POLYTECH NANTES

École d'ingénieurs de l'université de Nantes

Spécialité Informatique

INFRASTRUCTURE DES BASES DE DONNEES

Une mise en oeuvre avec ORACLE

MP Nachouki

Table des matières

Chapitre I

Le langage SQL

Sommaire

1 Introduction :

a Structured Query Language ou Langage d'interrogation structuré

Conçu par IBM dans les années 70, il est devenu le langage standard des systèmes de gestion de bases de données (SGBD) relationnelles (SGBDR).
C'est un L4G (langage de quatrième génération inspiré du modèle d'algèbre relationnel de Codd Il est à la fois :

1. un langage d'interrogation de la base (ordre SELECT)
2. un langage de manipulation des données : LMD (ordres UPDATE, INSERT, DELETE)
3. un langage de définition des données : LDD (ordres CREATE, ALTER, DROP, RENAME),
4. un langage de contrôle de l'accès aux données : LCD (ordres GRANT, REVOKE).

Le langage SQL est utilisé par les principaux SGBDRs : Oracle, POSTGRESS, DB2, SQLSERVER, ... Chacun de ces SGBDR possède cependant sa propre variante du langage. Ce support de cours présente un noyau de commandes disponibles sur l'ensemble de ces SGBDR, et leur implantation dans Oracle Version 10.

b Langage normalisé

SQL devient un standard ANSI en 1986, un standard ISO en 1987, mais les implémentations concrètes sont loin d'être standardisée.

Versions normalisées de SQL :

— SQL1 (1989) :
— SQL2 (1992) : version étendue à trois niveaux de conformité (entry, intermediate, full), version de référence bien que quelques SGBD supportent une partie de SQL3
— SQL3 (1999), norme étendue à la gestion d'objets complexes
— SQL 2003 ajoute des capacités XML au langage

c syntaxe SQL

— Séparateur d'instruction : point virgule
— Séparateur de mots-clés : espace, tab, fin de ligne
— Mots-clés du SQL non sensibles à la casse
— Noms d'objets (table, champ. . .) sensibles à la casse
— Entourer de '. . .' ou ". . ." les noms d'objets comportant un espace
— Valeurs de chaînes et date entre ' ' ou " " dans les expressions
— Commentaires : commentaire sur plusieurs lignes /* commentaires */, commentaire sur une ligne :–

2 Exemple

```
Fournisseur
    (f   : Chaîne,
    nom    : Chaîne,
    remise   : Réel,
    ville    : Chaîne)
Mafourniture
    (four    : Fournisseur,
    prod   : Produit,
    qte    : Entier)
Produit
    (P   :chaîne,
    libellé  :Chaîne
    couleur  :Chaîne)
```

3 Le langage de définition

a la commande CREATE

Création simple

Définition. La commande CREATE

```
CREATE TABLE nom_table
        [(nom_col1,
          nom_col2,
    ...)]
  ;
```

Définition des clés

- PRIMARY KEY (colonne1, colonne2, ...)
- FOREIGN KEY (colonne1, colonne2, ...)
- REFERENCES Nom-de-la-table-etrangere(colonne1,colonne2,...)

Contraintes

- CONSTRAINT
- DEFAULT
- NOT NULL
- UNIQUE
- CHECK

Les types de données

1. Numériques : NUMBER, DECIMAL, FLOAT, NUMBER[(longueur,[précision])]
2. Caractères et chaines : CHAR(longueur) , VARCHAR(longueur), LONG
3. date : DATE
4. Grands objets binaires : RAW, BLOB

Trigger gachette

— ON DELETE est suivi d'arguments entre accolades permettant de spécifier l'action à réaliser en cas d'effacement d'une ligne de la table faisant partie de la clé étrangère :
  — CASCADE indique la suppression en cascade des lignes de la table étrangère dont les clés étrangères correspondent aux clés primaires des lignes effacées
  — RESTRICT indique une erreur en cas d'effacement d'une valeur correspondant à la clé
  — SET NULL place la valeur NULL dans la ligne de la table étrangère en cas d'effacement d'une valeur correspondant à la clé
  — SET DEFAULT place la valeur par défaut (qui suit ce paramètre) dans la ligne de la table étrangère en cas d'effacement d'une valeur correspondant à la clé

Exemple. *Création des tables du jeu d'essai*

```
1  /* création de la table Fournisseur */
2  CREATE TABLE fournisseur
3      (f NUMBER(5) PRIMARY KEY,
4       nom VARCHAR(15),
5       remise NUMBER(5,2),
6       ville VARCHAR(15)CONSTRAINT typeville CHECK (ville IN ('Nantes', '
            Paris', 'Brest'))
7       );
8
9  /* création de la table Produit */
10 CREATE TABLE produits
11     (p NUMBER(5) PRIMARY KEY,
12      libelle VARCHAR(15),
13      couleur VARCHAR(15)CONSTRAINT typecouleur CHECK (couleur IN ('rouge'
            , 'vert', 'orange'))
14      );
15
16 /* création de la table Mafourniture */
17 CREATE TABLE mafourniture
18     (four NUMBER(5) REFERENCES fournisseur(f)ON DELETE CASCADE,
19    /*La suppression d'un fournisseur entraine automatiquement la
          suppression de toutes les
20            lignes de la table Mafourniture qui référencent ce fournisseur
                  *\
21      prod NUMBER(5),
22      qte NUMBER(7) NOT NULL CONSTRAINT qtepositive CHECK (qte>0),
```

```
23      PRIMARY KEY (four, prod),
24      FOREIGN KEY (prod) REFERENCES produits(p)
25      );
```

INITIALLY DEFERRED DEFERRABLE

Soient les tables suivantes :

```
1  CREATE TABLE poule (cID number(3) PRIMARY KEY,
2              eID number(3)   );
3
4
5  CREATE TABLE oeuf(eID number(3)PRIMARY KEY,
6          cID number(3)   );
```

En supposant que : "un oeuf est pondu par une poule et une poule nait d'un oeuf", on obtient :

```
1  CREATE TABLE poule (cID number(3) PRIMARY KEY,
2              eID number(3) REFERENCES oeuf(eID)   );
3
4
5  CREATE TABLE oeuf(eID number(3) PRIMARY KEY,
6          cID number(3) REFERENCES poule(cID)   );
```

Impossible d'insérer des tuples dans les deux tables, d'où, utilisation de la clauseINITIALLY DEFERRED DEFERRABLE qui permettra d'insérer le premier tuple dans chaque table.

```
1  CREATE TABLE poule (cID number(3) PRIMARY KEY,
2              eID number(3) REFERENCES oeuf(eID)
3    INITIALLY DEFERRED DEFERRABLE );
4
5
6  CREATE TABLE oeuf(eID number(3) PRIMARY KEY,
7          cID number(3) REFERENCES poule(cID)
8          INITIALLY DEFERRED DEFERRABLE);
```

Création avec insertion de données

```
1  CREATE TABLE nom_table
2    [
3        (nom_col1,nom_col2, ...)
4    ]
5  AS SELECT ... ;
```

Exemple. *Création de la table Mafourniture1 à partir de la table Mafourniture*

```
1  CREATE TABLE mafourniture1 as select * from mafourniture;
```

b la commande ALTER

— ADD : Ajouter une colonne ou une contrainte
— MODIFY : Modifier une colonne
— DROP : Supprimer une contrainte

```
ALTER TABLE table_name ADD column_name datatype;

The basic syntax of ALTER TABLE to DROP COLUMN in an existing table is
    as follows:

ALTER TABLE table_name DROP COLUMN column_name;

The basic syntax of ALTER TABLE to change the DATA TYPE of a column in a
    table is as follows:

ALTER TABLE table_name MODIFY COLUMN column_name datatype;

ALTER TABLE table_name MODIFY column_name datatype NOT NULL;

ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);

ALTER TABLE table_name
ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);


ALTER TABLE table_name
ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);

ALTER TABLE table_name
DROP CONSTRAINT MyUniqueConstraint;
```

Exemple. *Ajout d'une contrainte à Mafourniture*

```
ALTER TABLE mafourniture1 ADD CONSTRAINT qtepositive CHECK (qte>0);
```

c la commande RENAME

RENAME ancien nouveau;

d la commande DROP

DROP TABLE nom_table;

Exemple. *Suppression de la table Mafourniture*

```
DROP TABLE mafourniture1;
```

e création de domaines

création

définition d'un type associé à un certain nombre de règles de validité

```
CREATE DOMAIN nom_domaine
    [AS] type_donnée
  [DEFAULT valeur_défaut]
    [contrainte_de_domaine1
    [, contrainte_de_domaine2 ... ] ]
  [COLLATE nom_collation];
contrainte_de_domaine :: CONSTRAINT nom_contrainte
CHECK (regle_validation)
```

Exemple. *Création du domaine typecouleur*

```
CREATE DOMAIN typecouleur AS VARCHAR(15)
  CHECK (VALUE IN ('rouge', 'vert', 'orange'));
```

Modification

```
ALTER DOMAIN nom_domaine
  {SET DEFAULT valeur_défaut |
    DROP DEFAULT |
    ADD contrainte_de_domaine |
    DROP CONSTRAINT nom_contrainte }
```

Suppression

```
DROP DOMAIN nom_domaine
  [ CASCADE | RESTRICT ];
```

4 Le langage de manipulation

a La commande INSERT

Insertion de n-uplets dans une table

Définition. INSERT

```
INSERT INTO nom_table [(nom_col1,
      nom_col2, ...)]
  VALUES (val1, val2...);
```

Exemple. *Insertion d'un produit*

```
INSERT INTO produits VALUES(15, 'Cornichon','vert');
```

b la commande UPDATE

Définition. UPDATE

```
UPDATE nom_table
  SET nom_col1 = {expression1 |
      ( SELECT ...) }
WHERE predicat;
```

Exemple. *Modification d'un produit*

```
UPDATE produits SET couleur = 'rouge' WHERE libelle = 'Cornichon';
```

c la commande DELETE

Définition. DELETE

```
DELETE FROM nom_table
  [WHERE predicat];
```

Exemple. *Suppression d'un produit*

```
DELETE FROM produits WHERE libelle = 'Cornichon';
```

5 Consultation de la base

a Définition

Format général :

```
SELECT [DISTINCT| ALL]
    {*| <value exp.> [, <value exp.>]...}
    FROM relation [variable],
         relation [ variable]...
      [WHERE <search condition>]
     [GROUP BY <attribute>
                  [,<attribute >]...]
     [HAVING <search condition>]
      [ORDER BY <attribute.>[.{ASC | DESC}]
          [,<attribute.>[.{ASC | DESC}] ]...]
```

EXPRESSION DE VALEURS

— Calculs arithmétiques
— Fonctions agrégats

CONDITION DE RECHERCHE

— Sélection, projection, jointure
— Recherche textuelle
— Recherche par intervalle
— Recherche sur valeur nulle

Format général de la condition de recherche :

```
[<search condition>] ::=[NOT]
<nom-colonne>] \theta constante
[<nom-colonne>] \theta <nom-colonne>
[<nom-colonne>] LIKE <modèle-de-chaîne>
[<nom-colonne> IN <liste-de-valeurs>
[<nom-colonne> \theta (ALL , ANY , SOME) <liste-de-valeurs>]
[EXISTS <liste-de-valeurs> ]
[UNIQUE <liste-de-valeurs> ]
[<nom-colonne> BETWEEN constante AND constante ]
[<search condition> AND , OR <search condition> avec ][\theta ::= <, =,
    >, = , = , <> ]
```

Remarque : <liste-de-valeurs> peut être dynamiquement déterminée par une requête

b les opérateurs ensemblistes

INTERSECTION

SELECT ... **INTERSECT** **SELECT** ...

Exemple. *Réunion des tables Mafourniture et Mafourniture1*

```
SELECT * FROM mafourniture
  INTERSECT
SELECT * FROM mafourniture1;
```

SOUSTRACTION

SELECT ... MINUS **SELECT** ...

UNION

SELECT ... **UNION** **SELECT** ...

PRODUIT CARTESIEN

SELECT ... **FROM** R1,R2

c Projection

```
SELECT [DISTINCT]nom_col1, nom_col2, ...
FROM nom_table ;
```

Exemple. *Liste des fournisseurs qui fournissent quelque chose*

```
SELECT f FROM mafourniture;
```

d Sélection

```
SELECT *|liste-attributs
FROM nom_table
WHERE predicat ;
```

Les prédicats de sélection simples

Un prédicat simple est le résultat de la comparaison de deux expressions au moyen d'un opérateur de comparaison qui peut être :

— $=, <, >, <=, >=, NOT =$
— BETWEEN : expr1 BETWEEN expr2 AND expr3
— IN : expr1 IN (expr2, expr3, ...)
— LIKE : expr LIKE chaine (_ remplace exactement 1 caractère, % remplace un nombre quelconque de caractères)

Les prédicats de sélection composés

— AND
— OR
— NOT

Exemple. *liste des produits verts*

```
SELECT * FROM produits WHERE couleur = 'vert';
```

Exemple. *Liste des tables que j'ai créées*

```
SELECT TABLE_NAME FROM USER_TABLES;
```

e Le regroupement

Les principales fonctions d'agrégat

Les principales fonctions de calcul sur l'ensemble des valeurs d'une colonne sont les suivantes :

```
AVG([DISTINCT | ALL] expression)
COUNT(* | [DISTINCT | ALL] expression)
MAX([DISTINCT | ALL] expression)
MIN([DISTINCT | ALL] expression)
STDDEV([DISTINCT | ALL] expression)
SUM([DISTINCT | ALL] expression)
VARIANCE([DISTINCT | ALL] expression)
DISTINCT
```

Exemple. *Nombre de fournisseurs*

```
SELECT COUNT(*) FROM fournisseur
```

Exemple. *Remise maximum*

```
SELECT MAX(remise) FROM fournisseur
```

la commande GROUP BY

```
SELECT ...FROM...
WHERE ...
GROUP BY expr_1, expr_2, ...;
```

Dans la liste des colonnes résultat d'un SELECT comportant une fonction de groupe, ne peuvent figurer que des caractéristiques de groupe, c'est-à-dire : soit des fonctions de groupe ; soit des attributs figurant dans le GROUP BY.

Exemple. *Déterminer le nombre de fournisseurs de chaque ville*

```
SELECT VILLE,COUNT(f) FROM fournisseur
    GROUP BY VILLE;
```

Sélection des groupes

```

SELECT ...FROM...
WHERE ...
GROUP BY expr_1, expr_2, ...
HAVING ...
```

Exemple. *Déterminer le nombre de fournisseurs de chaque ville en éliminant les villes dont le nombre de fournisseurs est inférieur à 100*

```
SELECT VILLE,COUNT(f) FROM fournisseur
    GROUP BY VILLE
        HAVING COUNT(f) > 100 ;
```

f Calcul de sous totaux simples

calcul de l'*UNION de GROUP BY* de chaque sous-ensemble des attributs en paramètre, selon l'ordre des attributs en paramètre

```
SELECT att1, att2, ... fonct(atti)
FROM table
GROUP BY    ROLLUP att1, att2,...;
```

avec foct= SUM,COUNT,AVG,MIN ou MAX

Exemple :

f	nom	remise	ville	type
f1	Bidochon	5	Paris	MP
f2	Lagaffe	7	Paris	MP
f3	Deschiens	3	Reims	PF
f4	Mortimer	6	Dijon	PF
f5	Black	10	Riec	MP
f6	Achille Talon	1	Paris	PF

```
SELECT type, ville, sum(remise)
FROM table
GROUP BY  ROLLUP (type, ville);
```

g Généralisation CUBE

L'opération CUBE groupe les lignes sélectionnées sur la base des valeurs de toutes les combinaisons possibles des expressions dans la spécification, et renvoie une seule ligne d'informations succinctes sur chaque groupe. calcul de l'*UNION de GROUP BY* de chaque sous-ensemble des attributs en paramètre

```
SELECT att1, att2, ... fonct(atti)
FROM table
GROUP BY CUBE (att1, att2,..) ;
```

Exemple :

f	nom	remise	ville	type
f1	Bidochon	5	Paris	MP
f2	Lagaffe	7	Paris	MP
f3	Deschiens	3	Reims	PF
f4	Mortimer	6	Dijon	PF
f5	Black	10	Riec	MP
f6	Achille Talon	1	Paris	PF

```
SELECT type, ville, sum(remise)
FROM table
GROUP BY  cube (type, ville);
```

h Généralisation GROUPING SETS

```
SELECT att1, att2, ... fonct(atti)
FROM table
GROUP BY GROUPING SETS ( (att1), (tta2,att3), ....)
```

L'opération GROUPING SET groupe les lignes sélectionnées sur la base des valeurs de toutes les combinaisons possibles des expressions dans la spécification, et renvoie une seule ligne d'informations succinctes sur chaque groupe.

Exemple :

f	nom	remise	ville	type
f1	Bidochon	5	Paris	MP
f2	Lagaffe	7	Paris	MP
f3	Deschiens	3	Reims	PF
f4	Mortimer	6	Dijon	PF
f5	Black	10	Riec	MP
f6	Achille Talon	1	Paris	PF

```
SELECT type, ville, sum(remise)
FROM table
GROUP BY  grouping sets (type, ville);
```

```
SELECT type, ville, sum(remise)
FROM mafourniture
GROUP BY  grouping sets (type, ville , (type, ville ));
```

i Tri des résultats

```
ORDER BY {nom_col1 | num_col1 [DESC] [, nom_col2 |
          num_col2 [DESC] ,...]}
```

Exemple. *Tri des fournisseurs par no*

```
SELECT * FROM fournisseur
    ORDER BY f;
```

j la jointure

```
SELECT ...
FROM nom_table1, nom_table2...
WHERE predicat;
```

- Equi-jointure
- Auto-jointure
- θ jointures
- Jointure externe

Exemple. *Equi-jointure : Liste des fournisseurs et de leurs fournitures*

```
SELECT * FROM fournisseur, mafourniture
    WHERE fournisseur.f= mafourniture.four;
```

Exemple. *Auto-jointure : Liste des fournisseurs qui ont une remise supérieure à celle du fournisseur no 100*

```
SELECT f1.* FROM fournisseur f1, fournisseur f2
WHERE f2.f= 100 AND f1.remise > f2.remise;
```

Exemple. *Jointure externe : Liste des fournisseurs et, s'ils fournissent quelque chose, de leurs fournitures*

```
SELECT * FROM fournisseur, mafourniture
    WHERE fournisseur.f= mafourniture.four(+);
```

k Les jointures normalisées

Pourquoi ?

- Lecture plus facile :
  - Le type de jointure est spécifié directement dans la clause FROM
  - La condition de jointure est dissociée des conditions de recherches : se situe maintenant dans la clause ON et non plus dans la clause WHERE
- L'ancienne syntaxe est toujours valide

Jointure interne :

```
SELECT ... FROM <table gauche>
  [INNER]JOIN <table droite>
  ON <condition de jointure>
```

Exemple. *Liste des fournisseurs et de leurs fournitures*

```
SELECT * FROM fournisseur [INNER]JOIN mafourniture
  ON fournisseur.f= mafourniture.four;
```

Jointure externe :

```
SELECT ... FROM <table gauche> LEFT |
               RIGHT |
               FULL OUTER JOIN
  <table droite> ON condition de jointure
```

Exemple. *Liste des fournisseurs et, s'ils fournissent quelque chose, de leurs fournitures*

```
SELECT * FROM fournisseur LEFT OUTER JOIN mafourniture
  ON fournisseur.f= mafourniture.four;
```

Jointure naturelle :

```
SELECT ... FROM <table gauche>
       NATURAL JOIN <table droite>
       [USING <noms de colonnes>]
```

Exemple. *Liste des fournisseurs et de leurs fournitures*

```
SELECT * FROM fournisseur NATURAL JOIN mafourniture
  USING f;
```

Jointure croisée

```
SELECT ... FROM <table gauche> CROSS JOIN
               <table droite>;
```

Jointure d'union

```
SELECT ... FROM <table gauche> UNION JOIN
               <table droite>
```

1 Les requêtes imbriquées et corrélées

Pourquoi ?

— Sous-requête ramenant une seule ligne

Exemple. *Liste des fournisseurs qui ont une remise égale à celle du fournisseur no 100*

```
1 SELECT * FROM fournisseur
2 WHERE remise = (SELECT remise FROM fournisseur WHERE f=100);
```

— Sous-requête ramenant plusieurs lignes : Les opérateurs permettant de comparer une valeur à un ensemble de valeurs sont :

— l'opérateur IN : les opérateurs obtenus en ajoutant ANY ou ALL à la suite d'un opérateur de comparaison classique (=, !=, >, >=, <, <=)

— ANY : la comparaison est vraie si elle est vraie pour au moins un des éléments de l'ensemble.

— ALL : la comparaison sera vraie si elle est vraie pour tous les éléments de l'ensemble.

Exemple. *fournisseurs qui fournissent quelque chose*

```
1 SELECT * FROM fournisseur
2    WHERE f = (SELECT four FROM mafourniture);
```

Exemple. *Liste des fournisseurs qui ont une remise > à celle du fournisseur no 100*

```
1 SELECT * FROM fournisseur
2 WHERE remise > (SELECT remise FROM fournisseur WHERE f=100);
```

Sous-requêtes ramenant plusieurs colonnes

Il est possible de comparer le résultat d'une requête ramenant plusieurs colonnes à une liste de colonnes. La liste de colonnes figurera entre parenthèses à gauche de l'opérateur de comparaison.

Exemple. *Liste des fournisseurs qui ont une remise égale à celle du fournisseur no 100 et dont la ville est identique à celle du fournisseur no 100*

```
1 SELECT * FROM fournisseur
2 WHERE (remise,ville) = (SELECT remise, ville FROM fournisseur WHERE f
      =100);
```

Sous-requêtes corrélées

SQL sait également traiter une sous-interrogation faisant référence à une colonne de la table de l'interrogation principale. Le traitement dans ce cas est plus complexe car il faut évaluer la sous-interrogation pour chaque ligne de l'interrogation principale.

Exemple. *Liste des fournisseurs qui ne fournissent rien*

```
1 SELECT * FROM fournisseur
2 WHERE NOT EXISTS (SELECT * FROM mafourniture WHERE  mafourniture.four=
      fournisseur.f;
```

Remarque : Le prédicat EXISTS permet de tester l'existence ou l'absence de données dans la sous requête. Si la sous requête renvoie au moins une ligne, le prédicat est vrai. Dans le cas contraire, le prédicat est faux.

Division

Deux solutions

1. NOT EXISTS

   Exemple. *Liste des fournisseurs qui fournissent tous les produits verts*

```
SELECT * FROM fournisseur
WHERE NOT EXISTS (SELECT * FROM produits WHERE couleur = 'vert' AND
      NOT EXISTS (SELECT * FROM mafourniture WHERE mafourniture.
          four= fournisseur.f and mafourniture.prod=produits.p));
```

2. GROUP BY

   Exemple. *Liste des fournisseurs qui fournissent tous les produits verts*

```
SELECT four FROM mafourniture
  WHERE prod IN (SELECT p FROM produits WHERE couleur = 'vert')
  GROUP BY four
        HAVING COUNT(*)=
    (SELECT COUNT(*) FROM produits
      WHERE couleur = 'vert');
```

6 Le traitement des structures d'arbre

Il est possible de représenter selon le modèle relationnel des listes ou des structures d'arbre. Pour cela, il suffit d'introduire dans la relation un attribut représentant un lien vers l'élément suivant ou le prédécesseur dans la liste ou l'arbre :
Relation(clé, ... , *clé du prédécesseur*)
Parcours d'un arbre : il faut

— définir comment se fait le lien entre une ligne et la précédente, en indiquant la colonne clé et la colonne lien dans une clause *CONNECT BY* : **CONNECT BY** nom_col_1 = **PRIOR** nom_col_2

— Le mot clé *PRIOR* , accolé à l'une ou l'autre des colonnes, définit le sens du parcours. définir un (ou plusieurs) point de départ, par une clause *START WITH* placée après la clause CONNECT BY.

Chapitre II

Le langage PL-SQL

Sommaire

a PL/SQL : Langage de programmation

— utilisant les commandes SQL
— disposant d'un ensemble de fonctionnalités supplémentaires (programmation structurée)

1. Déclarations de variables, variables initialisées, constantes
2. Structures algorithmiques (itérations, alternatives)

b Apport de PL-SQL

PL/SQL regroupe les requêtes SQL en un seul bloc qui est envoyé au serveur en un seul appel

1. PL/SQL améliore les performances (moins de communications à travers le réseau)
2. C'est un langage portable : il peut fonctionner sur toute plateforme supportant Oracle Server
3. PL/SQL peut aussi coopérer avec les différents outils de développement d'application de Oracle Server (p.ex. Developer 2000)
4. Permet de créer des bibliothèques de code réutilisable

c Environnement de PL-SQL

1. *Block anonyme* bloc PL/SQL imbriqué dans une application ou créé interactivement
2. *Procédure enregistrée :* bloc nommé enregistré dans le serveur Oracle et qui peut être invoqué par son nom
3. *Procédure d'application* bloc nommé enregistré dans une application Developper 2000 ou dans une librairie partagée
4. *Package :* module PL/SQL qui regroupe un ensemble de procédures
5. *Trigger base de donnée* block associé à une table et déclenché automatiquement lors d'une requête
6. *Trigger d'application* block PL/SQL associé à un événement d'application

d Caractéristiques de PL-SQL

1. Structures de contrôle
   — traitements conditionnels
   — traitements répétitifs
2. Utilisation des curseurs
3. Gestion des erreurs
4. Gestion des variables

1 Structure d'un bloc PL-SQL

a bloc PL-SQL

```
1  DECLARE
2    /* section de declaration variables exception curseurs */
3  BEGIN
4    /* corps du bloc de programme: il s'agit de la seule zone
5  dont la presence est obligatoire */
6  EXCEPTION
7    /* gestion des exceptions */
8  END;
9  /
10   /* fin du bloc PL/SQL
```

Le corps du programme (entre le BEGIN et le END) contient des instructions PL/SQL (affectation, boucles, appels de procédure) ainsi que des instructions SQL. Il s'agit de la seule partie qui soit obligatoire. Les deux autres zones (déclaration et gestion des exceptions) sont facultatives. Les commandes du langage de définition (CREATE,DROP, RENAME, ALTER) ne sont pas autorisées dans un bloc PL/SQL. Chaque instruction se termine par un " ;".

b Corps d'un bloc PL-SQL

Un bloc Pl-SQL est tout entier enfermé entre BEGIN et END ; S'il est vide, le mot BEGIN est omis mais END ; demeure.

```
1  BEGIN ...
2  END;
```

— Saisie de données au clavier avec &nom
— Affectation
  1. variable := expression ou
  2. SELECT attrib INTO variable FROM table WHERE condition

2 Les variables

a Déclaration des variables

— Variables : DECLARE nom-variable TYPE ;
— Cas particuliers de types :
  — nom-variable *nom* – *table.nom* – *colonne%TYPE* ; (variable de même type que la colonne)
  — nom-variable *nom* – *table%ROWTYPE* ; (record avec les mêmes champs que la table)
— variable initialisée

— Constantes
— Curseur
— RECORD : Equivalent à struct du langage C

TYPE nomRecord IS RECORD (

champ1 type1,

champ2 type2,

...) ;

b Affectation des variables

— Affectation :=

```
1 DECLARE
2    a NUMBER;
3    b NUMBER;
4 BEGIN
5    a:=a+b;
6 END;
```

— Lecture de la base : SELECT INTO

```
1 DECLARE
2    a fournisseur.f\%TYPE;
3 BEGIN
4   SELECT f INTO a FROM fournisseur
5 WHERE f= 12;
6 END;
7 /
8 Attention: La requête SQL ne doit renvoyer qu'une et une seule
     valeur
```

3 Structures algorithmiques

a Structures algorithmiques : 3 types de boucles

Boucle LOOP - END LOOP "empaquetage des instructions"

```
1 LOOP
2   Bloc d'instructions avec
3   EXIT WHEN condition;
4 END LOOP;
```

Boucle FOR

```
FOR condition  -- l'indice de boucle n'est pas déclaré
  LOOP
   Bloc d'instructions;
END LOOP;
```

Boucle WHILE

```
WHILE condition
LOOP
   Bloc d'instructions;
END LOOP;
```

b Structures algorithmiques : Alternatives/conditionnelles

IF THEN ELSE

```
IF <condition> THEN
instructions;
[ ELSEIF <condition> THEN
instructions; ]
[ ELSE
instructions; ]
END IF;
```

CASE expression

```
CASE expression
  WHEN expr2 THEN instructions 2;
   ...
  ELSE instructions N;
END CASE;
```

c Structures algorithmiques : Branchement

```
GOTO nom ;
...

<<nom>>
   Bloc-Instructions ;
```

4 Les entrées-sorties

a saisie d'un mot au clavier

1. &nom : saisie d'un mot au clavier
2. un mot est un nombre ou une chaîne de caractères alphanumériques (éventuellement entre apostrophes)
3. Oracle substituera la suite des caractères saisis à &nom.

b Affichage à l'écran

1. On utilise le package DBMS-OUTPUT
2. Tout d'abord, avant le bloc PL/SQL, utiliser l'instruction : set server output on
3. Puis pour chaque affichage : dbms-output.put-line('texte'||X) ;
4. La concaténation de chaînes de caractères est réalisée avec : ||

5 Les curseurs

a Fonctionnalités

1. Toutes les requêtes SQL sont associées à un curseur
2. Ce curseur représente la zone mémoire utilisée pour parser et exécuter la requête
3. Le curseur peut être implicite (non déclaré par l'utilisateur) ou explicite
4. Les curseurs explicites servent à retourner plusieurs lignes avec un select

b Attributs des curseurs

Tous les curseurs ont des attributs que l'utilisateur peut utiliser

1. %ROWCOUNT : nombre de lignes traitées par le curseur
2. %FOUND : vrai si au moins une ligne a été traitée par la requête (ou le dernier fetch)
3. %NOTFOUND : vrai si aucune ligne n'a été traitée par la requête ou le dernier fetch
4. %ISOPEN : vrai si le curseur est ouvert(utile uniquement pour les curseurs explicites)

c Les curseurs : mise en oeuvre

déclaration

```
1 CURSOR nom-curs  IS SELECT ... ; -- déclaration du curseur lui-même
2 -- déclarations des variables utiles pour l'utilisation du curseur
```

utilisation

```
1 OPEN nom-curs; -- ouverture du curseur C'est à l'ouverture que la
      sélection est exécutée
2   FETCH nom-curs INTO ligne; -- Lecture de la ligne courante du
      curseur et stockage dans la variable ligne
3 CLOSE nom-curs; -- fermeture du curseur
```

exemple

```
1  DECLARE
2    CURSOR lesfournisseurs IS SELECT * FROM fournisseur WHERE ville = '
       Nantes';
3    unfournisseur fournisseur\%ROWTYPE;
4  BEGIN
5    OPEN lesfournisseurs;
6  LOOP
7    FETCH lesfournisseurs INTO unfournisseur;
8  ...
9  EXIT WHEN lesfournisseurs\%NOTFOUND;
10 END LOOP;
11 CLOSE lesfournisseurs;
12 END;
```

d Les curseurs paramétrés

définition

1. Un curseur paramétré peut servir plusieurs fois avec des valeurs des paramètres différentes
2. On doit fermer le curseur entre chaque utilisation de paramètres différents (sauf si on utilise «for» qui ferme automatiquement le curseur)

```
1 DECLARE
2   CURSOR nom_curseur ( param_1 type_1, param_2 type_2, ... ) IS SELECT
      ...
3 ...
4 BEGIN
5   OPEN nom_curseur ( valeur_1, valeur_2, ... );
6 ...
```

exemple

```
1  DECLARE
2    CURSOR lesfournisseurs (localite IN VARCHAR(30)) IS SELECT * FROM
       fournisseur WHERE ville = 'Nantes';
3    unfournisseur fournisseur\%ROWTYPE;
4  BEGIN
5  OPEN lesfournisseurs('Nantes');
6  LOOP
7    FETCH lesfournisseurs INTO unfournisseur;
8    ...
9    EXIT WHEN lesfournisseurs\%NOTFOUND;
10 END LOOP;
11 CLOSE lesfournisseurs;
12 OPEN lesfournisseurs('Paris');
13 LOOP
14   FETCH lesfournisseurs INTO unfournisseur;
15   ...
16   EXIT WHEN lesfournisseurs\%NOTFOUND;
17 END LOOP;
18 CLOSE lesfournisseurs;
19 END;
```

e Boucle FOR pour un curseur

1. Elle simplifie la programmation car elle évite d'utiliser explicitement les instruction open,fetch, close
2. Elle déclare implicitement une variable de type « row » associée au curseur

6 Le traitement des erreurs

a Présentation

Une exception est une erreur qui survient durant une exécution

1. exception prédéfinie par Oracle
2. exception définie par le programmeur

b les exceptions prédéfinies

1. NO-DATA-FOUND
2. TOO-MANY-ROWS
3. VALUE-ERROR (erreur arithmétique)
4. ZERO-DIVIDE
5. ...

c Le traitement des erreurs

```
1
2 BEGIN
3     ...
4   EXCEPTION
5     WHEN NO_DATA_FOUND THEN      ...
6     WHEN TOO_MANY_ROWS THEN
7 END;
```

d Le traitement des erreurs utilisateur

Exceptions utilisateur

```
1 /* Déclaration dans la section declare */
2   excpt1 EXCEPTION
3 /* Détection dans le corps du prgm */
4   IF ... THEN RAISE excpt1
5 /*Traitement dans la section Exception} */
6   WHEN <exception1> [OR <exception2> OR ...] THEN <instructions>
7   WHEN <exception3> [OR <exception2> OR ...] THEN <instructions>
8   WHEN OTHERS THEN  <instructions>
```

7 Procédures, Fonctions et paquetages

a Procédures, Fonctions et paquetages

Une procédure est un ensemble de code PL/SQL nommé, défini par l' utilisateur et généralement stocké dans la BDD

Une fonction est identique à une procédure à la différence qu' elle retourne une valeur

Un paquetage est le regroupement de plusieurs procédures et fonctions dans un objet distinct

b Principaux avantages

— Le code relatif aux règles de gestion est centralisé.
— Partage du code entre plusieurs applications
— Amélioration des performances, car le code stocké est pré-compilé

c Procédures

```
1 CREATE [OR REPLACE] PROCEDURE [schema.]procedure
2     [ (argument [IN | OUT | IN OUT] datatype
```

```
    [, argument [IN | OUT | IN OUT] datatype] ...)]
    {IS | AS}
  variables;
  BEGIN
  ...
  END;
```

Commentaires :

— IN (valeur par défaut) indique que le paramètre transmis par le programme appelant n'est pas modifiable par la procédure
— OUT indique que le paramètre est modifiable par la procédure
— IN OUT indique que le paramètre est transmis par le programme appelant et renseigné par la procédure

Exemple. *Ajouter une nouvelle fourniture*

```
CREATE OR REPLACE PROCEDURE ajoutfourniture(unfour IN NUMBER, unprod
    IN NUMBER) IS
moy mafourniture.qte%type;
BEGIN

/*appel de la fonction Moyenneqte pour calculer la quantité du produit*/
moy:= Moyenneqte (unprod);

INSERT INTO mafourniture VALUES(unfour, unprod, moy);
END;
```

d Les Fonctions

Une fonction est identique à une procédure à la différence qu'elle retourne obligatoirement une valeur d'où le mot clé obligatoire RETURN

```
CREATE [OR REPLACE] FUNCTION [schema.]function
    [ (argument [IN] datatype
    [, argument [IN] datatype] ...)]
    RETURN datatype
    {IS | AS}
  variables;
  BEGIN
  ...
  END;
```

Exemple. *Création de la fonction Moyenne qui calcule la moyenne des qtes d'un produit donné*

```
CREATE OR REPLACE FUNCTION Moyenneqte (leprod number)RETURN NUMBER IS
moy number;
BEGIN
```

```
4  SELECT AVG(qte) INTO moy FROM mafourniture
5          WHERE prod = leprod;
6
7  RETURN moy;
8
9  END;
```

Une instruction RETURN <expression> devra se trouver dans le corps pour spécifier quel résultat sera renvoyé.

e Les Paquetages

Un paquetage est un ensemble de procédures et fonctions regroupées dans un objet nommé.
Par exemple le paquetage Oracle DBMS-LOB regroupe toutes les fonctions et procédures manipulant les grands objets

Organisation d'un paquetage

Un paquetage est organisé en deux parties distinctes
— Une partie spécification : qui permet de spécifier à la fois les fonctions et procédures publiques ainsi que les déclarations des types, variables, constantes, exceptions et curseurs utilisés dans le paquetage et visibles par le programme appelant
— Une partie corps : qui contient les blocs et les spécifications de tous les objets publics listés dans la partie spécification Cette partie peut inclure des objets qui ne sont pas listés dans la partie spécification, et sont donc privés
— La déclaration de la partie spécification d'un paquetage s'effectue avec l'instruction CREATE [OR REPLACE] PACKAGE
— Celle de la partie corps avec l'instruction CREATE [OR REPLACE] PACKAGE BODY

f Visualisation du code source

Le code source des objets procéduraux est visible par l'intermédiaire des vues du dictionnaire de données
— USER-SOURCE pour les objets appartenant au schéma
— ALL-SOURCE pour les objets appartenant aux schémas accessibles
— DBA-SOURCE pour les objets appartenant à tous les schémas
Mise au point : SHOW-ERRORS

8 Triggers

a Les déclencheurs

Un déclencheur est un bloc PL/SQL associé à une vue ou une table, qui s'exécutera lorsqu'une instruction du langage de manipulation de données *DML* sera exécutée

Un déclencheur s'exécute dans le cadre d'une transaction. Il ne peut donc pas contenir d'instruction COMMIT ou ROLLBACK ou toute instruction générant une fin de transaction implicite

b Définition

Un trigger est un programme qui se déclenche automatiquement à la suite d'un événement. C'est donc un programme qui fait partie du schéma mais qui n'est pas appelé explicitement, à la différence d'une procédure stockée.

c Intérêt

Les triggers peuvent servir à :
— vérifier des contraintes que l'on ne peut pas définir de façon déclarative ;
— gérer la redondance d'information, après une dénormalisation du schéma ;
— collecter des informations sur les mises à jour de la base.

d Structure d'un trigger

```
1
2 CREATE [OR REPLACE] TRIGGER nom-trigger
3   instant listeevts
4   ON nom-table [FOR EACH ROW]
5   [WHEN ( condition ) ]
6     corps;
```

On définit :
— la table à laquelle le trigger est lié,
— les instructions du LMD qui déclenchent le trigger
— le moment où le trigger va se déclencher par rapport à l'instruction LMD (avant ou après)
— si le trigger se déclenche une seule fois pour toute l'instruction (trigger instruction), ou une fois pour chaque ligne modifiée/insérée/supprimée. (trigger ligne, avec l'option FOR EACH ROW)
— et éventuellement une condition supplémentaire de déclenchement (clause WHEN)

After ou Before ?

— Si le trigger doit déterminer si l'instruction LMD est autorisée, BEFORE (on peut annuler la MAJ si erreur avant écriture dans la base)
— le trigger doit "fabriquer" la valeur d'une colonne pour pouvoir ensuite l'écrire dans la table : BEFORE
— Si l'instruction LMD doit être terminée pour exécuter le corps du trigger : AFTER

Trigger ligne ou instruction ?

— Un trigger instruction se déclenche une fois, suite à une instruction LMD.
— Un trigger ligne (FOR EACH ROW) se déclenche pour chaque ligne modifiée par l'instruction LMD.

Trigger ligne ou instruction ?

Dans un trigger ligne, on peut faire référence à la ligne courante : Pour cette ligne, on a accès :
— à la valeur avant l'instruction DML (nommée :OLD)
— à la valeur après l'instruction (nommée :NEW).

Clause WHEN

On peut définir une condition pour un trigger ligne : le trigger se déclenchera pour chaque ligne vérifiant la condition.

Ordre d'exécution des triggers

Pour une instruction du LMD sur une table de la base, il peut y avoir 4 familles de triggers possibles selon l'instant (BEFORE, AFTER) et le type (instruction ou ligne).
Ces triggers se déclenchent dans l'ordre suivant :
— Triggers instruction BEFORE
— Triggers ligne BEFORE (déclenchés n fois)
— Triggers ligne AFTER (déclenchés n fois)
— Triggers instruction AFTER
Dans une même famille, on ne contrôle pas l'ordre d'exécution des triggers.

Corps du trigger

Le corps du trigger est écrit en PL/SQL. Par rapport à une procédure stockée, on peut utiliser des prédicats qui permettent de savoir quel événement a déclenché le trigger : prédicat INSERTING (resp. UPDATING, DELETING) qui vaut vrai ssi le trigger a été déclenché par un INSERT (resp UPDATE, DELETE).
Le corps d'un trigger ligne peut faire référence aux valeurs de la ligne courante (avant et après modif) par :old et :new.

Exemple. *Affecter automatiquement une quantité à une fourniture*

```
1  CREATE OR REPLACE TRIGGER affectationquantité
2          BEFORE INSERT OR UPDATE OF qte ON mafourniture
3          FOR EACH ROW
4  DECLARE
5  moy NUMBER;
6
7  BEGIN
8  SELECT AVG(qte) INTO moy FROM mafourniture
```

```
9          WHERE prod = :new.prod;
10      IF :new.qte <> '' THEN
11              raise_application_error(-20005, 'Qte non nulle');
12      ELSE  :new.qte := moy;
13      END IF;
14  END;
```

e Les déclencheurs sur VUE

— La syntaxe d'un déclencheur sur vue est identique à celle du déclencheur sur table, à la différence que la clause INSTEAD OF est ajoutée
— Ce type de déclencheur est particulier dans la mesure ou son exécution remplace celle de la commande DML à laquelle il est associé
— Ce type de déclencheur n'est définissable que sur les vues et lui seul peut être mis en place sur les vues

f Activation/désactivation d'un déclencheur

— Il est possible de désactiver un déclencheur avec la commande suivante

```
1  ALTER TRIGGER nom-déclencheur DISABLE;
```

— et de l'activer avec la commande suivante

```
1  ALTER TRIGGER nom-déclencheur ENABLE;
```

— De la même façon, on peut désactiver tous les déclencheurs définis sur une table

```
1  ALTER TABLE nom-table DISABLE ALL TRIGGERS;
```

— et de les activer avec la commande suivante

```
1  ALTER TABLE nom-table ENABLE ALL TRIGGERS;
```

g vues du dictionnaire de données

USER-TRIGGERS pour les déclencheurs appartenant au schéma

ALL-TRIGGERS pour les déclencheurs appartenant aux schémas accessibles

DBA-TRIGGERS pour les déclencheurs appartenant à tous les schémas

Mise au point : SHOW-ERRORS

Chapitre III

Administration de la base

Sommaire

La gestion des vues

h Intérêt

Une vue permet principalement de :
— Réduire la complexité d'une requête ;
— Limiter l'accès aux données ;
— Vérifier la cohérence des données

i Syntaxe

```
CREATE VIEW nom-vue [(nom-col1 , nom-col2 ,...)]
  AS SELECT ...;
  [with check option ]
```

La clause facultative WITH CHECK OPTION permet de définir, lors de la mise à jour de données via la vue, que la (les) conditions de la clause WHERE doit être vérifiée. Si elle n'est pas présente, aucune vérification ne sera faite

j La gestion des vues : utilisation

— SELECT : se fait comme sur une table,
— INSERT,UPDATE,DELETE ssi

1. La vue ne résulte pas d'une jointure ;
2. La clause SELECT ne contient pas de clause DISTINCT ;
3. Il n'y a pas de GROUP BY et HAVING dans la clause SELECT

Exemple. *Création de la vue fournisseurs nantais et ses fournitures*

```
CREATE VIEW frnantais AS SELECT * FROM fournisseur , mafourniture
  WHERE fournisseur.f= mafourniture.four
    AND fournisseur.ville = 'Nantes'
                  WITH CHECK OPTION;
```

k La gestion des vues : Trigger INSTEAD OF

```
CREATE TRIGGER nom-trigger
  INSTEAD OF type ordre
  ON nom-table;
  FOR EACH ROW
    actions ;
```

Les variables :OLD et :NEW sont utilisées dans l' action comme si l' événement avait lieu.

Exemple. *Trigger supprimant un fournisseur nantais*

```
CREATE OR REPLACE TRIGGER supfour
        INSTEAD OF DELETE ON frnantais
            FOR EACH ROW
BEGIN
delete from mafourniture  where four=:old.four ;
delete from fournisseur where f = :old.four;
END;
```

l La gestion des vues : les vue matérialisée

```
CREATE MATERIALIZED VIEW nom REFRESH type ON table AS SELECT ...
```

— Normalement, seule la définition de la vue est stockée dans la base. Une vue matérialisée crée une nouvelle table contenant les résultats de la requête utilisée dans la définition de la vue.
— Utilisation : dans un Data warehouse elles sont utilisées pour pré-calculer et stocker les données agrégées comme les sommes et moyennes .

1 La table DUAL

Oracle fournit une table appelée « DUAL » qui se compose d'une ligne et d'une colonne qui est utilisée pour tester des fonctions ou effectuer des calculs rapides.

```
DESC DUAL
```

Table	Column	Type de données
DUAL	DUMMY	VARCHAR2

Table Column Type de données DUAL DUMMY Varchar2

a La table DUAL et les pseudo-colonnes

La table DUAL permet par exemple de tester les pseudo-colonnes fournies par ORACLE :

```
/* date du jour */
SELECT SYSDATE FROM DUAL;
/* utilisateur courant */
SELECT user FROM DUAL;
```

b Les séquences ORACLE

Une séquence est un objet ORACLE qui permet de gérer une suite de nombres entiers. Elle permet en particulier de générer des clés uniques dans des tables de la base.

Exemple. *Affectation automatique d'un numéro lors de l'insertion d'un nouveau fournisseur*

```
/*Création d'une séquence*/
CREATE SEQUENCE nouveaufournisseur;

/* création du trigger permettant d'affecter un numéro au nouveau
    fournisseur créé */
CREATE OR REPLACE TRIGGER incrementfournisseur
BEFORE INSERT ON fournisseur
FOR EACH ROW
BEGIN
   SELECT nouveaufournisseur.NEXTVAL INTO :NEW.f FROM DUAL;
END;
```

2 Le dictionnaire de données

a Contenu du dictionnaire :

tablespaces, fichiers physiques, tables, contraintes, clusters, vues, index, synonymes, procédures, fonctions, packages, triggers, utilisateurs, droits d'accès, rôles, profils, audits, etc...

b 3 catégories de vues :

— 3 catégories de vues :

1. USER-XXX : décrit les objets appartenant à l'utilisateur connecté
2. ALL-XXX : décrit les objets accessibles à l'utilisateur connecté
3. DBA-XXX : décrit tous les objets de la base

— Les vues dont le nom commence par V$ sont des vues dynamiques donnant des informations sur l'état courant de l'instance de la base de données de son démarrage à son arrêt. Elles permettent par exemple de connaître les fichiers physiques actuellement utilisés par la base (logs, rollback segments, ...).

Une (méta) description du dictionnaire est donnée par la vue DICT. La commande :

```
SELECT * FROM DICT WHERE TABLE_NAME LIKE 'DBA%';
```

3 La gestion des droits

a Schéma

Un schéma permet de regrouper un ensemble d 'objets appartenant à un utilisateur particulier (Tables, Triggers, Constraints, Indexes, Views, Sequences, Stored program units, Synonyms, User-defined)

— Lorsqu'un utilisateur est créé, un schéma de même nom est créé et lui est associé.

— Les notions de username et schema sont très souvent interchangeables.
— Un schéma permet de regrouper un ensemble d 'objets appartenant à un utilisateur particulier.

b Création d'un utilisateur

```
CREATE USER nom
                    IDENTIFIED BY motdepasse/EXTERNALLY
                    DEFAULT TABLESPACE tablespace
                    TEMPORARY TABLESPACE tablespace
                    QUOTA [TAILLE M] on tablespace
                    QUOTA [TAILLE M] on tablespace
                    PASSWORD EXPIRE
                    PROFILE nom du profil
                    ACCOUNT LOCK/UNLOCK
```

Exemple. *Création de l'utilisateur Robert*

```
CREATE USER robert
        IDENTIFIED BY lebidochon
        DEFAULT TABLESPACE info4
```

c Modification d'un utilisateur :

```
ALTER USER name
                    IDENTIFIED BY motdepasse/EXTERNALLY
                    DEFAULT TABLESPACE tablespace
                    TEMPORARY TABLESPACE tablespace
                    QUOTA [TAILLE M] on tablespace
                    QUOTA [TAILLE M] on tablespace
                    PASSWORD EXPIRE
                    PROFILE nom du profil
                    ACCOUNT LOCK/UNLOCK
```

Exemple. *Modification du mot de passe de l'utilisateur Robert*

```
ALTER USER robert IDENTIFIED BY raymonde;
```

d Suppression d'un utilisateur :

```
DROP USER name [CASCADE];
```

Exemple. *Suppression de l'utilisateur Robert*

```
DROP USER robert CASCADE;
```

e les rôles

Un rôle est un objet de base de données. Il permet d'attribuer des privilèges à un groupe d'utilisateurs : Oracle utilise des rôles afin de faciliter la gestion de privilèges pour des groupes d'utilisateurs.

— il faut d'abord attribuer des privilèges à un rôle,
— puis assigner ce rôle aux utilisateurs en question.

Création d'un rôle

```
1 CREATE ROLE name IDENTIFIED BY password;
```

Exemple. *Création du role INFO4*

```
1 CREATE ROLE info4;
```

Suppression d'un rôle

```
1 DROP ROLE name [CASCADE];
```

f les privilèges

La gestion des droits : définition

Un privilège est le droit d'exécuter un type d'instruction SQL spécifique comme :

— Le droit de se connecter à une base de données (créer une session).
— Le droit de créer une table.
— Le droit de sélectionner des lignes dans une table.

g Les privilèges système

autorisent les utilisateurs à exécuter une action particulière portant sur tout le système ou sur un objet de schéma particulier (comme une table).

h Les privilèges objet

autorisent les utilisateurs à exécuter une action spécifique sur un objet particulier.

Ces privilèges sont assignés aux utilisateurs finaux pour leur permettre d'utiliser une application de base de données afin d'accomplir certaines tâches.

i les privilèges système exemples

— CREATE ANY INDEX
— ALTER ANY INDEX
— DROP ANY INDEX
— CREATE TABLE
— CREATE ANY TABLE
— ALTER ANY TABLE
— DROP ANY TABLE

Il y a environ 126 privilèges système. La clause ANY dans un privilège signifie que les utilisateurs ont ce privilège dans tous les schémas

j transmission des privilèges

Octroi de privilèges : GRANT

```
1 GRANT (ALL ou liste de opération)
2   [ON nom-de-table . nom-de-vue]
3   TO (PUBLIC / identificateur-d'utilisateur)
4   [WITH GRANT OPTION]
```

— opération := (INSERT ou DELETE ou UPDATE [liste de nom-de-colonne],EXECUTE).
— la clause WITH GRANT OPTION transmet le droit de transmettre le privilège (attention à la transmission en cascade des privilèges système)
— PUBLIC ne peut être utilisée que par l'administrateur.

Retrait de privilèges : REVOKE

```
1 REVOKE ALL  (ou liste d' opération)
2   ON nom-de-table ou nom-de-vue
3   FROM PUBLIC ou utilisateur
```

opération :=(INSERT ou DELETE ou UPDATE [liste de nom-de-colonne],EXECUTE). La clause WITH GRANT OPTION transmet le droit de transmettre le privilège.

Exemples

Octroi de privilèges système

Exemple. *Un privilège à un utilisateur*

```
1 GRANT CREATE TABLE TO robert;
```

Exemple. *Un privilège à un role*

```
1 GRANT CREATE USER TO info4;
```

Exemple. *Un role à un role*

```
1 GRANT CONNECT TO info4;
```

Exemple. *Un role à un utilisateur*

```
1 GRANT info4 TO robert;
```

Octroi de privilèges objets

Exemple. *Octroi du privilège de consultation de la table fournisseur à l'utilisateur robert*

```
1 GRANT SELECT ON fournisseur TO robert;
```

Exemple. *Octroi du privilège de consultation de la table fournisseur à tous les utilisateurs de la base*

```
1 GRANT SELECT ON fournisseur TO PUBLIC;
```

k les synonymes

alias pour n'importe quelle table, vue, cliché, séquence, procédure, fonction ou package.

```
1 CREATE [PUBLIC] SYNONYM objet FOR user.objet
```

Un synonyme PUBLIC peut être utilisé par tous les utilisateurs de la base.
Un synonyme PRIVE est dédié à un seul utilisateur.

Exemple. *Définition d'un PUBLIC synonyme fournisseur pour la table raymonde.fournisseur*

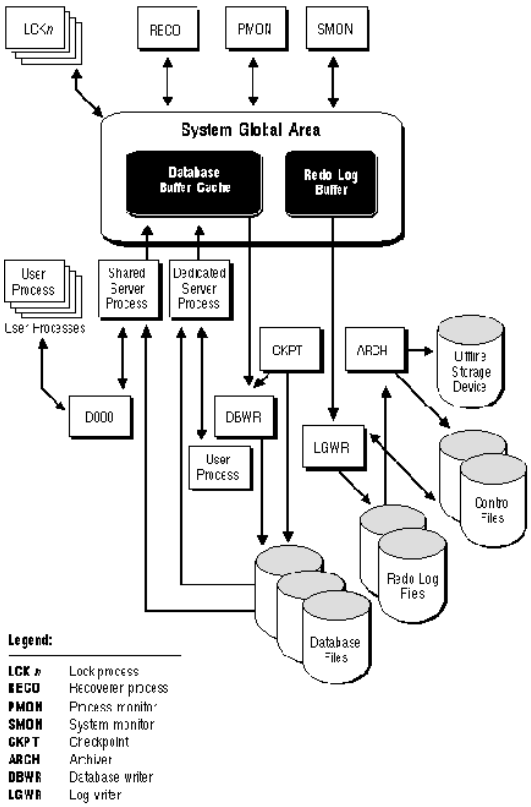
```
1 CREATE PUBLIC SYNONYM fournisseur FOR raymonde.fournisseur;
```

Chapitre IV

Le stockage ORACLE

Sommaire

Introduction



Une base de données Oracle est constituée de plusieurs éléments :

1. D'un espace mémoire(SGA,PGA)
2. De fichiers physiques stockés sur le serveur
3. De processus serveurs et de processus d'arrière plan qui servent d'interface entre la SGA et les fichiers de base de données.

On appelle instance Oracle les processus et la SGA d'une base de données Oracle.

1 l'espace mémoire

Ce niveau correspond à l'organisation des données en mémoire centrale. On distingue deux zones :

1. SGA (system global area) : ensemble de zones tampon allouées pour contenir des données et certaines info de contrôle d'une BD.Elle comprend :
   — Une zone partagée (Shared pool) utilisée principalement pour stocker les commandes SQL et analyser le plan d'exécution de ces commandes.
   — Le cache de données (DATABASE BUFFER CACHE) qui stocke les données demandées par les prgms SQL. Il permet de réduire le nombre de lectures sur le disque et de garder en mémoire les données souvent modifiées.

— Le cache de reprise (REDO LOG BUFFER) zone mémoire qui contient temporairement les MAJ validées avant qu'elles ne soient enregistrées dans les journeaux de reprise

2. PGA (Program global area)qui contient principlement les zones mémoire allouées aux processus

m les processus

— des processus serveur
— des processus d'arrière plan

Les processus d'arrière plan ont chacun une tâche déterminée pour la gestion des données. Ils apparaissent dès le lancement du SGBD Oracle et sont présents durant tout le fonctionnement de l'instance. les principaux sont :

1. *pmon :* utilisé pour nettoyer les ressources, les verrous et les processus utilisateurs.
2. *reco :* utilisé pour résoudre les problèmes de transactions réparties.
3. *dbwr :* utilisé pour écrire le contenu des buffers dans les fichiers de la base.
4. *smon :* utilisé pour la restauration de la BD au démarrage.
5. *lgwr :* utilisé pour l'écriture du contenu des buffers de journalisation dans les fichiers "redo log"
6. *d000 et s000 :* utilisés en fonctionnement multitâches

n les fichiers

Le SGBD Oracle gère trois principaux types de fichiers :

1. *les fichiers "données" :* ils regroupent les données des utilisateurs, les données du Dictionnaire des Données (DD) et les données des segments de travail (les segments temporaires, les "rollback segments" ou les images avant, le segment de démarrage et les segments différés).
2. *les fichiers "redo log" :* ils contiennent la journalisation des modifications successives et les changements d'état de la BD. Ils sont utilisés en cas de restauration de la BD suite à un incident de l'instance ou d'un disque.
3. *les fichiers de contrôle :* ces petits fichiers regroupent plusieurs paramètres de la BD tels que le nom des fichiers "base" et des fichiers "redo log", le nom de la BD, la date de création de la BD, des informations sur l'état de cohérence de la BD, ...

— un fichier d'initialisation ou de paramétrage (init file) et optionnellement un fichier de mot de passe (password file)
— des fichiers d'archivage des journaux

o Les vues utiles du dictionnaire de données

v$database description générale de la base

DBA_TABLESPACES description des tablespaces et fichiers

DBA_DATA_FILES description des redologs

v$logfile nfo sur l'historique de tous les redos issus du control file

v$log_history infos sur les groupes et les membres

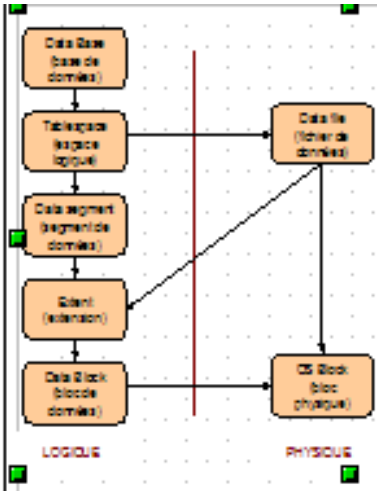
v$log infos sur les groupes et les membres

v$parameter TOUS les parametres d'init de l'instance, y compris CONTROL-FILES

v$controlfile nom des control files

1 Structure hiérarchique de stockage

a Hierarchie de stockage



b Tablespace

Les données d 'une BDD sont regroupées dans des espaces logiques plus petits appelés Tablespaces Un tablespace :

- N 'appartient qu'à une seule BDD.
- Peut être constitué de plusieurs fichiers physiques, mais constitue un seul espace logique de localisation.
- Est constitué d 'un ou plusieurs segments.
- Peut être mis en ligne à tout moment.
- Peut être mis hors ligne à tout moment sauf pour le Tablespace SYSTEM qui contient un Segment de données de type Rollback.
- Peut basculer d 'un état Read-Write à un état Read-only.

1. Tablespace SYSTEM :
   - Créé avec la base de données

— Nécessaire dans toute instance
— Contient le dictionnaire de données
— Contient le rollback segment SYSTEM

2. Tablespace non SYSTEM :
   — Permettent de répartir les différents types de segments : rollback, temporaires, données, index, ...
   — Permettent le contrôle de l 'espace alloué à chaque utilisateur.

c Les Data Files ou Fichiers de données

Il s 'agit de fichiers physiques dont la structure est conforme avec le système d 'exploitation sous lequel le Serveur de Base de Données fonctionne.

— Un data file n 'appartient qu'à un seul Tablespace.
— Oracle crée un Data File en allouant le total de l 'espace disque demandé.
— Le DBA peut :
   — Changer la taille d 'un Data File après sa création.
   — Obliger un Data File à croître dynamiquement en fonction de l' augmentation de taille des objets contenus dans leTablespace concerné.

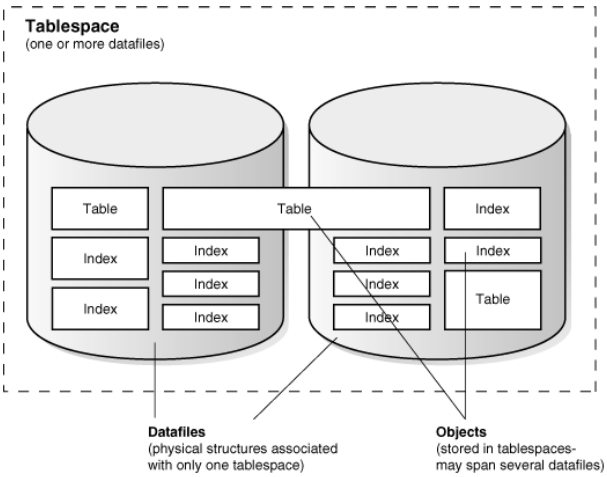
Informations sur les Tablespaces et Data File

Information sur les Tablespaces :
DBA_TABLESPACES
V$TABLESPACE
Information sur les Data files :
DBA_DATA_FILES
V$DATAFILE
Information sur les fichiers temporaires :
DBA_TEMP_FILES
V$TEMPFILE

d Segments

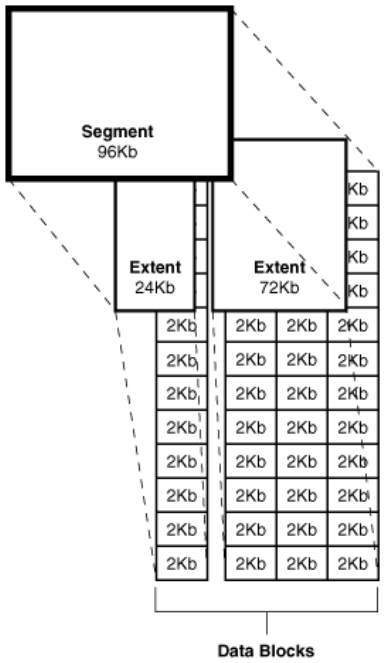
Il existe quatre types de segments :

1. *les segments de données* contiennent les enregistrements des tables, avec un segment de ce type par table ;
2. *les segments d'index* contiennent les enregistrements des index ; il y a un segment par index ;
3. *les segments temporaires* sont utilisés pour stocker des données pendant l'exécution des requêtes (par exemple pour les tris) ;
4. *les segments rollbacks* contiennent les informations permettant d'effectuer une reprise sur panne ou l'annulation d'une transaction



Un segment de données représente l 'espace allouée pour un type spécifique de structure de données dans un tablespace.

— Un Tablespace peut contenir un ou plusieurs segments.
— Un segment ne peut s'étendre sur plusieurs tablespaces.
— Un segment peut s'étendre sur plusieurs Data Files s'ils appartiennent à un même Tablespace.
— Chaque segment est constitué de un ou plusieurs Extents .
— Lorsque les extensions existantes sont pleines, Oracle alloue une autre extension pour ce segment, c'est pourquoi les extensions d'un segment peuvent être ou ne pas être contiguës sur le disque



Informations générales
OWNER
SEGMENT_NAME
SEGMENT_TYPE
TABLESPACE_NAME
Taille
EXTENTS
BLOCKS
BYTES
Paramètres de stockage
INITIAL_EXTENT
NEXT_EXTENT
MIN_EXTENTS
MAX_EXTENTS
PCT_INCREASE

e Les Extents (ou Extensions de Données)

Une extension est un ensemble de Data Blocks (ou blocs de données) contigus alloués pour stocker un type spécifique d'information.

- L' extension est l'unité de stockage constituant les segments.
- A la création d'un segment, celui-ci est constitué d'au moins une extension.
- Lorsqu'un segment grossit, des extensions lui sont ajoutées.
- Une extension ne peut s'étendre sur plusieurs Data Files.

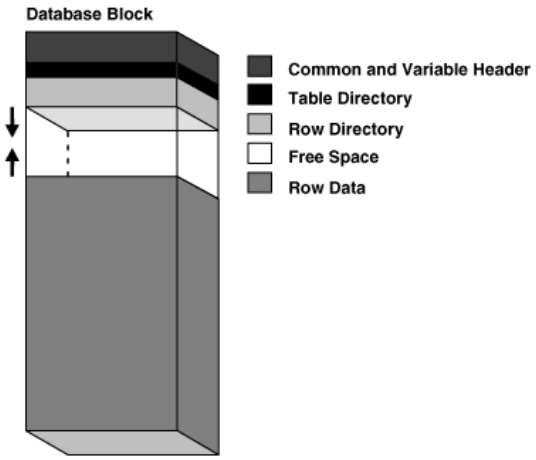
Interrogation de DBA_EXTENTS

- Informations générales
  - OWNER
  - SEGMENT-NAME
  - EXTENT-ID
  - TABLESPACE-NAME
- Taille
  - BLOCKS
  - BYTES
- Paramètres de localisation
  - TABLESPACE-NAME
  - RELATIVE-FNO
  - FILE-ID
  - BLOCK-ID

f Les blocs ORACLE

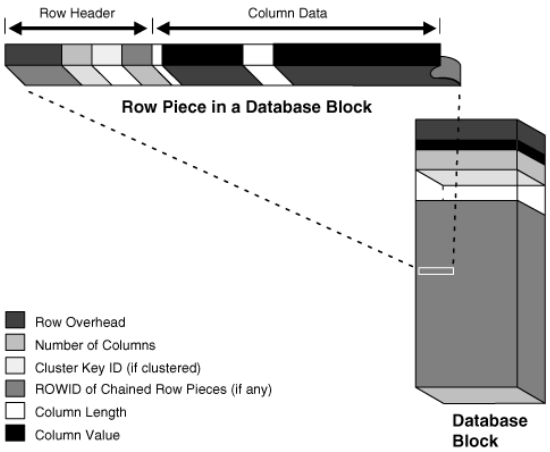
Le bloc est la plus petite unité de stockage gérée par ORACLE. La taille d'un bloc peut être choisie au moment de l'initialisation d'une base, et correspond obligatoirement à un multiple de

la taille des blocs du système d'exploitation.



Contenu d'un data block : La structure d'un bloc est identique quel que soit le type d'information qui y est stocké. Elle comprend :

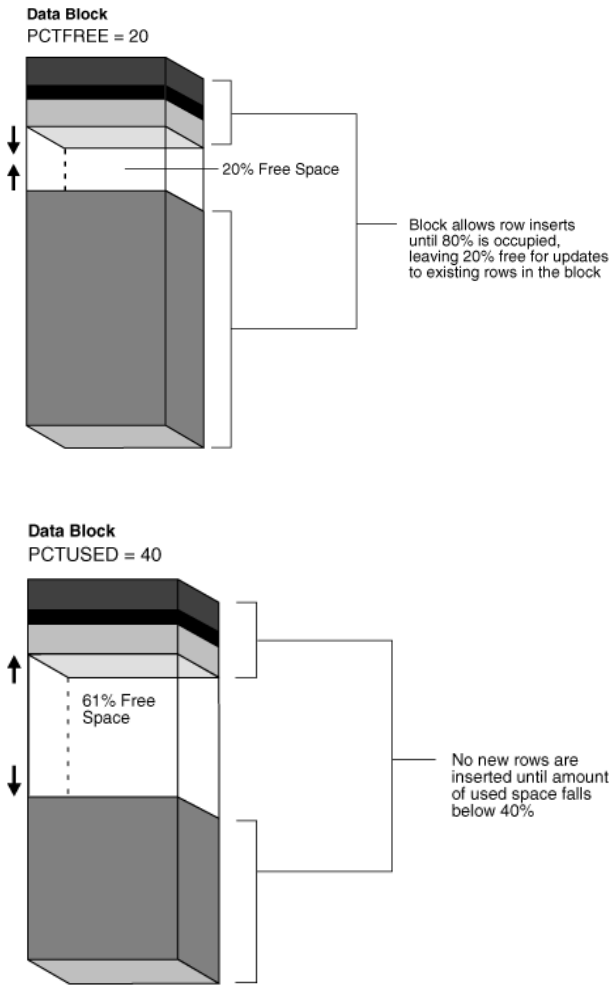
1. l'entête (header) contient l'adresse du bloc, et son type (données, index, etc) ;
2. le répertoire des tables donne la liste des tables pour lesquelles des informations sont stockées dans le bloc ;
3. le répertoire des enregistrements contient les adresses des enregistrements du bloc ;
4. un espace libre est laissé pour faciliter l'insertion de nouveaux enregistrements, ou l'agrandissement des enregistrements du bloc (par exemple un attribut à NULL auquel on donne une valeur par un UPDATE).



Paramètres de gestion de l'espace d'un Data Block pour les segments de données et d'index

PCTFREE indique le taux d'utilisation maximal au-delà duquel les insertions deviennent interdites :

PCTUSED indique le taux d'utilisation minimal en-deçà duquel ces insertions sont à nouveau possibles. Les valeurs de ces paramètres dépendent de l'application, ou plus précisément des



caractéristiques des données stockées dans une table particulière. Une petite valeur pour PCTFREE

permet aux insertions de remplir plus complètement le bloc, et peut donc mieux exploiter l'espace disque. Ce choix peut être valable pour des données qui sont rarement modifiées. En contrepartie une valeur plus importante de PCTFREE va occuper plus de blocs pour les mêmes données, mais offre plus de flexibilité pour des mises à jour fréquentes.

2 Les commandes

a création d'un tablespace

Syntaxe :

```
1 CREATE TABLESPACE tablespace-name
2   DATAFILE Datafile-Options Storage-options ;
```

Options de stockage :

INITIAL : taille du premier extent (Par défaut : 5*DB_BLOCK_SIZE).
NEXT Se rapporte à la taille de l'extent suivant.
MINEXTENTS : nombre d'extents alloués lors de la création du segment (Par défault 1).
PCTINCREASE : pourcentage de croissance de la taille de l'extent,
le n-iéme next est alors égale à next*(1+(pctincrease/100))e(n-2)

Création d'un TABLESPACE :

```
CREATE TABLESPACE tablespace
    DATAFILE filespec
        [ AUTOEXTEND { OFF
                     | ON [NEXT integer [K|M]]
                          [MAXSIZE [UNLIMITED | integer [K|M]] } ]
        [, filespec ...]
     [DEFAULT STORAGE storage_clause]
     [ONLINE | OFFLINE]
```

Exemple. *Exemple de tablespace*

```
CREATE TABLESPACE LESINFO4\_data
DATAFILE 'DATA01.dbf' size 100M,
         'DATA02.dbf' size 100M
DEFAULT STORAGE (initial 500K next 200K MAXEXTENTS 500 PCTINCREASE 10);
```

Pour ajouter un fichier à un tablespace :

```
ALTER TABLESPACE lesinfo4\_data ADD DATAFILE 'DATA03.dbf' SIZE 200M;\\
```

Vues du dictionnaire :
DBA_DAT_FILES
DBA_TABLESPACES
USER_TABLESPACES
DBA_TEMP_FILES
DBA_TS_QUOTAS
USER_TS_QUOTAS

b Création d'une TABLE

```

CREATE TABLE [schema.]table
      ( { column datatype [DEFAULT expr] [column_constraint] ...
        | table_constraint}
     [, { column datatype [DEFAULT expr] [column_constraint] ...
        | table_constraint} ]...)
     [ [PCTFREE integer] [PCTUSED integer]
       [INITRANS integer] [MAXTRANS integer]
```

```
9         [TABLESPACE tablespace]
10        [STORAGE INITIAL taille  NEXT taille MINEXTENTS n MAXEXTENTS n
              PCTINCREASE pct ]
11      [ ENABLE   enable_clause
12      | DISABLE disable_clause ] ...
13      [AS subquery]
```

crée une table dans le tablespace spécifié ou dans le tablespace SYSTEM par défaut avec les paramètres de stockage physique spécifiés par la clause STORAGE.

taille est un nombre qui précise la taille en KO ou MO du premier extent de la table et de l'extent suivant.

n précise respectivement le nombre d'extents alloué à la création et le nombre maximum d'extents.

pct précise la règle d'augmentation des extents futurs, en pourcentage

Principales Vues du dictionnaire :
DBA_TABLES
ALL_TABLES
USER_TABLES
ALL_TAB_COLUMNS
USER_TAB_COLUMNS
DBA_ALL_TABLES
ALL_ALL_TABLES
USER_ALL_TABLES
DBA_TAB_COMMENTS
ALL_TAB_COMMENTS
USER_TAB_COMMENTS
DBA_COL_COMMENTS
ALL_COL_COMMENTS
USER_COL_COMMENTS

Chapitre V

Les accélérateurs d'accès

Sommaire

Introduction
— Les index sont construits sur une ou plusieurs colonnes constituant leur clé 'd'accès'.
— Ils sont indépendants de la table accédée, ils peuvent être créés ou supprimés à tout moment sans affecter les tables ;
— Les requêtes SQL nomment les tables utilisées, mais pasl eurs index, et c'est l'optimiseur qui, décide, au lieu d'un parcours séquentiel, de l'utilisation d'un accélérateur d'accès.

A côté de l'accès séquentiel, il existe trois types d'accélérateurs d'accès :

1. arbre balancé,
2. bitmap,
3. hachage (implanté par Oracle sous la forme d'un cluster à une seule table avec hachage)

et des unités simultanément de données et d'accès :

1. les tables-index
2. les clusters

1 adresse physique d'un enregistrement

En règle générale un enregistrement est stocké dans un seul bloc.
L'adresse physique d'un enregistrement est le ROWID :

1. Le numéro de la page dans le fichier.
2. Le numéro du n-uplet dans la page.
3. Le numéro du fichier.

2 les index

La principale structure d'index utilisée par Oracle est l'arbre B+.
— Par défaut un arbre B+ est créé pour la clé primaire de chaque table, ce qui offre un double avantage :
  1. l'index permet d'optimiser les jointures ;
  2. au moment d'une insertion, l'index permet de vérifier très rapidement que la nouvelle clé n'existe pas déjà.
— Oracle maintient automatiquement l'arbre B+ au cours des insertions, suppressions et mises à jour affectant la table, et ce de manière transparente pour l'utilisateur.

a Création d'un index

Pour un utilisateur, créer un index dans son schéma suppose au moins l'une des conditions suivantes :
— la table ou le cluster à indexer est dans le schéma
— avoir le privilège INDEX sur la table à indexer

— avoir le privilège système CREATE ANY INDEX

```
CREATE [UNIQUE] INDEX index_name ON
   [schema. table (column [ASC $|$ DESC] [, column [ASC $|$ DESC]] ...)
   [TABLESPACE tablespace]
   [PCTFREE integer]
   [STORAGE ([NEXT integer [K $|$ M]] [MINEXTENTS integer]
   [MAXEXTENTS {integer}]);
```

Exemple. *Création d'un index secondaire sur le libellé dans la table PRODUIT*

```
CREATE INDEX libelle_idx1 ON
produit(libelle);
```

b Stockage des index

L'arbre B+ est placé dans le segment d'index associé à la table.
Au moment de la création d'un index, Oracle commence par trier la table dans un segment temporaire, puis construit l'arbre B+ de bas en haut afin d'obtenir un remplissage des blocs conforme au paramètre PCTFREE du tablespace. Au niveau des feuilles de l'arbre B+, on trouve, pour chaque valeur, le (cas de l'index unique) ou les (cas de l'index non-unique) ROWID des enregistrements associés à cette valeur. Un index requiert de l'espace de stockage pour contenir

ses données, appelé segment d'index, constitué d'extensions dont le nombre croit avec le nombre de lignes de la table indexée. L'utilisateur peut contrôler :

— le tablespace du segment d'index : est le tablespace nommé dans la commande CREATE INDEX,
— la clause de stockage/storage clause ou ensemble des paramètres d'allocation des extensions du segment d'index
— l'espace libre des blocs d'un segment d'index en fixant son paramètre PCTFREE.

En cas de clause de stockage non spécifiée pour l'index, les options de stockage par défaut ou du tablespace spécifié sont utilisées.

La performance des requêtes utilisant un index peut être améliorée en stockant un index et sa table sur des tablespaces différents localisés sur des unités de disque différentes.

3 le hachage

a le hachage : Structure appelée Hash Cluster

— Utilisée en deux étapes
1. On crée la structure avec tous ses paramètres

2. On affecte une ou plusieurs tables à la structure

— Le hachage dans Oracle n'est pas dynamique

b Hachage : création d'un hash cluster

```
1
2 CREATE CLUSTER [schema.] cluster (column datatype [, column datatype] ...
    ]
3   HASHKEYS integer [HASH IS expression]  [PCTFREE integer] [PCTUSED
      integer]
4   [SIZE integer [K $|$ M]] [storage-clause] [TABLESPACE tablespace];
```

— La clé de hachage est de type INTEGER ; Oracle fournit automatiquement une fonction avec de bonnes propriétés
— Nombre de valeurs de la fonction donné par HASHKEYS
— Taille de chaque entrée estimée par SIZE

c Hachage : Affectation à un Hash Cluster

On indique la structure d'affectation dans la commande CREATE TABLE

```
1 CREATE TABLE [schema.] table ( column-definition [, column-definition] ...
2   [CONSTRAINT constraint] ... )
3   [, CLUSTER [schema.] cluster ( column [, column] ...
4   [] ...);
```

— Assez délicat à paramétrer
— Demande un contrôle régulier par un DBA

Exemple. *Création d'un hash cluster lesproduits*

```
1 /*Création de la structure CLUSTER*/
2 CREATE CLUSTER lesproduits
3 (noprod number(5))
4 HASHKEYS 30;
5
6 /*Création de la table dans la structure CLUSTER*/
7 CREATE TABLE HPRODUIT
8     (p NUMBER(5) PRIMARY KEY,
9      libelle VARCHAR(15),
10     couleur VARCHAR(15)
11     )
12     CLUSTER lesproduits (p);
```

4 les bitmap

a les bitmap : pourquoi ?

Dans un index en B-arbre traditionnel, il y a stockage pour chaque valeur des attributs d'accès de la liste des rowId des lignes contenant cette valeur (dans Oracle, chaque valeur de la clé est répétée avec chaque rowId stocké).
Dans un index bitmap, à chaque valeur des attributs d'accès au lieu d'une liste de rowId, est associée une table de bits ou bitmap : chaque bit dans la table de bits correspond à une et une seule ligne de la table, et inversement. Si le bit est à 1, alors cela signifie que la ligne correspondante contient la valeur des attributs d'accès. Une fonction convertit la position d'un bit en une adresse de ligne, ainsi l'index bitmap fournit la même fonctionnalité qu'un index classique.

b Bitmaps : Avantages

— réduction du temps de réponse pour les requêtes type décisionnel/ applications entrepôts de données avec beaucoup de données
— réduction substantiel d'espace si le nombre des valeurs distinctes des attributs d'accès est petit

c Création d'un index bitmap

```
1 CREATE BITMAP INDEX [schema.] index ON
2   [schema.] table (column[ASC$|$DESC] [,column [ASC$|$DESC]]...)
3   [TABLESPACE tablespace]
4   [PCTFREE integer]
5   [storageComplément ...]
```

Exemple. *Création d'un index bitmap sur la couleur des produits*

```
1 CREATE BITMAP INDEX couleur_idx1 ON
2 produit(couleur);
```

5 les cluster

a Les clusters- introduction

Le cluster est une organisation physique des données qui consiste à regrouper physiquement (dans un même bloc disque) les lignes d'une ou plusieurs tables ayant une caractéristique commune (une même valeur dans une ou plusieurs colonnes) constituant la clé du cluster. La mise en cluster a trois objectifs :

1. accélérer la jointure selon la clé de cluster des tables mises en cluster,

2. accélérer la sélection des lignes d'une table ayant même valeur de clé, par le fait que ces lignes sont regroupées physiquement,
3. économiser de la place, du fait que chaque valeur de la clé du cluster n'est stockée qu'une seule fois

— La (ou les) colonne(s) en commun est appelée la clé (ou les attributs d'accès) du cluster. Les lignes de la table ou des tables qui partagent une même valeur de clé ou, cas d'un cluster à hachage, qui ont la même valeur par hachage sont physiquement stockées ensemble à l'intérieur de la base de données et du cluster.
— Chaque colonne de cluster doit avoir le même type et la même taille que la colonne correspondante de la table regroupée, mais pas obligatoirement le même nom.
— Un cluster est en arbre balancé (appelé aussi indexed cluster par Oracle) ou à hachage.
— Oracle ne crée pas un index en arbre balancé à la création d'un cluster en arbre balancé (même si le cluster est créé avec l'option INDEX). C'est à l'usager de créer l'arbre balancé du cluster, par la commande CREATE INDEX index ON CLUSTER cluster ;
— On peut placer jusqu'à 32 tables dans un cluster, en pratique 4 ou 5 tables.

Une table-index est une alternative à un cluster. La clé primaire de la table-index est équivalente à la clé d'accès d'un cluster aux valeurs NULL près des attributs d'accès.

b Les clusters- mise en oeuvre en trois étapes

1. Création d'un cluster

```
CREATE CLUSTER [schema.] cluster
  (column datatype [, column datatype]...)
  { HASHKEYS integer [HASH IS expression] $|$ [INDEX] }
  [PCTFREE integer] [PCTUSED integer]
  integer] [SIZE integer [K $|$ M]] [storage-clause]
  [TABLESPACE tablespace]
```

2. Création d'un cluster index

```
CREATE INDEX [schema.] index ON CLUSTER [schema.] cluster
  [PCTFREE integer]
  [TABLESPACE tablespace] [storage-clause]
```

3. Création d'une table dans un cluster

```
CREATE TABLE [schema.] table ( column-definition [, column-
    definition]...
  [, [CONSTRAINT constraint]
  CLUSTER [schema.] cluster ( column [, column]...)] )
```

SIZE spécifie l'espace en octets, Ko ou Mo, pour stocker toutes les lignes de même valeur de clé de cluster ou de même valeur calculée (implicitement un bloc).

Exemple. *Création d'un cluster fourniture regroupant pour chaque fournisseur, ses fournitures*

```
/*Création de la structure CLUSTER*/
CREATE CLUSTER fourniture
(nofournisseur number(5));

/* Renommage des tables */
RENAME fournisseur TO wfournisseur;
RENAME mafourniture TO wmafourniture;

/*Création de l'index du CLUSTER*/
CREATE INDEX unfournisseur ON CLUSTER fourniture;

/*Création de la table Fournisseur dans le CLUSTER*/
CREATE TABLE fournisseur
    (f NUMBER(5) PRIMARY KEY,
     nom VARCHAR(15),
     remise NUMBER(5,2),
     ville VARCHAR(15)
     )
     CLUSTER fourniture (f);


/*Création de la table Mafourniture dans le CLUSTER*/
CREATE TABLE mafourniture
    (four NUMBER(5) REFERENCES fournisseur(f)ON DELETE CASCADE,
     prod NUMBER(5),
     qte NUMBER(7) NOT NULL ,
     PRIMARY KEY (four, prod),
     FOREIGN KEY (prod) REFERENCES produits(p)
     )
     CLUSTER fourniture (four);
```

c Suppression d'un cluster

```
DROP CLUSTER [schema.]
   cluster [INCLUDING TABLES
   [CASCADE CONSTRAINTS]]
```

6 les tables-index

a Le principe

Une table-index ou table organisée en index (Index-Organized Table) est à la fois
— une méthode d'accès en arbre balancé nécessairement avec une clé primaire de façon à identifier chaque ligne, ou ensemble UNIQUE d'attributs d'accès NOT NULL, et

— la table accédée ordonnée suivant la clé primaire
Un enregistrement n'est pas identifié par son ROWID mais par sa clé primaire.

b Les tables index avantages et inconvénients

+ accès plus rapide pour les recherches basées sur la valeur de clé, ou sur un intervalle de valeur. La traversée d'index fournit en effet directement les enregistrements, alors qu'elle ne produit qu'une liste de ROWID dans le cas d'un arbre-B+.
+ un accès rapide par intervalle de la clé primaire parce que les lignes sont ordonnées suivant la clé primaire
+ un besoin en stockage plus faible : les valeurs de la clé primaire ne sont pas dupliquées à la fois dans l'index et dans la table
- Quand les enregistrements sont de taille importante car on ne peut alors mettre que peu d'enregistrements dans un nœud de l'arbre, ce qui risque de faire perdre à l'index une partie de son efficacité.

c les tables index Déclaration

```
1 CREATE TABLE [schema.] table
2   ( column-definition [, column_definition]...
3   [, out-line-constraint [,out-line-constraint]... ] )
4   ORGANIZATION INDEX
5   [PCTFREE integer][storage-clause] [TABLESPACE tablespace]);
```

Exemple. *Création d'une table organisée en index Produit2*

```
1 CREATE TABLE produit2
2    (prdt NUMBER(5) PRIMARY KEY,
3     libelle VARCHAR(15),
4     couleur VARCHAR(15))
5
6     ORGANIZATION INDEX ;
```

7 Principales vues du dictionnaire

— DBA-INDEXES (ALL, USER)
— DBA-INDEXTYPES (ALL, USER)
— DBA-IND-COLUMNS (ALL, USER)
— INDEX-STATS, INDEX-HISTOGRAM

Chapitre VI

Annexe

1 Annexe1 : Les principales vues du dictionnaire

```
1
2  DICTIONARY
3  USER_TABLES
4  USER_TAB_COLUMNS
5  USER_VIEWS
6  USER_INDEXES
7  USER_IND_COLUMNS
8  USER_CLUSTERS
9  USER_OBJECTS
10 USER_SEQUENCES
11 USER_SYNONYMS
12 USER_USERS
13 USER_CONSTRAINTS
14 USER_DB_LINKS
15 USER_TAB_PRIVS
16 USER_EXTENTS
17 USER_TS_QUOTAS
18 DBA_ROLES    All Roles which exist in the database
19 DBA_PROFILES     Display all profiles and their limits
20 USER_RESOURCE_LIMITS  Display resource limit of the user
21 USER_PASSWORD_LIMITS  Display password limits of the user
22 USER_CATALOG     Tables, Views, Synonyms and Sequences owned by the user
23 ALL_CATALOG    All tables, views, synonyms, sequences accessible to the
       user
24 DBA_CATALOG    All database Tables, Views, Synonyms, Sequences
25 USER_CLUSTERS    Descriptions of user's own clusters
26 ALL_CLUSTERS     Description of clusters accessible to the user
27 DBA_CLUSTERS     Description of all clusters in the database
28 USER_CLU_COLUMNS  Mapping of table columns to cluster columns
29 DBA_CLU_COLUMNS   Mapping of table columns to cluster columns
30 USER_COL_COMMENTS Comments on columns of user's tables and views
```

```
31 ALL_COL_COMMENTS  Comments on columns of accessible tables and views
32 DBA_COL_COMMENTS  Comments on columns of all tables and views
33 USER_COL_PRIVS    Grants on columns for which the user is the owner,
       grantor or grantee
34 ALL_COL_PRIVS    Grants on columns for which the user is the grantor,
       grantee, owner, or an enabled role or PUBLIC is the grantee
35 DBA_COL_PRIVS    All grants on columns in the database
36 USER_OBJECTS     Objects owned by the user
37 ALL_OBJECTS    Objects accessible to the user
38 DBA_OBJECTS    All objects in the database
39 USER_PROCEDURES    Description of the users own procedures
40 ALL_PROCEDURES     Description of all procedures available to the user
41 DBA_PROCEDURES     Description of all procedures
42 USER_ROLE_PRIVS    Roles granted to current user
43 DBA_ROLE_PRIVS     Roles granted to users and roles
44 USER_SYS_PRIVS     System privileges granted to current user
45 DBA_SYS_PRIVS    System privileges granted to users and roles
46 USER_SEQUENCES     Description of the user's own SEQUENCEs
47 ALL_SEQUENCES    Description of SEQUENCEs accessible to the user
48 DBA_SEQUENCES    Description of all SEQUENCEs in the database
49 DBA_SYNONYMS     All synonyms in the database
50 USER_SYNONYMS    The user's private synonyms
51 ALL_SYNONYMS     All synonyms for base objects accessible to the user
       and session
52 USER_TABLES    Description of the user's own relational tables
53 USER_OBJECT_TABLES  Description of the user's own object tables
54 USER_ALL_TABLES    Description of all object and relational tables owned
       by the user's
55 ALL_TABLES    Description of relational tables accessible to the user
56 ALL_OBJECT_TABLES Description of all object tables accessible to the
       user
57 ALL_ALL_TABLES    Description of all object and relational tables
       accessible to the user
58 DBA_TABLES    Description of all relational tables in the database
59 DBA_OBJECT_TABLES Description of all object tables in the database
60 DBA_ALL_TABLES    Description of all object and relational tables in the
        database
61 USER_TAB_COLS    Columns of user's tables, views and clusters
62 ALL_TAB_COLS     Columns of user's tables, views and clusters
63 DBA_TAB_COLS     Columns of user's tables, views and clusters
64 USER_TAB_COLUMNS  Columns of user's tables, views and clusters
65 ALL_TAB_COLUMNS Columns of user's tables, views and clusters
66 DBA_TAB_COLUMNS Columns of user's tables, views and clusters
67 USER_NESTED_TABLE_COLS  Columns of nested tables
68 ALL_NESTED_TABLE_COLS Columns of nested tables
69 DBA_NESTED_TABLE_COLS Columns of nested tables
70 USER_TAB_COL_STATISTICS Columns of user's tables, views and clusters
```

```
ALL_TAB_COL_STATISTICS  Columns of user's tables, views and clusters
DBA_TAB_COL_STATISTICS  Columns of user's tables, views and clusters
USER_TAB_HISTOGRAMS Histograms on columns of user's tables
ALL_TAB_HISTOGRAMS  Histograms on columns of all tables visible to user
DBA_TAB_HISTOGRAMS  Histograms on columns of all tables
USER_TAB_COMMENTS Comments on the tables and views owned by the user
ALL_TAB_COMMENTS  Comments on tables and views accessible to the user
DBA_TAB_COMMENTS  Comments on all tables and views in the database
USER_TAB_PRIVS    Grants on objects for which the user is the owner,
   grantor or grantee
ALL_TAB_PRIVS   Grants on objects for which the user is the grantor,
   grantee, owner, or an enabled role or PUBLIC is the grantee
DBA_TAB_PRIVS   All grants on objects in the database
USER_TAB_PRIVS_MADE All grants on objects owned by the user
ALL_TAB_PRIVS_MADE  User's grants and grants on user's objects
USER_TAB_PRIVS_RECD Grants on objects for which the user is the grantee
ALL_TAB_PRIVS_RECD  Grants on objects for which the user, PUBLIC or
   enabled role is the grantee
USER_USERS    Information about the current user
ALL_USERS   Information about all users of the database
DBA_USERS   Information about all users of the database
USER_VIEWS    Description of the user's own views
ALL_VIEWS   Description of views accessible to the user
DBA_VIEWS   Description of all views in the database
USER_CONSTRAINTS  Constraint definitions on user's own tables
ALL_CONSTRAINTS   Constraint definitions on accessible tables
DBA_CONSTRAINTS   Constraint definitions on all tables
USER_LOG_GROUPS   Log group definitions on user's own tables
ALL_LOG_GROUPS    Log group definitions on accessible tables
DBA_LOG_GROUPS    Log group definitions on all tables
USER_CLUSTER_HASH_EXPRESSIONS Hash functions for the user's hash
   clusters
ALL_CLUSTER_HASH_EXPRESSIONS  Hash functions for all accessible clusters
DBA_CLUSTER_HASH_EXPRESSIONS  Hash functions for all clusters
USER_CONS_COLUMNS Information about accessible columns in constraint
   definitions
ALL_CONS_COLUMNS  Information about accessible columns in constraint
   definitions
DBA_CONS_COLUMNS  Information about accessible columns in constraint
   definitions
USER_ERRORS   Current errors on stored objects owned by the user
ALL_ERRORS    Current errors on stored objects that user is allowed to
   create
DBA_ERRORS    Current errors on all stored objects in the database
USER_SOURCE   Source of stored objects accessible to the user
ALL_SOURCE    Current source on stored objects that user is allowed to
   create
```

```
109 DBA_SOURCE    Source of all stored objects in the database
110 USER_TRIGGERS   Triggers owned by the user
111 ALL_TRIGGERS    Triggers accessible to the current user
112 DBA_TRIGGERS    All triggers in the database
113 USER_INTERNAL_TRIGGERS  Description of the internal triggers on the user
       's own tables
114 ALL_INTERNAL_TRIGGERS Description of the internal triggers on the tables
        accessible to the user
115 DBA_INTERNAL_TRIGGERS Description of the internal triggers on all tables
       in the database
116 USER_TRIGGER_COLS Column usage in user's triggers
117 ALL_TRIGGER_COLS  Column usage in user's triggers or in triggers on user
       's tables
118 DBA_TRIGGER_COLS  Column usage in all triggers
119 USER_DEPENDENCIES Dependencies to and from a users objects
120 ALL_DEPENDENCIES  Dependencies to and from objects accessible to the
       user
121 DBA_DEPENDENCIES  Dependencies to and from objects
122 DBA_OBJECT_SIZE   Sizes, in bytes, of various pl/sql objects
123 USER_OBJECT_SIZE  Sizes, in bytes, of various pl/sql objects
124 USER_SEGMENTS   Storage allocated for all database segments
125 DBA_SEGMENTS    Storage allocated for all database segments
126 DBA_SEGMENTS_OLD  Storage allocated for all database segments
127 USER_EXTENTS    Extents comprising segments owned by the user
128 DBA_EXTENTS   Extents comprising all segments in the database
129 DBA_UNDO_EXTENTS  Extents comprising all segments in the system managed
       undo tablespaces
130 DBA_LMT_USED_EXTENTS  All extents in the locally managed tablespaces
131 DBA_DMT_USED_EXTENTS  All extents in the dictionary managed tablespaces
132 USER_FREE_SPACE   Free extents in tablespaces accessible to the user
133 DBA_FREE_SPACE    Free extents in all tablespaces
134 DBA_LMT_FREE_SPACE  Free extents in all locally managed tablespaces
135 DBA_DMT_FREE_SPACE  Free extents in all dictionary managed tablespaces
136 DBA_FREE_SPACE_COALESCED Statistics on Coalesced Space in Tablespaces
137 DBA_DATA_FILES    Information about database data files
138 USER_TABLESPACES  Description of accessible tablespaces
139 DBA_TABLESPACES   Description of all tablespaces
140 DBA_TEMP_FILES    Information about database temp files
141
142
143
144 COLUMN_PRIVILEGES Grants on columns for which the user is the grantor,
       grantee, owner, or an enabled role or PUBLIC is the grantee
145
146 DICTIONARY    Description of data dictionary tables and views
147 DICT_COLUMNS    Description of columns in data dictionary tables and
       views
```

```
148 DUAL   -
149 GLOBAL_NAME   global database name
150
151 ROLE_ROLE_PRIVS   Roles which are granted to roles
152 ROLE_SYS_PRIVS    System privileges granted to roles
153 ROLE_TAB_PRIVS    Table privileges granted to roles
154 SESSION_PRIVS   Privileges which the user currently has set
155 SESSION_ROLES   Roles which the user currently has enabled.
156 TABLE_PRIVILEGE   Grants on objects for which the user is the grantor,
       grantee, owner, or an enabled    role or PUBLIC is the grantee
157 CAT Synonym for USER_CATALOG
158 CLU Synonym for USER_CLUSTERS
159 DICT  Synonym for DICTIONARY
160 IND Synonym for USER_INDEXES
161 OBJ Synonym for USER_OBJECTS
162 SEQ Synonym for USER_SEQUENCES
163 SYN Synonym for USER_SYNONYMS
164 TABS  Synonym for USER_TABLES
165 COLS  Synonym for USER_TAB_COLUMNS
```

2 Annexe 2 : Fonctions arithmétiques

Dans ce paragraphe, ont été regroupées les fonctions ayant un ou plusieurs nombres comme arguments, et renvoyant une valeur numérique. [ROUND(n ,m)]

ABS(nb) Renvoie la valeur absolue de nb.

CEIL(nb) Renvoie le plus petit entier supérieur ou égal à nb.

COS(n) Renvoie le cosinus de n, n étant un angle exprimé en radians.

COSH(n) Renvoie le cosinus hyperbolique de n.

EXP(n) Renvoie e puissance n.

FLOOR(nb) Renvoie le plus grand entier inférieur ou égal à nb.

LN(n) Renvoie le logarithme népérien de n qui doit être un entier strictement positif.

LOG(m,n) Renvoie le logarithme en base m de n. m doit être un entier strictement supérieur à 1, et n un entier strictement positif.

MOD(m,n) Renvoie le reste de la division entière de m par n, si n vaut 0 alors renvoie m. Attention, utilisée avec au moins un de ses arguments négatifs, cette fonction donne des résultats qui peuvent être différents d'un modulo classique. Cette fonction ne donne pas toujours un résultat dont le signe du diviseur.

POWER(m,n) Renvoie m puissance n, m et n peuvent être des nombres quelconques entiers ou réels mais si m est négatif n doit être un entier.

ROUND(n[,m]) Si m est positif, renvoie n arrondi (et non pas tronqué) à m chiffres après la virgule. Si m est négatif, renvoie n arrondi à m chiffres avant la virgule. m doit être un entier et il vaut 0 par défaut.

SIGN(nb) Renvoie -1 si nb est négatif, 0 si nb est nul, 1 si nb est positif.

SIN(n) Renvoie le sinus de n, n étant un angle exprimé en radians.

SINH(n) Renvoie le sinus hyperbolique de n.

SQRT(nb) Renvoie la racine carrée de nb qui doit être un entier positif ou nul.

TAN(n) Renvoie la tangente de n, n étant un angle exprimé en radians.

TANH(n) Renvoie la tangente hyperbolique de n.

TRUNC(n[,m)] Si m est positif, renvoie n arrondi tronqué à m chiffres après la virgule. Si m est négatif, renvoie n tronqué à m chiffres avant la virgule. m doit être un entier et il vaut 0 par défaut.

3 Annexe 3 : Expressions et fonctions sur les chaînes de caractères

a Opérateur sur les chaînes de caractères

Il existe un seul opérateur sur les chaînes de caractères : la concaténation. Cet opérateur se note au moyen de deux caractères |(barre verticale) accolés. Le résultat d'une concaténation est une chaîne de caractères obtenue en écrivant d'abord la chaîne à gauche de || puis celle à droite de ||. SELECT libellé ||'/'|| couleur FROM Produit ;

b Fonctions sur les chaînes de caractères

Le paragraphe suivant contient les fonctions travaillant sur les chaînes de caractères et renvoyant des chaînes de caractères.

```
1
2  CONCAT(chaîne1,chaîne2):
3    Renvoie la chaîne obtenue en concaténant chaîne1 à chaîne2. Cette
       fonction est équivalente à l'opérateur de concaténation | |.
4  INITCAP(chaîne):
5    Renvoie chaîne en ayant mis la première lettre de chaque mot en
       majuscule et toutes les autres en minuscule. Les séparateurs de
       mots sont les espaces et les  caractères non alphanumériques.
6  LOWER(chaîne)
7    Renvoie chaîne en ayant mis toutes ses lettres en minuscules.
8  LPAD(chaîne,long,[char])
9    Renvoie la chaîne obtenue en complétant, ou en tronquant, chaîne pour
       qu'elle ait comme longueur long en ajoutant éventuellement à gauche
        le caractère (ou la  chaîne de caractères) char. La valeur par
       défaut de char est un espace.
10 LTRIM(chaîne[,ens])
11   Renvoie la chaîne obtenue en parcourant à partir de la gauche chaîne
       et en supprimant tous les caractères qui sont dans ens. On s'arrête
        quand on trouve un  caractère qui n'est pas dans ens. La valeur
        de defaut de ens est un espace.
```

```
12 REPLACE(chaine,avant,après
13   Renvoie chaine dans laquelle toutes les occurrences de la chaîne de
       caractères avant ont été remplacés par la chaîne de caractères
       après.
14 RPAD(chaîne,n,[char])
15   Renvoie la chaîne obtenue en complétant, ou en tronquant, chaîne pour
       qu'elle ait comme longueur long en ajoutant éventuellement à droite
        le caractère (ou la  chaîne de caractères) char. La valeur par
       défaut de char est un espace.
16 RTRIM(chaîne[,ens])
17   Renvoie la chaîne obtenue en parcourant à partir de la droite chaîne
       et en supprimant tous les caractères qui sont dans ens. On s'arrête
        quand on trouve un  caractère qui n'est pas dans ens. La valeur
        de defaut de ens est un espace.
18 SOUNDEX(chaîne)
19   Renvoie la chaîne de caratères constituée de la représentation
       phonétique des mots de chaîne.
20 SUBSTR(chaîne,m[,n])
21   Renvoie la partie de chaîne commençant au caractère m et ayant une
       longueur de n.
22 TRANSLATE(chaîne, avant, après)
23   Renvoie une chaîne de caractères en remplaçant chaque caractère de
       chaîne présent dans avant par le caractère situé à la même position
        dans après. Les  caractères de chaîne non présents dans avant ne
        sont pas modifiés. avant peut contenir plus de caractères que
       après, dans ce cas les caractères de avant sans  correspondants
       dans après seront supprimés de chaîne .
24 UPPER(chaîne)
25   Renvoie chaîne en ayant mis toutes ses lettres en majuscules.
26 INSTR(chaîne, sous-chaîne, debut, occ)
27   Renvoie la position du premier caractère de chaîne correspondant à l'
       occurrence occ de sous-chaîne en commençant la recherche à la
       position début.
28 LENGTH(chaîne)
29   Renvoie la longueur de chaîne, exprimée en nombre de caractères.
```

4 Annexe 4 : Expressions et fonctions sur les dates

a Opérateurs sur les dates

Au moyen des opérateurs arithmétiques + et - il est possible de construire les expressions suivantes :

— date +/- nombre : le résultat est une date obtenue en ajoutant le nombre de jours nombre à la date date.

— date2 - date1 : le résultat est le nombre de jours entre les deux dates.

b Fonctions sur les dates

```
1  ADD_MONTHS(date,n)
2    Renvoie la date obtenue en ajoutant n mois à date. n peut être un
       entier quelconque. Si le mois obtenu a moins de jours que le jour
       de date, le jour obtenu est le  dernier du mois.
3  LAST_DAY(date)
4    Renvoie la date du dernier jour du mois de date.
5  MONTHS_BETWEEN(date2, date1)
6    Renvoie le nombre de mois entre date2 et date1, si date2 est après
       date1 le résultat est positif, sinon le résultat est négatif. Si
       les jours date2 et date1 sont les  mêmes, ou si ce sont les
       derniers jours du mois, le résultat est un entier. La partie
       fractionnaire est calculée en considérant chaque jour comme 1/31ème
        de mois
7  NEXT_DAY(date,nom_du_jour)
8    Renvoie la date du prochain jour de la semaine dont le nom est
       nom_de_jour.
9  ROUND(date[,précision])
10   Renvoie date arrondie à l'unité spécifiée dans précision. L'unité de
       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,... Par défaut la précision est le jour.
11 SYSDATE
12   Renvoie la date et l'heure courantes du système d'exploitation hôte.
13 TRUNC(date[,précision])
14   Renvoie date tronquée à l'unité spécifiée dans précision. Les
       paramètres sont analogues à ceux de la fonction ROUND.
```