

Les fonctions SQL

Introduction

Les fonctions sont un dispositif très puissant du SQL. Les fonctions peuvent être utilisées pour :

- Effectuer des calculs de données
- Modifier des parties de données
- Manipuler les résultats pour les groupes de lignes
- Gérer le format d'affichage des nombres et des dates
- Convertir un type de données de colonne dans un autre

Les fonctions SQL peuvent avoir des arguments mais retournent toujours une seule valeur.

La plupart des fonctions décrites dans ce cours sont spécifiques à la version SQL d'Oracle.

Les fonctions Single-Row :

Ces fonctions opèrent sur une seule ligne et renvoient un seul résultat par ligne. Il existe différents types de fonctions single-row. Ce cours couvre les types de fonction suivants :

- Caractères
- Nombre
- Date
- Conversion
- Générales

Les fonctions Multiple-Row

Les fonctions peuvent manipuler des groupes de lignes pour retourner un seul résultat par groupe de lignes. On appelle aussi ces fonctions des fonctions de groupe (elles seront étudiées dans une autre leçon).

Les fonctions Single-Row

Les fonctions single-row sont utilisées pour manipuler des données. Elles peuvent accepter un ou plusieurs arguments et retournent une valeur pour chaque ligne renvoyée par la requête.

Types d'argument utilisables :

- Constantes définies par les utilisateurs
- Variables
- Noms de colonne
- Expressions

Fonctionnalités des fonctions single-row :

- Agissent sur chaque enregistrement retourné dans la requête.
- Retournent un résultat par ligne.
- Peuvent retourner une donnée de type de données différent de l'initiale.
- Peuvent accepter un ou plusieurs arguments.
- Peuvent être utilisées dans les clauses SELECT, WHERE et ORDER BY
- Peuvent être imbriquées

Syntaxe :

```
function_name [(arg1, arg2,...)]
```

Dans la syntaxe :

- `function_name` : est le nom de la fonction
- `arg1, arg2` : désignent tous les arguments pouvant être utilisés dans la fonction (nom de colonne, expression, etc.)

Types de fonctions single-row :

- Fonctions de caractères : Acceptent en argument des chaînes de caractères et retournent soit une chaîne de caractère soit un nombre
- Fonctions de nombres : Acceptent des valeurs numériques et retournent des valeurs numériques
- Fonctions de dates : Opèrent sur des valeurs de type de données DATE (Toutes les fonctions de date retournent une valeur de type DATE excepté la fonction MONTHS_BETWEEN qui retourne un nombre.)
- Fonctions de conversion : Convertissent une valeur d'un type de donnée dans un autre
- Fonctions générales : NVL, NVL2, NULLIF, COALESCE, CASE, DECODE

Les fonctions opérant sur les caractères

Présentation

Les fonctions single-row opérant sur les chaînes de caractères acceptent en paramètre des chaînes de caractères et retournent soit une valeur numérique, soit une chaîne de caractères.

Ces fonctions sont divisées en deux groupes :

- Fonctions de manipulation de casse
- Fonctions de manipulation des chaînes de caractères

Fonctions	Description
LOWER (column expression)	Convertie une chaîne de caractère en minuscule
UPPER (column expression)	Convertie une chaîne de caractères en majuscule
INITCAP (column expression)	Convertie la première lettre de chaque mot en majuscule et les autres lettres en minuscule
CONCAT (column1	Concatène le premier argument avec le deuxième; équivalente à

Fonctions	Description
expression1, column2 expression2)	l'opérateur de concaténation ()
SUBSTR (column expression, m [,n])	Extrait une partie d'une chaîne de caractère à partir de la position spécifiée par m, de longueur spécifiée par n. (Si la valeur de m est négative, le compteur démarre depuis la fin de la chaîne de caractère. Si n n'est pas spécifié, tous les caractères à partir de la position m sont retournés)
LENGTH (column expression)	Retourne la longueur d'une chaîne de caractères
INSTR (column expression,'string' [,m][,n])	Retourne la position d'une chaîne de caractère ('string') dans une autre. Vous pouvez indiquer la position de départ de la recherche avec m, et la nième occurrence de la valeur recherchée dans la chaîne de caractère. Par défaut, les valeurs de m et n sont à 1, ce qui signifie que la recherche commence au début de la chaîne de caractère et renvoie la première occurrence.
LPAD (column expression, n, 'string')	Permet d'avoir à l'affichage une chaîne de n caractères, si nombre de caractères la valeur retournée par column expression est inférieur à n, le vide est rempli par 'string' à gauche de la valeur.
RPAD (column expression, n, 'string')	Permet d'avoir à l'affichage une chaîne de n caractères, si nombre de caractères la valeur retournée par column expression est inférieur à n, le vide est rempli par 'string' à droite de la valeur.
TRIM (leading trailing both, trim_character FROM trim_source)	Permet de couper les caractères trim_character au début (leading), à la fin (trailing) ou les deux (both) d'une chaîne de caractère trim_source. Si trim_character ou trim_source est une chaîne littérale entourez-la d'apostrophes.
REPLACE (text, search_string, replacement_string)	Cherche l'expression search_string dans l'expression expr. Si la chaîne de caractère est trouvée, elle est remplacée par l'expression replacement_string

Fonctions manipulant la casse

LOWER, UPPER, et INITCAP sont les trois fonctions de conversion de casse.

- LOWER : Convertie une chaîne de caractères en minuscule
- UPPER : Convertie une chaîne de caractère en majuscule
- INITCAP : Convertie la première lettre de chaque mot en majuscule et les autres lettres en minuscules

```
SELECT 'The job id for ' || UPPER(last_name) ||
       ' is ' || LOWER(job_id) AS "EMPLOYEE DETAILS"
FROM   employees;
```

L'exemple ci-dessous affiche le numéro d'employé, le nom, et le numéro de département pour l'employé Higgins.

```
SELECT employee_id, last_name, department_id
```

```
FROM employees
WHERE last_name = 'higgins';

no rows selected
```

La clause WHERE recherche le nom d'employé higgins. Comme tous les enregistrements de la table EMPLOYEES sont stockés avec une casse particulière, le nom higgins n'est pas trouvé dans la table et donc aucune ligne n'est sélectionnée.

```
SELECT employee_id, last_name, department_id
FROM employees
WHERE LOWER(last_name) = 'higgins';
```

La clause WHERE de ce second ordre SQL compare le nom d'employé dans la table EMPLOYEES à higgins, en convertissant la colonne LAST_NAME en minuscule. Les deux noms sont en minuscule, une correspondance est donc trouvée et une ligne est sélectionnée. La clause WHERE peut être réécrite de la manière suivante et fournir le même résultat :

```
...WHERE last_name = 'Higgins'
```

Remarquez qu'à l'affichage le nom apparaît tel qu'il a été stocké dans la base de données. Pour afficher le même nom avec toutes les lettres en majuscule utilisez la fonction UPPER dans la clause SELECT :

```
SELECT employee_id, UPPER(last_name), department_id
FROM employees
WHERE LOWER(last_name) = 'higgins';
```

Fonctions de caractères

CONCAT, SUBSTR, LENGTH, INSTR, LPAD, RPAD, et TRIM sont des fonctions manipulant les chaînes de caractères.

- CONCAT : Joint des valeurs ensemble (vous êtes limité à deux paramètres dans cette fonction)
- SUBSTR : Extrait une chaîne de caractère d'une taille déterminée.
- LENGTH : Retourne la longueur d'une chaîne de caractères.
- INSTR : Trouve la position d'un caractère.
- LPAD : Remplit le début de la chaîne avec le caractère spécifié pour atteindre la longueur demandée.
- RPAD : Remplit la fin de la chaîne avec le caractère spécifié pour atteindre la longueur demandée.
- TRIM : Supprime un caractère spécifié au début, à la fin ou les deux, d'une chaîne de caractères.

Fonctions	Résultat
CONCAT('Hello', 'World')	HelloWorld
SUBSTR('HelloWorld',1,5)	Hello
LENGTH('HelloWorld')	10
INSTR('HelloWorld', 'W')	6
LPAD(salary,10,'*')	*****24000

Fonctions	Résultat
RPAD(salary, 10, '*')	24000*****
REPLACE('JACK and JUE','J','BL')	BLACK and BLUE
TRIM('H' FROM 'HelloWorld')	elloWorld

Vous pouvez utiliser les fonctions telles que UPPER et LOWER avec une variable de substitution
UPPER('&job_title') qui permet à l'utilisateur de ne pas avoir à entrer un poste dans une casse spécifique.

L'exemple ci-dessous affiche le prénom et le nom concaténés, la longueur du nom de l'employé ainsi que la position de la lettre a dans leur nom pour tous les employés qui ont dans leur identifiant de poste la chaîne de caractère REP à partir de la 4 lettre.

```
SELECT employee_id, CONCAT(first_name, last_name) NAME, job_id,
       LENGTH(last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(job_id, 4) = 'REP';
```

L'exemple suivant modifie l'ordre SQL précédent pour afficher les informations de tous les employés dont le nom se termine par un n.

```
SELECT employee_id, CONCAT(first_name, last_name) NAME,
       LENGTH(last_name), INSTR(last_name, 'a') "Contains 'a'?"
FROM   employees
WHERE  SUBSTR(last_name, -1, 1) = 'n';
```

Les fonctions opérant sur des nombres

Présentation

Les fonctions opérant sur les nombres acceptent en paramètre des valeurs numériques et retournent des valeurs numériques. Voici une liste non exhaustive des fonctions :

Fonctions	Description
ROUND(column expression, n)	Arrondie une valeur numérique à n décimales près. Si n est positif, la valeur sera arrondie à n décimales après la virgule. Si n est négatif, la valeur sera arrondie à n décimales avant la virgule. Si vous ne précisez pas n, la valeur sera arrondie à 0 décimale près.
TRUNC(column expression, n)	Permet de tronquer les valeurs à n décimales près. Si n est positif, la valeur sera tronquée à n décimales après la virgule. Si n est négatif, la valeur sera arrondie à n décimales avant la virgule. Si vous ne précisez pas n, la valeur sera tronquée à 0 décimale près.

Fonctions	Description
MOD(m,n)	Permet de retourner le reste de la division de la valeur de m par n.

Utilisation de la fonction ROUND

La fonction ROUND arrondie une valeur numérique à n décimales près. Si le second argument est omis ou s'il est égal à 0, la valeur est arrondie à zéro décimale près. Si le second argument est 2, la valeur est arrondie à 2 décimales près. Réciproquement, si le deuxième argument est -2, la valeur est arrondie à deux décimales près à gauche (arrondie à la dizaine la plus proche).

La fonction ROUND peut être utilisée avec les dates.

```
SELECT ROUND(45.923,2), ROUND(45.923), ROUND(45.923,-1)
FROM DUAL;
```

Table DUAL : Le propriétaire de la table DUAL est l'utilisateur SYS et elle est accessible à tous les utilisateurs. DUAL contient une seule colonne DUMMY et une ligne avec la valeur X. La table DUAL est pratique quand vous voulez retourner une valeur une seule fois (par exemple, la valeur d'une constante, d'une pseudo-colonne ou d'une expression qui ne provient pas d'une table avec des données d'utilisateurs). La table DUAL est généralement utilisée pour compléter la syntaxe des clauses SELECT car les clauses SELECT et FROM sont obligatoires. Certains calculs n'ont pas besoin d'utiliser les valeurs de table.

Utilisation de la fonction TRUNC

La fonction TRUNC permet de tronquer les valeurs à n décimales près. TRUNC manipule les mêmes types d'arguments que la fonction ROUND. Si le second argument est omis ou s'il est égal à 0, la valeur est tronquée à zéro décimale près. Si le second argument est 2, la valeur est tronquée à 2 décimales près. Réciproquement, si le deuxième argument est -2, la valeur est tronquée à deux décimales près à gauche.

Comme la fonction ROUND, la fonction TRUNC peut être utilisée avec les dates.

```
SELECT TRUNC(45.923,2), TRUNC(45.923), TRUNC(45.923,-1)
FROM DUAL;
```

Utilisation de la fonction MOD

La fonction MOD renvoie le reste de la division du premier argument par le second argument. L'exemple suivant affiche le reste de la division du salaire par 5000 pour tous les employés dont l'intitulé de poste est SA_REP.

La fonction MOD est souvent utilisée pour déterminer si une valeur est paire ou impaire.

```
SELECT last_name, salary, MOD(salary, 5000)
```

```
FROM    employees
WHERE   job_id = 'SA_REP';
```

Travailler avec les dates

Format de date Oracle

Le serveur Oracle stocke les dates dans un format numérique interne, représentant le siècle, l'année, le mois, le jour, les heures, les minutes et les secondes. L'affichage par défaut et le format d'entrée pour n'importe quelle date est DD-MON-RR. Les dates Oracle valides se situent entre le 1er janvier 4712 av. J.-C. et le 31 décembre 9999 apr. J.-C.

Dans l'exemple qui suit, les valeurs de la colonne HIRE_DATE sont affichées dans le format par défaut DD-MON-RR. Cependant, les dates ne sont pas stockées dans ce format dans la base de données. Tous les composants de la date et de l'heure sont stockés. Ainsi, une date d'embauche comme 17-JUN-87 affiche le jour, le mois et l'année, or les informations sur le siècle et l'heure lui sont également associées. La donnée complète peut être le 17 Juin 1987, 5:10:43 p.m.

```
SELECT last_name, hire_date
FROM    employees
WHERE   hire_date < '01-FEB-88';
```

Dans la base de données sont stockées de la manière suivante:

CENTURY YEAR MONTH DAY HOUR MINUTE SECOND

19	87	06	17	17	10	43
----	----	----	----	----	----	----

Quand une ligne avec une colonne de date est insérée dans une table, l'information sur le siècle est recueillie de la fonction SYSDATE. Cependant, quand la colonne de date est affichée à l'écran, le composant siècle n'est pas affiché (par défaut).

Le type de données DATE stocke toujours en interne l'information sur l'année sous la forme de quatre chiffres : deux chiffres pour le siècle et deux chiffres pour l'année. Par exemple, le serveur Oracle stocke l'année 1987 ou 2004 et pas 87 ou 04.

La fonction SYSDATE

SYSDATE est une fonction de date qui retourne la date et l'heure courant du serveur de la base de données. Vous pouvez utiliser SYSDATE comme si vous utilisiez n'importe quel autre nom de colonne. Par exemple, vous pouvez afficher la date courante en sélectionnant SYSDATE d'une table. Il est fréquent de sélectionner SYSDATE de la table DUAL.

Exemple :

Affichez la date courante en utilisant la table DUAL :

```
SELECT SYSDATE
FROM DUAL;
```

Opérations arithmétiques sur les dates

Parce que la base de données stocke les dates comme des nombres, vous pouvez réaliser des calculs en utilisant des opérateurs arithmétiques comme l'addition et la soustraction. Vous pouvez ajouter ou soustraire aussi bien des nombres constants que les dates.

Vous pouvez réaliser les opérations suivantes :

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

Si une date récente est soustraite à une date antérieure alors la différence est un nombre négatif.

L'exemple qui suit affiche le nom de l'employé et le nombre de semaines travaillées pour tous les employés du département 90. On soustrait la date à laquelle l'employé a été embauché à la date courante (SYSDATE) et divise le résultat par 7 pour calculer le nombre de semaines pendant lesquelles un employé a travaillé dans l'entreprise.

```
SELECT last_name, (SYSDATE-hire_date)/7 AS WEEKS
FROM employees
WHERE department_id = 90;
```

SYSDATE est une fonction SQL qui retourne la date et l'heure courante. Votre résultat peut être différent de celui de l'exemple.

Les fonctions de date

Les fonctions de date opèrent sur les dates Oracle. Toutes les fonctions de date retournent une valeur de type de données DATE exceptée MONTHS_BETWEEN qui retourne une valeur numérique.

MONTHS_BETWEEN(date1, date2)

Trouve le nombre de mois entre date1 et date2. Le résultat peut être positif ou négatif. Si la date de date1 est plus récente que celle de date2, le résultat est positif. Si la date de date1 est plus ancienne que celle de date2, le résultat est négatif. La partie non entière du résultat représente une portion du mois.

```
MONTHS_BETWEEN('01-SEP-95', '11-JAN-94')
```

```
result : 19.6774194
```

ADD_MONTHS(date, n)

Ajoute n nombre de mois calendaires à la date. La valeur de n doit être un nombre entier et ne peut pas être négative.

```
ADD_MONTHS('11-JAN-94', 6)
```

```
result : '11-JUL-94'
```

NEXT_DAY(date, 'char')

Renvoie la date du prochain jour de la semaine spécifié ('char'). La valeur de char peut être un nombre représentant le jour de la semaine ou une chaîne de caractères.

```
NEXT_DAY('01-SEP-95', 'FRIDAY')
```

```
result : '08-SEP-95'
```

LAST_DAY(date)

Renvoie la date du dernier jour du mois contenu dans la date spécifiée par date.

```
LAST_DAY('01-FEB-95')
```

```
result : '28-FEB-95'
```

ROUND(date[, 'fmt'])

Retourne la date arrondie à l'unité spécifiée par le modèle de format fmt. Si le modèle de format fmt est omit, la date est arrondie au jour le plus près.

TRUNC(date[, 'fmt'])

Retourne la date avec une portion de temps du jour tronqué à l'unité spécifiée par le modèle de format fmt. Si le modèle de format fmt est omit, la date est tronquée au jour le plus près.

Cette liste est un sous ensemble des fonctions de date disponibles. Les modèles de format sont décrites plus loin.

On affiche les l'identifiant d'employé, la date d'embauche, le nombre de mois travaillés, la date après six mois d'essaye, le premier vendredi après la date d'embauche et le dernier jour du mois d'embauche pour tous les employés qui ont été embauché depuis mois de 80 mois.

```
SELECT employee_id, hire_date,  
       MONTHS_BETWEEN (SYSDATE, hire_date) TENURE,  
       ADD_MONTHS (hire_date, 6) REVIEW,
```

```

        NEXT_DAY (hire_date, 'FRIDAY'),
        LAST_DAY(hire_date)
FROM    employees
WHERE   MONTHS_BETWEEN (SYSDATE, hire_date) < 100;

```

Les fonctions ROUND et TRUNC peuvent être utilisées avec des nombres et les dates. Quand ces fonctions sont utilisées avec dates, les dernières sont arrondies ou tronquées à un modèle de format spécifié. Donc vous pouvez arrondir ou tronquer les dates jusqu'à l'année ou mois le plus près.

Supposez que SYSDATE = '25-JUL-03' :

L'exemple suivant affiche le numéro d'employé, la date d'embauche et le mois d'embauche grâce à la fonction ROUND et TRUNC. Seulement les employés embauchés en 1997 sont affichés.

```

SELECT employee_id, hire_date,
       ROUND(hire_date, 'MONTH'), TRUNC(hire_date, 'MONTH')
FROM    employees
WHERE   hire_date LIKE '%97';

```

Fonctions de conversion

Présentation

Les colonnes des tables dans une base de donnée Oracle peuvent être définies non seulement en utilisant les types de données Oracle mais ANSI, DB2, et SQL/DS. Cependant, le serveur Oracle convertit en interne ces types de données en types de données Oracle.

Dans certains cas, le serveur Oracle utilise des données d'un certain type lorsque des données de type différent sont attendues. Lorsque ce cas se présente, le serveur Oracle peut automatiquement convertir les données dans le type de données adapté. Cette conversion de type de données peut être réalisée implicitement par le serveur Oracle ou explicitement par l'utilisateur.

La conversion implicite fonctionne selon les règles expliquées dans la partie suivante.

La conversion explicite se fait avec des fonctions de conversion. Les fonctions de conversion convertissent une valeur d'un type de données dans un autre type de données. En général, la forme des noms des fonctions suit la convention type de données TO type de données. Le premier type de données est le type à convertir et le deuxième est le type retourné.

Bien qu'une conversion implicite de types de données soit disponible, il est recommandé de réaliser une conversion explicite des types de données pour assurer la fiabilité des requêtes SQL.

Conversion implicite de types de données

Pour les assignements, le serveur Oracle est capable de faire les conversions suivantes automatiquement :

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

L'assignement réussit si le serveur Oracle peut convertir le type de donnée de la valeur assignée dans le type de donnée cible.

Par exemple, l'expression `hire_date > '01-JAN-90'` entraîne une conversion implicite de la chaîne de caractère '01-JAN-90' à une date.

Pour évaluer une expression, le serveur Oracle est capable de faire les conversions suivantes automatiquement :

De	A
VARCHAR2 ou CHAR	NUMBER
VARCHAR2 ou CHAR	DATE

En général, le serveur Oracle utilise une règle pour les expressions quand une conversion de type de donnée est nécessaire.

Par exemple, de l'expression `salary = '20000'` entraîne une conversion implicite de la chaîne de caractère '20000' vers le nombre 20000.

les conversions de CHAR à NUMBER se produisent avec succès uniquement si la chaîne de caractère représente un nombre valide.

Conversion explicite de types de données

SQL fournit trois fonctions pour convertir un type de données d'une valeur dans un autre type de données :

Fonction	But
<code>TO_CHAR(number date [, fmt] [, nlsparams])</code>	Convertit un nombre ou une date en une chaîne de caractère VARCHAR2 avec le modèle de format fmt. Conversion de nombre : Le paramètre nlsparams spécifie les caractères qui sont retournés par des éléments format nombre (Caractère décimal, Séparateur

Fonction	But
	de groupe, Symbole monétaire local, Symbole monétaire international). Si le paramètre nlsparams ou les autres paramètres ne sont pas indiqués, la fonction utilise les valeurs des paramètres par défaut de la session.
	Conversion de date : Le paramètre nlsparams spécifie la langue dans laquelle le nom du mois, le nom du jour et les abréviations sont retournés. Si ce paramètre est omis, cette fonction utilise le langage par défaut de la session.
TO_NUMBER(char [, fmt] [, nlsparams])	Convertit une chaîne de caractère contenant des chiffres en un nombre correspondant au format spécifié par le modèle de format optionnel fmt. Le paramètre nlsparams a le même fonctionnement que dans la fonction TO_CHAR pour la conversion de nombre.
TO_DATE(char [, fmt] [, nlsparams])	Convertit une chaîne de caractère représentant en une date en une valeur date suivant le format spécifié (fmt). Si fmt n'est pas indiqué, le format est DD-MON-YY. Le paramètre nlsparams a le même fonctionnement que dans la fonction TO_CHAR pour la conversion de date.
	La liste de fonctions mentionnées dans ce cours n'inclut qu'une partie des fonctions de conversion disponibles.

Utilisation de la fonction TO_CHAR avec les dates

Auparavant, toutes les valeurs de date Oracle étaient affichées dans le format DD-MON-YY. Vous pouvez utiliser la fonction TO_CHAR pour convertir une date dans un autre format.

```
TO_CHAR(date, 'format_model')
```

Règles :

- Le modèle de format doit être entouré d'apostrophes et est sensible à la casse.
- Le modèle de format peut inclure n'importe quel élément de format de date valide. Assurez vous de séparer la valeur de date du modèle de format par une virgule.
- Des espaces sont ajoutés automatiquement après le nom du jour et le nom du mois.
- Pour éliminer les espaces ou supprimer les zéros à gauche, il faut utiliser l'élément fm (fill mode).
- Vous pouvez mettre en forme les colonnes de résultant avec la commande iSQL*Plus COLUMN (expliqué dans un cours ultérieur)

```
SELECT employee_id, TO_CHAR(hire_date, 'MM/YY') Month_Hired
FROM employees
WHERE last_name = 'Higgins';
```

Eléments du format de date :

Elément	Description
SCC ou CC	Siècle; le serveur utilise le préfixe – avec les siècles av. J.-C.
YYYY ou SYYYY	Année; le serveur utilise le préfixe – avec les années av. J.-C.
YYY ou YY ou Y	3, 2 ou 1 derniers chiffres de l'année
Y, YYY	Année avec une virgule après les milliers
IYYY, IYY, IY, I	Année sur 4, 3, 2 ou 1 chiffres, basée sur le standard ISO
SYEAR ou YEAR	Année en lettres; le serveur utilise le préfixe – avec les années av. J.-C.
BC ou AD	Indique l'année av. J.-C. ou apr. J.-C.
B.C. ou A.D.	Indique l'année av. J.-C. ou apr. J.-C. avec des points
Q	Trimestre de l'année
MM	Mois sur deux chiffres
MONTH	Nom du mois suivi d'espaces pour atteindre neuf caractères
MON	Abréviation du mois en trois lettres
RM	Mois en chiffres romains
WW ou W	Semaine de l'année ou du mois
DDD or DD or D	Jour de l'année, du mois ou de la semaine
DAY	Nom du jour suivi d'espaces pour atteindre neuf caractères
DY	Abréviation du nom du jour en trois lettres
J	Jour julien (nombre de jours depuis le 31 Décembre 4713 av. J.-C.)

Eléments du format l'heure :

Utilisez les formats listés dans les tables suivantes pour montrer l'information de temps et des littéraux ainsi que pour renvoyer des nombres en toutes lettres.

Elément	Description
AM ou PM	Indicateur méridien (AM : ante meridiem, PM : post meridiem)
HH ou HH12 ou HH24	Heure du jour, ou heure (1–12), ou heure (0–23)
MI	Minute (0–59)
SS	Seconde (0–59)
SSSSS	Secondes après minuit (0–86399)

Autres formats :

Elément	Description
/ . ,	La ponctuation est reproduite dans le résultat.
“of the”	La chaîne en guillemets est reproduite dans le résultat

Spécifier les suffixes pour influencer sur l’affichage des nombres :

Elément	Description
TH	Nombre ordinal (par exemple, DDTH pour 4TH)
SP	Nombre en toutes lettres (par exemple, DDSP pour FOUR)
SPTH ou THSP	Nombre ordinal en toutes lettres (par exemple, DDSPTH pour FOURTH)

Exemples

L’ordre SQL suivant affiche le nom et la date d’embauche pour tous les employés. Les dates d’embauches apparaîtront comme 17 June 1987.

```
SELECT last_name,  
       TO_CHAR(hire_date, 'fmDD Month YYYY') AS HIREDATE  
FROM   employees;
```

L’exemple précédent a été modifié pour afficher les dates au format : “Seventeenth of June 1987 12:00:00 AM.”

```
SELECT last_name,  
       TO_CHAR(hire_date, 'fmDdspth "of" Month YYYY fmHH:MI:SS AM')  
HIREDATE  
FROM   employees;
```

Remarquez que Month respecte le modèle de format : la première lettre est en majuscule et les autres en minuscule.

Utilisation de la fonction TO_CHAR avec des nombres.

Quand vous travaillez avec des nombres comme des chaînes de caractères, vous devez convertir les valeurs numériques en type de données caractères en utilisant la fonction TO_CHAR. TO_CHAR transforme une valeur de type NUMBER en VARCHAR2. Cette technique est très pratique lors des concaténations.

```
TO_CHAR(number, 'format_model')
```

Eléments du format de nombre :

Si vous convertissez un nombre en type de donnée caractère, vous pouvez utiliser les formats suivants :

Elément	Description	Exemple	Résultat
9	Détermine le nombre de chiffres affichés (le nombre de 9 définit la longueur maximale à l’affichage).	999999	1234
0	Force l’affichage des zéros au début.	099999	001234

Elément	Description	Exemple	Résultat
\$	Le signe dollar flottant (\$).	\$999999	\$1234
L	Symbole monétaire local flottant.	L999999	FF1234
D	Retourne le séparateur décimal dans la position spécifiée. Le point (.) par défaut.	99D99	99.99
.	Retourne le séparateur décimal point (.) dans la position spécifiée.	999999.99	1234.00
G	Retourne le séparateur de groupes dans la position spécifiée. Vous pouvez spécifier plusieurs séparateurs de groupes dans un modèle de format de nombre.	9G999	1,234
,	Virgule (,) dans la position spécifiée.	999,999	1,234
MI	Ajoute le signe moins (–) à droite (pour les valeurs négatives).	999999MI	1234-
PR	Entoure les nombres négatifs de parenthèses.	999999PR	<1234>
EEEE	Notation scientifique (le format doit contenir quatre E)	99.999EEEE	1.234E+03
U	Retourne à la position spécifiée le symbole monétaire "Euro" (ou autre)	U9999	€1234
V	Multiplie par 10 ⁿ fois (n = nombre de chiffres 9 après V).	9999V99	123400
S	Retourne la valeur négative ou positive.	S9999	-1234 ou +1234
B	Affiche un espace à la place des valeurs égales à zéro, pas 0.	B9999.99	1234.00

Règles:

- Le serveur Oracle affiche une chaîne de # à la place de nombre dont le nombre de chiffres est supérieur au nombre de chiffres fourni dans le model de format.
- Le serveur Oracle arrondit la valeur décimale stockée au nombre de décimales fournies dans le model de format.

Exemple:

L'exemple suivant affiche le salaire de l'employé Ernst. Le signe dollar, un séparateur de milliers et les cents sont affichés.

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY
FROM   employees
WHERE  last_name = 'Ernst';
```

Utilisation de la fonction TO_NUMBER et TO_DATE

Vous pouvez convertir une chaîne de caractères en un nombre ou en une date. Pour accomplir cette tâche, utilisez la fonction TO_NUMBER ou TO_DATE. Le modèle de format que vous choisissez est basé sur les éléments de format décrit précédemment.

Conversion d'une chaîne de caractères en un nombre en utilisant la fonction TO_NUMBER :

```
TO_NUMBER(char[, 'format_model'])
```

Conversion d'une chaîne de caractère en une date en utilisant la fonction TO_DATE :

```
TO_DATE(char[, 'format_model'])
```

Ces fonctions ont un modificateur fx. Ce modificateur spécifie la correspondance exacte de la chaîne caractères passée en argument au modèle de format de date de la fonction TO_DATE :

- La ponctuation et le texte entre guillemets dans la chaîne de caractère passée en argument doit correspondre exactement (sauf pour la casse) au modèle de format.
- La chaîne de caractères ne peut pas avoir d'espaces supplémentaires. Sans fx, Oracle ignore les espaces supplémentaires.
- Les données numériques dans la chaîne de caractère doivent avoir le même nombre de digit que dans le modèle de format utilisé. Sans fx, les zéros au début des nombres dans la chaîne de caractères sont ignorés.

L'exemple suivant est sensé d'afficher le nom et la date d'embauche pour tous les employés qui ont commencé à travailler le 24 mai 1999. Du fait que le modificateur fx est utilisé, une correspondance exacte est requise et les espaces après le mot May ne sont pas reconnus :

```
SELECT last_name, hire_date
FROM   employees
WHERE  hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY');
```

```
WHERE hire_date = TO_DATE('May 24, 1999', 'fxMonth DD, YYYY')
      *
```

ERROR at line 3:

ORA-01858: a non-numeric character was found where a numeric was expected

Format de Date RR

Le format de date RR est similaire à l'élément YY, mais vous pouvez l'utiliser pour spécifier différents siècles. Utilisez le format de date RR à la place de YY pour que le siècle de retour varie selon l'année spécifiée sur deux chiffres et les deux derniers chiffres de l'année en cours. Le tableau ci-dessous récapitule le comportement de l'élément RR.

Année en cours	Date donnée	Interprétée (RR)	Interprétée (YY)
1994	27-OCT-95	1995	1995
1994	27-OCT-17	2017	1917
2001	27-OCT-17	2017	2017
Si l'année spécifiée sur deux chiffres est :			
		0-49	50-99
Si les deux chiffres de l'année courante	0-49	La date de retour est dans le siècle en cours	La date de retour est dans le siècle précédent le siècle en

Année en cours	Date donnée	Interprétée (RR)	Interprétée (YY)
sont :			cours
	50-99	La date de retour est dans le siècle suivant le siècle en cours	La date de retour est dans le siècle en cours

Exemple:

Pour récupérer des employés dont la date d'embauche est inférieure à 1990, le format RR peut être utilisé. Du fait que l'année en cours est supérieure à 1999, le format RR interprète la portion de l'année de la date de 1950 à 1999.

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-YYYY')
FROM employees
WHERE hire_date < TO_DATE('01-Jan-90', 'DD-Mon-RR');
```

La commande suivante, au contraire, ne retourne aucune ligne car le format YY interprète la portion de l'année de la date dans le siècle en cours (2090).

```
SELECT last_name, TO_CHAR(hire_date, 'DD-Mon-yyyy')
FROM employees WHERE TO_DATE(hire_date, 'DD-Mon-yy') < '01-Jan-1990';
```

no rows selected.

Fonctions imbriquées

Les fonctions single-row peuvent être imbriquées sans limite. Les fonctions imbriquées sont évaluées de la fonction la plus imbriquée à la fonction la moins imbriquée.

Les exemples suivants montrent la flexibilité de ces fonctions :

```
SELECT last_name,
       UPPER (CONCAT (SUBSTR (LAST_NAME, 1, 8), '_US'))
FROM employees
WHERE department_id = 60;
```

Cet exemple affiche les noms des employés qui travaillent dans le département 60.

L'évaluation de l'ordre SQL implique trois étapes :

- La fonction la plus imbriquée récupère les huit premiers caractères du nom. Result1 = SUBSTR (LAST_NAME, 1, 8)
- La fonction moins imbriquée concatène le résultat avec _US. Result2 = CONCAT (Result1, '_US')
- La fonction la plus à l'extérieure convertit le résultat en majuscule. FinalResult = UPPER (Result2)

L'expression entière devient l'entête de colonne car aucun alias n'a été précisé.

Cet exemple suivant affiche la date du prochain vendredi qui est six mois plus tard que la date d'embauche. La date résultante a la forme suivante : Friday, August 13th, 1999. Les résultats sont triés par date d'embauche.

```
SELECT      TO_CHAR(NEXT_DAY(ADD_MONTHS(hire_date, 6), 'FRIDAY'), 'fmDay,  
Month DDth, YYYY') "Next 6 Month Review"  
FROM        employees  
ORDER BY hire_date;
```

Les fonctions générales

Présentation

Les fonctions suivantes marchent avec n'importe quel type de données et concernent l'utilisation des valeurs NULL :

Fonction	Description
NVL(expr1, expr2)	Convertit une valeur NULL en une valeur voulue
NVL2(expr1, expr2, expr3) expr3)	Si expr1 n'est pas NULL, NVL2 retourne expr2. Si expr1 est NULL, NVL2 retourne expr3. L'argument expr1 peut prendre n'importe quel type de données
NULLIF(expr1, expr2)	Compare deux expressions, retourne NULL si elles sont égales et la première expression si elles ne le sont pas.
COALESCE(expr1, expr2, ..., expr3) ..., exprn)	Retourne la première expression non NULL dans la liste des expressions.

La fonction NVL

Pour convertir une valeur NULL en une valeur spécifiée, utilisez la fonction NVL.

Syntaxe :

```
NVL (expr1, expr2)
```

Dans la syntaxe :

- expr1 est la valeur ou l'expression source qui peut contenir une valeur NULL
- expr2 est la valeur cible pour la conversion de la valeur NULL

Vous pouvez utiliser la fonction NVL pour convertir n'importe quel type de données, mais la valeur de retour doit être de même type de données qu'expr1.

Conversions NVL pour différents types de données :

Type de données	Exemple de conversion
-----------------	-----------------------

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:

Pour calculer la compensation annuelle de tous les employés, vous devez multiplier le salaire mensuel par 12 et ajouter le pourcentage de commission au résultat :

```
SELECT last_name, salary, commission_pct,
       (salary*12) + (salary*12*commission_pct) AN_SAL
FROM   employees;
```

Remarquez que la compensation annuelle est calculée seulement pour les employés qui touchent une commission. Si la valeur d'une colonne de l'expression est NULL alors le résultat est NULL. Pour calculer les valeurs pour tous les employés, vous devez convertir la valeur NULL en un nombre avant d'appliquer l'opérateur arithmétique.

Dans l'exemple suivant, la fonction NVL est utilisée pour convertir les valeurs NULL en 0.

```
SELECT last_name, salary, NVL(commission_pct,0),
       (salary*12) + (salary*12*NVL(commission_pct,0)) AN_SAL
FROM   employees;
```

La fonction NVL2

La fonction NVL2 examine la première expression. Si la première expression n'est pas NULL, alors la fonction NVL2 retourne la seconde expression. Si la première expression est NULL, alors la troisième expression est retournée.

Syntaxe :

```
NVL2(expr1, expr2, expr3)
```

Dans la syntaxe :

- expr1 est la valeur source ou l'expression qui pourrait être NULL
- expr2 est la valeur retournée si expr1 n'est pas NULL
- expr3 est la valeur qui est retournée si expr1 est NULL

L'argument expr1 peut être de n'importe quel type de données. Les arguments expr2 et expr3 peuvent être de n'importe quel type de donnée sauf LONG. Si le type de donnée est différent entre expr2 et expr3, le serveur Oracle convertit expr3 au type de donnée d'expr2 avant de les comparer sauf si expr3 est une constante NULL. Dans le dernier cas, une conversion de type de données n'est pas nécessaire. Le type de données de la valeur de retour est toujours le même que le type de donnée d'expr2 à moins que expr2 soit une donnée caractère, dans ce cas, la valeur de retour est de type VARCHAR2.

Exemple:

```
SELECT last_name, salary, commission_pct,
       NVL2(commission_pct, 'SAL+COMM', 'SAL') income
FROM   employees
WHERE  department_id IN (50, 80);
```

Dans l'exemple ci-dessus, la colonne COMMISSION_PCT est examinée. Si la valeur n'est pas NULL, la seconde expression SAL+COMM est retournée. Si la valeur de la colonne est NULL, alors la troisième expression SAL est retournée.

La fonction NULLIF

La fonction NULLIF compare deux expressions. Si elles sont égales, la fonction retourne NULL. Si elles ne le sont pas, la fonction retourne la première expression. Vous ne pouvez pas spécifier une valeur littérale NULL pour la première expression.

Syntaxe :

```
NULLIF (expr1, expr2)
```

Dans la syntaxe :

- expr1 est la valeur source comparée à expr2
- expr2 est la valeur source comparée à expr1 (Si elle n'est pas égale à expr1, expr1 est retournée)

Exemple :

```
SELECT first_name, LENGTH(first_name) "expr1",
       last_name, LENGTH(last_name) "expr2",
       NULLIF(LENGTH(first_name), LENGTH(last_name)) result
FROM   employees;
```

Dans cet exemple, la longueur du prénom dans la table EMPLOYEES est comparée à la longueur du nom dans la table EMPLOYEES. Si les longueurs sont égales, une valeur NULL est affichée. Si elles ne sont pas égales la longueur du prénom est affichée.

La fonction NULLIF est logiquement équivalente à l'expression CASE suivante.
L'expression CASE est décrite plus loin.

```
CASE
  WHEN expr1 = expr2
  THEN NULL
  ELSE expr1
END
```

La fonction COALESCE

La fonction COALESCE retourne la première expression non NULL dans la liste d'arguments.

Syntaxe :

`COALESCE (expr1, expr2, ... exprn)`

Dans la syntaxe :

- `expr1` retourne cette expression si elle n'est pas NULL.
- `expr2` retourne cette expression si la première expression est NULL et celle-ci ne l'est pas.
- `exprn` retourne cette expression si toutes les expressions précédentes sont NULL

Toutes les expressions doivent être du même type de données.

Exemple :

```
SELECT    last_name,  
          COALESCE(manager_id, commission_pct, -1) comm  
FROM      employees  
ORDER BY  commission_pct;
```

Dans cet exemple, si la valeur de `MANAGER_ID` n'est pas NULL, elle est affichée. Si la valeur de `MANAGER_ID` est NULL, la valeur de `COMMISSION_PCT` est affichée. Si les valeurs de `MANAGER_ID` et `COMMISSION_PCT` sont NULL, alors la valeur -1 est affichée.

Les expressions conditionnelles

Présentation

Deux méthodes sont utilisées pour implémenter un traitement conditionnel (de type IF-THEN-ELSE) dans un ordre SQL :

- l'expression **CASE**
- la fonction **DECODE**

L'expression **CASE** est conforme au standard ANSI SQL. La fonction **DECODE** est propre à la syntaxe Oracle.

L'expression CASE

L'expression **CASE** vous permet de reproduire la logique IF-THEN-ELSE dans un ordre SQL sans avoir à appeler des procédures.

Syntaxe :

```
CASE expr  
  WHEN comparison_expr1 THEN return_expr1
```

```

[WHEN comparison_expr2 THEN return_expr2
 WHEN comparison_exprn THEN return_exprn
 ELSE else_expr]
END

```

Dans une expression **CASE** simple, le serveur Oracle cherche la première paire **WHEN ... THEN** pour laquelle la valeur d'*expr* est égale à la valeur de *comparison_expr* et retourne *return_expr*.

Si aucune paire **WHEN ... THEN** ne satisfait la condition et si une clause **ELSE** existe, alors le serveur Oracle retourne *else_expr*. Autrement, le serveur Oracle retourne une valeur **NULL**. Vous ne pouvez pas utiliser la valeur littérale **NULL** pour toutes les expressions *return_expr* et l'expression *else_expr*.

Toutes les expressions (*expr*, *comparison_expr* et *return_expr*) doivent être du même type de données. Le type de données peut être **CHAR**, **VARCHAR2**, **NCHAR** ou **NVARCHAR2**.

Exemple :

```

SELECT last_name, job_id, salary,
       CASE job_id
         WHEN 'IT_PROG' THEN 1.10*salary
         WHEN 'ST_CLERK' THEN 1.15*salary
         WHEN 'SA_REP' THEN 1.20*salary
         ELSE salary
       END "REVISED_SALARY"
FROM   employees;

```

Dans l'ordre SQL de l'exemple, la valeur de **JOB_ID** est évaluée. Si sa valeur est **IT_PROG**, le salaire de l'employé est augmenté de 10% ; si elle est égale à **ST_CLERK**, le salaire est augmenté de 15% ; si la valeur est égale à **SA_REP**, le salaire est augmenté de 20%. Le salaire n'est pas modifié pour les autres postes.

Le même ordre peut être écrit avec la fonction **DECODE**.

L'exemple ci-dessous montre une autre syntaxe de la clause **CASE** appelée *searched CASE*. Dans une expression *searched CASE*, la recherche se produit de gauche à droite jusqu'à ce qu'une occurrence de la condition énumérée soit évaluée à vrai, alors l'instruction correspondante est retournée. Si aucune condition n'est évaluée à **TRUE** et si une clause **ELSE** existe, l'expression de la clause **ELSE** est retournée ; autrement une valeur **NULL** est retournée.

```

SELECT last_name, salary,
       (CASE WHEN salary<5000 THEN 'Low'
            WHEN salary<10000 THEN 'Medium'
            WHEN salary<20000 THEN 'Good'
            ELSE 'Excellent'
          END) qualified_salary
FROM   employees;

```

La fonction DECODE

La fonction **DECODE** fonctionne d'une manière similaire à la logique IF-THEN-ELSE qui est utilisée dans plusieurs langages.

```
DECODE(col|expression, search1, result1
      [, search2, result2,...,]
      [, default])
```

La fonction **DECODE** compare *expression* à chaque valeur *search*. Si l'expression est identique à la valeur *search*, la fonction retourne la valeur *result*. Si la valeur par défaut n'est pas définie, la fonction renvoie une valeur **NULL** lorsqu'aucune comparaison n'est validée.

Exemple :

```
SELECT last_name, job_id, salary,
       DECODE(job_id, 'IT_PROG', 1.10*salary,
                  'ST_CLERK', 1.15*salary,
                  'SA_REP', 1.20*salary,
                  salary) REVISED_SALARY
FROM   employees;
```

Dans l'ordre SQL de l'exemple, la valeur de **JOB_ID** est évaluée. Si sa valeur est **IT_PROG**, le salaire de l'employé est augmenté de 10% ; si elle est égale à **ST_CLERK**, le salaire est augmenté de 15% ; si la valeur est égale à **SA_REP**, le salaire est augmenté de 20%. Le salaire n'est pas modifié pour les autres postes.

La même instruction peut être écrite en pseudo-code IF-THEN-ELSE :

```
IF job_id = 'IT_PROG' THEN salary = salary*1.10
IF job_id = 'ST_CLERK' THEN salary = salary*1.15
IF job_id = 'SA_REP' THEN salary = salary*1.20
ELSE salary = salary
```

L'exemple ci-dessous montre un autre cas d'utilisation de la fonction **DECODE**. Le taux d'imposition est déterminé pour chaque employé du département 80 en se basant sur son salaire mensuel.

```
SELECT last_name, salary,
       DECODE (TRUNC(salary/2000, 0),
              0, 0.00,
              1, 0.09,
              2, 0.20,
              3, 0.30,
              4, 0.40,
              5, 0.42,
              6, 0.44,
              0.45) TAX_RATE
FROM   employees
WHERE  department_id = 80;
```

Rang de salaire mensuel	Taux
\$0.00 - 1,999.99	00%
\$2,000.00 - 3,999.99	09%

Rang de salaire mensuel	Taux
\$4,000.00 - 5,999.99	20%
\$6,000.00 - 7,999.99	30%
\$8,000.00 - 9,999.99	40%
\$10,000.00 - 11,999.99	42%
\$12,200.00 - 13,999.99	44%
\$14,000.00 ou supérieur	45%

Générer des agrégats avec les fonctions de groupe

Présentation

Différentes des fonctions single-row, les fonctions de groupe opèrent sur un ensemble de lignes pour donner un résultat par groupe. Ces ensembles peuvent comprendre la table entière ou bien la table découpée en groupe.

Différents types de fonctions de groupe

Chacune de ces fonctions accepte un argument. Le tableau suivant identifie les options que vous pouvez utiliser.

Fonction	Description
AVG ([DISTINCT ALL] n)	Retourne la moyenne des valeurs de <i>n</i> , ignore les valeurs NULL .
COUNT ({* [DISTINCT ALL] expr})	Retourne le nombre d'enregistrements pour <i>expr</i> différents de NULL . COUNT (*) retourne le nombre total d'enregistrements retournés en incluant les valeurs NULL et les doublons.
MAX ([DISTINCT ALL] expr)	Retourne la plus grande valeur de l' <i>expr</i> , ignore les valeurs NULL .
MIN ([DISTINCT ALL] expr)	Retourne la plus petite valeur de l' <i>expr</i> , ignore les valeurs NULL .
STDDEV ([DISTINCT ALL] n)	Retourne la variance standard de <i>n</i> , ignore les valeurs NULL .
SUM ([DISTINCT ALL] n)	Retourne la somme de toutes les valeurs du <i>n</i> , ignore les valeurs NULL , ignore les valeurs NULL .
VARIANCE ([DISTINCT ALL] n)	Retourne la variance de <i>n</i> , ignore les valeurs NULL .

- **AVG**
- **COUNT**
- **MAX**
- **MIN**
- **STDDEV**
- **SUM**
- **VARIANCE**

Toutes ces fonctions de groupe ignorent les valeurs **NULL** sauf **COUNT(*)**.

Syntaxe des fonctions de groupe

```
SELECT [column,] group_function(column), ...  
FROM table  
[WHERE condition]  
[GROUP BY column]  
[ORDER BY column];
```

- **DISTINCT** indique que la fonction ne prendra pas en compte les valeurs dupliquées.
ALL indique que la fonction devra considérer toutes les valeurs incluant les doublons.
ALL étant par défaut, vous n'avez pas besoin de le spécifier.
- Le type de données des arguments *expr* peut être **CHAR**, **VARCHAR2**, **NUMBER** ou **DATE**.
- Toutes ces fonctions de groupe ignorent les valeurs nulles. Pour substituer les valeurs **NULL** dans un groupe, utilisez les fonctions **NVL**, **NVL2** ou **COALESCE**.

Utilisation des fonctions AVG et SUM

Vous pouvez utiliser les fonctions **AVG** et **SUM** pour les données numériques.

L'exemple affiche la moyenne de salaires mensuels, le salaire mensuel le plus élevé et le plus bas ainsi que la somme de salaires mensuels pour tous les représentants de ventes.

```
SELECT AVG(salary), MAX(salary),  
       MIN(salary), SUM(salary)  
FROM employees  
WHERE job_id LIKE '%REP%';
```

Utilisation des fonctions MIN et MAX

Vous pouvez utiliser les fonctions **MAX** et **MIN** pour les données de type numérique, caractères et date. Cet exemple affiche le plus récent et le plus ancien des employés.

```
SELECT MIN(hire_date), MAX(hire_date)  
FROM employees;
```

L'exemple suivant affiche le nom du premier et dernier employé de la liste de tous les employés triés par ordre alphabétique.

```
SELECT MIN(last_name), MAX(last_name)  
FROM employees;
```

Les fonctions **AVG**, **SUM**, **VARIANCE** et **STDDEV** ne peuvent être utilisées qu'avec des données numériques. **MAX** et **MIN** ne peuvent pas être utilisés avec les données de type **LOB** ou **LONG**.

Utilisation de la fonction COUNT

La fonction **COUNT** a trois formats :

- **COUNT (*)**
- **COUNT (expr)**
- **COUNT (DISTINCT expr)**

COUNT (*) renvoie le nombre de lignes dans une table qui satisfont les critères de la requête **SELECT**, en prenant en compte les doublons et les lignes contenant des valeurs **NULL** dans les colonnes. Si la clause **WHERE** est incluse dans la requête **SELECT**, **COUNT (*)** renvoie le nombre de lignes qui satisfont la condition dans la clause **WHERE**.

En revanche, **COUNT (expr)** renvoie le nombre de valeurs non **NULL** qui sont dans la colonne identifiée par l'*expr*. **COUNT (DISTINCT expr)** renvoie le nombre de valeurs uniques et non **NULL** qui sont dans la colonne spécifiée par l'*expr*.

Exemples:

L'exemple suivant affiche le nombre d'employés qui sont dans le département 50.

COUNT(*) retourne le nombre de lignes de la table satisfaisant les critères de la recherche:

```
SELECT COUNT(*)
FROM employees
WHERE department_id = 50;
```

L'exemple suivant affiche le nombre d'employés du département 80 qui touchent une commission.

COUNT(expr) retourne le nombre de lignes avec des valeurs non **NULL** pour l'*expr* :

```
SELECT COUNT(commission_pct)
FROM employees
WHERE department_id = 80;
```

Utilisation du mot-clé DISTINCT

Utilisez le mot-clé **DISTINCT** pour supprimer les doublons dans la colonne.

COUNT (DISTINCT expr) retourne le nombre de valeurs distincts et non **NULL** de l'*expr*.

L'exemple suivant affiche les numéros de département distincts présents dans la table **EMPLOYEES**.

```
SELECT COUNT(DISTINCT department_id)
FROM employees;
```

Les fonctions de groupe et les valeurs NULL

Toutes les fonctions de groupes ignorent les valeurs **NULL** dans la colonne.

La fonction **NVL** force les fonctions de groupe à prendre en compte les valeurs **NULL**.

Exemples :

La moyenne n'est calculée que pour les lignes de la table où la colonne **COMMISSION_PCT** contient une valeur valide. La moyenne est calculée comme le total des commissions qui a été payé à tous les employés, divisée par le nombre des employés ayant reçu une commission (quatre).

Les fonctions de groupe ignorent les valeurs **NULL** dans la colonne :

```
SELECT AVG(commission_pct)
FROM employees;
```

La moyenne est calculée à partir de toutes les lignes de la table sans prendre en compte les valeurs **NULL** stockées dans la colonne **COMMISSION_PCT**. La moyenne est calculée comme le total de commissions payées à tous les employés, divisée par le nombre total des employés dans l'entreprise (20).

La fonction **NVL** force les fonctions de groupe à prendre en compte les valeurs **NULL** :

```
SELECT AVG(NVL(commission_pct, 0))
FROM employees;
```

Création de groupe de données

Jusque là, toutes les fonctions de groupe ont traité les tables comme un grand groupe d'informations.

Parfois, vous aurez besoin de diviser les informations de la table en plus petits groupes. Ceci peut être fait en employant la clause **GROUP BY**.

Clause **GROUP BY**

Vous pouvez utiliser la clause **GROUP BY** pour diviser les enregistrements d'une table en plusieurs groupes. Les fonctions de groupe peuvent alors être utilisées pour retourner les informations relatives à chaque groupe

- Si vous incluez une fonction de groupe dans une clause **SELECT**, vous ne pouvez pas sélectionner les résultats individuels. Les colonnes n'apparaissant pas dans les fonctions de groupes doivent être présentes dans la clause **GROUP BY**. Vous recevez un message d'erreur si vous n'incluez pas la liste de colonne dans la clause **GROUP BY**.
- La clause **WHERE** peut être utilisée pour pré-exclure des enregistrements avant la division en groupes.
- Vous devez inclure les colonnes dans la clause **GROUP BY**.
- Vous ne pouvez pas utiliser des alias de colonne dans la clause **GROUP BY**.
- Vous pouvez diviser les enregistrements d'une table en plus petits groupes en utilisant la clause **GROUP BY**.

```
SELECT column, group_function(column)
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[ORDER BY column];
```

Dans la syntaxe :

group_by_expression indique les colonnes dont les valeurs déterminent la base des regroupements de lignes

En utilisant la clause **GROUP BY**, assurez-vous que toutes les colonnes dans la liste **SELECT** qui ne sont pas des fonctions de groupes soient bien incluses dans la clause **GROUP BY**.

L'exemple suivant montre le numéro de département et le salaire moyen pour chaque département.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

Voici comment la requête **SELECT** contenant une clause **GROUP BY** est évaluée :

- La clause **SELECT** indique les colonnes à rechercher, comme suit :
 - La colonne du numéro de département dans la table **EMPLOYEES**.
 - La moyenne de tous salaires dans le groupe que vous avez indiqués dans la clause **GROUP BY**.
- La clause **FROM** indique les tables auxquelles le serveur Oracle doit accéder : ici, la table **EMPLOYEES**.
- La clause **WHERE** indique les lignes à récupérer. Puisqu'il n'y a aucune clause **WHERE**, toutes les lignes sont retournées.
- La clause **GROUP BY** indique comment les lignes doivent être groupées. Les lignes sont groupées par numéro de département, ainsi la fonction **AVG**, qui est appliquée à la colonne de salaire, calcule le salaire moyen pour chaque département.

La colonne contenue dans la clause **GROUP BY** ne doit pas obligatoirement être présente dans la clause **SELECT**.

L'exemple suivant affiche la moyenne de salaires pour chaque département sans afficher le numéro de département respectif.

```
SELECT AVG(salary)
FROM employees
GROUP BY department_id;
```

Vous pouvez utiliser une fonction de groupe dans la clause **ORDER BY**.

L'exemple suivant affiche la moyenne de salaires pour chaque département. Les résultats sont triés par les moyennes de salaires.

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
ORDER BY AVG(salary);
```

Les groupes imbriqués

Parfois, vous avez besoin de voir les résultats des groupes à l'intérieur des groupes. Cet exemple montre un rapport détaillé affichant le salaire total qui est payé pour chaque poste dans chaque département.

La table **EMPLOYEES** est groupée en premier lieu par numéro de département et ensuite par l'identifiant de poste à l'intérieur du premier groupe. Par exemple, les quatre **ST_CLERK** dans le département 50 sont regroupés ensemble et un seul résultat (le salaire total) est produit pour tous les **ST_CLERK** dans le groupe.

Vous pouvez obtenir des résumés de résultats pour les groupes et les sous-groupes en spécifiant plus d'une colonne dans la clause **GROUP BY**. L'ordre de tri par défaut des résultats correspond à l'ordre des colonnes dans la clause **GROUP BY**.

```
SELECT department_id dept_id, job_id, SUM(salary)
FROM employees
GROUP BY department_id, job_id;
```

Dans cet exemple précédant, la requête **SELECT** contient une clause **GROUP BY** qui est évalué ainsi :

- La clause **SELECT** indique la colonne à récupérer :
 - Le numéro de département de la table **EMPLOYEES**.
 - L'identifiant de poste de la table **EMPLOYEES**.
 - La somme de tous les salaires dans le groupe que vous avez indiqué dans la clause **GROUP BY**.
- La clause **FROM** indique les tables auxquelles la base de données devra accéder : la table **EMPLOYEES**.
- La clause **GROUP BY** indique comment les lignes doivent être regroupées :
 - Tout d'abord, les lignes sont groupées par le numéro de département.
 - Ensuite, les lignes sont groupées par le l'identifiant du poste dans le groupe des numéros de département.

Ainsi la fonction **SUM** est appliquée à la colonne salaire pour tous les identifiants de poste dans chaque groupe de numéro de département.

Les requêtes invalides utilisant les fonctions de groupe

A chaque fois que vous utilisez un mélange d'éléments individuels (par exemple **DEPARTMENT_ID**) et des fonctions de groupe (**COUNT**) dans la même requête, vous devez inclure les éléments individuels (dans cet exemple **DEPARTMENT_ID**) dans la clause **GROUP BY**. Si la clause **GROUP BY** est absente, alors un message d'erreur "not a single-group group function" apparaîtra et un astérisque (*) pointera sur la colonne concernée.

```
SELECT department_id, COUNT(last_name)
FROM employees;
SELECT department_id, COUNT(last_name)
```

```
*  
ERROR at line 1:  
ORA-00937: not a single-group group function
```

Vous pouvez corriger cette erreur en ajoutant la clause **GROUP BY**.

```
SELECT department_id, count(last_name)  
FROM employees  
GROUP BY department_id;
```

Les colonnes n'apparaissant pas dans les fonctions de groupes doivent être présentes dans la clause **GROUP BY**.

Si vous voulez restreindre les groupes vous devez utiliser la clause **HAVING**, la clause **WHERE** ne peut pas être utilisée pour restreindre les groupes.

L'exemple suivant retourne une erreur parce que la clause **WHERE** est utilisée pour restreindre des moyennes de salaire de tous les départements qui ont un salaire moyen supérieur à 8000\$.

```
SELECT department_id, AVG(salary)  
FROM employees  
WHERE AVG(salary) > 8000  
GROUP BY department_id;  
WHERE AVG(salary) > 8000  
*  
ERROR at line 3:  
ORA-00934: group function is not allowed here
```

Vous pouvez corriger l'erreur dans l'exemple précédent en utilisant la clause **HAVING** pour restreindre les groupes :

```
SELECT department_id, AVG(salary)  
FROM employees  
HAVING AVG(salary) > 8000  
GROUP BY department_id;
```

Restreindre le résultat des groupes

De la même manière que vous utilisez la clause **WHERE** pour restreindre les lignes que vous sélectionnez, vous pouvez utiliser la clause **HAVING** pour restreindre les groupes. Pour trouver le salaire maximum de chaque département ayant un salaire maximum supérieur à 10 000\$, vous avez besoin de réaliser les étapes suivantes :

1. Trouver le salaire moyen pour chaque département en les regroupant par numéro du département.
2. Limiter les groupes aux départements qui ont un salaire maximum supérieur à 10 000\$.

Restreindre les résultats des groupes avec la clause **HAVING**

Vous utilisez la clause **HAVING** pour spécifier quels groupes seront affichés.

Le serveur Oracle réalise les étapes suivantes lorsque vous utilisez la clause **HAVING** :

1. Les lignes sont groupées.
2. La fonction de groupe est appliquée.
3. Les groupes qui correspondent aux critères de la clause **HAVING** sont affichés.

La clause **HAVING** peut précéder la clause **GROUP BY** mais il est plus logique de placer la clause **GROUP BY** avant. Les groupes sont formés et les fonctions de groupe sont calculées avant que la clause **HAVING** soit appliquée aux groupes de la liste de **SELECT**.

```
SELECT column, group_function
FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING group_condition]
[ORDER BY column];
```

Dans la syntaxe :

group_condition restreint les groupes de lignes retournées aux groupes de lignes qui correspondent à la condition spécifiée.

Utilisation de la clause HAVING

Vous pouvez utiliser la clause **HAVING** sans utiliser une fonction de groupe dans la liste de **SELECT**.

Si vous limitez les lignes basées sur le résultat d'une fonction de groupe, vous devez avoir la clause **GROUP BY** aussi bien que la clause **HAVING**.

L'exemple suivant affiche le numéro de département et le salaire maximum des départements dont le salaire maximum est supérieur à 10 000\$.

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary)>10000;
```

L'exemple suivant affiche le numéro de département et la moyenne des salaires des départements qui ont un salaire maximum plus grand que 10 000\$:

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary)>10000;
```

L'exemple suivant affiche l'identifiant de poste et le salaire total mensuel pour chaque type de poste dont la rémunération totale excède 13 000\$. L'exemple exclut les commerciaux (**SA_REP**) et trie la liste par le salaire total mensuel.

```
SELECT job_id, SUM(salary) PAYROLL
FROM employees
WHERE job_id NOT LIKE '%REP%'
GROUP BY job_id
HAVING SUM(salary) > 13000
```

```
ORDER BY SUM(salary);
```

Les fonctions de groupe imbriquées

Les fonctions de groupes peuvent être imbriquées à deux niveaux.

L'exemple suivant affiche le maximum des salaires moyen.

```
SELECT MAX(AVG(salary))  
FROM employees  
GROUP BY department_id;
```