par *format*. Si le format est omis, *date* est arrondie au jour le plus près.
TRUNC(*date* [,'*format*'])	Retourne *date* tronquée à l'unité spécifié par *format*. Si le format est omis, *date* est tronquée au jour le plus près.

format peut avoir les valeurs : DAY, MONTH ou WEEK.

Exemple :

Fonctions	**Résultats**
MONTHS_BETWEEN ('01-SEP-95','11-JAN-94')	19.6774194
ADD_MONTHS ('11-JAN-94',6)	'11-JUL-94'
NEXT_DAY ('01-SEP-95','FRIDAY')	'08-SEP-95'
LAST_DAY('01-SEP-95')	'30-SEP-95'
ROUND('25-JUL-95','MONTH')	'01-AUG-95'
ROUND('25-JUL-95','YEAR')	'01-JAN-96'
TRUNC('25-JUL-95','MONTH')	'01-JUL-95'
TRUNC('25-JUL-95','YEAR')	'01-JAN-95'

4.5 Fonctions de conversions de types de données

Il existe deux types de fonctions de conversion de types de données :

- les fonctions de conversion explicite (effectuées par l'utilisateur) .
- les fonctions de conversion implicite (effectuées par le serveur Oracle).

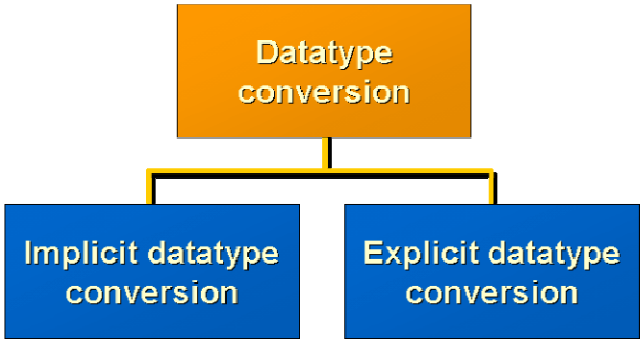


Figure 12 : Les fonctions de conversion de types de données

Bien que la conversion implicite de type de données soit disponible, il est recommandé d'effectuer des conversions de type de données explicites pour assurer la fiabilité des ordres SQL.

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

Assignement		
	De	**à**
	VARCHAR2 ou CHAR	NUMBER
	VARCHAR2 ou CHAR	DATE
	NUMBER	VARCHAR2
	DATE	VARCHAR2
Evaluation d'expression		
	De	**à**
	VARCHAR2 ou CHAR	NUMBER
	VARCHAR2 ou CHAR	DATE

4.5.1 Conversion explicite de types de données

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

- **TO_CHAR**(number | date [,'format']) : convertit un nombre ou une date en une chaîne de caractères
- **TO_NUMBER**(char ['format']) : convertit une chaînes de caractères en un nombre
- **TO_DATE**(char [,'format']) : convertit une chaîne de caractères en une date.

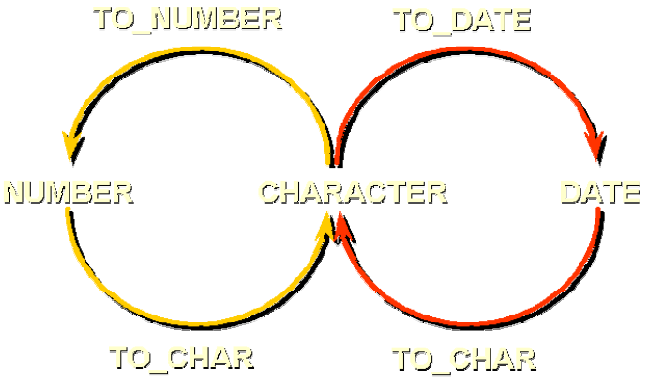


Figure 13 : Conversion explicite de types de données

4.5.2 La fonction TO_CHAR avec des dates

Voici les différents formats de conversion de la fonction **TO_CHAR** avec des dates :

YYYY	Æ année sur quatre chiffres
YEAR	Æ année écrite en toutes lettres
MM	Æ le mois sur deux caractères
MONTH	Æ le mois en toutes lettres
DY	Æ le jour de la semaine en trois lettres
DAY	Æ le jour de la semaine en toutes lettres
WW ou W	Æ semaine de l'année ou du mois
DDD	Æ jour de l'année
DD	Æ jour du mois
D	Æ jour de la semaine
J	Æ le nombre de jour depuis le 31 décembre 4713 BC
Q	Æ quart de l'année
MON	Æ le mois sur trois caractères
RM	Æ numéro romain du mois
CC	Æ siècle
fmDAY	Æ supprime les espaces
AM ou PM	Æ indicateur de méridien
HH ou HH12 ou HH24	Æ heure du jour
MI	Æ minutes (0-59)
SS	Æ secondes (0-59)
SSSS	Æ secondes (0-86399)
TH	Æ nombre ordinal
SP	Æ nombre écrit en toutes lettres
SPTH ou THSP	Æ nombre ordinal écrit en toutes lettres

Le format doit être entouré de simples côtes. Le format 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 "padded" avec des blancs. Pour éliminer les blancs ou supprimer les zéros, il faut utiliser l'élément "fill mode".

Exemple :

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

ENAME      HIREDATE
---------- -----------------
KING       17 November 1981
BLAKE      1 May 1981
CLARK      9 June 1981
JONES      2 April 1981
MARTIN     28 September 1981
ALLEN      20 February 1981
...
14 rows selected.
```

4.5.3 La fonction TO_CHAR avec des nombres

Voici les différents formats de conversion de la fonction **TO_CHAR** avec des nombres :

ELEMENT	EXAMPLE	RESULT	MEANING
9	999999	1234	
0	99999	1234	
$	$999999	$1234	
.	99999.99	1234.00	
,	999,999	1,234	
L	L999999	FF1234	francs
L	L999999	DM1234	Deutche marks
L	L999999	$1234	dollars
MI	999999MI	1234-	
PR	999999PR	<1234>	
EEEE	99.999EEE	$1.234^{E}+03$	
V	9999V99	123400	
B	B9999.99	1234	

Le serveur Oracle affiche une chaîne de caractères composée de signes # à la place du résultat si la valeur fournit 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 = 'SCOTT';

SALARY
--------
  $3,000

1 row selected.
```

Autres fonctions de conversion de types de données

TO_NUMBER(*char* [,*format*])	Convertit la chaîne de caractères *char* en un nombre selon le format spécifié par le paramètre optionnel *format*.
TO_DATE(*char* [,*format*])	Convertit la chaîne de caractères *char* représentant une date en une valeur de type Date selon le format spécifié par le paramètre optionnel *format*. Le format par défaut est DD-MON-YY.
CHR(*number*)	Retourne le caractère ayant l'équivalent binaire de *number* comme une valeur VARCHAR2 dans un jeu de caractères de base de données.
CAST(*value* **AS** *datatype*)	Convertit le type de données de la valeur *value* à un autre type de données *datatype*.

4.5.5 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.

Exemples :

La date '01/02/01' au format RR sera interprétée comme étant en 2001 alors qu'au format YY, elle sera interprétée comme étant en 1901.

Current Year	Specified Date	RR Format	YY Format
1995	27-oct-95	1995	1995
1995	27-oct-17	2017	1917
2001	27-oct-17	2017	2017
2001	27-oct-95	1995	2095

Utilisation du format RR :

		Si l'année spécifiée sur deux chiffres est :	
		0 - 49	**50 - 99**
Si l'année courante sur deux chiffres 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 générales

4.6.1 La fonction NVL

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

NVL (expr1,expr2) :

expr1 : valeur source ou expression pouvant contenir une valeur nulle
expr2 : valeur de substitution

expr1 et *expr2* doivent être du même type de données.

Conversion de types de données variés :

Datatype	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, sal, NVL(TO_CHAR(comm),'no commission') comm
  2  FROM      emp;

ENAME             SAL COMM
---------- ---------- -------------------
SMITH              800 pas de commission
ALLEN             1600 300
WARD              1250 500
JONES             2975 pas de commission
MARTIN            1250 1400
BLAKE             2850 pas de commission
CLARK             2450 pas de commission
...
14 rows selected.
```

Æ 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 caractère "pas de commission".

Exemple 2 :

```
SQL> SELECT    ename, sal, comm, (sal*12)+NVL(comm,0) salaire anuel
  2  FROM      emp;

ENAME            SAL      COMM SALAIRE ANNUEL
---------- --------- --------- --------------
KING            5000                    60000
BLAKE           2850                    34200
CLARK           2450                    29400
JONES           2975                    35700
MARTIN          1250      1400          16400
ALLEN           1600       300          19500
...
14 rows selected.
```

Æ La fonction NVL remplace les valeurs nulles de la colonne COMM par le nombre 0 afin de pouvoir calculer le salaire annuel de ceux qui ne touchent pas de commission.

4.6.2 La 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*,]
[…,…]
[*resultat par défaut*])

La fonction DECODE décode l'expression après l'avoir comparé à chaque valeur *search*. Si l'expression est la même que *search*, la valeur *result* est retournée. Si la valeur par défaut est omis et qu'aucune valeur *search* ne correspond à l'expression, une valeur nulle est retournée.

Exemple 1 :

```
SQL> SELECT    job, sal,
  2            DECODE(job,'ANALYST', SAL*1.1,
  3                       'CLERK',   SAL*1.15,
  4                       'MANAGER', SAL*1.20,
  5                                  SAL)
  6            REVISED_SALARY
  7  FROM      emp;
JOB              SAL REVISED_SALARY
--------- --------- --------------
PRESIDENT       5000           5000
MANAGER         2850           3420
MANAGER         2450           2940
...
14 rows selected.
```

Æ 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 = 'ANALYST' THEN revised_salary = sal * 1,1 ;
ELSIF job = 'CLERK' THEN revised_salary = sal * 1,15 ;
ELSIF job = 'MANAGER' THEN revised_salary = sal * 1,20 ;
ELSE revised_sal = sal ;
```

Exemple 2 :

```
SQL> SELECT    ename, sal,
  2            DECODE(TRUNC(sal/1000, 0),
  3                                    0, 0.00,
  4                                    1, 0.09,
  5                                    2, 0.20,
  6                                    3, 0.30,
  7                                    4, 0.40,
  8                                    5, 0.42,
  9                                    6, 0.44,
 10                                       0.45) TAX_RATE
 11  FROM      emp
 12  WHERE     deptno = 30;

ENAME             SAL   TAX_RATE
---------- ---------- ----------
ALLEN            1600        ,09
WARD             1250        ,09
MARTIN           1250        ,09
BLAKE            2850         ,2
TURNER           1500        ,09
JAMES             950          0

6 rows selected.
```

Æ Cette requête affiche te taux de la taxe affectée à chaque employé du département numéro 30. Ce taux est calculé en fonction de la valeur tronquée du salaire divisé pas 1000.

4.6.3 Les fonctions imbriquées

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

F3 (F2 (F1 (*col*,*arg1*), *arg2*), *arg3*)

Ordre d'évaluation : F1, F2 puis F3

Exemple :

```
SQL> SELECT    ename,
  2            NVL(TO_CHAR(mgr),'No Manager')
  3  FROM      emp
  4  WHERE     mgr IS NULL;

ENAME      NVL(TO_CHAR(MGR),'NOMANAGER')
---------- ----------------------------
KING       No Manager

1 row selected.
```

Æ La fonction TO_CHAR convertit en CHAR la valeur contenue dans la colonne JOB afin quel soit du même type que le deuxième argument de la fonction NVL.