Chapitre 13 : À propos d'Oracle10*g*

Ce chapitre décrit dans un premier temps la procédure d'installation d'Oracle10*g* *Release* 1 (10.1.0.2), puis les nouvelles fonctionnalités apparues dans le langage SQL. Nous ne détaillerons pas ici les nouvelles commandes relatives à l'administration, au *tuning* d'une base et aux extensions objet. Pour ces dernières, vous pouvez consulter [SOU 04] (voir fin du chapitre).

Mise en service d'Oracle10*g*

Installation

Si tout se passe bien, comptez moins d'une heure pour installer Oracle10*g* (selon la puissance de votre machine). La configuration requise minimale est un processeur Pentium IV et 512 Mo de RAM, les processus de la base occupant déjà 400 Mo. L'installation basique requiert 2 Go d'espace disque.

La procédure d'installation est plus simple que pour la version 9*i* qui nécessitait trois CD-Rom et davantage de menus. Le produit Oracle Database Oracle10*g* est réparti sur deux CD-Rom : le premier permet d'installer la base, la console d'administration et les interfaces de commande SQL*Plus, tandis que le second, appelé *companion*, contient les produits moins courants (précompilateurs, pilotes et diverses extensions).

Commencez tout d'abord par extraire le fichier zippé dans un répertoire temporaire, puis exécutez `setup.exe`.

Vous devez ensuite choisir le répertoire cible (choisissez un répertoire vide si vous n'utilisez pas celui proposé par Oracle) et le type du produit (ici *Personal*). Donnez un nom à la base (ici `bdcs10g`), et un mot de passe aux comptes système. L'installation avancée permet de paramétrer, entre autres, la langue. Nous décrivons ici la plus simple installation.

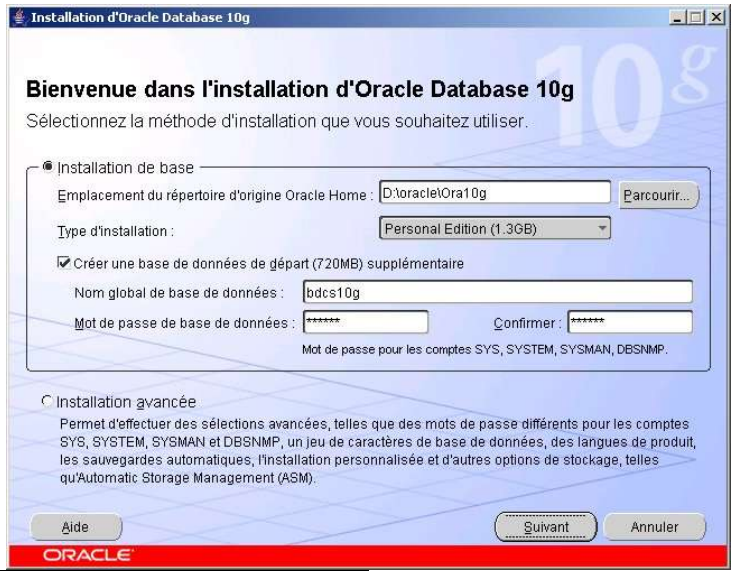


Figure 13-1 Répertoire cible d'installation

La fenêtre suivante offre la possibilité de faire migrer (*upgrade*) une base d'une précédente version dans la base 10*g* créée (ici, l'écran illustre l'existence d'une base Oracle9*i* de nom `BDCS`).

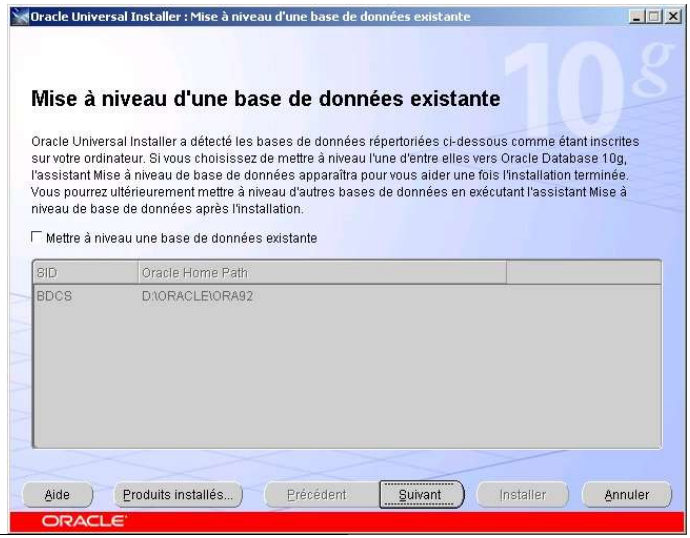


Figure 13-2 Migration éventuelle d'une ancienne base

Dans la fenêtre de résumé, Oracle vous informe sur la place qu'il va occuper : assurez-vous que vous disposez d'un espace disque suffisant. Si vous désirez installer d'autres produits absents de la liste, il faudra utiliser le deuxième CD-Rom (*companion*).

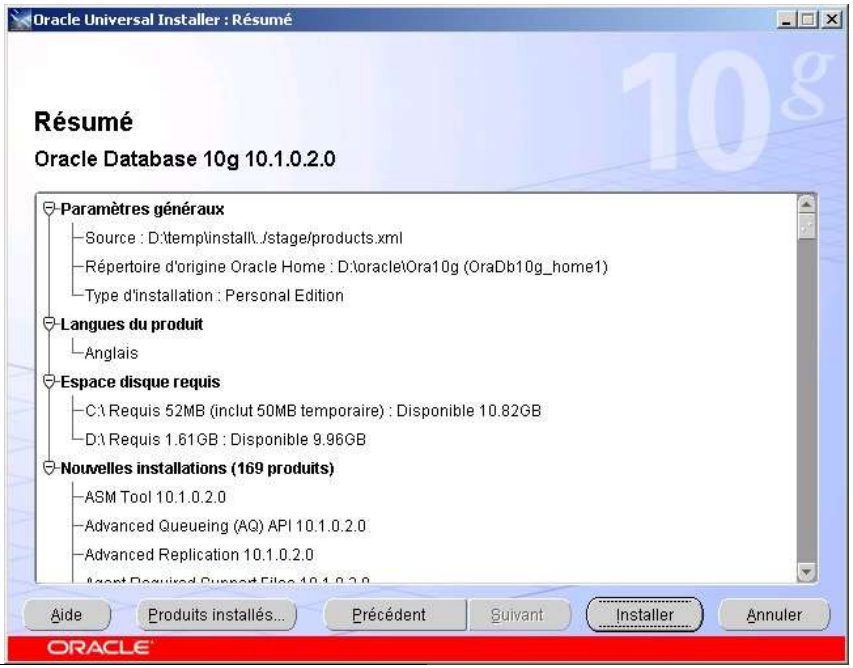


Figure 13-3 Résumé

De longues minutes vont s'écouler avant que vous ne puissiez modifier les mots des passe des comptes d'administration (ici, aucune modification n'a été faite). Personnellement, j'ai rencontré, à deux reprises, deux problèmes. À 43 % d'installation, m'a été signalé un échec de l'initialisation *Oracle Cluster Registry* : il suffit juste de cliquer sur `OK` et le processus redémarre. À 63 %, l'installation bloque de nouveau quelques minutes (échec du démarrage du service *OracleCSService*) : en cliquant sur `Réessayer`, le processus se remet en route.

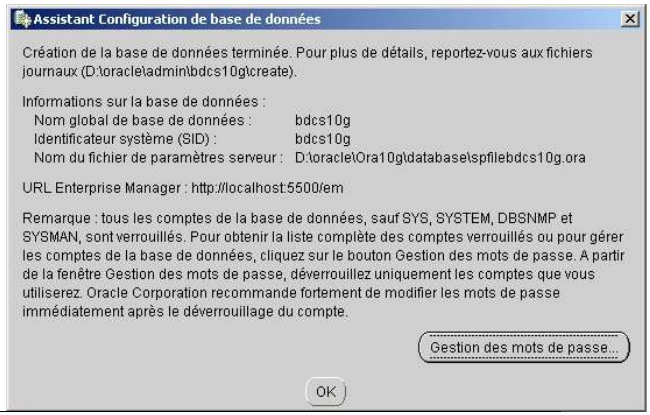


Figure 13-4 Modification éventuelle des mots de passe

Puis l'installeur exécute différents assistants qui doivent tous se terminer avec succès.

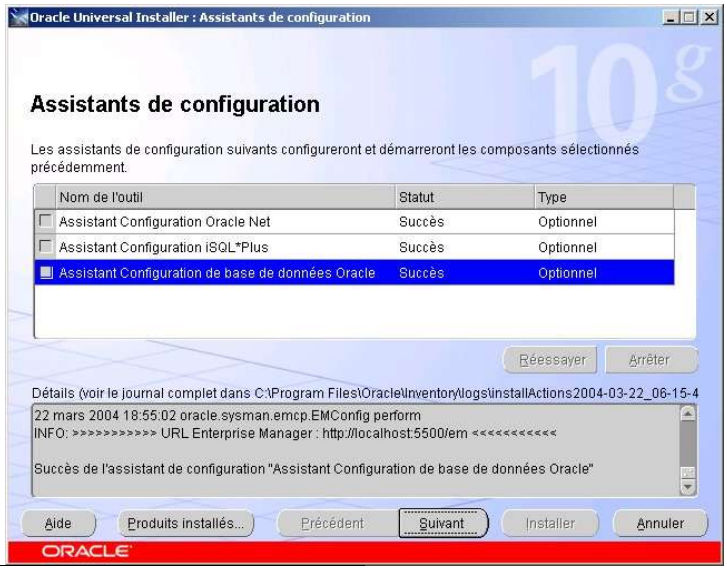


Figure 13-5 Assistants

Une fois Oracle installé, cliquez sur `Quitter`. Le navigateur se lance pour exécuter la nouvelle version de l'outil d'administration *Enterprise Manager*. Pour créer un utilisateur, il faudra donc se connecter avec le compte système (mot de passe défini au préalable lors de l'installation). L'écran suivant est obtenu en choisissant l'onglet `Administration` et l'hyperlien `Utilisateurs`. Pensez à affecter manuellement (hyperlien `Quotas`) une valeur en Mo pour les espaces `USERS` et `TEMP`. Il semble que l'interface Web actuelle ne permette pas d'affecter un quota illimité.

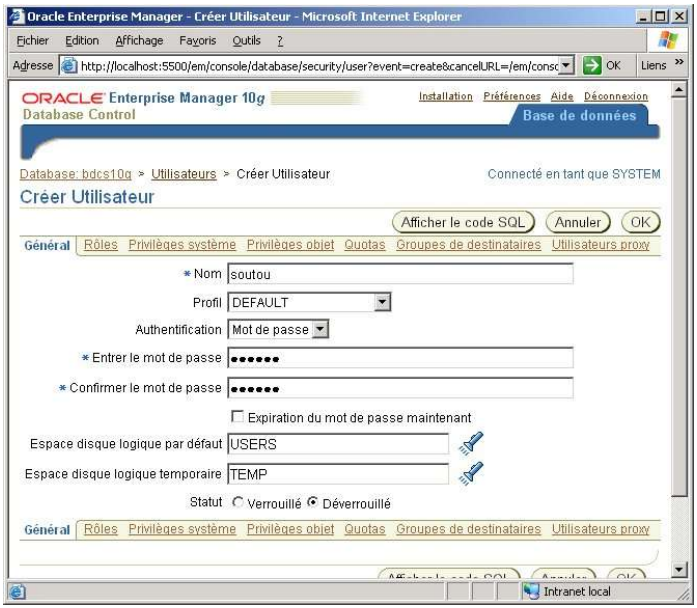


Figure 13-6 Création d'un utilisateur

Le rôle `CONNECT` est affecté par défaut à un nouvel utilisateur (il permet entre autres de pouvoir se connecter ; attention, il n'est pas suffisant pour créer des tables, types, etc.). Le rôle `RESOURCE` n'existant plus sous cette version, il faudra donc alimenter les privilèges d'un utilisateur explicitement (en relançant si nécessaire la console d'administration `http://localhost:5500/em` et en modifiant l'utilisateur au niveau des privilèges système). L'écran suivant illustre ce propos : en autorisant l'utilisateur *Soutou*, on crée dans son schéma des procédures, déclencheurs, types, vues, et tables.

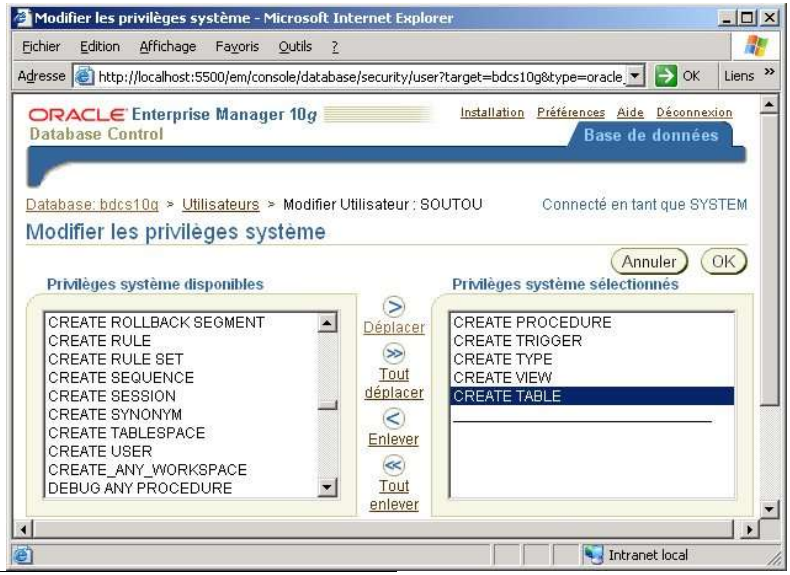


Figure 13-7 Affectation de privilèges à un utilisateur

Oracle a mis en place huit services dont cinq sont automatiquement lancés.

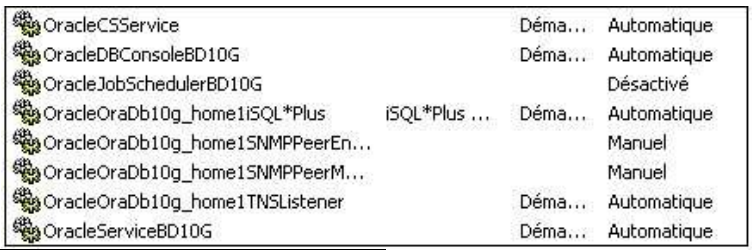


Figure 13-8 Services mis en place

Si vous n'utilisez pas souvent Oracle, pensez à arrêter ces services et à les positionner sur `Manuel`.

Désinstallation d'Oracle

La désintallation d'Oracle comme l'installation de produits supplémentaires n'est pas aisée. En bref, voici la procédure rapide et efficace à employer pour tout nettoyer :

1. Arrêtez tous les services d'Oracle.

2. Entrez dans la Base de registres (Menu `Démarrer/Exécuter…regedit`) et supprimez les clés suivantes :

   - `ORACLE` dans `HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE` (si vous n'avez pas d'autres instances Oracle). Dans le cas inverse, supprimer seulement les clés `ORA_CRS_HOME`, `KEY_OraDb10g_home1`, `Ocr`, `SCR`, et `SYSMAN`.
   - Relatives à Oracle dans l'entrée `HKEY_CLASSES_ROOT`.
   - Pour les systèmes non XP, supprimez les clés correspondant aux huit services dans `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services`. Pour les systèmes XP, ils se trouvent dans `HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services` et `ControlSet002`.

3. Enlevez les chemins mis en place au niveau des variables d'environnement (panneau de configuration, Système onglet `Avancé` sous XP).

4. Supprimez, s'il existe, le répertoire `C:\Documents and Settings\`*`utilisateur`*`\Local Settings\Temp\OraInstall`*`Date`*.

5. Enlevez les entrées du menu Démarrer (sous XP `C:\Documents and Settings\All Users\Menu Démarrer\Programmes`).

6. Supprimez le répertoire `Oracle` sous `Program Files`. Rédémarrez votre ordinateur.

7. Supprimez le répertoire d'installation où vous avez installé Oracle (ici `D:\Oracle\Ora10g` illustré à la figure 13-1). Si `oci.dll` dans le répertoire `bin` vous cause tracas, arrêtez le processus `SVCHOST.EXE` (celui qui occupe le plus d'espace en mémoire) et supprimez ce fichier. Videz la corbeille. Rédémarrez votre ordinateur.

Une fois que tout cela est fait, il est également conseillé de défragmenter l'unité de disque qui a contenu la base, avant d'entreprendre une nouvelle installation. Enfin, par prudence, utilisez un nom de base différent à chaque nouvelle installation.

*i*SQL*Plus

Cette interface est proposée depuis Oracle9*i*. Pour lancer *i*SQL*Plus, inscrivez l'URL suivante `http://`*`nomMachine`*`:5560/isqlplus` dans la barre d'adresses de votre navigateur. La première fenêtre permet de se connecter. La deuxième fenêtre (qui est la fenêtre principale) présente de nombreuses possibilités et ressemble un peu à l'interface Web de MySQL.

*Figure 13-9 Interface Web iSQL*Plus*

Connexion à Oracle

Quel que soit le mode de connexion que vous allez choisir, il faudra saisir au moins deux paramètres (nom d'utilisateur et mot de passe). Le troisième paramètre est optionnel : il s'agit du descripteur de connexion (aussi appelé indentificateur de connexion à la figure 13-12) qui indique la base cible. Si vous n'avez qu'une seule base (instance) et que vous êtes connecté sur la machine qui l'héberge, nul besoin de renseigner ce paramètre. Dans le cas contraire, il faut utiliser un descripteur de connexion qui aura été défini par l'outil Net Configuration Assistant et qui se trouve dans le fichier `tnsnames.ora` situé dans votre répertoire d'installation … `\NETWORK\ADMIN`.

Dans le fichier généré lors de l'installation, la partie qui décrit la connexion locale est la suivante. (Le descripteur de connexion est surligné : il s'agit par défaut celui de la base locale en gras.)

```
BDCS10G =
  (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = localhost)(PORT = 1521))
    (CONNECT_DATA = (SERVER = DEDICATED)
      (SERVICE_NAME = bdcs10g) ) )
```

Vous pouvez volontairement distinguer ces deux identificateurs pour des connexions à des bases distantes ou vers d'autres instances locales. Utilisez alors des noms de descripteurs de connexion du type `CX`*nomBase*. Pour les connexions en mode ligne de commande, le descripteur de connexion devra être suffixé par le nom d'utilisateur, par exemple : `soutou@cx`*nomBase*.

Nouvelles fonctionnalités

Flottants

Deux types numériques apparaissent : `BINARY_FLOAT` et `BINARY_DOUBLE` qui permettent de représenter des grands nombres (plus importants que ceux définis par `NUMBER`) sous la forme de

flottants. Les nombres flottants peuvent disposer d'une décimale située à tout endroit (de la première position à la dernière) ou ne pas avoir de décimale du tout. Un exposant peut être éventuellement utilisé (exemple : 1.777 e^{-20}). Une échelle de valeurs ne peut être imposée à un flottant puisque le nombre de chiffres apparaissant après la décimale n'est pas restreint.

Le stockage des flottants diffère de celui des `NUMBER` en ce sens que le mécanisme de représentation interne est propre à Oracle. Pour une colonne `NUMBER`, les nombres à virgule ont une précision décimale. Pour les types `BINARY_FLOAT` et `BINARY_DOUBLE`, les nombres à virgule ont une précision exprimée en binaire.

Tableau 13-1 Types de flottants

Type	Description	Commentaire pour une colonne
`BINARY_FLOAT`	Flottant simple précision.	Sur 5 octets (un représentant la longueur). Valeur entière maximale $3.4 \times 10^{+38}$, valeur entière minimale $-3.4 \times 10^{+38}$. Plus petite valeur positive 1.2×10^{-38}, plus petite valeur négative -1.2×10^{-38}.
`BINARY_DOUBLE`	Flottant double précision.	Sur 9 octets (un représentant la longueur). Valeur entière maximale $1.79 \times 10^{+308}$, valeur entière minimale $-1.79 \times 10^{+308}$. Plus petite valeur positive 2.3×10^{-308}, plus petite valeur négative -2.3×10^{-308}.

Oracle fournit également le type ANSI `FLOAT` qui peut aussi s'écrire `FLOAT(`*n*`)`. L'entier *n* (de 1 à 126) indique la précision binaire. Afin de convertir une précision binaire en précision décimale, il convient de multiplier l'entier par 0.30103. La conversion inverse nécessite de multiplier *n* par 3.32193. Le maximum de 126 bits est à peu près équivalent à une précision de 38 décimales.

L'écriture d'un flottant est la suivante :

```
[+|-] {chiffre [chiffre]... [.] [chiffre [chiffre]...] . chiffre [chiffre]...}
      [e[+|-] chiffre [chiffre]...] [f|d]
```

- e (ou E) indique la notation scientifique (mantisse et exposant).
- f (ou F) indique que le nombre est de type `BINARY_FLOAT`.
- d (ou D) indique que le nombre est de type `BINARY_DOUBLE`.

Si le type n'est pas explicitement précisé, l'expression est considérée comme de type `NUMBER`.

Valeurs spéciales

La recommandation IEEE 754 définit des valeurs spéciales pour les flottants : l'infini positif (`+INF`), l'infini néfatif (`-INF`) et `NaN` (*Not a Number*) qui est utilisé pour représenter les résultats des opérations indéfinies. L'obtention de ces valeurs se réalise par les opérations suivantes : dépassement de capacité (*overflow*) pour obtenir `-INF`, `+INF`, opération invalide retourne `NaN`, la

division par zéro peut retourner -INF, +INF ou NaN. Les opérateurs SQL NAN et INFINITE permettent de tester ces valeurs spéciales sur des flottants.

Le script suivant crée une table, insère deux flottants, modifie les chiffres pour insérer des valeurs infinies (la première résultant d'une division par zéro, la seconde d'un dépassement de capacité).

```
CREATE TABLE  Flottants (bfloat BINARY_FLOAT, bdouble BINARY_DOUBLE);
INSERT INTO   Flottants VALUES (+3.4e+38f, +1.77e+308d);
SELECT * FROM Flottants;

    BFLOAT    BDOUBLE
---------- ----------
  3,4E+038  1,77E+308

UPDATE Flottants SET bfloat = bfloat/0, bdouble= 2*bdouble;
SELECT * FROM Flottants WHERE bfloat IS INFINITE OR bdouble IS INFINITE;

    BFLOAT    BDOUBLE
---------- ----------
       Inf        Inf
```

Fonctions pour les flottants

Plusieurs fonctions sont disponibles pour manipuler des flottants.

TO_BINARY_DOUBLE

Comme son nom l'indique, cette fonction transforme une expression en flottant de type BINARY_DOUBLE. La syntaxe est la suivante :

```
TO_BINARY_DOUBLE(expression [,'format' [, 'nlsparam' ] ])
```

- *format* et *nlsparam* ont la même signification que dans TO_CHAR ;
- *expression* représente une valeur numérique ou 'INF', '-INF', 'NaN'.

Le script suivant présente l'utilisation de cette fonction.

```
SELECT TO_BINARY_DOUBLE(13.56767) FROM DUAL;
TO_BINARY_DOUBLE(13.56767)
--------------------------
                1,357E+001

SELECT TO_BINARY_DOUBLE('-INF') FROM DUAL;
TO_BINARY_DOUBLE('-INF')
------------------------
                    -Inf
```

TO_BINARY_FLOAT

Cette fonction transforme une expression en flottant de type BINARY_FLOAT. La syntaxe est la suivante :

```
TO_BINARY_FLOAT(expression[,'format'[,'nlsparam']]).
```

La signification des paramètres est identique à la fonction précédente.

DUMP

La fonction DUMP n'est pas dédiée aux flottants mais elle peut être utile pour mieux visualiser leur représentation. Cette fonction décrit la représentation interne de toute information sous la forme d'une chaîne de caractères incluant le code du type de données, la taille en octets, et la valeur de chaque octet. Sa syntaxe est la suivante :

```
DUMP(expression[, FormatRetour [, position [, longueur ] ] ] )
```

- *FormatRetour* :
  - 8 pour retourner une notation octale.
  - 10 pour retourner une notation décimale.
  - 16 pour retourner une notation hexadécimale.
  - 17 pour retourner des caractères distincts.
- *position* et *longueur* combinent la portion de la représentation interne à retourner (par défaut toute l'expression est décodée).

Voici deux exemples d'utilisation de cette fonction. La confirmation qu'un flottant de type BINARY_DOUBLE est représenté sur 8 octets apparaît ici clairement. La valeur de chaque octet en décimale est précisée dans la liste de valeurs retournées.

```
SELECT DUMP(TO_BINARY_DOUBLE(13.56767),10) FROM DUAL;
DUMP(TO_BINARY_DOUBLE(13.56767),10)
----------------------------------------
Typ=101 Len=8: 192,43,34,165,164,105,215,52

SELECT DUMP('C.Soutou', 10) "C.Soutou en ASCII" FROM DUAL;
C.Soutou en ASCII
----------------------------------------
Typ=96 Len=8: 67,46,83,111,117,116,111,117
```

NANVL

La fonction NANVL permet de substituer la valeur NaN (*Not a Number*) contenue dans un flotant par une autre valeur donnée et compréhensible (par exemple : zéro ou NULL). La syntaxe de cette fonction est la suivante :

```
NANVL(expression, substitution)
```

- *expression* désigne la valeur à substituer (tout type numérique ou non numérique pouvant être implicitement converti en numérique). Si l'expression n'est pas NaN, la valeur de l'expression est retournée. Sinon la valeur *substitution* est retournée.

Le code suivant décrit l'utilisation de cette fonction appliquée à deux flottants. L'opérateur IS NAN est utilisée dans la deuxième requête. Dans la troisième requête, l'opérateur NANVL permet de substituer la valeur 0 au premier flottant et -1 au second quand ces deux valeurs sont indéterminées.

```
INSERT INTO Flottants VALUES (+3.4e+38f,+1.77e+308d) ;
INSERT INTO Flottants VALUES ('NaN','NaN') ;
```

```
SELECT * FROM Flottants;
    BFLOAT    BDOUBLE
---------- ----------
  3,4E+038  1,77E+308
       Nan        Nan

SELECT * FROM Flottants WHERE bfloat IS NOT NAN AND bdouble IS NOT NAN;
    BFLOAT    BDOUBLE
---------- ----------
  3,4E+038  1,77E+308

SELECT NANVL(bfloat,0), NANVL(bdouble,-1) FROM Flottants;
NANVL(BFLOAT,0) NANVL(BDOUBLE,-1)
--------------- -----------------
       3,4E+038         1,77E+308
              0         -1,0E+000
```

REMAINDER

La fonction `REMAINDER` retourne le reste de la division de *m* par *n*. La fonction `MOD` (étudiée au chapitre 4) est quelque peu similaire à `REMAINDER`. Elles diffèrent par le fait que `MOD` utilise à ses fins l'opérateur `FLOOR` alors que `REMAINDER` utilise `ROUND`. La syntaxe de cette fonction est la suivante :

```
REMAINDER(m, n)
```

- *m* désigne la valeur à diviser (tout type numérique ou non numérique pouvant être implicitement converti en numérique). *n* désigne de la même manière le diviseur.
- Si *n* = 0 ou si *m* est infini, et si les arguments sont de type `NUMBER`, la valeur retournée est une erreur. Dans le cas de flottants (`BINARY_FLOAT` or `BINARY_DOUBLE`), la valeur retournée est `NaN` (*Not a Number*).
- Si *n* est différent de zéro, la fonction retourne la valeur *m* - (*n***N*) avec *N* plus grand entier plus proche du résultat *m*/*n*.
- Si *m* est un flottant et si le résultat vaut zéro, alors le signe du résultat est du signe de *m*. Si *m* est un `NUMBER` et si le résultat vaut zéro, alors le résultat n'est pas signé.

Le code suivant décrit l'utilisation de cette fonction appliquée à deux flottants de différents types également valués (1234,56). La valeur retournée n'est pas zéro du fait da la différence des types.

```
INSERT INTO Flottants VALUES (1234.56,1234.56);

SELECT * FROM Flottants;
    BFLOAT    BDOUBLE
---------- ----------
1,235E+003 1,235E+003

SELECT bfloat, bdouble, REMAINDER(bfloat, bdouble) FROM Flottants;
    BFLOAT    BDOUBLE REMAINDER(BFLOAT,BDOUBLE)
---------- ---------- -------------------------
1,235E+003 1,235E+003                5,859E-005
```

Fusion (MERGE)

L'instruction MERGE étudiée au chapitre 4 dispose d'une nouvelle syntaxe qui permet soit de mettre à jour (UPDATE ou DELETE), soit d'insérer (INSERT), soit d'effectuer les trois types d'opération sur des données dans une table cible. Cela évite d'écrire des insertions ou des mises à jour ou suppressions multiples en plusieurs commandes.

Syntaxe

La nouvelle syntaxe de cette instruction est la suivante :

```
MERGE INTO [schéma.] nomTableCible [alias]
      USING [schéma.] { nomTableSource | nomVue | requête } [alias]
      ON (condition)
   [WHEN MATCHED THEN
         UPDATE SET col1 = {expression1 | DEFAULT}
                   [,col2 = {expression2 | DEFAULT}]...
        [WHERE condition]
        [DELETE WHERE condition] ]
   [WHEN NOT MATCHED THEN
         INSERT [ (col1 [, col2]...) ]
         VALUES ( {expression1 [,expression2]... | DEFAULT } )
         [WHERE condition] ] ;
```

Le choix de l'opération dans la table cible est toujours conditionné par la clause ON. Pour chaque enregistrement de la table cible qui vérifie la condition, l'enregistrement correspondant de la table source est modifié. Les données de la table cible qui ne vérifient pas la condition déclenchent une insertion dans la table cible, basée sur des valeurs d'enregistrements de la table source.

La clause DELETE… permet de vider des enregistrements de la table cible, tout en la remplissant ou en la modifiant. Les seuls enregistrements affectés sont ceux qui sont concernés par la fusion. Cette clause évalue seulement les valeurs mises à jour (pas les valeurs originales qui sont évaluées par la directive UPDATE SET... WHERE *condition*). Si un enregistrement de la table cible satisfait à la condition du DELETE, mais n'est pas inclus dans la jointure définie par la directive ON, il ne sera pas détruit.

La clause WHERE… de l'instruction INSERT filtre les insertions par une condition sur la table source.

> Il n'est pas possible de modifier une colonne référencée dans la clause de jointure ON.

Exemple

Supposons qu'on désire ajouter à la paye de chaque pilote de grade 'CDB' un bonus. Si un bonus est donné à un pilote n'ayant pas encore eu de prime, il faudra ajouter ce pilote en affectant sa paye au bonus reçu. On désire aussi supprimer les primes des pilotes modifiés si la valeur de leur paye est inférieure à 90. La figure suivante illustre cet exemple qui, sans l'utilisation de l'instruction MERGE, requiert l'utilisation d'une instruction UPDATE , DELETE et INSERT (qui serait multiligne si plusieurs pilotes n'étaient pas référencés dans la table Primes).

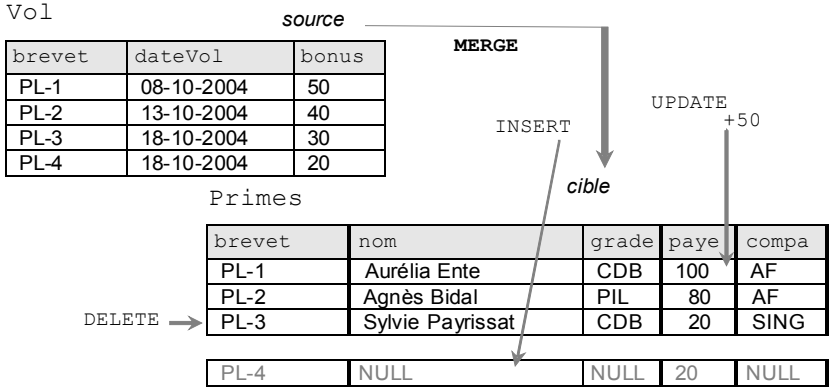


Figure 13-10 Mises à jour conditionnées

Le tableau suivant décrit l'instruction `MERGE` à utiliser et le résultat produit.

Web

Tableau 13-2 Fusion par `MERGE`

Requête	Résultat sous SQL*Plus

`MERGE INTO Primes p`
` USING (SELECT brevet, bonus FROM Vol) v`
` ON (p.brevet = v.brevet)`
`  WHEN MATCHED THEN`
`  UPDATE SET p.paye = p.paye + v.bonus`
`  WHERE grade = 'CDB'`
`  DELETE WHERE paye < 90`
`  WHEN NOT MATCHED THEN`
`  INSERT (brevet, paye)`
`  VALUES (v.brevet, v.bonus);`
	`SQL> SELECT * FROM Primes ;`
	
	`BREVET NOM GRADE PAYE COMP`
	`------ ------------ ----- ------ ----`
	`PL-1 Aurélia Ente CDB 150 AF`
	`PL-2 Agnès Bidal PIL 80 AF`
	`PL-4 20`

Requêtes hiérarchiques

Plusieurs nouvelles fonctions sont disponibles pour interroger des hiérarchies. Les exemples qui suivent utilisent la table `Trajets` présentée au chapitre 4, section « Requêtes hiérarchiques ».

SYS_CONNECT_BY_PATH

La fonction `SYS_CONNECT_BY_PATH` extrait le chemin (sous la forme d'une chaîne `VARCHAR2`) à partir de la racine (ou des racines si aucune clause `START WITH` n'est indiquée jusqu'aux feuilles terminales). La syntaxe de cette fonction est la suivante :

`SYS_CONNECT_BY_PATH`(*colonne*, *caractère*)

colonne et *caractère* sont de type `CHAR`, `VARCHAR2`, `NCHAR`, ou `NVARCHAR2`. Le premier paramètre désigne la colonne de la table qui compose la hiérarchie définie par la clause `CONNECT BY` et qu'on désire afficher. Le second paramètre indique le séparateur utilisé pour l'affichage du chemin complet.

La requête suivante extrait tous les chemins complets partant de Paris.

Web

```
COL chemin FORMAT A30 HEADING "Hélas tout part de Paris..."
SELECT LPAD(' ',2*LEVEL-1) || SYS_CONNECT_BY_PATH(arrivée,'/') chemin, tempsVol
FROM Trajets
   START WITH depart = 'Paris'
   CONNECT BY PRIOR arrivée = départ;
```

```
Hélas tout part de Paris...      TEMPSVOL
------------------------------ ----------
 /Blagnac                               1
   /Blagnac/Pau                        ,4
 /Lyon                                 ,8
   /Lyon/Grenoble                      ,3
     /Lyon/Grenoble/Gap                ,35
   /Lyon/Valence                       ,2
     /Lyon/Valence/Ales                ,25
 /Marseille                            ,9
   /Marseille/Frejus                   ,2
   /Marseille/Toulon                   ,15
   /Marseille/Nimes                    ,35
```

CONNECT_BY_ROOT

L'opérateur CONNECT_BY_ROOT étend la fonctionnalité de la condition CONNECT BY [PRIOR] en permettant de qualifier une colonne et de retourner non seulement un enregistrement parent de l'enregistrement courant, mais également tous ses ancêtres. Cet opérateur ne peut pas être utilisé dans une clause START WITH ou CONNECT BY.

La requête suivante extrait les chemins complets ayant deux escales. L'opérateur CONNECT_BY_ROOT permet ici d'afficher la première escale.

```
COL chemin FORMAT A30 HEADING "Chemin..."
SELECT arrivée "De Paris à", CONNECT_BY_ROOT arrivée,
       SYS_CONNECT_BY_PATH(départ,'/') chemin
     FROM Trajets WHERE LEVEL > 2
     CONNECT BY PRIOR arrivée = départ;
```

```
De Paris à CONNECT_BY Chemin...
---------- ---------- ------------------------------
Gap        Lyon       /Paris/Lyon/Grenoble
Ales       Lyon       /Paris/Lyon/Valence
```

CONNECT_BY_ISLEAF

La pseudo colonne CONNECT_BY_ISLEAF retourne la valeur 1 si l'enregistrement courant est une feuille de la hiérarchie désignée par la condition dans la clause CONNECT BY. Dans le cas inverse, cette pseudo colonne vaut 0. Cette information permet de savoir si un enregistrement courant est un nœud ou une feuille de la hiérarchie.

La requête suivante extrait les chemins complets des trajets avec les destinations finales. L'opérateur CONNECT_BY_ISLEAF permet ici d'afficher seulement les terminaisons de la hiérarchie.

```
COL chemin FORMAT A30 HEADING "Chemin..."
SELECT arrivée, CONNECT_BY_ISLEAF "IsLeaf", LEVEL,
       SYS_CONNECT_BY_PATH(départ,'/') chemin
```

```
       FROM Trajets WHERE CONNECT_BY_ISLEAF = 1
       START WITH départ='Paris'
       CONNECT BY PRIOR arrivée = départ;

ARRIVÉE        IsLeaf      LEVEL Chemin...
---------- ---------- ---------- ------------------------------
Pau                 1          2 /Paris/Blagnac
Gap                 1          3 /Paris/Lyon/Grenoble
Ales                1          3 /Paris/Lyon/Valence
Frejus              1          2 /Paris/Marseille
Toulon              1          2 /Paris/Marseille
Nimes               1          2 /Paris/Marseille
```

La requête suivante extrait les chemins complets des trajets avec les destinations au bout de deux escales non terminales.

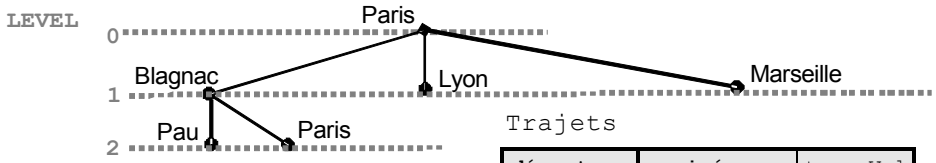
```
COL chemin FORMAT A35 HEADING "Chemin 2 escales non terminales..."
SELECT arrivée, SYS_CONNECT_BY_PATH(départ, '/') chemin
    FROM Trajets
    WHERE CONNECT_BY_ISLEAF = 0 AND LEVEL = 2
    START WITH depart = 'Paris'
    CONNECT BY PRIOR arrivée = départ;

ARRIVÉE    Chemin 2 escales non terminales...
---------- -----------------------------------
Grenoble   /Paris/Lyon
Valence    /Paris/Lyon
```

CONNECT_BY_ISCYCLE

La pseudo colonne `CONNECT_BY_ISCYCLE` retourne la valeur 1 si l'enregistrement courant est associé à un enregistrement enfant qui est également son ancêtre dans la hiérarchie désignée par la condition dans la clause `CONNECT BY`. Dans le cas inverse, cette pseudo colonne vaut 0. Cette pseudo colonne n'a de sens que si le paramètre `NOCYCLE` a été spécifié dans la clause `CONNECT BY`. Ce paramètre permet de retourner un résultat récursif qui échouerait sans cette option. La syntaxe de la définition du parcours de la hiérarchie est la suivante (elle est à placer après la condition `WHERE` de la requête) :

```
[ START WITH condition ]
CONNECT BY [ NOCYCLE ] condition
```

Considérons la hiérarchie suivante qui inclut un cycle. Il sera nécessaire d'utiliser le paramètre `NOCYCLE` et la pseudo colonne `CONNECT_BY_ISCYCLE` pour que le cycle n'entraîne pas d'interférences dans les différentes requêtes qui parcoureront la hiérarchie.



Trajets

départ	arrivée	tempsVol
Paris	Blagnac	1
Paris	Lyon	0.8
Paris	Marseille	0.9
Blagnac	Pau	0.4
Blagnac	Paris	0.95

Figure 13-11 Hiérarchie avec un cycle

La requête suivante extrait les chemins complets des trajets avec les destinations finales et intermédiaires. L'opérateur `CONNECT_BY_ISCYCLE` permet ici de trouver le cycle.

```
COL chemin FORMAT A30 HEADING "Chemin..."
SELECT arrivée "De Paris à", CONNECT_BY_ISCYCLE, LEVEL,
       SYS_CONNECT_BY_PATH(départ, '/') chemin
     FROM Trajets
     START WITH depart = 'Paris'
     CONNECT BY NOCYCLE PRIOR arrivée = départ;
```

```
De Paris à CONNECT_BY_ISCYCLE      LEVEL Chemin...
---------- ------------------ ---------- ------------------------------
Blagnac                     0          1 /Paris
Pau                         0          2 /Paris/Blagnac
Paris                       1          2 /Paris/Blagnac
Lyon                        0          3 /Paris/Blagnac/Paris
Marseille                   0          3 /Paris/Blagnac/Paris
Lyon                        0          1 /Paris
Marseille                   0          1 /Paris
```

La requête suivante extrait les chemins complets des trajets avec les destinations finales et intermédiaires sans que le cycle n'interfère dans le résultat.

```
SELECT arrivée "De Paris à", LEVEL, SYS_CONNECT_BY_PATH(départ, '/') chemin
       FROM Trajets
       WHERE CONNECT_BY_ISCYCLE = 0 AND LEVEL < 3
       START WITH départ = 'Paris'
       CONNECT BY NOCYCLE PRIOR arrivée = départ;
```

```
De Paris à      LEVEL Chemin...
---------- ---------- ------------------------------
Blagnac             1 /Paris
Pau                 2 /Paris/Blagnac
Lyon                1 /Paris
Marseille           1 /Paris
```

Expressions régulières

Les expressions régulières ont un fort rapport avec la notion de format de données ou de grammaire associée. Par exemple, un numéro de téléphone en France s'écrit sur 10 chiffres, le plus souvent indiqués par groupes de 2 entre tirets (ainsi : 05-62-74-75-71). Les deux premiers

chiffres indiquent une région (05 indique ici le Sud-Ouest). Un autre exemple concerne les numéros d'immatriculation des véhicules composés d'une série de chiffres, de lettres et de chiffres représentant le département d'appartenance.

Les expressions régulières sont manipulées sous SQL ou PL/SQL par les nouveaux opérateurs `REGEXP_LIKE`, `REGEXP_REPLACE`, `REGEXP_INSTR` et `REGEXP_SUBSTR`. Le tableau suivant décrit les principaux éléments permettant de composer une expression régulière.

Tableau 13-3 Éléments décrivant une expression régulière

Élément	Description
`\`	Le caractère *backslash* (barre oblique inverse) permet d'annuler l'effet d'un caractère significatif suivant (opérateur, par exemple).
`*`	Désigne aucune ou plusieurs occurrences.
`+`	Désigne une ou plusieurs occurrences.
`?`	Désigne au plus une occurrence.
`|`	Opérateur spécifiant une alternative.
`^`	Désigne le début d'une ligne de caractères.
`$`	Désigne la fin d'une ligne de caractères.
`.`	Désigne tout caractère excepté la valeur `NULL`.
`[ ]`	Désigne une liste devant vérifier une expression continue dans la liste. Une liste ne devant pas vérifier une expression contenue dans la liste devra commencer par le caractère "^".
`( )`	Désigne une expression groupée et traitée comme une simple sous-expression.
`{m}`	Signifie exactement *m* fois.
`{m,}`	Signifie au moins *m* fois.
`{m,n}`	Signifie au moins *m* fois mais pas plus de *n* fois.
`[::]`	Spécifie la classe de caractères (précisée dans le tableau suivant)
`[==]`	Spécifie la classe d'équivalence (ex : '[=a=]' filtrera ä, â, à…).

.Le tableau suivant recence les classes d'équivalence disponibles.

Tableau 13-4 Classes d'équivalence

Classe	Explication
`[:alnum:]`	Caractères alphanumériques.
`[:alpha:]`	Caractères alphabétiques.
`[:blank:]`	Caractères d'espacement.
`[:cntrl:]`	Caractères de contrôle.

Classe	Explication
[:digit:]	Chiffres.
[:graph:]	Caractères de la forme [:punct:], [:upper:], [:lower:], et [:digit:]
[:lower:]	Caractères alphabétiques minuscules.
[:print:]	Caractères imprimables.
[:punct:]	Caractères de ponctuation.
[:space:]	Caractères espaces (non affichables).
[:upper:]	Caractères alphabétiques majuscules.
[:xdigit:]	Caractères hexadécimaux.

Quelques exemples

Considérons les données suivantes décrivant des parcs Américains issu de [GEN 03]. La structure de la table Parcs est la suivante : endroit VARCHAR2(7), telephone VARCHAR2(15), description VARCHAR2(400).

Parcs

endroit	telephone	description
P1	(231) 436-4100	Michigan's first state park encompasses approximately 1800 acres of Mackinac Island. The centerpiece is Fort Mackinac, built in 1780 by the British to protect the Great Lakes Fur Trade. For information by phone, dial 800-44-PARKS or 517-373-1214.
P2	(906) 289-4215	Located almost at the very tip of the Keewenaw Penninsula, Fort Wilkens is a restored army fort built during the copper rush. Camping is available. For the modern campground, phone (800) 447-2757. For group-camping, phone 906.289.4215. For information on canoe, kayak, and other boat rentals, call the concession office at (906) 289-4210.
P3	(906) 863-9747	This scenic site is centered around an impressive waterfall.A rustic, picnic area with waterpump is available.
P4	(906) 658-3338	A 217-acre park located on the site of an old lumber town, Deer Park. Shower and toilet facilities are available, as are campsites withelectricity.
P5	(906) 885-5275	Michigan's largest state parkconsists of some 60,000 acres of mostly virgin timber. Over 90 miles of trails are available to backpackers and hikers. Downhill skiing is available in winter.Rustic cabins are available. To reserve a cabin, call (906) 885-5275.
P6	NULL	One of the largest waterfalls east of the Mississippi is found within this park's 40,000+ acres. Upper Tahquamenon Falls is some 50 feet high, 200 feet across, and supports a flow that has been known to reach 50,000 gallons/second. The park phone is 906.492.3415.

Figure 13-12 Jeu d'essai

Fonction REGEXP_LIKE

La fonction booléenne `REGEXP_LIKE` permet d'identifier des enregistrements vérifiant une condition à propos d'une expression régulière. Cette fonction s'utilise majoritairement dans la clause `WHERE` d'une requête. La syntaxe de cette fonction est la suivante :

```
REGEXP_LIKE (chaineSource, grammaire [,paramètre …] )
```

`paramètre` est un texte littéral qui permet de moduler l'expression régulière. Les valeurs de ce paramètre peuvent être :

- `'i'` si on ne tient pas compte de la casse ;
- `'c'` si on tient compte de la casse ;
- `'n'` permet d'utiliser le caractère "." en tant que fin de ligne ;
- `'m'` permet de traiter la chaîne source comme plusieurs lignes. Oracle interprète "^" et "$" comme le début et la fin de chaque sous-ligne.

Si aucun paramètre n'est utilisé, la sensibilité à la casse est définie par la valeur de `NLS_SORT`, le caractère « . » ne termine pas une ligne et la chaîne est traitée comme une seule ligne.

Exemples pour l'extraction

Le tableau suivant illustre quelques utilisations de cette fonction manipulant des expressions régulières. Le filtre porte sur la colonne `description` qui comporte plusieurs lignes. Nous testons ici les différents formats des numéros de téléphone.

Web

Tableau 13-5 Utilisations de la fonction `REGEXP_LIKE`

Expression	Requête	Résultat SQL*Plus

`xxx-xxxx`	`SELECT endroit`
	`FROM Parcs`
	`WHERE REGEXP_LIKE (description, '...-....');`
		`ENDROIT`
		`-------`
		`P1`
		`P2`
		`P4`
		`P5`

Idem, x étant un chiffre, élimine par exemple l'expression "217-acre".	`SELECT endroit FROM Parcs`
	`WHERE REGEXP_LIKE (description, '[0-9][0-9][0-9]-[0-9][0-9][0-9][0-9]');`
	ou
	`SELECT endroit FROM Parcs`
	`WHERE REGEXP_LIKE (description, '[0-9]{3}-[0-9]{4,4}');`
		`ENDROIT`
		`-------`
		`P1`
		`P2`
		`P5`

Idem en autorisant aussi les nombres séparés par des points.	`SELECT endroit FROM Parcs`
	`WHERE REGEXP_LIKE (description, '[0-9]{3}[-.][0-9]{4,4}');`
		`ENDROIT`
		`-------`
		`P1`
		`P2`
		`P5`
		`P6`

Le tableau suivant illustre quelques autres expressions régulières extraites du jeu d'essai décrit ci-après.

Test

col
bonjour
Maître
enfant
maître
mòbile
pájaro
zurück

Test2

col
resume
résumé
résume
resumé
rasumé
ràsume

Figure 13-13 Jeu d'essai

Web

Tableau 13-6 Utilisation de classe de caractères

Expression	Requête	Résultat
Chaînes de 6 caractères et plus en minuscules.	`SELECT col FROM Test WHERE REGEXP_LIKE(col,'([[:lower:]]){6}');`	`COL ---------- bonjour enfant maître mòbile pájaro zurück`
Chaînes de 6 caractères en minuscules.	`SELECT col FROM Test WHERE REGEXP_LIKE(col,'^[[:lower:]]{6}$');`	`COL ---------- enfant maître mòbile pájaro zurück`
Chaînes de 6 caractères commençant par une majuscule, le reste en minuscules.	`SELECT col FROM Test WHERE REGEXP_LIKE(col,'^[[:upper:]]{1}[[:lower:]]{5}$');`	`COL ---------- Maître`
Classe d'équivalence du « e » en deuxième et dernière position.	`SELECT col FROM Test2 WHERE REGEXP_LIKE(col,'r[[=e=]]sum[[=e=]]');`	`COL ---------- resume résumé résume resumé`
Classe d'équivalence de « a » et de « e ».	`SELECT col FROM Test2 WHERE REGEXP_LIKE(col,'r[[=a=]]sum[[=e=]]');`	`COL ---------- rasumé ràsume`

Définition d'une contrainte

La fonction `REGEXP_LIKE` permet également de définir des contraintes au niveau des colonnes de tables afin de s'assurer du format des données. L'ajout de la contrainte suivante garantit que la colonne `telephone` contient à présent des valeurs de la forme « (*xxx*) *xxx-xxxx* ».

```
ALTER TABLE Parcs
   ADD (CONSTRAINT ck_format_telephone
        CHECK (REGEXP_LIKE(telephone,
```

```
'^\([[:digit:]]{3}\) [[:digit:]]{3}-[[:digit:]]{4}$'));
```

Étudions à présent les fonctions par lesquelles on peut manipuler des chaînes de caractères tout en utilisant des expressions régulières.

Fonction REGEXP_REPLACE

La fonction `REGEXP_REPLACE` étend la fonction `REPLACE` en permettant de modifier une chaîne de caractères à partir d'une expression régulière. Par défaut, la fonction remplace une chaîne source par chaque occurrence d'une expression régulière donnée. Cette fonction retourne un `VARCHAR2` si le premier paramètre n'est pas une donnée de type *LOB*. Dans le cas inverse, la fonction retourne une donnée de type `CLOB`. La syntaxe de cette fonction est la suivante :

```
REGEXP_REPLACE (source, modèle [,remplace
     [,position [, occurrence [, paramètre ] ] ] ] )
```

- *source* indique la chaîne à examiner (une colonne de type `CHAR`, `VARCHAR2`, `NCHAR`, `NVARCHAR2`, `CLOB`, ou `NCLOB`).
- *modèle* désigne l'expression régulière (jusqu'à 512 octets).
- *remplace* décrit, sous la forme de références arrières (jusqu'à 500 expressions « \n » avec *n* chiffres de 1 à 9), de quelle manière la chaîne source va être transformée. Si le paramètre *remplace* est un `CLOB` ou `NCLOB`, alors Oracle le tronque à 32 Ko
- *position* est un entier indiquant la position de début de recherche (par défaut 1).
- *occurrence* est un entier précisant le remplacement (0 pour remplacer toutes les occurrences qui conviennent à l'expression régulière, *n* pour remplacer la $n^{ième}$).
- `paramètre` a la même signification que dans l'utilisation de la fonction `REGEXP_LIKE`.

Le tableau suivant illustre quelques utilisations de cette fonction de remplacement. Le premier exemple remplace chaque caractère non nul par son équivalent suivi d'un tiret. Le deuxième remplace plusieurs espaces par un seul.

Dans le troisième exemple, nous rendons homogène (à l'affichage) les différents formats des numéros de téléphone de type « *xxx*xxx*xxxx » (x* étant un chiffre et * étant un tiret ou un point) présents dans la colonne `description` (pour l'endroit de code 'P1') par le format « *(xxx) xxx-xxxx »*. On remarque que le numéro de téléphone codé en partie à l'aide de lettres n'a pas été modifié car il ne respecte pas l'expression régulière. Utilisée dans un `UPDATE`, cette fonction pourrait permettre de modifier cette colonne en conséquence.

Web

Tableau 13-7 Utilisation de la fonction `REGEXP_REPLACE`

Requête	Résultat SQL*Plus

`CREATE TABLE Test (col VARCHAR2(30));`
`INSERT INTO Test VALUES ('Castanet');`
`INSERT INTO Test VALUES ('Blagnac');`
`INSERT INTO Test VALUES ('Paris');`
 	
`SELECT REGEXP_REPLACE(col, '(.)', '\1-')`
`  FROM Test ;`
	`REGEXP_REPLACE(COL,'(.)','\1-')`
	`--------------------------------`
	`C-a-s-t-a-n-e-t-`
	`B-l-a-g-n-a-c-`
	`P-a-r-i-s-`

```
SELECT REGEXP_REPLACE('IUT,    1    Place
G.    Brassens,  Blagnac',
 '( ){2,}', ' ') "Exemple 2" FROM DUAL;
```

```
Exemple 2
--------------------------------
IUT, 1 Place G. Brassens, Blagnac
```

```
SELECT REGEXP_REPLACE(description,
   '([[:digit:]]{3})[-.]([[:digit:]]{3})
     [-.]([[:digit:]]{4})', '(\1) \2-\3')
       FROM Parcs WHERE endroit = 'P1';
```

```
DESCRIPTION
----------------------------
Michigan's first state park
encompasses approximately 1800 acres
of Mackinac Island. The centerpiece is
Fort Mackinac, built in 1780 by the
British to protect the Great Lakes Fur
Trade. For information by phone, dial
800-44-PARKS or (517) 373-1214.
```

Fonction REGEXP_INSTR

La fonction `REGEXP_INSTR` étend la fonction `INSTR` en permettant de rechercher une chaîne de caractères à partir d'une expression régulière. Cette fonction retourne un entier indiquant le début (ou la fin) d'une sous-chaîne vérifiant l'expression régulière, ceci en fonction d'un paramètre de retour. Si aucune sous-chaîne ne convient, la fonction retourne 0. La syntaxe de cette fonction est la suivante :

```
REGEXP_INSTR (source, modèle
       [, position [, occurrence [, optionRetour [, paramètre ] ] ] ] )
```

- *source* indique la chaîne à examiner (une colonne de type `CHAR`, `VARCHAR2`, `NCHAR`, `NVARCHAR2`, `CLOB`, ou `NCLOB`).
- *modèle* désigne l'expression régulière (jusqu'à 512 octets).
- *position* est un entier positif indiquant la position de début de recherche (par défaut 1).
- *occurrence* est un entier positif précisant quelle est l'occurrence de l'expression recherchée (par défaut, 1 indiquant que la première occurrence est à examiner, *n* pour examiner la $n^{ième}$).
- *optionRetour* codifie ce qui doit être retourné :
  - 0 si la position du premier caractère de l'occurrence extraite doit être retournée (option par défaut).
  - 1 si la position du premier caractère suivant l'occurrence extraite doit être retournée.
- *paramètre* a la même signification que dans l'utilisation des fonctions `REGEXP_LIKE` et `REGEXP_REPLACE`.

Le tableau suivant illustre quelques utilisations de cette fonction de recherche.

Le premier exemple examine la chaîne décrivant une adresse, recherche les occurrences des caractères non blancs en débutant au premier caractère et retourne la première position du quatrième mot (15 correspond à la position qui débute avec « 31703 »).

Le deuxième exemple examine la chaîne et analyse les mots de sept lettres commençant par s, r, ou p (casse indifférente). La recherche débute au troisième caractère et retourne la position du premier caractère suivant la seconde occurrence du type de mot recherché (ici, 28 correspond à la position du « *S* » de « *Shores* » ; « *Parkway* » et « *Redwood* » étant deux mots qui respectent l'expression régulière).

Dans le troisième exemple, nous extrayons les endroits dont la description inclut une surface (définies en acres mais hétérogènes au niveau de l'expression). Utilisée conjointement à SUBSTR (qui extrait une sous-chaîne), les fonctions REGEXP_INSTR permettent de délimiter les différentes expressions décrivant une surface (*1800 acres*, *217-acre*, *60,000 acres* et *40,000+ acres*). L'expression régulière est divisée par une barre verticale qui filtre à la fois les mots*acres* et *acre*. Les deuxième et troisième appels à REGEXP_INSTR servent à déterminer la taille de l'expression.

Web

Tableau 13-8 Utilisations de la fonction REGEXP_INSTR

Requête	Résultat SQL*Plus

`SELECT REGEXP_INSTR('IUT Dept GTR, 31703 Blagnac',`
`'[^ ]+', 1, 4) FROM DUAL;`
	`Exemple 1`
	`----------`
	`15`

`SELECT`
`REGEXP_INSTR('500 Oracle Parkway, Redwood Shores, CA',`
`'[s|r|p][[:alpha:]]{6}', 3, 2, 1, 'i')`
`"Exemple 2" FROM DUAL;`
	`Exemple 2`
	`----------`
	`28`

`SELECT endroit, SUBSTR(description,`
`REGEXP_INSTR(description,`
`'[^ ]+ acres | [^ ]+-acre',1,1,0,'i'),`
`REGEXP_INSTR(description,`
`'[^ ]+ acres | [^ ]+-acre',1,1,1,'i')`
`- REGEXP_INSTR(description,`
`'[^ ]+ acres | [^ ]+-acre',1,1,0,'i'))`
`"SURFACE"`
`FROM Parcs`
`WHERE REGEXP_LIKE(description,`
`'[^ ]+ acres | [^ ]+-acre','i');`
	`ENDROIT SURFACE`
	`-------- -------------`
	`P1 1800 acres`
	`P4 217-acre`
	`P5 60,000 acres`
	`P6 40,000+ acres`

Fonction REGEXP_SUBSTR

La fonction REGEXP_SUBSTR étend la fonction SUBSTR en permettant d'extraire une sous-chaîne à partir d'une expression régulière. Le fonctionnement de cette fonction est similaire à celui de REGEXP_INSTR sauf qu'au lieu de retourner la position d'une sous-chaîne, REGEXP_SUBSTR retourne la sous-chaîne elle-même. La syntaxe de cette fonction est la suivante :

```
REGEXP_SUBSTR (source, modèle
      [, position [, occurrence [, paramètre ] ] ] )
```

- *source* indique la chaîne à examiner (une colonne de type CHAR, VARCHAR2, NCHAR, NVARCHAR2, CLOB, ou NCLOB).
- *modèle* désigne l'expression régulière (jusqu'à 512 octets).
- *position* est un entier positif indiquant la position de début de recherche (par défaut 1).
- *occurrence* est un entier positif précisant quelle est l'occurrence de l'expression recherchée (par défaut, 1 indiquant que la première occurrence est à examiner, *n* pour examiner la $n^{ième}$).
- *paramètre* a la même signification que dans l'utilisation des fonctions REGEXP_LIKE et REGEXP_REPLACE et REGEXP_INSTSR.

Le tableau suivant illustre quelques utilisations de cette fonction d'extraction reposant sur les exemples précédents. Le premier exemple retourne la chaîne correspondant au quatrième mot. Le deuxième exemple retourne la chaîne correspondant à la seconde occurrence d'un mot de sept

lettres commençant par s, r, ou p (casse indifférente). Dans le troisième exemple, nous simplifions l'extraction précédemment étudiée.

Web

Tableau 13-9 Utilisations de la fonction `REGEXP_SUBSTR`

Requête	Résultat SQL*Plus

`SELECT REGEXP_SUBSTR('IUT Dept GTR, 31703 Blagnac', '[^ ]+', 1, 4) "Ex. 1" FROM DUAL;`	`Ex. 1`
	`-----`
	`31703`

`SELECT REGEXP_SUBSTR('500 Oracle Parkway, Redwood Shores, CA', '[s|r|p][[:alpha:]]{6}', 1, 2, 'i') "Ex. 2" FROM DUAL;`	`Ex. 2`
	`-------`
	`Redwood`

`COLUMN surface format a13 heading "Ex. 3"`
`SELECT endroit,`
`REGEXP_SUBSTR(description,'[^ ]+[- ]acres?',1,1,'i')`
`surface`
`FROM Parcs`
`WHERE REGEXP_LIKE(description, '[^ ]+[- ]acres?','i');`
	`ENDROIT  Ex. 3`
	`--------  -------------`
	`P1  1800 acres`
	`P4  217-acre`
	`P5  60,000 acres`
	`P6  40,000+ acres`

Bibliographie

[GEN 03] J. GENNICK, *First Expressions,* Oracle Magazine*,* pp 95-98, sept-oct. 2003.

[SOU 04] C. SOUTOU, *Programmer objet sous Oracle*, Vuibert 2004.