Langage SQL

Mémento Syntaxique

Oracle

R. Chapuis Année 2011 - 2012

Modifié A. ARSANE

# Les types de données

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caractère** | **CHAR** | permet de stocker une chaîne de caractères de longueur fixe . | CHAR (longueur) | *longueur* : nombre de caractères qu’il est possible de stocker dans le champ. Cette valeur est facultative. La longueur par défaut est d'un caractère. La longueur maximale est de 2000 caractères. |
| **Caractère** | **VARCHAR2** | permet de stocker une chaîne de caractères de longueur variable. | VARCHAR2 (longueur) | *longueur* : nombre de caractères qu’il est possible de stocker dans le champ. Cette valeur est facultative. La longueur par défaut est d'un caractère. La longueur maximale est de 4000 caractères. |
| **Numérique** | **NUMBER** | correspond à une donnée stockée sous forme "décimal flottant" avec une mantisse de longueur variable . | NUMBER [(précision [, échelle])] | les paramètres  *précision* et é*chelle*s sont facultatifs. P*récision* : nombre entier de chiffres significatifs. *Echelle* : nombre de chiffres à droite de la marque décimale |
| **Date** | **DATE** | permet de stocker des informations comprenant une date et/ou une heure. | DATE | le format standard est DD-MON-YY |
| **Long** | **LONG** | permet de stocker des chaînes de caractères d’une longueur maximale de 2 gigas octets. | LONG |  |
| **Binaire** | **RAW** | permet de stocker des données de type binaire de longueur fixe. | RAW(n) | *n* : nombre d'octets. La longueur maximale est de 2000 octets. |
| **Binaire Long** | **LONG RAW** | Type identique à LONG pour des données de type binaire. | LONG RAW |  |

# SQL et les Opérateurs de base

## Opérateur de Projection

SELECT [DISTINCT] liste\_attributs  
FROM nom-de-table | requête

Où :

1. attribut = nom\_table.nom\_colonne *ou*

nom-colonne *ou*

expression. ou

requête.

1. liste\_attribut = attribut, attribut, ... *ou* \* (tous les attributs de la relation)
2. DISTINCT élimine les lignes résultats identiques.

### Tri du résultat d'une requête

SELECT [DISTINCT] liste\_attributs  
FROM nom-de-table  
ORDER BY nom\_col1 | numéro\_Col1 | Expression [DESC] [, nom\_col2 | numéro\_Col2 | Expression [DESC] ] ...

## Opérateur de sélection

SELECT \*  
FROM nom-de-table  
WHERE prédicat

### Prédicat de sélection

#### Prédicat Simple

Un prédicat simple est le résultat de la comparaison de deux expressions (nom colonne, requête, expression calculée) au moyen d’un opérateur de comparaison :

|  |  |
| --- | --- |
| = | égal |
| != ou <> | différent |
| > | supérieur |
| >= | supérieur ou égal |
| < | inférieur |
| <= | inférieur ou égal. |

Il faut ajouter à ces opérateurs de comparaison classiques les prédicats suivants :

|  |  |
| --- | --- |
| **between** | expr1 BETWEEN expr2 AND expr3 |
|  | Vrai si expression\_1 est compris entre expression\_2 et expression\_3, bornes incluses; |
| **IN** | expression\_1 IN (expression\_2, expression\_3...) |
|  | Vrai si expression\_1 est égale à l’une des expressions de la liste entre parenthèses; |
| **LIKE** | expr LIKE chaîne |
|  | où chaîne est une chaîne de caractères pouvant contenir l’un des caractères génériques de substitution : **\_** (souligné) qui remplace un seul caractère ou  **%** qui remplace une chaîne de caractères de longueur quelconque, y compris de longueur nulle. |
| **IS [NOT] NULL** | test de la valeur NULL |

## Opérateur Produit Cartésien

SELECT \*  
FROM Nom-de-table-1, Nom-de-table-2

## Opérateur de Jointure

### Equijointure

SELECT \*  
FROM Nom-de-table-1 | requête, Nom-de-table-2 | requête  
WHERE pivot

avec pivot :

[table1.]colonne = [table2.]colonne

### Jointure externe

Elle s'exprime au niveau du pivot : l’opérateur (+), placé après le nom d’une des deux colonnes qui composent le pivot de jointure, indique la table subordonnée (où manquent des éléments) :

[table1.]colonne(+) = [table2.]colonne

ou

[table1.]colonne = [table2.]colonne(+)

### Thétajointure

Une thétajointure est une jointure dont l'expression du pivot utilise des opérateurs autre que l’égalité, tel que : <, <=, >, >= , != ou <>.

### Autojointure

Une autojointure est la jointure d’une table à elle-même.

Il est alors obligatoire d’utiliser des *alias* pour différencier les différentes copies.

### Alias de nom de table

Un a*lias* est un synonyme donné à un nom de table, valable uniquement à l’intérieur de la requête (synonyme local).

SELECT \*  
FROM Nom-de-table-1| requête *alias\_1*, Nom-de-table | requête-2 *alias\_2*

WHERE *alias\_1*.col\_1 = *alias\_2*.col\_2 ...

### Jointures multiples et sélections simultanées

SELECT ……  
FROM Nom-de-table-1, Nom-de-table-2, ..., Nom-de-table-n  
WHERE pivot-1 AND pivot-2 ...  
[AND condition de sélection sur table-1]  
[AND condition de sélection sur table-2]  
[ ... ]

## Opérateurs ensemblistes : union, intersection, différence

SELECT liste\_attributs FROM ... [WHERE ... ]  
{UNION | INTERSECT | MINUS}  
SELECT liste\_attributs FROM ... [WHERE ... ]

# Les Sous-requêtes

## Syntaxe :

SELECT \*  
FROM nom-de-table  
WHERE attribut opérateur (SELECT ... ) ;

ou

SELECT \*  
FROM nom-de-table  
WHERE [NOT] EXISTS (SELECT ... ) ;

## Opérateurs

1. opérateur simple : =, !=, <>, <, >, <=, >=
2. IN : La condition est vraie si elle est vérifiée pour une des valeurs renvoyées par la sous-requête.
3. EXISTS :le prédicat est vrai si l'exécution de la sous-requête renvoie au moins une ligne.
4. opérateur simple (=, !=, <>, <, >, <=, >= ) précédé de ALL ou ANY :
5. ANY : La comparaison sera vraie si elle est vraie pour au moins une des valeurs renvoyées par la sous-requête.
6. ALL : La comparaison sera vraie si elle est vraie pour chacune des valeurs renvoyées par la sous-requête.

# Expressions et Fonctions

## Expression arithmétique

### Opérateurs

Une expression arithmétique peut contenir :

1. des noms de colonnes,
2. des constantes,
3. des fonctions arithmétiques;

combinées au moyen des opérateurs :

|  |  |
| --- | --- |
| + | addition, |
| ‑ | soustraction, |
| \* | multiplication, |
| / | division. |

### Fonctions

Les principales fonctions arithmétiques implicites sont :

POWER(n, m) :

élève n à la puissance m. (m doit être une valeur entière et peut être négatif).

ROUND (n [,d]) :

arrondit n à dix puissance d (par défaut d=0).

TRUNC (n [,d]) :

tronque n à dix puissance d (par défaut d=0).

CEIL (n) :

prend la valeur de l’entier directement supérieur ou égal à n.

FLOOR (n) :

prend la valeur de la partie entière de n (entier directement inférieur ou égal à n).

ABS(n)

valeur absolue de n.

MOD(n,m)

donne le reste de la division de n par m.

SIGN(n)

vaut :

|  |  |
| --- | --- |
| 1 | si n est positif |
| 0 | si n est égal à 0 |
| -1 | si n est négatif |

SQRT(n)

racine carrée de n (prend la valeur NULL si n est négatif).

*Etc.*

## Expression sur chaînes de caractères

### Opérateur de concaténation

Cet opérateur se note au moyen de deux caractères "|" (barre verticale) accolés.

### Fonctions

LENGTH (chaîne)

renvoie comme valeur la longueur de la chaîne.

SUBSTR (chaîne, pos [, long])

extrait de la *chaîne* une sous-chaîne de longueur *long* commençant en position *pos* de la *chaîne*.

Le paramètre longueur est facultatif; par défaut, la sous-chaîne va jusqu’à l’extrémité de la chaîne.

INSTR (chaîne, sous-chaîne [, pos [, n])

renvoie comme valeur la position de la *sous-chaîne* dans la *chaîne* (les positions sont numérotées à partir de 1). Zéro signifie que la sous-chaîne n’a pas été trouvée dans la chaîne.

La recherche commence à la position *pos* de la chaîne (paramètre facultatif qui vaut 1 par défaut). Une valeur négative de *pos* signifie une position par rapport à la fin de la chaîne.

Le dernier paramètre *n* permet de rechercher la n-ième occurrence de la sous-chaîne dans la chaîne. Ce paramètre facultatif vaut 1 par défaut.

UPPER (chaîne)

convertit les minuscules en majuscules

LOWER (chaîne)

convertit les majuscules en minuscules .

INITCAP (chaîne)

met en majuscule la première lettre de chaque mot de la *chaîne*, et toutes les autres lettres en minuscule. Sont considérés comme séparateurs de mots, tous les caractères qui ne sont pas des lettres.

SOUNDEX (chaîne)

calcule une valeur phonétique qui peut être comparée dans un prédicat à la valeur phonétique d’une autre chaîne.

On peut ainsi, par exemple, utiliser comme critères de recherche des mots dont on ne connaît pas l’orthographe exacte.

LPAD (chaîne, long [,car])

complète (ou tronque) la *chaîne* à la longueur *long*. La *chaîne* est éventuellement complétée à gauche par le caractère (ou la chaîne de caractères) *car*.

Le paramètre *car* est optionnel. Par défaut, la chaîne est complétée par des blancs.

RPAD (chaîne, long [,car])

fonction analogue à LPAD, la chaîne étant complétée ou tronquée à droite.

LTRIM (chaîne, car)

supprime les caractères de l'extrémité gauche de la chaîne tant qu’ils appartiennent à l’ensemble de caractères de référence. Exemple : LTRIM (ename, 'SC').

RTRIM (chaîne, car)

fonction analogue à LTRIM, les caractères étant supprimés à l’extrémité droite de la chaîne.

TRANSLATE (chaîne, car\_source, car\_cible)

effectue le codage d’une chaîne de caractères :

*car\_source* et *car\_cible* sont des chaînes de caractères considérées comme des ensembles de caractères. La fonction TRANSLATE remplace chaque caractère de la chaîne présent dans l’ensemble de caractères *car\_source* par le caractère correspondant (de même position) de l’ensemble *car\_cible*.

REPLACE (chaîne, chaîne\_source, chaîne\_remplacement)

remplace dans la chaîne de caractères toutes les séquences chaîne\_source par la séquence chaîne\_remplacement.

Etc.

## Expression de type DATE

### Opérateurs

Le langage SQL offre deux opérations sur le type date :

1. l'ajout d'une durée, exprimée en nombre de jours, à une date pour obtenir une nouvelle date :

date +/‑ nombre

Le résultat est une date obtenue en ajoutant le nombre de jours *nombre* à une expression de type date.

1. l'obtention du nombre de jours séparant deux dates :

date2 ‑ date1

Le résultat est le nombre de jours entre les deux dates. Le résultat peut être une valeur décimale si les valeurs de date1 et/ou date2 contiennent une notion d’heure.

### Fonctions

ADD\_MONTHS (date, nombre)

ajoute ou soustrait un certain nombre de mois à une date; le résultat est une date.

MONTHS\_BETWEEN (date2, date1)

renvoie comme valeur la différence date2-date1 exprimée en nombre de mois.

La partie fractionnaire du résultat est calculée en considérant chaque jour comme égal à 1/31-ième de mois.

LAST\_DAY (date)

renvoie comme valeur la date du dernier jour du mois correspondant à *date*

NEXT\_DAY (date, nom de jour)

donne la date du prochain jour de la semaine spécifiÈ dans *nom de jour*.

ROUND (date [, précision])

arrondit la date à la précision spécifiée. La précision est indiquée en utilisant un des masques de mise en forme de la date. On peut ainsi arrondir une date à l’année, au mois, à la minute, etc.

Par défaut, la précision est le jour.

TRUNC (date [, précision ])

Tronque la date à la précision spécifiée.

Les paramètres sont analogues à ceux de la fonction ROUND.

SYSDATE

SYSDATE est un mot réservé que l’on peut utiliser dans une expression de type date et qui a pour valeur la date et l’heure courante du système d’exploitation hôte.

## Fonctions de conversion

### Fonction de formatage d’un nombre : TO\_CHAR()

La fonction TO\_CHAR permet de convertir un nombre en chaîne de caractères en fonction d’un masque :

TO\_CHAR (nombre, masque)

*nombre* est une expression de type numérique

*masque* est une chaîne de caractères pouvant contenir les caractères suivants :

1. caractère de substitution :

|  |  |
| --- | --- |
| 9 | représente un chiffre (non représenté dans le cas d'un zéro non significatif) |
| 0 | représente un chiffre (présent même si non significatif) |
| . | point décimal apparent |
| V | définit la position de la séparation partie entière - partie fractionnaire |

1. caractères de simple insertion :

|  |  |
| --- | --- |
| , | une virgule apparaîtra à cet endroit |
| $ | un $ précédera le premier chiffre significatif |

1. autres :

|  |  |
| --- | --- |
| B | le nombre sera représenté par des blancs s’il vaut zéro |
| EEEE | le nombre est représenté avec un exposant (le spécifier avant MI ou PR) |
| MI | le signe négatif sera à droite (à droite du masque) |
| PR | une valeur négative sera entre < > (à droite du masque) |

### Fonction de conversion d'une chaîne de caractères en nombre : TO\_NUMBER()

La fonction TO\_NUMBER convertit la chaîne de caractères en numérique.

TO\_NUMBER (chaîne)

La *chaîne* de caractères représente une valeur numérique et doit être conforme aux règles d’écriture d’une constante numérique.

### Fonction de conversion d’une date en chaîne de caractères : TO\_CHAR()

TO\_CHAR (date, masque)

*masque* indique quelle partie de la date doit apparaître et correspond à une combinaison des codes suivants :

|  |  |
| --- | --- |
| SCC | siècle (avec signe) |
| CC | siècle |
| SY,YYY | année (avec signe et virgule) |
| Y,YYY | année (avec virgule) |
| YYYY | année |
| YYY | trois derniers chiffres de l’année |
| YY | deux derniers chiffres de l’année |
| Y | dernier chiffre de l’année |
| Q | numéro du trimestre dans l’année |
| WW | numéro de la semaine dans l’année |
| W | numéro de la semaine dans le mois |
| MM | numéro du mois |
| DDD | numéro du jour dans l’année |
| DD | numéro du jour dans le mois |
| D | numéro du jour dans la semaine |
| HH ou HH12 | heure (sur 12 heures) |
| HH24 | heure (sur 24 heures) |
| MI | minutes |
| SS | secondes |
| SSSSS | secondes après minuit |
| J | jour du calendrier Julien |

Les formats suivants permettent d’obtenir des dates en lettres (en anglais):

|  |  |
| --- | --- |
| SYEAR ou YEAR | année en toutes lettres |
| MONTH | nom du mois |
| MON | nom du mois abrégé sur 3 lettres |
| DAY | nom du jour |
| DY | nom du jour abrégé sur 3 lettres |
| AM ou PM | indication AM ou PM |
| BC ou AD | indication BC (avant Jésus Christ) ou AD (après Jésus Christ) |

Les suffixes suivants modifient la présentation du nombre auquel ils sont accolés:

|  |  |
| --- | --- |
| TH | ajout du suffixe ordinal ST, ND, RD ou TH |
| SP | nombre en toutes lettres |

Tout caractère spécial inséré dans le format sera reproduit dans la chaîne de caractères résultat.

### Fonction de conversion d’une chaîne de caractères en date : TO\_DATE()

La fonction TO\_DATE convertit une chaîne de caractères en une donnée de type date :

TO\_DATE (chaîne, masque)

L’expression du masque est identique à celui de la fonction TO\_CHAR.

## Autres Fonctions

DECODE (Crit, val1, res1 [,val2, res2], def)

Cette fonction permet de choisir une valeur parmi une liste d’expressions, en fonction de la valeur prise par une expression servant de critère de sélection. Le résultat est *res1* si l’expression *Crit* a la valeur *val1*; *res2* si *Crit* a la valeur *va*l2, ....; si l’expression *Crit* n’est égale à aucune des expressions *val1*, *val2*... , le résultat est la valeur par défaut *def*.

Les expressions résultat (*res1*, *res2*, ..., *def*) peuvent être de types différents: caractère et numérique, ou caractère et date (le résultat est du type de la première expression rencontrée dans la fonction DECODE).

La fonction DECODE permet également de mélanger dans une colonne résultat des informations venant de plusieurs colonnes d’une même table.

ASCII (chaîne)

donne un nombre qui représente le code (ASCII ou EBCDIC) selon la machine hôte) du premier caractère de la chaîne.

CHR (n)

donne le caractère dont le code (ASCII ou EBCDIC) est égal à l’expression numérique n.

GREATEST (exp1, exp2,...)

donne la plus grande des valeurs des expressions arguments. Cette expression peut être de type arithmétique, caractère ou date.

LEAST (exp1, exp2,...)

renvoie la plus petite valeur des expressions arguments. Cette expression peut être de type arithmétique, caractère ou date.

NVL (expr1, expr2)

Une valeur NULL en SQL est une valeur non définie.

Lorsque l’un des termes d’une expression à la valeur NULL, l’expression entière prend la valeur NULL.

D’autre part, un prédicat comportant une comparaison avec une expression ayant la valeur NULL prendra toujours la valeur faux.

La fonction NVL permet de remplacer une valeur NULL par une valeur significative :

NVL(expr1, expr2) prend la valeur expr2, si expr1 est NULL, sinon NVL(expr1, expr2) prend la valeur expr1.

# Groupement des données

## Fonctions d’agrégat

|  |  |
| --- | --- |
| AVG(expression) | moyenne des valeurs d’une colonne |
| SUM(expression) | somme des valeurs d’une colonne |
| MIN(expression) | prend pour valeur la plus petite des valeurs |
| MAX(expression) | prend pour valeur la plus grande des valeurs |
| COUNT(expression) | dénombre une collection de valeurs |
| VARIANCE(expression) | variance |
| STDDEV (expression) | écart‑type, ou déviation standard |
|  |  |

Remarques **:**

|  |  |
| --- | --- |
| Cas particulier de la fonction : COUNT() |  |
| COUNT(\*) | nombre de lignes satisfaisant à la condition WHERE. |
| COUNT(expression.) | nombre de lignes ayant une valeur non NULL pour une expression. |
| COUNT(DISTINCT expression.) | nombre de valeurs distinctes et non NULL pour une expression. |

## Syntaxe

SELECT col [,col ... ]  
FROM table [, table ..]  
[WHERE prédicat]  
[GROUP BY expression [, expression ...]  
[HAVING prédicat] ]

# Traitement des structures arborescentes

## Syntaxe

SELECT col [,col ... ]  
FROM table [, table ..]  
[WHERE prédicat]  
CONNECT BY condition  
START WITH prédicat

Où :

CONNECT BY condition

condition s’exprime par :

colonne-1 = PRIOR colonne-2  
ou  
PRIOR colonne-1 = colonne-2

# Langage de manipulation des données

|  |  |
| --- | --- |
| **UPDATE** | mise à jour de lignes |
| **INSERT** | ajout de lignes |
| **DELETE** | suppression des lignes sélectionnées |
| **TRUNCATE** | suppression des lignes d’une table |

## Modification des données - UPDATE

UPDATE table  
SET colonne-1 = {expresion-1 | (SELECT ...) }  
 [, colonne-2 = {expression-2 | (SELECT ...) } ... ]  
[WHERE prédicat]

## Insertion de lignes - INSERT

### Insertion d’une ligne

INSERT INTO table [ (colonne-1[, colonne-2] ) ]   
VALUES (valeur-1 [, valeur-2 ..] )

### Insertion de plusieurs lignes

INSERT INTO table [(colonne-1, colonne-2,...)]  
SELECT ...

### Création et insertion simultanées

CREATE TABLE nom\_table  
AS SELECT ...

## Suppression de lignes - DELETE

DELETE FROM table  
[WHERE prédicat ]

## Suppression de lignes - TRUNCATE

TRUNCATE TABLE table  
[DROP | REUSE STORAGE ]

|  |  |
| --- | --- |
| DROP STORAGE | après suppression des lignes de la table, les blocs mémoire libérés sont réaffectés à la base, tout en conservant l’extension initiale pour la table. |
| REUSE STORAGE | permet de conserver l’espace mémoire alloué à la table. |

## Gestion des transactions

### Validation d’une transaction

COMMIT

### Annulation d’une transaction

ROLLBACK

# Langage PL/SQL

## Structure d'un bloc

[**DECLARE**

Déclaration des variables locales au bloc, constantes, exceptions, curseurs.

*Section facultative*.

]

**BEBIN**

Instruction PL/SQL*.*

*Section* obligatoire.

[**BEGIN**

possibilité de blocs imbriqués

**END;** ]

[**EXCEPTION**

Traitement des erreurs.

*Section facultative*.

]

**END;**

## Liste des instructions utilisables dans un bloc PL/SQL :

1. des instructions d’affectation,
2. des instructions SQL :
3. CLOSE
4. COMMIT
5. DELETE
6. FETCH
7. INSERT
8. LOCK TABLE
9. OPEN
10. ROLLBACK
11. SAVEPOINT
12. SELECT
13. SET TRANSACTION
14. UPDATE
15. des instructions de contrôle itératif ou répétitif,
16. des instructions de gestion de curseurs,
17. des instructions de gestion des erreurs.

Chaque instruction est terminée par le caractère "**;**"

## Types de variable

### Types scalaires

1. Variable de type SQL :

nom variable nom\_type

ou nom\_type peut être : CHAR, NUMBER, DATE, VARCHAR2

1. Variable par référence à une colonne d'une table du dictionnaire de   
   données :

nom variable nom table. nom colonne % type.

1. Autres types scalaires définis :

nom variable nom\_type

ou nom\_type peut être : BOOLEAN, BINARY\_INTEGER, DECIMAL, FLOAT, INTEGER,REAL, SMALLINT, ROWID,....

1. Variable de lien

### Types composés

#### Enregistrement

soit par référence à une description de table existante :

nom variable nom table.ROWTYPE

soit par une description explicite comportant deux étapes :

1. déclaration du type enregistrement :

TYPE nom\_type IS RECORD  
(nom\_champ type\_champ, nom\_champ type\_champ, ...);

1. déclaration de la variable de type enregistrement :

nom\_variable nom\_type;

#### Table

La déclaration comporte deux étapes :

1. déclaration du type de la table.

TYPE nom\_type IS TABLE OF type\_champ,  
INDEX BY BINARY\_INTEGER;

1. déclaration de la variable de type table :

nom\_variable nom\_type

### Variable et constante

1. Attribution d'une valeur initiale :

nom variable type := valeur

nom variable type DEFAULT valeur

1. Déclaration de constante :

nom variable CONSTANT type := valeur

## Instructions PL/SQL.

### Opérateur d'affectation

nom variable := valeur

### Ordre SELECT inclus dans un bloc PL/SQL

SELECT liste d’expressions  
INTO liste de variables  
FROM .......

### Instructions de contrôle

#### Structure alternative : IF

Première forme :

IF condition THEN  
instructions;  
END IF;

Deuxième forme :

IF condition THEN  
instructions;  
ELSE  
instructions;  
END IF;

Troisième forme

IF condition THEN  
instructions;  
ELSIF condition THEN  
instructions;  
ELSE  
instructions;.  
END IF;

#### Structures répétitives

LOOP

LOOP  
 instructions  
END LOOP;

EXIT sortie inconditionnelle de boucle

EXIT permet de sortir de la boucle en donnant le contrôle à l'instruction qui suit END LOOP.

EXIT WHEN sortie conditionnelle de boucle

EXIT WHEN condition

Permet de sortir de la boucle si la condition est vrai.

FOR

FOR variable\_indice IN [REVERSE] valeur\_début .. valeur\_fin  
LOOP  
 instructions;  
END LOOP;

où :

|  |  |
| --- | --- |
| variable\_indice | variable locale à la boucle non déclarée dans DECLARE |
| valeur\_début | variable locale précédemment déclarée et initialisée, ou constante |
| valeur\_fin | variable locale déclarée et initialisée, ou constante. |
|  |  |

WHILE

WHILE condition  
LOOP  
 instructions;  
END LOOP;

## Curseur

### Déclaration du curseur

CURSOR nom curseur IS requête

Un curseur peut être défini avec des paramètres :

CURSOR nom curseur (nom\_paramètre type [:= valeur par défaut] [,…])   
IS requête

### Ouverture du curseur

OPEN nom\_curseur

ou :

OPEN nom\_curseur (paramètres effectifs)

### Fermeture du curseur

CLOSE nom\_curseur

### Traitement des lignes

FETCH nom\_curseur INTO liste\_variables.

#### Forme syntaxique condensée

DECLARE  
 CURSOR nom\_curseur IS requête;  
BEGIN  
FOR nom\_enregistrement IN nom curseur [(paramètres effectifs)]  
LOOP  
 traitement;  
END LOOP;

### Statut d'un curseur

|  |  |
| --- | --- |
| 1. %FOUND | vrai si exécution correcte de l’ordre SQL |
| 1. %NOTFOUND | vrai si exécution incorrecte de l’ordre SQL |
| 1. %ISOPEN | vrai si curseur ouvert |
| 1. %ROWCOUNT | nombre de lignes traitées par l’ordre SQL, évolue à chaque ligne distribuée |

## Gestion des erreurs standard

### Identification de l'erreur

|  |  |
| --- | --- |
| *code erreur  SQLCODE* | *nom erreur* |
|  |  |
| +100 | NO\_DATA\_FOUND |
| -1 | DUP\_VAL\_ON\_INDEX |
| -6502 | VALUE\_ERROR |
| -1001 | INVALID CURSOR |
| -1722 | INVALID NUMBER |
| -6501 | PROGRAM ERROR |
| -1017 | LOGIN DENIED |
| -1422 | TOO\_MANY\_ROWS |

### Description du traitement de l'erreur

EXCEPTION  
 WHEN nom\_erreur1 THEN   
 Traitement erreur\_1;  
 .................  
 WHEN nom\_erreur\_i [OR nom\_erreur\_j] THEN   
 Traitement erreur\_n;  
 WHEN OTHERS THEN  
 Traitement commun autres erreurs;

### Gestion des erreurs (anomalies) utilisateur

#### Déclaration de l'anomalie

DECLARE  
nom\_anomalie EXCEPTION;

#### Déclenchement du traitement

RAISE nom\_anomalie;

# Langage de Description des Données

## Création d'une table

CREATE TABLE [schéma.]nom\_table  
(colonne [,colonne [ DEFAULT expression ] [ contrainte de colonne] ]...)  
[Contrainte de table ]  
[PCTFREE pourcentage] [PCTUSED pourcentage]  
[INITRANS nombre] [MAXTRANS nombre]  
[TABLESPACE nom\_tablespace]  
[STORAGE valeurs]  
[CLUSTER nom\_cluster (colonne [,colonne] ...)]  
[ENABLE contrainte ]  
[DESABLE contrainte]  
[AS query]

avec :

|  |  |
| --- | --- |
|  |  |
| *schéma* | propriétaire de la table, par défaut l'utilisateur qui crée la table. |
| *nom* | le nom de la table doit être unique pour ce schéma. |
| *colonne* | définit le nom de colonne et son type. Un nom de table doit être unique pour la table, mais plusieurs tables peuvent avoir des noms de colonne identiques. Une table peut avoir au maximum 254 colonnes. |
| *contrainte colonne* | permet de définir une contrainte particulière pour la colonne. |
| *contrainte table* | permet de définir des contraintes au niveau de la table. |
| *Default expression* | permet de définir une valeur par défaut pour la colonne qui sera prise si aucune valeur n'est spécifiée dans une commande INSERT. Ce peut être une constante, USER ou SYSDATE. |
| *PCTFREE, PCTUSED* | paramètres de gestion de l'espace physique de la table |
| *INITRANS, MAXTRANS* | paramètres de gestion des transactions concurrentes sur la table. |
| *TABLESPACE* | nom du tablespace dans lequel la table doit être créée. Par défaut la table est créée dans le tablespace SYSTEM ou dans le tablespace affecté à l'utilisateur par la commande CREATE USER. |
| *STORAGE valeurs* | identique à la clause DEFAULT STORAGE de création de tablespace. Son utilisation permet de spécifier des valeurs particulières pour la table à créer en lieu et place des valeurs définies au niveau du tablespace. |
| *CLUSTER* | utilisé si la table est mise en cluster. |
| *ENABLE contrainte* | permet de définir et d'activer une contrainte d'intégrité, non déjà définie dans la clause *colonne*. |
| *DESABLE contrainte* | permet de désactiver une contrainte d'intégrité définie dans la clause *colonne*. |
| *AS query* | expression d'une requête permettant la création de lignes dans la table en même temps que la création de la table. Cette clause doit être la dernière de l'ordre de création. Si l'ordre CREATE possède des descriptions de colonnes, celles-ci ne doivent préciser que le nom de colonne; le type associé est déduit du type de l'attribut correspondant dans la requête |
|  |  |

*pour une contrainte de colonne :*

[CONSTRAINT nom\_contrainte]  
{[NOT] NULL  
| {UNIQUE | PRIMARY KEY }  
| REFERENCE [schéma.]table[(colonne)]  
 [ON DELETE CASCADE]  
| CHECK (condition)}  
{[USING INDEX paramètres se stockage ]  
 [EXCEPTIONS INTO [schéma.]table  
 | DISABLE] }

*pour une contrainte de table :*

[CONSTRAINT nom\_contrainte]  
{{UNIQUE | PRIMARY KEY } (colonne [,colonne] ... )  
| FOREIGN KEY (colonne [,colonne] ...)  
 REFERENCE [schéma.]table[(colonne [,colonne] ...)]  
 [ON DELETE CASCADE]  
| CHECK (condition)}  
{[USING INDEX paramètres se stockage ]  
 [EXCEPTIONS INTO [schéma.]table  
 | DISABLE] }

avec :

|  |  |
| --- | --- |
| USING INDEX | spécifie les paramètres de stockage pour l'index créé pour une clause PRIMARY ou UNIQUE. |
| EXCEPTION INTO | indique la table dans laquelle seront envoyés les messages d'erreur en cas de violation des contraintes. Cette table doit être définie auparavant. |
| DISABLE | permet de désactiver la contrainte qui est activée par défaut. |

## Modification de la description d'une table

### Ajout d'une (plusieurs) colonne(s)

ALTER TABLE nom\_table  
ADD (colonne [,colonne] ...)

### Modification de la description d'une colonne

ALTER TABLE nom\_table  
MODIFY (colonne [,colonne] ....)

### Modification des paramètres de stockage

ALTER TABLE [schéma.]nom\_table  
[PCTFREE pourcentage] [PCTUSED pourcentage]  
[INITRANS nombre] [MAXTRANS nombre]  
[STORAGE valeurs]  
[DROP contrainte]

### Modification des contraintes d'intégrité

ALTER TABLE [schéma.]table  
[DROP {PRIMARY KEY  
 | UNIQUE (colonne [,colonne] ..)  
 | CONSTRAINT nom\_contrainte }  
 [(CASCADE] ]  
[ENABLE nom\_contrainte | DISABLE nom\_contrainte]

## Suppression d'une table

DROP TABLE [schéma.]nom\_table  
[CASCADE CONSTRAINTS]

# Interface avec un langage hôte

## Ordres SLQ intégré (Embedded SQL)

|  |  |
| --- | --- |
|  |  |
| *Ordres intéractifs* | *Ordres non intéractifs* |
|  |  |
| ALTER SESSION | CLOSE |
| ALTER | CONNECT |
| COMMENT | DESCRIBE |
| COMMIT | EXECUTE |
| ANALYSE | FETCH |
| AUDIT | OPEN |
| CREATE | PREPARE |
| DELETE |  |
| DROP |  |
| EXPLAIN PLAN |  |
| GRANT |  |
| INSERT |  |
| LOCK TABLE |  |
| NOAUDIT |  |
| RENAME |  |
| REVOKE |  |
| ROLLBACK |  |
| SAVEPOINT |  |
| SELECT |  |
| SET ROLE |  |
| SET TRANSACTION |  |
| TRUNCATE |  |
| UPDATE |  |
|  |  |

## Variable hôte

Déclaration :

EXEC SQL BEGIN DECLARE SECTION

Description des variables selon syntaxe du langage hôte

EXEC SQL END DECLARE SECTION

## Connexion

### Connexion à une seule base

EXEC SQL CONNECT :nom-utilisateur IDENTIFIED BY :mot-passe;

ou

EXEC SQL CONNECT :connexion

### Connexions multiples

EXEC SQL CONNECT :nom-utilisateur IDENTIFIED BY :mot-passe   
AT nom-base USING :chaîne-de-connexion

où

*chaîne-de-connexion* est une variable hôte qui contient le protocole et l’adresse de la machine distante.

Lorsque plusieurs bases sont accessibles dans un même programme, il faut spécifier, pour chaque ordre SQL, la base visée en indiquant son nom de base dans la clause **AT** de l'ordre EXEC SQL, sous la forme :

EXEC SQL AT nom-base ordre-SQL

## Ordre SELECT inclus

EXEC SQL SELECT liste d’expressions  
INTO liste de variables  
FROM .......

## utilisation d'un CURSEUR.

### Déclaration

EXEC SQL DECLARE nom-curseur   
CURSOR FOR ordre-SQL

### Ouverture curseur

EXEC SQL OPEN nom-curseur

### Fermeture curseur

EXEC SQL CLOSE nom-curseur

### Distribution des lignes

EXEC SQL FETCH nom-curseur INTO liste variables hôtes

## Gestion des transactions

### Validation

EXEC SQL COMMIT WORK [RELEASE]

### Annulation

EXEC SQL ROLLBACK WORK [RELEASE]

### Gestion transactions avec points de reprise

EXEC SQL SAVEPOINT nom

EXEC SQL ROLLBACK TO SAVEPOINT nom

## Contrôle des transferts

### Variables indicatrices

Une variable indicatrice a une utilisation différente selon quelle est associée à :

1. *une requête*. Sa valeur indique le comportement de la requête et permet la detection d'éventuelles anomalies :

|  |  |
| --- | --- |
|  |  |
| **0** | la valeur renvoyée est non nulle et a pu être stockée dans la variable hôte associée. |
| **-1** | aucune valeur renvoyée, la variable hôte n'a pas été modifiée. |
| **> 0** | la valeur renvoyée a dû être tronquée pour être transférée dans la variable hôte. Si la valeur renvoyée est de type chaîne de caractères, la valeur de la variable indicatrice contient la longueur de la chaîne avant transaction. |
|  |  |

1. *une opération de mise à jour.* Elle permet alors d'attribuer une valeur NULL à une colonne par exécution d'un ordre INSERT ou UPDATE, en associant une variable indicatrice ayant la valeur -1.

### Zone de communication

EXEC SQL INCLUDE SQLCA

### SQLCODE

SQLCODE donne le status de l'ordre SQL :

|  |  |
| --- | --- |
| 0 | exécution normale |
| >0 | avertissement (warning) |
| <0 | erreur fatale |

## Traitements des erreurs

EXEC SQL WHENEVER <événement> <action>

où un événement peut être :

|  |  |
| --- | --- |
|  |  |
| **SQLERROR** | détection de la présence d'une erreur (SQLERROR <0). |
| **SQLWARNING** | détection de la présence d'une anomalie indiquée dans une des zones SQLWARN(2) à SQLWARN(8)). |
| **NOT FOUND** | détection de la fin de distribution de lignes pour une instruction FETCH (SQLCODE = + 1403). |

où action peut être :

|  |  |
| --- | --- |
|  |  |
| **STOP** | arrêt de l'exécution du programme; si une transaction est en cours, elle est annulée. |
| **CONTINUE** | l'exécution du programme continue en séquence; permet de neutraliser l'effet de WHENEVER. |
| **GO TO** | branchement à l'étiquette spécifiée. |
| **DO nom-procédure** | le programme transfert le contrôle à une procédure. |
|  |  |

# Les déclencheurs

## Déclencheur par ordre

Un déclencheur par ordre est exécuté une seule fois pour l’ensemble des lignes manipulées par l’événement.

### Création

La commande qui permet de créer un déclencheur et d’en stocker la description dans la base de données est

CREATE [OR REPLACE] TRIGGER [schéma.]nom\_déclencheur  
séquence  
événement [OR événement]   
ON nom\_table  
bloc\_PL/SQL

avec :

|  |  |
| --- | --- |
|  |  |
| Séquence | BEFORE ou AFTER |
| Événement | AFTER ou UPDATE ou DELETE |
| nom\_table | nom de la table à laquelle le déclencheur est liée |
| bloc\_PL/SQL | traitement à réaliser |
|  |  |

Remarque :

Dans le cas d’un ordre UPDATE il est possible de limiter la mise en œuvre du traitement à la modification de certaines colonnes :

UPDATE OF nom\_colonne [, nom\_colonne ..... ]

## Déclencheur ligne

Un déclencheur ligne est exécuté pour chacune des lignes manipulées par l’exécution de l’événement.

### Création

L’option FOR EACH ROW permet de déclarer un déclencheur ligne.

CREATE [OR REPLACE] TRIGGER [schéma.]nom\_déclencheur  
séquence  
événement [OR événement]   
ON nom\_table  
[REFERENCING {[ OLD [AS] ancien] | [NEW [AS] nouveau]}]  
FOR EACH ROW  
[WHEN condition]  
bloc\_PL/SQL

avec :

|  |  |
| --- | --- |
|  |  |
| Séquence | BEFORE ou AFTER. |
| Événement | AFTER ou UPDATE ou DELETE. |
| nom\_table | nom de la table à laquelle le déclencheur est lié. |
| Condition | prédicat exécuté pour chaque ligne manipulée. Il faut que la condition soit vraie pour que le traitement associé au déclencheur soit exécuté. L’expression de la condition ne peut pas contenir de requête SQL. |
| REFERENCING | permet de changer l’indicatif de référence de OLD par ancien et / ou de NEW par nouveau. |
| bloc\_PL/SQL | traitement à réaliser. |
|  |  |

Remarques :

La condition de la clause WHERE peut être utilisée pour restreindre l’action du déclencheur à certaines lignes de la table.

On peut faire référence, dans la condition de la clause WHERE ou dans le corps du traitement associé au déclencheur, à la valeur d’une colonne avant modification en préfixant le nom de colonne par **OLD**, et / ou à la valeur après modification en préfixant le nom de colonne par **NEW.** Selon l’ordre SQL en cause, la valeur prise en compte est :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | |  | |
|  | **OLD** | | **NEW** | |
| **INSERT** | NULL | | valeur créée | |
| **DELETE** | Valeur avant suppression | | NULL | |
| **UPDATE** | Valeur avant modification | | valeur après modification | |
|  |  | |  | |
|  | |  | |

## Suppression d’un déclencheur

Une déclaration d'un déclencheur peut être supprimée par l’ordre

DROP TRIGGER nom\_déclencheur

## Activation - désactivation d’un déclencheur

### Désactivation

Il est possible de désactiver un déclencheur particulier par l’ordre

ALTER TRIGGER nom\_déclencheur DISABLE

ou de désactiver tous les déclencheurs associés à une même table par :

ALTER TABLE nom\_table DISABLE ALL TRIGGERS.

### Activation

Il est possible de réactiver un déclencheur particulier par l’ordre :

ALTER TRIGGER nom\_déclencheur ENABLE

ou de réactiver tous les déclencheurs d’une même table par :

ALTER TABLE nom\_table ENABLE ALL TRIGGERS.

# Procédures et fonctions stockées

## Création d’une procédure

CREATE [OR REPLACE] PROCEDURE nom\_procédure  
[( argument [mode] typologie [, argument [mode] typologie ] ... )]  
[IS | AS] bloc PL/SQL

avec :

|  |  |
| --- | --- |
|  |  |
| nom\_procédure | nom attribué à la procédure. |
| Argument | nom du paramètre formel. |
| Mode | définit si le paramètre formel est en :  entrée : **IN** en sortie : **OUT** en entrée - sortie **IN OUT**  Le mode par défaut est IN. |
| Typologie | définit le type de donnée du paramètre. Tous les types définis dans le langage PL/SQL sont utilisables pour définir un argument. |
| bloc PL/SQL | corps de la procédure. |
|  |  |

## Création d’une fonction

CREATE [OR REPLACE] FUNCTION nom\_fonction  
[( argument [IN] typologie [, argument [mode] typologie ] ... )]  
RETURN typologie\_retour  
[IS | AS] bloc PL/SQL

avec

|  |  |
| --- | --- |
|  |  |
| nom\_fonction | nom attribué à la fonction. |
| Argument | nom du paramètre formel. |
| IN | définit le mode du paramètre formel qui est obligatoirement en entrée. |
| Typologie | définit le type de donnée du paramètre. Tous les types définis dans le langage PL/SQL sont utilisables pour définir un argument, mais sans spécification de taille pour les types explicites. |
| RETURN typologie\_retour | définit le type de la valeur retournée par la fonction. |
| bloc PL/SQL | corps de la procédure qui doit contenir une instruction RETURN(variable\_résultat), ou variable\_résultat sert à transmettre la valeur de la fonction. |
|  |  |

### Les paramètres

Les paramètres permettent de transférer des données entre procédure ou fonction et environnement appelant. Il existe trois modes de transfert :

|  |  |  |
| --- | --- | --- |
|  | |  |
| IN (défaut) | Permet de passer une valeur de l’environnement appelant vers la procédure ou la fonction; | |
| OUT | Permet à une procédure de retourner une valeur à l’environnement appelant; | |
| IN OUT | Permet d’utiliser un même paramètre pour passer une valeur de l’environnement appelant vers la procédure, puis d’obtenir une valeur en retour de l’exécution de la procédure. | |
|  |  | |

## Suppression d’une procédure ou fonction

Une procédure peut être supprimée par l’ordre :

DROP PROCEDURE nom\_procédure

Une fonction peut être supprimée par l’ordre :

DROP FUNCTION nom\_fonction

## Utilisation d’une procédure ou fonction stockée

### Appel aux procédures et fonctions

Le mode d’appel est différent selon l’environnement à partir duquel elle est utilisée :

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **Environnement d’appel** |  | **commande** | **Mode de transmission des paramètres** |
| Interactif | SQL\*PLUS | EXECUTE | substitution |
| Inclus dans une application cliente | Developer | appel direct | champs hôte |
|  | PRO\*xx | ordre EXEC SQL | variable hôte |
|  | Bloc PL/SQL | appel direct | variable locale |
|  | autre procédure ou fonction stockée ou déclencheur | appel direct | variable globale ou locale |

### A partir de SQL\*PLUS

L’appel à une procédure stockée se fait par la commande :

EXECUTE nom\_procédure [(liste de paramètres effectifs)]

L’appel à une fonction stockée se fait par :

EXECUTE :variable locale := nom\_fonction [(liste de paramètres effectifs)]

## Gestion des erreurs

### Gestion des erreurs par une section EXCEPTION

Il est possible de gérer la totalité des erreurs (SGBD et utilisateur) à l'intérieur d'une procédure ou d'une fonction en utilisant une section EXCEPTION (Voir le chapitre sur le langage PL/SQL).

### Erreur générée par l'utilisateur

La génération d'un diagnostic d'erreur utilisateur s'effectue, au sein d'une procédure ou d'une fonction par utilisation de la procédure standard

RAISE\_APLICATION\_ERROR (numéro\_erreur, 'texte erreur')

avec :

|  |  |  |  |
| --- | --- | --- | --- |
|  | | |  |
| numéro\_erreur | numéro fourni pour l'erreur utilisateur, doit être compris entre -20000 et -20999. | |
| texte erreur | message d'erreur associé au numéro précédent. | |
|  |  | |

# Groupement de procédures - Packages

## Développement d'un package.

Chaque partie du package doit être créée et compilée séparément.

### Création de la partie SPECIFICATION

La création et compilation s'effectue par la commande :

CREATE [OR REPLACE] PACKAGE nom\_package   
[ IS | AS ]  
 {[déclaration de procédure; ]  
 | [déclaration de fonction; ]  
 | [déclaration de variable; ]  
 | [déclaration de curseur; ]  
 | [déclaration d'exception;] ...}  
END nom\_package

### Création de la partie BODY

La création et compilation s'effectue par la commande

CREATE [OR REPLACE] PACKAGE BODY nom\_package  
[ IS | AS ]  
 {[déclaration de variable; ]  
 | [déclaration de fonction; ]  
 | [déclaration de procédure; ]  
 | [déclaration de curseur, ]  
 | [déclaration d'exception ; ] ... }  
END nom\_package

## Modification d'un package existant

Pour modifier la partie spécification ou la partie corps d'un package il suffit de modifier le texte source correspondant et d'exécuter l'un des ordres

REPLACE PACKAGE nom\_package  
  
REPLACE PACKAGE BODY nom\_package

## Suppression d'un package

### Suppression d'un package entier

Elle se fait par :

DROP PACKAGE nom\_package

### Suppression du corps d'un package

Elle se fait par

DROP PACKAGE BODY nom\_package