Le langage SQL

SQL : Interroger une base

SQL : Interroger une base

Introduction

Ce chapitre expose la partie du langage sql permettant de retrouver des informations stockées dans une base de données. Il s'agit, comme cela a déjà été dit, d'un langage déclaratif dont la syntaxe est très simple (comme beaucoup de langages de ce type) ce qui permet de se concentrer sur le problème à résoudre.

Les exemples cités dans ce chapitre ont tous étés testés sous ORACLE, un des systèmes de gestion de bases de données relationnels les plus répandus sur le marché.

Ces exemples sont bâtis sur une base de données composée des deux relations suivantes :

- emp (nom, num, fonction, n_sup, embauche, salaire, comm, n_dept)

```
NOM          NUM    FONCTION      N_SUP   EMBAUCHE  SALAIRE COMM N_DEPT
-------- ------ -------------   ------  --------- ------- ----- ------
MARTIN    16712  directeur      25717   23-MAY-90   40000         30
DUPONT    17574  administratif  16712   03-MAY-95    9000         30
DUPOND    26691  commercial     27047   04-APR-88   25000  2500   20
LAMBERT   25012  administratif  27047   14-APR-91   12000         20
JOUBERT   25717  président              10-OCT-82   50000         30
LEBRETON  16034  commercial     27047   01-JUN-91   15000  0      20
MARTIN    17147  commercial     27047   10-DEC-93   20000  500    20
PAQUEL    27546  commercial     27047   03-SEP-83   22000  2000   20
LEFEBVRE  25935  commercial     27047   11-JAN-84   23500  1500   20
GARDARIN  15155  ingénieur      24533   22-MAR-85   24000         10
SIMON     26834  ingénieur      24533   04-OCT-88   20000         10
DELOBEL   16278  ingénieur      24533   16-NOV-94   21000         10
ADIBA     25067  ingénieur      24533   05-OCT-87   30000         10
CODD      24533  directeur      25717   12-SEP-75   55000         10
LAMERE    27047  directeur      25717   07-SEP-89   45000         20
BALIN     17232  administratif  24533   03-OCT-87   13500         10
BARA      24831  administratif  16712   10-SEP-88   15000         30
```

- dept(n_dept, nom, lieu)

```
N_DEPT     NOM            LIEU
------ -------------- -------------
10         recherche      Rennes
20         vente          Metz
30         direction      Gif
40         fabrication    Toulon
```

La commande `SELECT` constitue, à elle seule, le langage permettant d'interroger une base de données. Elle permet :

- de sélectionner certaines colonnes d'une table : c'est l'opération de ***projection*** ;
- de sélectionner certaines lignes d'une table en fonction de leur contenu : c'est l'opération de ***restriction*** ;
- de combiner des informations venant de plusieurs tables : ce sont les opérations de ***jointure***, ***union***, ***intersection***, ***différence relationnelle*** ;

- de combiner entre elles ces différentes opérations.

Une interrogation, on parle plutôt de requête, est une combinaison d'opérations portant sur des tables (relations) et dont le résultat est lui-même une table dont l'existence est éphémère (le temps de la requête).

On peut introduire un commentaire à l'intérieur d'une commande sql en l'encadrant par /* */.

Interroger simplement une base

Sélection de colonnes ou projection

La commande SELECT la plus simple a la syntaxe suivante :

```
SELECT *
FROM  nom_table ;
```

Dans laquelle :

- nom_table : est le nom de la table sur laquelle porte la sélection.
- *:signifie que toutes les colonnes de la table sont sélectionnées.

Par défaut toutes les lignes sont sélectionnées. On peut limiter la sélection à certaines colonnes, en indiquant une liste de noms de colonnes à la place de l'astérisque.

```
SELECT nom_col1, nom_col2, ...
FROM  nom_table ;
```

Exercice : Donner le nom et la fonction de chaque employé.

La clause DISTINCT ajoutée derrière la commande SELECT permet d'éliminer les duplications.

Exercice : Quelles sont toutes les fonctions différentes ?

Sélection de lignes ou restriction

La clause WHERE permet de spécifier quelles sont les lignes à sélectionner. Elle est suivie d'un prédicat qui sera évalué pour chaque ligne de la table. Les lignes pour lesquelles le prédicat est vrai seront sélectionnées.

La syntaxe est la suivante :

```
SELECT *
FROM  nom_table
WHERE prédicat ;
```

Un prédicat n'est ni plus ni moins que la façon dont on exprime une propriété. Les prédicats, qu'ils soient simples ou composés, sont constitués à partir d'expressions que l'on compare entre elles.

Expression simple

Une expression simple peut être :

- une variable désignée par un nom de colonne,
- une constante.

Les expressions peuvent être de trois types : numérique, chaîne de caractères ou date. A chacun de ces types correspond un format de constante :

Constante numérique

nombre contenant éventuellement un signe, un point décimal et une puissance de dix. Ex : -10, 2.5, 1.2 E-10

Constante chaîne de caractères

une chaîne de caractères entre apostrophes. Ex :'MARTIN' (Attention, une lettre en majuscules n'est pas considérée comme égale à la même lettre en minuscule).

Constante date

une chaîne de caractères entre apostrophes au format suivant : jour-mois-année où le jour est sur deux chiffres, le mois est désigné par les trois premières lettres de son nom en anglais, l'année est sur deux chiffres. Ex : '01-FEB-85'

On peut, en SQL, exprimer des expressions plus complexes en utilisant des opérateurs et des fonctions étudiés dans le chapitre Expressions et fonctions.

Prédicat simple

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

=	égal
!=	différent
<	inférieur
<=	inférieur ou égal
>	supérieur
>=	supérieur ou égal

Les trois types d'expressions peuvent être comparés au moyen de ces opérateurs :

- Pour les types date, la relation d'ordre est l'ordre chronologique.
- Pour les types caractère, la relation d'ordre est l'ordre alphabétique.

Il faut ajouter à ces opérateurs arithmétiques classiques les opérateurs suivants :

`expr1` BERTWEEN expr2 AND expr3

vrai si `expr1` est compris entre `expr2` et `expr3`, bornes incluses

`expr1` IN (expr2, expr3, ...)

Quels sont les employés dont le nom commence par M ?

Prédicats composés

Les opérateurs logiques AND (et) et OR (ou inclusif) peuvent être utilisés pour combiner entre eux plusieurs prédicats. L'opérateur NOT placé devant un prédicat en inverse le sens.

L'opérateur AND est prioritaire par rapport à l'opérateur OR. Des parenthèses peuvent être utilisées pour imposer une priorité dans l'évaluation du prédicat ou simplement pour rendre plus claire l'expression logique.

Exercices : Quels sont les employés du département 30 ayant un salaire supérieur à 25000 ?

Quels sont les employés directeurs, ou commerciaux et travaillant dans le département 10 ?

La requête précédente donnerait le même résultat sans les parenthèses, résultat différent de celui du `SELECT` suivant.

Exercice : Quels sont les employés directeurs ou commerciaux, et travaillant dans le département 10 ?

Valeurs `NULL`

Pour sql, une valeur `NULL` est une valeur non définie. Il est possible d'ajouter une ligne à une table sans spécifier de valeur pour les colonnes non obligatoires : ces colonnes absentes auront la valeur `NULL`.

Par exemple les employés dont la rémunération ne prend pas en compte de commission auront une valeur `NULL`, c'est-à-dire indéfinie, comme commission.

L'opérateur `IS NULL` permet de tester la valeur `NULL` : le prédicat `expr IS NULL` est vrai si l'expression a la valeur `NULL` (c'est-à-dire si elle est indéfinie).

Exercice : Quels sont les employés dont la commission a la valeur `NULL` ?

L'opérateur `IS NOT NULL` permet de construire un prédicat vrai si la valeur n'est pas `NULL` (et donc le prédicat `expr` IS NOT NULL est vrai si `expr` est définie)

Remarques

- La valeur `NULL` est différente de la valeur zéro qui, elle, qui est une valeur bien définie.
- Le prédicat expr = NULL est toujours faux, et ne permet donc pas de tester si l'expression a la valeur `NULL`.
- Une expression de la forme `NULL + val` donne `NULL` comme résultat quelle que puisse être la valeur de `val`.

Nom de colonne

Les colonnes constituant le résultat d'un `SELECT` peuvent être renommées dans le `SELECT`, ceci est utile en particulier lorsque la colonne résultat est une expression. Pour cela, il suffit de faire suivre l'expression définissant la colonne d'un nom, selon les règles suivantes :

- le nom (30 caractères maximum) est inséré derrière l'expression définissant la colonne,

séparé de cette dernière par un espace.

- si le nom contient des séparateurs (espace, caractère spécial), ou s'il est identique à un mot clé de SQL (ex : `DATE`), il doit être mis entre guillemets "".

Ce nom est celui sous lequel la colonne sera connue des interfaces externes. Sous `SQLPLUS`, par exemple, il constituera le titre par défaut de la colonne, et servira de référence pour définir un format pour la colonne.

Exercice : Salaire de chaque employé.

Remarque : Attention, ce nom n'est pas connu à l'intérieur du `SELECT`.

Classer le résultat d'une interrogation

Les lignes constituant le résultat d'un `SELECT` sont obtenues dans un ordre indéterminé. On peut, dans un `SELECT`, demander que le résultat soit classé dans un ordre ascendant ou descendant, en fonction du contenu d'une ou plusieurs colonnes (jusqu'à 16 critères de classement possibles). Les critères de classement sont spécifiés dans une clause `ORDER BY` dont la syntaxe est la suivante :

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

Le classement se fait d'abord selon la première colonne spécifiée dans l'`ORDER BY` puis les lignes ayant la même valeur dans la première colonne sont classées selon la deuxième colonne de l'`ORDER BY`, etc... Pour chaque colonne, le classement peut être ascendant (par défaut) ou descendant (`DESC`). L'`ORDER BY` peut faire référence à une colonne par son nom ou par sa position dans la liste des colonnes présentes derrière le `SELECT` (la première colonne sélectionnée a le numéro 1, la deuxième a le numéro 2, ...).

Exercice : Donner tous les employés classés par fonction, et pour chaque fonction classés par salaire décroissant

Remarque : Dans un classement les valeurs `NULL` sont toujours en tête quel que soit l'ordre du classement (ascendant ou descendant).

Requêtes avancées

Les jointures

La jointure est une opération permettant de combiner des informations venant de plusieurs tables.

Jointure externe

Lorsqu'une ligne d'une table figurant dans une jointure n'a pas de correspondant dans les autres tables, elle ne satisfait pas au critère d'équi-jointure et donc ne figure pas dans le résultat de la jointure.
Une option permet de faire figurer dans le résultat les lignes satisfaisant la condition d'équi-jointure plus celles n'ayant pas de correspondant. Cette option s'obtient en accolant (+) au nom de colonne de la table dans laquelle manquent des éléments, dans la condition d'équi-jointure.

Exercice : Le département 40 ne figurait pas dans le résultat du `SELECT` précédent. Faites-le figurer dans le résultat du `SELECT` suivant.

Le (+) peut s'interpréter comme l'ajout d'une ligne fictive dont toutes les colonnes ont la valeur `NULL`, et qui réalise la correspondance avec les lignes de l'autre table qui n'ont pas de correspondant réel. Dans l'exemple ci-dessus, la valeur de `nom` associée au département 40 est la valeur `NULL`.

Exercice : Retrouver les départements n'ayant aucun employé.

Les sous-interrogations

Une caractéristique puissante de SQL est la possibilité qu'un critère de recherche employé dans une clause `WHERE` (expression à droite d'un opérateur de comparaison) soit lui-même le résultat d'un `SELECT` ; c'est ce qu'on appelle une sous-interrogation.

Sous-interrogation ramenant une seule valeur

Exercice : Quels sont les employés ayant la même fonction que coda ?

Remarques

- une sous-interrogation qui ne ramène aucune ligne se termine avec un code d'erreur.
- une sous-interrogation ramenant plusieurs lignes provoquera aussi, dans ce cas, une erreur (pour traiter correctement ce cas, voir paragraphe ci-dessous)

Sous-interrogation ramenant plusieurs lignes

Une sous-interrogation peut ramener plusieurs lignes à condition que l'opérateur de comparaison admette à sa droite un ensemble de valeurs. 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.

Exercice : Quels sont les employés gagnant plus que tous les employés du département 30.

Sous-interrogation ramenant plusieurs colonnes

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

Exercice : Quels sont les employés ayant même fonction et même supérieur que `CODD`?

Sous-interrogation synchronisée avec l'interrogation principale

Dans les exemples précédents, la sous-interrogation était évaluée d'abord, puis le résultat pouvait être utilisé pour exécuter l'interrogation principale. 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.

Exercice : Quels sont les employés ne travaillant pas dans le même département que leur supérieur hiérarchique ?

Il a fallu ici renommer la table `emp` de l'interrogation principale pour pouvoir la référencer dans la

sous-interrogation.
n_sup IS NOT NULL est nécessaire car dans le cas de JOUBERT la colonne n_sup est NULL et la sous-requête ne ramène alors aucune valeur.

Sous-interrogation ramenant au moins une ligne

L'opérateur EXISTS permet de construire un prédicat vrai si la sous-interrogation qui suit ramène au moins une ligne.

Exercice : Quels sont les employés travaillant dans un département qui a procédé à des embauches depuis le début de l'année 94 ?

Remarque : On peut inverser le sens de l'opérateur EXISTS en le faisant précéder de NOT.

Sous-interrogations multiples

Un SELECT peut comporter plusieurs sous-interrogations, soit imbriquées, soit au même niveau dans différents prédicats combinés par des AND ou des OR.

Exercice : Liste des employés du département 10 ayant même fonction que quelqu'un du département de DUPONT.

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)

C'est le cas dans la table emp où chaque ligne contient un "pointeur" vers le supérieur hiérarchique : n_sup. Chaque employé a un seul supérieur hiérarchique. Par contre, un employé peut être le supérieur hiérarchique de plusieurs employés. Le lien n_sup définit donc une structure d'arbre.

Parcours d'un arbre

En utilisant la clé num et le lien vers le supérieur hiérarchique n_sup, il est possible de parcourir l'arbre hiérarchique implicitement décrit par la table emp. SQL permet d'obtenir comme résultat d'un SELECT les lignes dans l'ordre de parcours de l'arbre (ou de la liste). Pour cela 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.

Un arbre (ou une liste) sera construit à partir de chaque ligne répondant au prédicat. En l'absence de clause START WITH, un arbre sera construit à partir de chaque ligne.

Exercice : Quels sont toutes les personnes dont CODD est le supérieur hiérarchique (CODD compris) ?

Niveau : LEVEL

Il est possible d'obtenir pour chaque ligne le niveau correspondant dans l'arbre ou dans la liste, la première ligne sélectionnée étant de niveau 1. Ce niveau s'obtient dans la variable LEVEL, que l'on peut utiliser dans le SELECT comme résultat. Cette variable peut aussi être présente dans un prédicat.

Exercice : Donner la liste des subordonnés de JOUBERT, avec pour chacun d'eux son niveau.

Sélection de lignes

Clause WHERE

Le prédicat de la clause WHERE permet d'éliminer certaines lignes. Mais le parcours de l'arbre n'est pas interrompu lorsqu'une ligne ne répondant pas au prédicat est rencontrée.

Exercice : Quelles sont toutes les personnes dont JOUBERT est le supérieur hiérarchique (JOUBERT compris et CODD non compris) ?

CODD ne figure pas dans le résultat , mais tous ses subordonnés y figurent.

Clause CONNECT BY

Il est possible de rajouter des conditions dans le CONNECT BY. Lorsque le prédicat du CONNECT BY

n'est plus satisfait, le parcours de l'arbre s'arrête.

Exercice : Quelles sont toutes les personnes dont `JOUBERT` est le supérieur hiérarchique (`JOUBERT` compris et `CODD` et ses subordonnés non compris) ?

Dans la requête précédente, `nom !='CODD'` n'était pas un des prédicats du `CONNECT BY` alors qu'ici il l'est.

Restrictions

Boucle

SQL détecte les boucles dans les `CONNECT BY`. Le `SELECT` est interrompu avec un code d'erreur.

Niveaux

Le parcours descendant d'un arbre nécessite, lorsqu'on a atteint l'extrémité d'une branche, de remonter jusqu'au nœud le plus proche pour parcourir la branche suivante.
Le nombre de niveaux qu'il est ainsi possible de remonter est limité à 256.

Jointure

Un `SELECT` avec clause `CONNECT BY` **ne doit pas être une jointure.**

Les opérateurs ensemblistes

Les opérateurs ensemblistes permettent de "joindre" des tables verticalement c'est-à-dire de **combiner dans un résultat unique des lignes provenant de deux interrogations.** Les lignes peuvent venir de tables différentes mais après projection on doit obtenir des tables ayant même schéma de relation.

Les opérateurs ensemblistes sont les suivants :

- l'union : `UNION`
- l'intersection : `INTERSECT`
- la différence relationnelle : `MINUS`

La syntaxe d'utilisation est la même pour ces trois opérateurs :

```
SELECT ...   {UNION | INTERSECT | MINUS } SELECT ...
```

Dans une requête utilisant des opérateurs ensemblistes :

- Tous les `SELECT` doivent avoir le même nombre de colonnes sélectionnées, et leur types doivent être un à un identiques. Les conversions éventuelles doivent être faites à l'intérieur du `SELECT` à l'aide des fonctions de conversion.
- Les doubles sont éliminés (`DISTINCT` implicite).
- Les noms de colonnes (titres) sont ceux du premier `SELECT`.
- La largeur des colonnes est la plus grande parmi tous les `SELECT`.
- Dans une requête on ne peut trouver qu'un seul `ORDER BY`. S'il est présent, il doit être mis dans le dernier `SELECT` et il ne peut faire référence qu'aux numéros des colonnes et non pas à leurs noms (car les noms peuvent être différents dans chacune des interrogations).

L'on peut combiner le résultat de plus de deux `SELECT` au moyen des opérateurs `UNION`, `INTERSECT`, `MINUS`.

```
SELECT ... UNION SELECT ... MINUS SELECT ...
```

Dans ce cas l'expression est évaluée de gauche à droite, mais on peut modifier l'ordre d'évaluation en utilisant des parenthèses.

```
SELECT ...
UNION  (SELECT ...
        MINUS
        SELECT ...)
```

Expressions et fonctions

Expressions et Fonctions simples

Une expression est un ensemble de variables (contenu d'une colonne), de constantes et de fonctions combinées au moyen d'opérateurs. Les fonctions prennent une valeur dépendant de leurs arguments qui peuvent être eux-mêmes des expressions.

Les expressions peuvent figurer :

- en tant que colonne résultat d'un `SELECT`,
- dans une clause `WHERE`,
- dans une clause `ORDER BY`.

Il existe trois types d'expressions correspondant chacun à un type de données de SQL : **arithmétique, chaîne de caractère, date**. A chaque type correspondent des opérateurs et des fonctions spécifiques.

SQL autorise les mélanges de types dans les expressions et effectuera les conversions nécessaires : dans une expression mélangeant dates et chaînes de caractères, les chaînes de caractères seront converties en dates, dans une expression mélangeant nombres et chaînes de caractères, les chaînes de caractères seront converties en nombre.

Expressions et fonctions arithmétiques

Une expression arithmétique peut contenir :

- des noms de colonnes
- des constantes
- des fonctions arithmétiques

combinés au moyen des opérateurs arithmétiques.

Opérateurs arithmétiques

Les opérateurs arithmétiques présents dans sql sont les suivants :

- `+` addition ou `+` unaire
- `-` soustraction ou `-` unaire
- `*` multiplication
- `/` division

Remarque : la division par 0 provoque une fin avec code d'erreur.

Priorité des opérateurs

Une expression arithmétique peut comporter plusieurs opérateurs. Dans ce cas, le résultat de l'expression peut varier selon l'ordre dans lequel sont effectuées les opérations. Les opérateurs de

multiplication et de division sont prioritaires par rapport aux opérateurs d'addition et de soustraction. Des parenthèses peuvent être utilisées pour forcer l'évaluation de l'expression dans un ordre différent de celui découlant de la priorité des opérateurs.

Exercices : Donner pour chaque commercial son revenu (salaire + commission).

Donner la liste des commerciaux classée par commission sur salaire décroissant.

Donner la liste des employés dont la commission est inférieure à 5% du salaire.

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.

Exercice : Donner pour chaque employé son salaire journalier.

Expressions et fonctions sur les chaînes de caractères

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 `||`.

Fonctions sur les chaînes de caractères

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

`CONCAT(chaîne1,chaîne2)`

Renvoie la chaîne obtenue en concaténant `chaîne1` à `chaîne2`. Cette fonction est équivalente à l'opérateur de concaténation `| |`.

`INITCAP(chaîne)`

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.

`LOWER(chaîne)`

Renvoie `chaîne` en ayant mis toutes ses lettres en minuscules.

`LPAD(chaîne, long,[char])`

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.

`LTRIM(chaîne[,ens])`

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 défaut de `ens` est un espace.

`REPLACE(chaîne, avant, après)`

Renvoie `chaîne` dans laquelle toutes les occurrences de la chaîne de caractères `avant` ont été remplacés par la chaîne de caractères `après`.

`RPAD(chaîne, n,[char])`

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.

`RTRIM(chaîne[,ens])`

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 défaut de `ens` est un espace.

`SOUNDEX(chaîne)`

Renvoie la chaîne de caractères constituée de la représentation phonétique des mots de `chaîne`.

`SUBSTR(chaîne, m[,n])`

Renvoie la partie de `chaîne` commençant au caractère `m` et ayant une longueur de `n`.

`TRANSLATE(chaîne, avant, après)`

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` .

`UPPER(chaîne)`

Renvoie `chaîne` en ayant mis toutes ses lettres en majuscules.

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

`INSTR(chaîne, sous-chaîne, début, occ)`

Renvoie la position du premier caractère de `chaîne` correspondant à l'occurrence `occ` de `sous-chaîne` en commençant la recherche à la position `début`.

`LENGTH(chaîne)`

Renvoie la longueur de `chaîne`, exprimée en nombre de caractères.

Expressions et fonctions sur les dates

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.

Fonctions sur les dates

`ADD_MONTHS(date, n)`

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.

`LAST_DAY(date)`

Renvoie la date du dernier jour du mois de `date`.

`MONTHS_BETWEEN(date2, date1)`

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

`NEXT_DAY(date, nom_du_jour)`

Renvoie la date du prochain jour de la semaine dont le nom est `nom_de_jour`.

`ROUND(date[,précision])`

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.

`SYSDATE`

Renvoie la date et l'heure courantes du système d'exploitation hôte.

`TRUNC(date[,précision])`

Renvoie `date` tronquée à l'unité spécifiée dans `précision`. Les paramètres sont analogues à ceux de la fonction `ROUND`.

Exercices : Donner la date du lundi suivant l'embauche de chaque employé.

Donner la date date d'embauche de chaque employé arrondie à l'année.

Donner pour chaque employé le nombre de jours depuis son embauche.

Fonctions de conversion

`ASCII(chaîne)`

Renvoie le nombre correspondant au code ascii du premier caractère de `chaîne`.

`CHR(nombre)`

Renvoie le caractère dont `nombre` est le code ascii.

`TO_CHAR(nombre, format)`

Renvoie la chaîne de caractères en obtenue en convertissant `nombre` en fonction de `format`.

`Format` est une chaîne de caractères pouvant contenir les caractères suivants :

`9`	représente un chiffre (non représenté si non significatif)
`0`	représente un chiffre (représenté même si non significatif)
`.`	point décimal apparent
`V`	définit la position du point décimal non apparent
`,`	une virgule apparaîtra à cet endroit
`$`	un $ précédera le premier chiffre significatif
`B`	le nombre sera représenté par des blancs s'il vaut 0
`EEEE`	le nombre sera représenté avec un exposant (le spécifier avant `MI` ou PR)
`MI`	le signe négatif sera à droite
`PR`	un nombre négatif sera entre <>

`TO_CHAR(date, format)`

Renvoie conversion d'une date en chaîne de caractères. Le format indique quelle partie de la date doit apparaître, c'est 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`	3 derniers chiffres de l'année
`yy`	2 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 avant ou après jésus christ

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

`Th` ajout du suffixe ordinat st, nd, rd, th

`Sp` nombre en toutes lettres

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

`TO_DATE(chaîne, format)`

Permet de convertir une chaîne de caractères en donnée de type date. Le format est identique à celui de la fonction `TO_CHAR`.

`TO_NUMBER(chaîne)`

Convertit chaîne en sa valeur numérique.

Remarque : On peut également insérer dans le format une chaîne de caractères quelconque, à condition de la placer entre guillemets"".

Exercices : Donner la liste de tous les employés dont le nom ressemble à `DUPONT`.

Donner la liste de tous les noms des employés en ayant supprimé tous les 'L' et les 'E' en tête des noms.

Donner la liste de tous les noms des employés en ayant remplacé les A et les M par des * dans les noms.

Afficher tous les salaires avec un $ en tête et au moins trois chiffres (dont deux décimales).

Autres fonctions

`GREATEST(expr1, expr2,...)`

Renvoie la plus grande des valeurs `expr1, expr2,...`. Toutes les expressions sont converties au format de `expr1` avant comparaison.

`LEAST`

Renvoie la plus petite des valeurs `expr1, expr2,...`. Toutes les expressions sont converties au format de `expr1` avant comparaison.

`NVL(expr_1, expr_2)`

Prend la valeur `expr_1`, sauf si `expr_1` est `NULL` auquel cas `NVL` prend la valeur `expr_2`.

Une valeur `NULL` en SQL est une valeur non définie.
Lorsque l'un des termes d'une expression a 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.

`DECODE(crit, val_1, res_1 [, val_2, res_2 ...], 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 récupéré est :

-

Les fonctions de groupe

Les fonctions de groupe

Dans les exemples précédents, chaque ligne résultat d'un `SELECT` était le résultat de calculs sur les valeurs d'une seule ligne de la table consultée. Il existe un autre type de `SELECT` qui permet d'effectuer des calculs sur l'ensemble des valeurs d'une colonne. Ces calculs sur l'ensemble des valeurs d'une colonne se font au moyen de l'une des fonctions suivantes :

`AVG([DISTINCT | ALL] expression)`

Renvoie la **moyenne** des valeurs d' `expression`.

`COUNT(* | [DISTINCT | ALL] expression)`

Renvoie **le nombre de lignes du résultat** de la requête. Si `expression` est présent, on ne compte que les lignes pour lesquelles cette expression n'est pas `NULL`.

`MAX([DISTINCT | ALL] expression)`

Renvoie la plus petite des valeurs d'`expression`.

`MIN([DISTINCT | ALL] expression)`

Renvoie la plus grande des valeurs d'`expression`.

`STDDEV([DISTINCT | ALL] expression)`

Renvoie **l'écart-type** des valeurs d'`expression`.

`SUM([DISTINCT | ALL] expression)`

Renvoie **la somme** des valeurs

`VARIANCE([DISTINCT | ALL] expression)`

Renvoie **la variance** des valeurs d'`expression`.

`DISTINCT` Indique à la fonction de groupe de ne prendre en compte que des valeurs distinctes.
`ALL` Indique à la fonction de groupe de prendre en compte toutes les valeurs, c'est la valeur par défaut.

Exercices : Donner le total des salaires du département 10.

Donner le nom, la fonction et le salaire de l'employé (ou des employés) ayant le salaire le plus élevé.

Remarques

- Ces `SELECT` sont différents de ceux vus précédemment. Il est, par exemple, impossible de demander en résultat à la fois une colonne et une fonction de groupe.
- un `SELECT` comportant une fonction de groupe peut être utilisé dans une sous-interrogation.

Valeurs `NULL`

Aucune des fonctions de groupe ne tient compte des valeurs `NULL` à l'exception de `count(*)`. Ainsi, `SUM(col)` est la somme des valeurs non `NULL` de la colonne `col`. De même `AVG` est la somme des valeurs non `NULL` divisée par le nombre de valeurs non `NULL`.

Calcul sur plusieurs groupes

Il est possible de subdiviser la table en groupes, chaque groupe étant l'ensemble des lignes ayant

une valeur commune. C'est la clause `GROUP BY` qui permet de découper la table en plusieurs groupes :

```
GROUP BY expr_1, expr_2, ...
```

Si on a une seule expression, ceci définit les groupes comme les ensembles de lignes pour lesquelles cette expression prend la même valeur. Si plusieurs expressions sont présentes les groupes sont définis de la façon suivante : parmi toutes les lignes pour lesquelles `expr_1` prend la même valeur, on regroupe celles ayant `expr_2` identique, ... Un `SELECT` de groupe avec une clause `GROUP BY` donnera une ligne résultat pour chaque groupe.

Exercice : Total des salaires pour chaque département

Remarque : 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 expressions figurant dans le `GROUP BY`.

Sélection des groupes

De la même façon qu'il est possible de sélectionner certaines lignes au moyen de la clause `WHERE`, il est possible dans un `SELECT` comportant une fonction de groupe de sélectionner par la clause `HAVING`, qui se place après la clause `GROUP BY`.

Le prédicat dans la clause `HAVING` suit les mêmes règles de syntaxe qu'un prédicat figurant dans une clause `WHERE`.

Cependant, il ne peut porter que sur des caractéristiques du groupe : fonction de groupe ou expression figurant dans la clause `GROUP BY`, dans ce cas la clause `HAVING` doit être placée après la clause `GROUP BY`.

Exercice : Donner la liste des salaires moyens par fonction pour les groupes ayant plus de deux employés.

Remarque : Un `SELECT` de groupe peut contenir à la fois une clause `WHERE` et une clause `HAVING`. La clause `WHERE` sera d'abord appliquée pour sélectionner les lignes, puis les groupes seront constitués à partir des lignes sélectionnées, et les fonctions de groupe seront évaluées.

Exercice : Donner le nombre d'ingénieurs ou de commerciaux des départements ayant au moins deux employés de ces catégories.

Une clause `HAVING` peut comporter une sous-interrogation.

Exercice : Quel est le département ayant le plus d'employés ?

Fonction de groupe à deux niveaux

Il est possible d'appliquer au résultat d'un `SELECT` avec `GROUP BY` un deuxième niveau de fonction de groupe.

Exercice : la fonction `MAX` peut être appliquée aux nombres d'employés de chaque département pour obtenir le nombre d'employés du département ayant le plus d'employés.

SQL : Modifier une base

Définition

Le langage de manipulation de données est le langage permettant de modifier les informations contenues dans une base de données.
L'unité manipulée est la ligne. Il existe trois commandes sql permettant d'effectuer les trois types de modifications des données : **ajout, modification et suppression.**

Ajout de lignes

La commande `INSERT` permet d'insérer une ligne dans une table en spécifiant les valeurs à insérer.
La syntaxe est la suivante :

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

La liste des noms de colonne est optionnelle. Si elle est omise, la liste des colonnes sera par défaut la liste de l'ensemble des colonnes de la table dans l'ordre de la création de la table.
Si une liste de colonnes est spécifiée, les colonnes ne figurant pas dans la liste auront la valeur `NULL`.

Il est possible d'insérer dans une table des lignes provenant d'une autre table. La syntaxe est la suivante :

```
INSERT INTO nom_table(nom_col1, nom_col2, ...)
SELECT ...
```

Le `SELECT` peut contenir n'importe quelle clause sauf un `ORDER BY` qui impliquerait un classement des lignes contraire à l'esprit du relationnel.

Exemple : Insérer dans la table bonus les noms et salaires des directeurs.

```
INSERT INTO bonus
SELECT nom, salaire
FROM emp
WHERE fonction = 'directeur';
```

Modification de lignes

La commande `UPDATE` permet de modifier les valeurs d'une ou plusieurs colonnes, dans une ou plusieurs lignes existantes d'une table. La syntaxe est la suivante :

```
UPDATE nom_table
SET nom_col1 = {expression1 | ( SELECT ...) },
    nom_col2 = {expression2 | ( SELECT ...) }
WHERE prédicat
```

Les valeurs des colonnes `nom_col1, nom_col2, ...` sont modifiées dans toutes les lignes satisfaisant au prédicat. En l'absence d'une clause `WHERE`, toutes les lignes sont mises à jour.
Les expressions `expression1, expression2,...` peuvent faire référence aux anciennes valeurs de la ligne.

Exemple : Augmenter de 10% les ingénieurs.

```
UPDATE emp
SET salaire = salaire * 1.1
WHERE fonction = 'ingénieur' ;
```

Suppression de lignes

La commande `DELETE` permet de supprimer des lignes d'une table.
La syntaxe est la suivante :

```
DELETE FROM nom_table
WHERE prédicat ;
```

Toutes les lignes pour lesquelles `prédicat` est évalué à vrai sont supprimées. En l'absence de clause `WHERE`, toutes les lignes de la table sont supprimées.

SQL : Définir une base

Définition

Le langage de définition des données est le langage permettant de créer ou de modifier le schéma d'une relation et donc d'une table.
Il permet de créer, de modifier et de supprimer non seulement les tables, mais aussi les vues, les index et les clusters.

Les tables

Créer une table

La table est la structure de base contenant les données des utilisateurs. Quand on crée une table, on peut spécifier les informations suivantes :

- la définition des colonnes,
- les contraintes d'intégrité,
- La tablespace contenant la table,
- les caractéristiques de stockage,
- le cluster contenant la table,
- les données résultant d'une éventuelle requête.

Création simple

La commande de création de table la plus simple ne comportera que le nom et le type de chaque colonne de la table. L'on peut créer une table par la commande `CREATE TABLE` en spécifiant le nom et le type de chaque colonne. A la création, la table sera vide mais un certain espace lui sera alloué. La syntaxe est la suivante :

```
CREATE TABLE nom_table
       (nom_col1 TYPE1,
        nom_col2 TYPE2,
        ...)
```

L'option `NOT NULL` assure qu'ORACLE interdit lors d'un `INSERT` ou d'un `UPDATE` que cette colonne contienne la valeur `NULL`, par défaut elle est autorisée.

Création avec Insertion de données

On peut insérer des données dans une table lors de sa création par la commande suivante :

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

On peut ainsi, en un seul ordre SQL créer une table et la remplir avec des données provenant du résultat d'un `SELECT`.

On n'a pas besoin alors de spécifier de type pour les colonnes : les types des données sont ceux

provenant du SELECT. Si des conversions de type sont à faire, on peut dans le SELECT utiliser les fonctions TO_CHAR, TO_DATE, TO_NUMBER.

Par défaut les noms des colonnes de la nouvelle table sont les noms des colonnes du SELECT. Si des expressions apparaissent dans le SELECT, les colonnes correspondantes doivent **impérativement** être renommées.

Le SELECT peut contenir des fonctions de groupes mais pas d'ORDER BY car les lignes d'une table ne peuvent pas être classées.

On peut, et même on doit, quand on crée une table définir les contraintes d'intégrité que devront respecter les données que l'on mettra dans la table (voir un peu plus bas).

Les types de données

Les types de données peuvent être :

`NUMBER[(longueur,[précision]`

Ce type de données permet de stocker des données numériques à la fois entières et réelles dont la valeur est comprise entre 10^-130 et 10^125 avec une précision de 38 chiffres.

`Longueur`

précise le nombre maximum de chiffres significatifs stockés (par défaut 38),

`Précision`

donne le nombre maximum de chiffres après la virgule (par défaut 38), sa valeur peut être comprise entre -84 et 127. Une valeur négative signifie que le nombre est arrondi à gauche de la virgule.

`CHAR(longueur)`

Ce type de données permet de stocker des chaînes de caractères de longueur fixe. `longueur` doit être inférieur à 255, sa valeur par défaut est 1.

`VARCHAR(longueur)`

Ce type de données permet de stocker des chaînes de caractères de longueur variable. `longueur` doit être inférieur à 2000, il n'y a pas de valeur par défaut.

`DATE`

Ce type de données permet de stocker des données constituées d'une date et d'une heure.

`RAW(longueur)`

Ce type de données permet de stocker des caractères non imprimables.

`LONG`

Ce type de données permet des stocker des chaînes de caractères de longueur variable et inférieure à 2^31 -1. Les colonnes de ce type sont soumises à certaines restrictions ;

- une table ne peut pas contenir plus d'une colonne de ce type ;
- les colonnes de ce type ne peuvent pas apparaître dans des contraintes d'intégrité ;
- les colonnes de ce type ne peuvent pas être indexées ;
- les colonnes de ce type ne peuvent pas apparaître dans des clauses : `WHERE`, `GROUP BY`, `ORDER BY` ou `CONNECT BY` ainsi que dans un `DISTINCT`.

Contraintes d'intégrité

A la création d'une table, les contraintes d'intégrité se déclarent de la façon suivante :

```
CREATE TABLE nom_table (
nom_col_1 type_1,
nom_col_2 type_2,
...
nom_col_n type_n
CONSTRAINT [nom_contrainte_1] contrainte_1,
CONSTRAINT [nom_contrainte_2] contrainte_2,
...
CONSTRAINT [nom_contrainte_m] contrainte_m
);
```

Ou bien de la façon suivante :

```
CREATE TABLE nom_table (
nom_col_1 type_1  CONSTRAINT [nom_contrainte_1_1] contrainte_1_1
                  CONSTRAINT [nom_contrainte_1_2] contrainte_1_2
                ...
                  CONSTRAINT [nom_contrainte_1_m] contrainte_1_m,

nom_col_2 type_2  CONSTRAINT [nom_contrainte_2_1] contrainte_2_1
                  CONSTRAINT [nom_contrainte_2_2] contrainte_2_2
                ...
                  CONSTRAINT [nom_contrainte_2_p] contrainte_2_p,

...
nom_col_n type_n  CONSTRAINT [nom_contrainte_n_1] contrainte_n_1
                  CONSTRAINT [nom_contrainte_n_2] contrainte_n_2
                ...
                  CONSTRAINT [nom_contrainte_n_q] contrainte_n_q
);
```

Les contraintes différentes que l'on peut déclarer sont les suivantes :

`NOT NULL`

La colonne ne peut pas contenir de valeurs `NULL`.

`UNIQUE`

Chaque ligne de la table doit avoir une valeur différente ou `NULL` pour cette (ou ces) colonne.

`PRIMARY KEY`

Chaque ligne de la table doit avoir une valeur différente pour cette (ou ces) colonne. les valeurs `NULL` sont rejetées.

`FOREIGN KEY`

Cette colonne fait référence à une colonne clé d'une autre table.

`CHECK`

Permet de spécifier les valeurs acceptables pour une colonne.

Modifier une table

On peut modifier **dynamiquement** la définition d'une table grâce à la commande `ALTER TABLE`. Deux types de modifications sont possibles : ajout d'une colonne et modification d'une colonne existante.

Il n'est pas possible de supprimer une colonne. Par contre une colonne qui n'est plus utilisée peut être mise à la valeur `NULL`, auquel cas elle n'occupe plus d'espace disque. Si on désire vraiment

supprimer une colonne, il faut :

- se créer une nouvelle table sans la colonne en question
- détruire l'ancienne table,
- donner à la nouvelle table le nom de l'ancienne.

Ajouter une colonne

La commande suivante permet d'ajouter une ou plusieurs colonnes à une table existante :

```
ALTER TABLE nom_table ADD
      (nom_col1 TYPE1,
       nom_col2 TYPE2,
       ...)
```

Les types possibles sont les mêmes que ceux décrits avec la commande `CREATE TABLE`.

Si la table contient déjà des lignes, la nouvelle colonne aura des valeurs `NULL` pour les lignes existantes.

Modifier une colonne

Il est possible de modifier la définition d'une colonne, à condition que la nouvelle définition soit compatible avec le contenu de la colonne et en respectant les contraintes suivantes :

- dans tous les cas il est possible d'augmenter la taille d'une colonne ;
- il est possible de diminuer la taille, ou même de changer le type d'une colonne vide ;
- on peut spécifier `NOT NULL` si la colonne ne contient aucune valeur `NULL` ;
- on peut dans tous les cas spécifier `NULL` pour autoriser les valeurs `NULL`.

```
ALTER TABLE nom_table MODIFY
      (nom_col1 nouveau_TYPE1,
       nom_col2 nouveau_TYPE2,
       ...)
```

Supprimer une table

La commande `DROP TABLE` permet de supprimer une table, sa syntaxe est la suivante :

```
DROP TABLE nom_table ;
```

La table `nom_table` est alors supprimée. La définition de la table ainsi que son contenu sont détruits, et l'espace occupé par la table est libéré.

Renommer une table

On a la possibilité de changer le nom d'une table par la commande `RENAME`, la syntaxe est la suivante :

```
RENAME ancien_nom TO nouveau_nom ;
```

Les vues

Les vues permettent d'assurer l'objectif d'**indépendance logique**. Grâce à elles, chaque utilisateur pourra avoir sa vision propre des données.

On a vu que le résultat d'un `SELECT` est lui-même une table.
Une telle table, qui n'existe pas dans la base mais est créée dynamiquement lors de l'exécution du `SELECT`, peut être vue comme une table réelle par les utilisateurs. Pour cela, il suffit de cataloguer le `SELECT` en tant que vue.

Les utilisateurs pourront consulter la base, ou modifier la base (avec certaines restrictions) à travers la vue, c'est-à-dire manipuler la table résultat du `SELECT` comme si c'était une table réelle.

Créer une vue

La commande `CREATE VIEW` permet de créer une vue en spécifiant le `SELECT` constituant la définition de la vue :

```
CREATE VIEW nom_vue [(nom_col1,...)]
AS SELECT ...
WITH CHECK OPTION ;
```

La spécification des noms de colonnes de la vue est facultative. Par défaut, les noms des colonnes de la vue sont les mêmes que les noms des colonnes résultat du `SELECT` (si certaines colonnes résultat du `SELECT` sont des expressions, il faut renommer ces colonnes dans le `SELECT`, ou spécifier les noms de colonne de la vue).

Une fois créée, une vue s'utilise comme une table. Il n'y a pas de duplication des informations mais stockage de la définition de la vue.

Exemple : Création d'une vue constituant une restriction de la table `emp` aux employés du département 10.

```
CREATE VIEW emp10 AS
SELECT *
FROM emp
WHERE n_dept = 10 ;
```

Le `CHECK OPTION` permet de vérifier que la mise à jour ou l'insertion faite à travers la vue ne produisent que des lignes qui font partie de la sélection de la vue.

Ainsi donc, si la vue `emp10` a été créée avec `CHECK OPTION` on ne pourra à travers cette vue ni modifier, ni insérer des employés ne faisant pas partie du département 10.

Il est possible d'effectuer des `INSERT` et des `UPDATE` à travers des vues, sous deux conditions :

- le `SELECT` définissant la vue ne doit pas comporter de jointure,
- les colonnes résultat du `SELECT` doivent être des colonnes réelles et non pas des expressions.

Exemple : Modification des salaires du département 10 à travers la vue `emp10`.

```
UPDATE emp10
SET sal = sal *1.1;
```

Toutes les lignes de la table `emp`, telles que le contenu de la colonne `n_dept` est égal à 10 seront modifiées.

Supprimer une vue

Une vue peut être détruite par la commande :

```
DROP  VIEW nom_vue;
```

Renommer une vue

On peut renommer une vue par la commande :

```
RENAME ancien_nom TO nouveau_nom;
```

Les index

Introduction - Généralités

Selon le modèle relationnel les sélections peuvent être faites en utilisant le contenu de n'importe quelle colonne et les lignes sont stockées dans n'importe quel ordre.

Considérons le `SELECT` suivant :

```
SELECT *
FROM emp
WHERE nom = 'MARTIN'
```

Un moyen de retrouver la ou les lignes pour lesquelles ,font color=rednom est égal à `MARTIN` est de balayer toute la table.

Un tel moyen d'accès conduit à des temps de réponse prohibitifs pour des tables dépassant quelques centaines de lignes.

Une solution offerte par tous les systèmes de gestion de bases de données est la création d'index, qui permettra de satisfaire aux requêtes les plus fréquentes avec des temps de réponse acceptables.

Un index sera matérialisé par la création de blocs disque contenant des couples (valeurs d'index, numéro de bloc) donnant le numéro de bloc disque dans lequel se trouvent les lignes correspondant à chaque valeur d'index.

Structure d'un index

Les index sont des structures permettant de retrouver une ligne dans une table à partir de la valeur d'une colonne ou d'un ensemble de colonnes. Un index contient la liste triée des valeurs des colonnes indexées avec les adresses des lignes (numéro de bloc dans la partition et numéro de ligne dans le bloc) correspondantes.

Tous les index oracle sont stockés sous forme d'arbres équilibrés (btree) : une structure arborescente permet de retrouver rapidement dans l'index la valeur de clé cherchée, et donc l'adresse de la ligne correspondante dans la table.

Dans un tel arbre, toutes les feuilles sont à la même profondeur, et donc la recherche prend approximativement le même temps quelle que soit la valeur de la clé.

Lorsqu'un bloc d'index est plein, il est éclaté en deux blocs ; en conséquence, tous les blocs d'index ont un taux de remplissage variant de 50% à 100%. Sans index on balaie séquentiellement toute la table quelle que soit la position de l'élément recherché.

Utilisation des index

L'adjonction d'un index à une table ralentit les mises à jour (insertion, suppression, modification de la clé) mais accélère beaucoup la recherche d'une ligne dans la table.

L'index accélère la recherche d'une ligne à partir d'une valeur donnée de clé, mais aussi la recherche des lignes ayant une valeur d'index supérieure ou inférieure à une valeur donnée, car les valeurs de clés sont triées dans l'index.

Exemple : Les requêtes suivantes bénéficieront d'un index sur le champ `n_dept`.

```
SELECT * FROM emp WHERE num = 16034  ;
SELECT * FROM emp WHERE num >= 27234 ;
SELECT * FROM emp WHERE num BETWEEN 16034  AND 27234;
```

Un index est utilisable même si le critère de recherche est constitué seulement du début de la clé.

Exemple : La requête suivante bénéficiera d'un index sur la colonne `nom`.

```
SELECT *
FROM emp
WHERE nom LIKE 'M%' ;
```

Par contre si le début de la clé n'est pas connu, l'index est inutilisable.

Exemple : La requête suivante ne bénéficiera pas d'un index sur le champ `nom`.

```
SELECT *
FROM emp
WHERE ename LIKE '?????????' ;
```

Valeurs `NULL`

Elles ne sont pas représentées dans l'index, ceci afin de minimiser le volume nécessaire pour stocker l'index. En contrepartie, l'index ne sera d'aucune utilité pour retrouver les valeurs `NULL` lorsque le critère de recherche est du type `IS NULL`.

Conversions

L'index n'est utilisable que si le critère de sélection est le contenu de la colonne indexée, sans aucune transformation. Par exemple un index sur `salaire` ne sera pas utilisé pour la requête suivante :

```
SELECT * FROM emp
WHERE salaire * 12  300000 ;
```

Attention en particulier aux conversions de type qui peuvent empêcher l'utilisation de l'index.

SQL est un langage typé, chaque type de données (numérique, caractère, date) ayant ses propres opérateurs, ses propres fonctions et sa propre relation d'ordre. En conséquence, si dans une expression, figurent à la fois un nombre et une chaîne de caractères, SQL convertira la chaîne de caractères en nombre. De même si dans une expression, figurent à la fois une chaîne de caractères et une date, SQL convertira la chaîne de caractères en date.

Or, dans un prédicat du type :

```
WHERE fonction(col_indexée) = constante
```

SQL ne peut pas utiliser l'index.

Ceci peut se produire , de façon insidieuse, lorsque SQL est obligé d'ajouter un appel à une fonction de conversion à cause d'une discordance de type.

Exemple : Le prédicat suivant ne bénéficiera pas d'un index sur le champ `embauche`.

```
SELECT * FROM emp
WHERE embauche LIKE '????' ;
```

En effet, SQL est obligé d'effectuer une conversion, et le prédicat qui sera évalué est :

```
WHERE TO_CHAR(embauche) LIKE '????'
```

Le critère de recherche est une fonction de `embauche`, et non le champ `embauche` lui-même, dans ce cas l'index est inutilisable.

Choix des index

Indexer en priorité :

1. les clés primaires
2. les colonnes servant de critère de jointure
3. les colonnes servant souvent de critère de recherche

Ne pas indexer :

1. les colonnes contenant peu de valeurs distinctes (index alors peu efficace)
2. les colonnes fréquemment modifiées

Créer un index

Un index peut être créé par la commande suivante :

```
CREATE [UNIQUE]  INDEX nom_index
ON nom_table (nom_col1 , nom_col2, ...)
[PCTFREE nombre]
[COMPRESS | NOCOMPRESS]
[ROWS = nombre_lignes] ;
```

dans laquelle :

- L'option `UNIQUE` indique que l'on interdit que deux lignes aient la même valeur dans la colonne indexée.
- `PCTFREE` précise le pourcentage de place laissée libre dans les blocs d'index à la création de l'index. Cette place libre évitera une réorganisation de l'index des les premières insertions de nouvelles clés. La valeur par défaut est 20% .
- `NOCOMPRESS` indique que l'on ne veut pas comprimer les clés.
- `nombre_lignes` est une estimation du nombre de lignes, permettant d'optimiser l'algorithme de classement..

Un index peut être créé dynamiquement sur une table contenant déjà des lignes. Il sera ensuite tenu à jour automatiquement lors des modifications de la table.

Un index peut porter sur plusieurs colonnes, la clé d'accès sera alors la concaténation des différentes colonnes.

On peut créer plusieurs index indépendants sur une même table.

Les requêtes SQL sont transparentes au fait qu'il existe un index ou non. C'est l'optimiseur du système de gestion de bases de données qui, au moment de l'exécution de chaque requête, recherche s'il peut s'aider ou non d'un index.

Index comprimé et non comprimé

Les clés dans les index peuvent être comprimées ou non. La compression est une technique permettant de réduire dans des proportions très importantes (d'autant plus que la clé est longue) le volume de l'index.

En contrepartie, il faut parfois un traitement supplémentaire pour recomposer la clé lors des mises à

jour de l'index.

Par défaut, les index sont comprimés, les avantages de réduction de taille l'emportant sur les inconvénients dans la plupart des cas.

sql sait exécuter certaines requêtes directement au niveau de l'index sans passer par le segment de données, si l'index est non comprimé et si tous les champs résultats de la requête sont dans l'index.

Exemple : L'index crée par :

```
CREATE INDEX x
ON emp (num, nom)
nocompress ;
```

permettra de répondre à la question :

```
SELECT nom
FROM emp
WHERE num
```

sans lire la table puisque toutes les informations se trouvent dans l'index et que l'index est non concaténé.

Index concaténé

Un index concaténé est un index portant sur plusieurs colonnes.

Exemple :

```
CREATE INDEX xemp
ON (n_dept,num) ;
```

Les index concaténés peuvent être utilisés pour matérialiser une clé composée de plusieurs colonnes.

SQL sait utiliser un index concaténé même si le critère de recherche ne porte pas sur toutes les colonnes présentes dans l'index.

Exemple : L'index ci-dessus est utilisable si l'on ne connaît que le numéro de département.

```
SELECT nom
FROM emp
WHERE n_dept = 20 ;
```

Supprimer un index

Un index peut être supprimé dynamiquement par la commande :

```
DROP  INDEX nom_index ;
```

L'espace libéré reste attaché au segment d'index de la table : il pourra être utilisé pour un autre index sur la même table.

L'espace ne sera rendu à la partition que lors de la suppression de la table.

Les clusters

Introduction - Généralités

Définition : 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 :

- accélérer la jointure selon la clé de cluster des tables mises en cluster,
- accélérer la sélection des lignes d'une table ayant même valeur de clé, par le fait que ces lignes sont regroupées physiquement,
- économiser de la place, du fait que chaque valeur de la clé du cluster ne sera stockée qu'une seule fois.

Le regroupement en cluster est totalement transparent à l'utilisateur : des tables mises en cluster sont toujours vues comme des tables indépendantes.

Par exemple on pourrait mettre en cluster les tables `emp` et `dept` selon `n_dept`. Ces tables seraient réorganisées de la façon suivante : un bloc de cluster serait créé pour chaque numéro de département, ce bloc contenant à la fois les lignes de la table `empet` de la table `dept` correspondant à ce numéro de département. La jointure entre les tables `emp` et `dept` selon `n_dept` deviendrait alors beaucoup plus rapide, puisqu'elle serait déjà réalisée dans l'organisation physique des tables.

Pour que l'on puisse mettre une table en cluster il faut que l'une au moins des colonnes faisant partie du cluster soit définie comme obligatoire (`NOT NULL`).

On peut indexer les colonnes d'une table en cluster, y compris les colonnes correspondant à la clé ou à une partie de la clé du cluster. La clé elle-même est automatiquement indexée, on peut éventuellement la réindexer pour créer un index unique servant à contrôler son unicité.

Créer un cluster

Avant de pouvoir mettre en cluster une ou plusieurs tables il faut créer le cluster au moyen de la commande `CREATE CLUSTER` dont la syntaxe est la suivante :

```
CREATE CLUSTER nom_cluster
       (cle1 type1,
        cle2 type2,
        ...)
```

où l'on donne un nom au cluster, et où l'on définit le nom et le type des colonnes constituant la clé du cluster.

```
CREATE CLUSTER nom_cluster
       (cle1 type1,
        cle2 type2,
        ...)
[SIZE taille_du_bloc]
[COMPRESS | NOCOMPRESS]
[SPACE nom_de_space_definition]
```

dans laquelle :

`SIZE`

est la taille d'un bloc de cluster. Cette taille peut varier de 1/6 de bloc oracle à 1 bloc oracle (2k octets sur vax/vms), ce paramètre doit être choisi de façon à avoir un bon remplissage des blocs.

`COMPRESS|NOCOMPRESS`

est relatif à l'index qui sera crée sur la clé du cluster

`SPACE`

spécifie le `SPACE DEFINITION` qui définira les paramètres d'allocation d'espace pour le cluster.

```
CREATE CLUSTER DEM
(DEPNO NUMBER)
SIZE 512;
```

Mise en cluster d'une table

En principe c'est dès sa création qu'il faut spécifier si une table sera implantée dans un cluster.

Lors de la création de la table

L'option cluster de l'ordre `CREATE TABLE` permet de spécifier que la table doit être mise en cluster. Le cluster doit déjà exister.

```
CREATE TABLE nom_table
(nom_col1 TYPE1 NOT NULL,
(nom_col2 TYPE2 NOT NULL,
                              ...)
CLUSTER NOM_CLUSTER  (nom_coli, nom_colj...)
```

`nom_coli, nom_colj` sont des noms de colonnes de la table, elles seront identifiées une à une aux colonnes clés du cluster spécifiées à la création du cluster.

Table déjà existante

En principe cela n'est pas possible, il faut donc procéder de la façon suivante :

- créer une nouvelle table avec l'option cluster et y copier le contenu de la table initiale ;
- supprimer l'ancienne table ;
- renommer éventuellement la nouvelle table.

Retrait d'une table d'un cluster

Pour retirer une table d'un cluster il faut :

- créer une nouvelle table en dehors du cluster et copier la table en cluster dans la nouvelle table ;
- détruire l'ancienne table ;
- la nouvelle table pourra alors être renommée pour prendre le nom de l'ancienne.

ceci ne détruit pas la table, mais la reconstruit en dehors du cluster.

Supprimer un cluster

Un cluster ne contenant aucune table peut être supprimé par la commande :

```
DROP CLUSTER nom_cluster ;
```

Remarque : Les performances du cluster ne sont valables que si on n'a pas de blocs chaînés (ex de grande table).

Contrôle des accès à la base et sécurité des données

place du nom d'utilisateur.

```
GRANT SELECT, UPDATE ON emp TO PUBLIC;
```

Un utilisateur ayant accordé un privilège peut le reprendre à l'aide de l'ordre `REVOKE`.

```
REVOKE PRIVILEGE ON {nom_table | nom_vue} FROM  nom_utilisateur;
```

Droits d'accès aux vues

Le propriétaire d'une table peut donner des droits de lecture non pas sur la table entière, mais sur une vue basée sur la table. Ainsi, seules les informations faisant partie de la vue seront accessibles.

```
CREATE VIEW emp1 AS
SELECT nom, fonction, embauche
FROM scott.emp
WHERE n_dept != 10 ;

GRANT SELECT ON emp1 TO PUBLIC;
```

Tous les utilisateurs pourront sélectionner les employés à travers la vue `emp1` : ils ne verront que les colonnes `nom`, `fonction`, `embauche` de la table `emp` et n'auront pas accès aux employés du département 10. Avec l'option de contrôle (`CHECK OPTION`), les vues peuvent servir à réaliser les contrôles lors des mises à jour.

Exemple : La vue suivante ne permettrait pas d'insérer dans la table des employés `emp` un employé dont le numéro de département ne figurerait pas dans la table des départements `dept` :

```
CREATE VIEW majemp AS
SELECT *
FROM emp
WHERE n_dept IN (SELECT n_dept
            FROM dept)
WITH CHECK OPTION ;
```

Comme il est impossible de faire une mise à jour sur une vue comportant une jointure il faut transformer le critère de jointure en une sous-interrogation.

Pour oracle , le nom complet d'une table ou d'une vue est le nom donné par le créateur, préfixé par le nom du créateur. Par exemple `scott.emp` est le nom complet de la table `emp` créée par l'utilisateur `scott` . Ceci permet à plusieurs utilisateurs de créer des objets de même nom sans qu'il y ait confusion. En revanche, pour accéder à un objet dont on n'est pas le créateur, il faut le désigner par son nom complet incluant le nom du créateur.

Gestion des transactions

Définition

Une transaction est un ensemble de modifications de la base qui forme un tout indivisible. Il faut effectuer ces modifications entièrement ou pas du tout, sous peine de laisser la base dans un état incohérent.

Les Systèmes de Gestion de Bases de Données permettent aux utilisateurs de gérer leurs transactions. Ils peuvent à tout moment :

- Valider la transaction en cours par la commande `COMMIT`. Les modifications deviennent définitives et visibles à tous les utilisateurs.
- Annuler la transaction en cours par la commande `ROLLBACK`. Toutes les modifications depuis le début de la transaction sont alors défaites.

En cours de transaction, seul l'utilisateur ayant effectué les modifications les voit.

Ce mécanisme est utilisé par les systèmes de gestion de bases de données pour assurer l'intégrité de la base en cas de fin anormale d'une tâche utilisateur : il y a automatiquement `ROLLBACK` des transactions non terminées.

ORACLE est un système transactionnel qui assure la cohérence des données en cas de mise à jour de la base, même si plusieurs utilisateurs lisent ou modifient les mêmes données simultanément.

ORACLE utilise un mécanisme de verrouillage pour empêcher deux utilisateurs d'effectuer des transactions incompatibles et régler les problèmes pouvant survenir.

ORACLE permet le verrouillage de certaines unités (table ou ligne) automatiquement ou sur demande de l'utilisateur.
Les verrous sont libérés en fin de transaction.

Cohérence d'une interrogation

Un utilisateur qui interroge une table (même très grande) est garanti de voir toutes les données telles qu'elles étaient au moment du début de l'interrogation, même si d'autres utilisateurs modifient la table et valident leurs modifications pendant ce temps.

Les sgbd dont oracle utilisent alors le fichier image avant pour assurer cette cohérence. Le `COMMIT` des utilisateurs modifiant la table n'est pas différé à la fin de l'interrogation.

Remarque : Dès que l'on interroge une table, un verrou est placé sur la définition de la table, c'est à dire qu'un autre utilisateur ne peut pas détruire la table, l'indexer, la mettre en cluster ou modifier sa définition, jusqu'à ce que l'interrogation soit terminée.

Cohérence de plusieurs interrogations successives

Si l'utilisateur désire que l'on ne modifie pas une table pendant une session de travail, celui-ci peut vérouiller la table en mode partagé au moyen de l'ordre sql suivant :

```
LOCK TABLE  nom_table IN SHARE MODE NOWAIT;
```

où l'option `NOWAIT`, qui peut s'adjoindre à toutes les commandes de verrouillage, spécifie que le process qui demande le verrou n'est pas mis en attente si celui-ci n'est pas disponible.

La table n'est alors accessible aux autres utilisateurs qu'en lecture jusqu'à la fin de la transaction de celui qui l'a verrouillée (les autres utilisateurs peuvent aussi verrouiller la table en share mode).

Les modifications des autres utilisateurs seront suspendues.

Mise à jour

Pour s'assurer l'accès exclusif en modification à une table, l'on peut verrouiller cette table en mode exclusif par la commande :

```
LOCK TABLE  nom_table IN EXCLUSIVE MODE NOWAIT;
```

La table n'est alors accessible aux autres utilisateurs qu'en lecture et ils ne peuvent plus la verrouiller en mode exclusif, ni en mode mise à jour partagée, ni en mode partagé jusqu'à la fin de la transaction. Ce verrou est également obtenu automatiquement dès que l'on effectue un `UPDATE`, `INSERT` ou `DELETE` sur une table. C'est le mode par défaut de mise à jour d'une table.

SQL : Dictionnaire de données

Tous les systèmes de gestion de bases de données relationnels contiennent un dictionnaire de données intégré. C'est un **ensemble de tables et de vues** dans lesquelles sont stockées les descriptions des objets de la base, et qui sont tenues à jour automatiquement par le système de gestion de bases de données.

Ces tables ou vues, comme toutes les tables ou vues, peuvent être consultées au moyen du langage SQL.

Description du dictionnaire des données

La vue DICTIONARY contient la description des tables et vues du dictionnaire de données, le contenu de cette vue varie en fonction du type d'utilisateur qui l'interroge (dba ou non). Les tables et vues du dictionnaire de données ont chacune un nom préfixé par :

User pour les vues contenant la description des objets appartenant à l'utilisateur qui interroge le dictionnaire.

DBA pour les vues accessibles uniquement aux utilisateurs ayant le privilège dba.

ALL pour les vues contenant la description de tous les objets auxquels a accès l'utilisateur qui interroge le dictionnaire. Ces vues contiennent non seulement les objets créés par l'utilisateur mais aussi ceux pour lesquels on lui a explicitement donné des droits d'accès ainsi que ceux qui sont accessibles a tout le monde (accès PUBLIC).

Ce chapitre contient la description d'un sous-ensemble des vues du dictionnaire de données.

Vues décrivant les objets de l'utilisateur

USER_CATALOG

Liste des objets appartenant à l'utilisateur.

USER_CLUSTERS

Description des clusters créés par l'utilisateur.

USER_COL_COMMENTS

Commentaires sur les colonnes des tables et des vues créées par l'utilisateur.

USER_INDEXES

Description des index créés par l'utilisateur.

USER_IND_COLUMNS

Liste des colonnes indexées par l'utilisateur courant ou indexant les tables de l'utilisateur courant.

USER_OBJECTS

Description des objets appartenant à l'utilisateur.

USER_SYNONYMS

Liste de synonymes créés par l'utilisateur.

USER_TABLES

Description des tables créées par l'utilisateur.

USER_TAB_COLUMNS

Description des colonnes de chaque table ou vue créée par l'utilisateur courant. Chaque ligne de la vue col décrit une colonne.

USER_TAB_COMMENTS

Commentaires sur les tables et les vues créées par l'utilisateur.

USER_TAB_GRANTS

Droits sur les objets donnés par l'utilisateur ou dont l'utilisateur est le bénéficiaire.

USER_TAB_GRANTS_MADE

Droits sur ses objets, tables ou vues, donnés par l'utilisateur.

USER_TAB_GRANTS_RECD

Droits sur des objets donnés à l'utilisateur.

USER_USERS

Informations sur l'utilisateur courant.

USER_VIEWS

Texte des vues créées par l'utilisateur.

Vues décrivant les objets auxquels l'utilisateur a accès

ALL_CATALOG

Liste de tous les objets accessibles par l'utilisateur.

ALL_COL_COMMENTS

Commentaires sur les colonnes des tables et des vues accessibles par l'utilisateur.

ALL_INDEXES

Description des index accessibles par l'utilisateur.

ALL_IND_COLUMNS

Liste des colonnes indexées appartenant aux tables accessibles par l'utilisateur.

ALL_OBJECTS

Description des objets accessibles par l'utilisateur.

ALL_SYNONYMS

Liste de tous les synonymes accessibles par l'utilisateur.

ALL_TABLES

Description des tables accessibles par l'utilisateur.

ALL_TAB_COLUMNS

Description de toutes les colonnes des tables ou vues accessibles par l'utilisateur.

ALL_TAB_COMMENTS

Commentaires sur les tables et les vues accessibles par l'utilisateur.

ALL_TAB_GRANTS

Droits sur les objets donnés par l'utilisateur ou dont l'utilisateur est le bénéficiaire.

ALL_TAB_GRANTS_MADE

Droits sur ses objets, tables ou vues, donnés par l'utilisateur.

ALL_TAB_GRANTS_RECD

Droits sur des objets donnés à l'utilisateur.

ALL_USERS

Informations sur tous les utilisateurs.

ALL_VIEWS

Synonymes

Les noms des vues étant assez longs, les synonymes suivants ont étés définis :

CAT	Synonyme pour USER_CATALOG
CLU	Synonyme pour USER_CLUSTERS
COLS	Synonyme pour USER_TAB_COLUMNS
DICT	Synonyme pour DICTIONARY
IND	Synonyme pour USER_INDEXES
MYPRIV	Synonyme pour USER_USERS
OBJ	Synonyme pour USER_OBJECTS
SEQ	Synonyme pour USER_SEQUENCES
SYN	Synonyme pour USER_SYNONYMS
TABS	Synonyme pour USER_TABLES

Références

Alfred V. Aho and Jeffrey D. Ullman. *Foundations of Computer science*. Computer Science Press, 1982.

M. Bouzeghoub, M. Jouve, and P. Pucheral. *Systèmes de Bases de Données : des techniques d'implantation à la conception de schémas*. Eyrolles, 1990.

G. Gardarin and P. Valduriez. *Bases de Données relationnelles : analyse et comparaison des systèmes*. Eyrolles, 1985.

Georges Gardarin. *Bases de Données : Les systèmes et leurs langages*. Eyrolles, 1982.

P. C. Kanellakis. Elements of relational database theory. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science Volume B : Formal model and semantics*. Elsevier, the MIT press, 1990.