Les fonctions de lignes

1. *Les fonctions de caractères :*

LOWER(arg) : renvoie l'argument mais en minuscule.

UPPER(arg) : renvoie l'argument mais en majuscule.

INITCAP(arg) : met en majuscule la première lettre de chaque mot, les lettres restantes

CONCAT(arg1,arg2) : renvoie la concaténation des deux arguments. Elle est équivalente à l'opérateur ||.

SUBSTR(arg,m[,n]) : retourne la sous chaîne de l'argument qui commence de la même position et de longueur n. Si n est omise, alors la sous-chaîne renvoyée s'étend de la position m jusqu'à la fin de l'argument. Si m est négative, alors l'extraction se fait à partir de la fin de l'argument. Finalement, si n est relativement grande alors le traitement se fait comme si elle était omise.

LENGTH(arg) : retourne la taille de l'argument en nombre de caractères.

INSTR(arg,cc[,m][,n]) : retourne la position de cc dans l'argument en commençant par la position m et dont l'occurrence est n. Par défaut m et n sont mis à 1.

LPAD(arg,m[,cc]) : rembourre l'argument à gauche plusieurs fois par les caractères de cc jusqu'à ce que la taille de la chaîne retournée soit de m. Par défaut, cc est égale à l'espace.

RPAD(arg,m[,cc]) : rembourre l'argument à droite plusieurs fois par les caractères de cc jusqu'à ce que la taille de la chaîne retournée soit de m. Par défaut, cc est égale à l'espace.

LTRIM (arg,cc) : renvoie l'argument dont elle élague chaque caractère de cc si ce caractère se trouve à son extrémité gauche. Le résultat retourné ne doit contenir à sa gauche aucun caractère de cc.

RTRIM(arg,cc) : renvoie l'argument duquel elle élague chaque caractère de cc si ce caractère se trouve à son extrémité droite. Le résultat retourné ne doit contenir à sa droite aucun caractère de cc.

TRIM([LEADING|TRAILING|BOTH] cc FROM arg) : si LEADING est précisé alors la fonction est équivalente à LTRIM, si TRAILING est précisé alors la fonction est équivalente à RTRIM sinon avec BOTH, l'élagage se fait des deux cotés. BOTH est considéré par défaut.

REPLACE (arg,cc1[,cc2]) : remplace la sous chaîne cc_1 par la sous chaîne cc_2 dans l'argument. Si cc_2 est omise alors toutes les occurrences de cc1 sont enlevées de l'argument.

2. *Les fonctions numériques :*

ROUND(arg,n) : arrondit l'argument à la précision n, c-à-d à n chiffres décimaux. Si n=0, alors l'argument est arrondi sur les unités, sinon (cas où n<0) l'argument est arrondit sur le |n|+1ème chiffre à gauche de la virgule.

TRUNC(arg,n) : tronque l'argument à la précision n.

CEIL(arg) : retourne le plus petit entier supérieur à l'argument.

FLOOR(arg) : retourne le plus grand entier inférieur à l'argument.

POWER(arg,n) : arg à la puissance n.

EXP(arg) : exponentielle de arg.

SQRT(arg) : racine carré de arg.

SIGN(arg) : **-1** si col <0, **1** si col>0, **0** sinon.

ABS(arg) : valeur absolue de arg.

MOD(arg, n) : arg modulo n.

LOG(m,arg) : logarithme de base m de arg.

SIN(arg) : sinus de arg.

COS(arg) : cosinus de arg.

3. *Les fonctions de date :*

Le format de date par défaut est : DD-MON-YY. Ceci dit, Oracle stocke une date suivant un format numérique interne plus détaillé : siècle, année, mois, jour, heure, minute et seconde. Le pseudo colonne SYSDATE retourne la date système (inclut aussi l'heure système). Pour afficher l'heure système, on peut utiliser la table DUAL. Le propriétaire de cette table est l'utilisateur SYS, mais tous les utilisateurs peuvent y accéder.

Par exemple, pour connaître la date système, on peut exécuter cette requête :

SELECT SYSDATE FROM DUAL ;

On peut faire **des calculs arithmétiques** sur les DATE :

DATE1 + n : ajoute ou soustrait n jours de DATE1 (selon le signe de n) et retourne la date résultat.

DATE1 – DATE2 : retourne la différence entre les deux dates en nombre réel de jours. Le résultat est souvent un nombre réel car l'opération de soustraction tient compte des heures des deux dates.

DATE1 + n/24 : Ajoute n heures à la date.

Les **fonctions** de date sont les suivantes :

MONTHS_BETWEEN(date1,date2) : retourne en nombre réel la différence en mois entre les deux dates.

ADD_MONTHS(date1,n) : retourne la date qui vient n mois après date1.

NEXT_DAY(date1,char/n) : char∈{'Lundi','Mardi',…,'Dimanche'}, n∈{1,2,…,7}. Elle retourne le prochain jour char dans la semaine de date1.

LAST_DAY(date1) : retourne la date du dernier jour du mois de date1.

ROUND(date1[,fmt]) : retourne date1 arrondie selon le format *fmt.* Par exemple, si *fmt*='MM', alors l'arrondissement est fait sur le mois, le résultat retourné est la date du 1er jour du mois de date1 si DD≤15, et la date du 1er jour du mois suivant sinon. Si *fmt*='YY', alors le résultat retourné est la date du 1er jour de l'année de date1 si MM≤6, et la date du 1er jour de l'année suivante sinon. Si *fmt* est omise, alors date1 est arrondie au jour le plus proche. Lorsque *fmt* est omise, alors l'arrondissement est fait sur le jour, la fonction retourne la même date si l'heure est < à 12 :00, et la date du lendemain si l'heure est ≥ à 12 :00.

TRUNC(date1[,fmt]) : retourne date1 tronquée selon *fmt*.

4. *Les fonctions de conversion :*

Ce sont les fonctions de conversion entre types de données. Le serveur Oracle peut **convertir implicitement**, sans l'utilisation d'une fonction :

VARCHAR2/CHAR → NUMBER

VARCHAR2/CHAR → DATE

DATE → VARCHAR2/CHAR

NUMBER → VARCHAR2/CHAR

Ainsi, la concaténation des colonnes EMPNO et ENAME de la table EMP est possible sans convertir la colonne EMPLNO qui est numérique ; la conversion est faite de manière automatique.

La conversion explicite est faite à travers un ensemble de fonctions de conversion :

TO_CHAR(Date1,„fmt") : convertit une date en chaîne de caractères et l'affiche dans le format „fmt" indiqué.
Exemple : SELECT TO_CHAR(SYSDATE,„DD/MM/YYYY") FROM DUAL; →affiche : 22/10/2004

TO_CHAR(NOMBRE,„fmt") : convertit un nombre en chaîne de caractère dans le format spécifié.
Exemple : TO_CHAR(900,„$9.999") →affiche „$900.000"

TO_NUMBER(char,"fmt") : convertit une chaîne de caractère en nombre.
Exemple : TO_NUMBER(„$1500","$9999") → affiche 1500.

TO_DATE(char,"fmt") : convertit char en date selon le format de date mentionnée.
Exemple : TO_DATE("23/10/2004",„DD/MM/YYYY") →retourne la date dans le format lisible par Oracle, à savoir "23-OCT-04".

5. *Les fonctions opérant sur tous les types de données :*

DECODE(arg,SEARCH1,val1,[SEARCH2,val2,…],default_arg) :Cette fonction retourne vali si arg=SEARCHi, si aucune valeur SEARCHi ne correspond à l'argument arg, alors default_arg sera retournée.

NVL(arg,val) : retourne val si l'argument est NULL sinon la valeur de l'argument.

GREATEST(arg1,arg2,…) : retourne l'argument la plus grande. S'il s'agit de DATE, alors elle retourne la date la plus récente.

LEAST(arg1,arg2,…) : retourne la valeur la plus petite.

VSIZE(arg) : retourne en nombre de bytes (octets) la taille de la colonne ou de la valeur.

Les fonctions de groupe

1. *Les fonctions de groupe :*

AVG([DISTINCT|ALL] arg) : retourne la moyenne des arguments pour chaque groupe tout en ignorant les valeurs NULL.

COUNT(*|[DISTINCT]|[ALL] arg) : retourne pour chaque groupe le nombre de lignes où l'argument est différent de NULL. Pour retourner le nombre de lignes y compris les lignes dupliquées et ayant des champs NULL, on utilise COUNT(*).

MAX([DISTINCT|ALL] arg) : retourne la valeur maximale de l'argument pour chaque groupe.

MIN([DISTINCT|ALL] arg) : retourne la valeur minimale de l'argument pour chaque groupe.

SUM([DISTINCT]/[ALL] arg) : retourne la somme des valeurs de l'argument.

VARIANACE([DISTINCT]/[ALL] arg): retourne la variance des valeurs de l'argument.

STDDEV([DISTINCT]/[ALL] arg) : retourne l'écart type (standard deviation) des valeurs de l'argument.

Remarque : Toutes les fonctions de groupe ignorent les valeurs NULL.

2. *La clause GROUP BY*

Si nous avons besoin de diviser la table en petits groupes, la clause GROUP BY est utilisée. Cette clause est suivie d'une colonne. Les lignes qui sont incluses dans chaque groupe résultant ont la même valeur pour cette colonne.

Remarque : Il est à noter qu'on ne peut mettre dans la clause SELECT que les fonctions de groupe et/ou les colonnes qui suivent la clause GROUP BY.

3. *Groupe dans des groupes :*

Parfois, nous avons besoin d'informations concernant des groupes à l'intérieur de groupes. Pour ceci, il faudrait lister plus qu'une colonne dans la clause GROUP BY.

4. *La clause HAVING :*

Tout comme la clause WHERE qui sert à restreindre les lignes retournées, cette clause sert à restreindre les groupes retournés. La clause HAVING est suivie d'une condition simple ou composée qui concerne les colonnes qui suivent la clause GROUP BY, ou/et des fonctions de groupe.