2022

# Les fondamentaux SQL

## BASE DE DONNÉES

DIDIER TEXIER

DTX

Version 08-2022 - 10385

# Les fondamentaux SQL


1 Convention d'écriture ..... 9
I. Introduction ..... 10
I.1 Une base de données relationnelle c'est quoi ? ..... 10
I.2 Environnement de travail ..... 12
I.3 Le dictionnaire de données ..... 12
2 Le modèle relationnel ..... 14
2.1 Définitions ..... 14
2.2 Le Modèle Conceptuel de Données (MCD) ..... 15
2.3 Exemples de tables ..... 17
3 SQL, les différents langages ..... 18
4 L'ordre SELECT ..... 19
4.1 Syntaxe ..... 19
4.2 L'opérateur concat ( || ) ..... 19
5 Restreindre et trier les données ..... 21
5.1 L'opérateur WHERE ..... 21
5.2 Variable de substitution ..... 22
5.3 La clause ORDER BY ..... 23
5.3.1 Cas de l'absence de valeur (NULL) ..... 23
1.1 Variable de substitution ..... 23
6 Les jointures ..... 25
6.1 Les jointures ..... 25
6.1.1 Alias de table ou synonyme local ..... 26
6.1.2 Auto-Jointure ..... 27
6.1.3 Les jointures externes ..... 27
7 Les fonctions de groupe ..... 29
7.1.1 Regroupement de lignes : clause GROUP BY ..... 29
7.1.2 Restrictions sur groupes : Clause HAVING ..... 30
7.1.3 Les expressions ROLLUP et CUBE ..... 30
7.1.4 La fonction analytique LISTAGG ..... 30
8 Les sous-interrogations ..... 32
8.1 Sous-interrogations mono-ligne ..... 32
8.2 Sous-interrogations multilignes ..... 32
8.2.1 Les opérateurs de comparaison multilignes ..... 33
8.2.2 L'opérateur existentiel EXISTS ..... 33
8.3 Les requêtes corrélées ..... 34
9 Opérateurs ensemblistes ..... 35

10 MANIPULATIONS DES DONNÉES ..... 37
10.1 AJOUT DE LIGNE : L'ORDRE INSERT ..... 37
10.2 COPIER DES LIGNES D'UNE AUTRE TABLE ..... 37
10.3 MISE À JOUR DE LIGNES : L'ORDRE UPDATE ..... 37
10.4 SUPPRESSION DE LIGNES : L'ORDRE DELETE ..... 38
10.5 L'ORDRE TRUNCATE ..... 38
10.6 INSERTION MULTI-TABLES ..... 38
I.4 FUSION DE LIGNES : L'INSTRUCTION MERGE ..... 40
I.4.1 SYNTAXE ..... 40
11 LES ORDRES LDD ..... 42
11.1 LE CREATE TABLE ..... 42
11.1.1 LES VUES DU DICTIONNAIRE ..... 43
11.1.2 PRIVILÈGES ..... 43
11.2 LES TYPES DE DONNÉES ..... 43
11.3 LES CONTRAINTES ..... 44
11.4 MODIFICATION DE LA STRUCTURE D'UNE TABLE : ALTER TABLE ..... 44
11.5 LES COMMENTAIRES ..... 45
11.6 LES VUES ..... 46
11.6.1 CRÉATION ..... 46
11.6.2 UTILISATION DES VUES ..... 46
11.6.3 SUPPRESSION D'UNE VUE ..... 47
11.7 LES SYNONYMES ..... 47
11.7.1 SYNTAXE ..... 47
11.7.2 PRIVILÈGES ..... 47
11.7.3 SUPPRESSION DES SYNONYMES ..... 47
11.7.4 VUES DU DICTIONNAIRE ..... 47
11.8 LES SÉQUENCES ..... 47
11.8.1 SYNTAXE ..... 48
11.8.2 PRIVILÈGES ..... 48
11.8.3 UTILISATION ..... 48
11.8.4 MODIFICATION ..... 48
11.8.5 SUPPRESSION ..... 48
11.8.6 VUES DU DICTIONNAIRE ..... 49
11.9 LES INDEX ..... 49
11.9.1 INDEX B-TREE ..... 49
12 ANNEXES ..... 51
12.1 FONCTIONS PREDEFINIES ..... 51
12.1.1 FONCTIONS NUMERIQUES ..... 51
12.1.2 FONCTIONS SUR CHAINES ..... 52

12.1.3 FONCTIONS SUR DATE ..... 54
12.1.4 FONCTIONS DE CONVERSION ..... 55
12.1.5 FONCTIONS PARTICULIERES ..... 55
12.1.6 FONCTIONS RELATIVES À LA VALEUR *NULL*. ..... 55
12.1.7 LES FONCTIONS DE GROUPE ..... 56
12.2 MODÈLE HR ..... 57
12.4 COMMANDES SQL*PLUS ..... 58
*12.4.1* LE FICHIER *LOGIN.SQL* ..... 58
12.5 SQLDEVELOPER ..... 60
13 SQLCL ..... 61
13.1 QUELQUES COMMANDES ..... 61
13.2 QUELQUES EXEMPLES DE FORMAT : ..... 61
13.3 EASTER EGGS ..... 61
14 BIBLIOGRAPHIE / SITOGRAPHIE ..... 62
II. LES PRIVILÈGES ..... 63
III. VUES DU DICTIONNAIRE ..... 64
IV. ACRONYMES ..... 65
V. INDEX ..... 66

*Toute technologie suffisamment avancée est indiscernable de la magie.*

*Arthur C. Clarke*

# AVANT PROPOS

Ce support de cours est un outil personnel, il ne constitue pas un guide de référence.

C'est un outil pédagogique élaboré dans un souci de concision : il décrit les actions essentielles à connaître pour appréhender le sujet de la formation.

**Objectifs pédagogiques :**

- Manipuler de l'information à l'aide des différentes opérations de l'algèbre relationnelle
- Appliquer les opérations de l'algèbre relationnelle aux requêtes SQL
- Composer des requêtes SQL simples
- Composer des requêtes SQL avancées

**Prérequis :** Être familier avec les concepts basiques des langages de programmation :

- Les fonctions
- Les types de variables (entier, décimal, réel, chaîne de caractères, booléen, date)
- Les opérateurs logiques (ET, OU, NON)
- Les booléens TRUE et FALSE

Si vous ne les connaissez pas, ce n'est pas encore rédhibitoire.

*Bien que le langage SQL soit normalisé, il existe, néanmoins des différences entre les différents systèmes de gestion de base de données relationnelle (SGBDR).*
*Les exemples fournis dans ce support seront écrits avec la syntaxe SQL Oracle.*

# 1 Convention d'écriture

La police `courier` est utilisée pour les exemples de commandes SQL :

```
SELECT last_name FROM Employees ;
```

Les MAJUSCULES son utilisées pour les mots clé SQL.

Les minuscules sont utilisées pour les noms des colonnes et le nom des tables seront écrits avec l'initiale en Majuscule.

Les termes Oracle sont en *italiques*.

Dans la syntaxe d'une instruction :

**SELECT** [**DISTINCT**] {* | NomCol1 | ExprSQL [AS etiq1] [, Nomcol2 | ExprSQL [etiq2] … ] }
**FROM** NomTable ;

Les symboles suivants définissent :

- [ ] le caractère optionnel d'une directive
- { } une liste d'éléments alternatifs
- | le choix possible parmi une liste d'éléments
- AS (souligné) un terme par défaut

# I. Introduction

Une base de données (BD) est en ensemble **structuré** d'informations.

## I.1 Une base de données relationnelle c'est quoi ?

> ***Une base de données relationnelle est un ensemble d'informations dont l'organisation respecte des règles précises et qui peut être interrogée par le langage SQL.***

À l'origine était ***System R***, système de gestion de base de données développé par IBM dans les années 1970.

Edgar Frank « Ted » CODD[1], informaticien mathématicien d'origine britannique et chercheur chez IBM, pose les fondements du modèle relationnel.

Le langage utilisé alors pour l'interrogation de la base de données était le ***SEQUEL*** *(*Structured English QUEry Language) qui deviendra SQL en 1975.

SQL (*Structured Query Language*, en français langage de requête structurée) est un langage informatique normalisé par l'ANSI (*American National Standards Institute*) servant à effectuer des opérations sur des bases de données relationnelles.

C'est en 1979, qu'Oracle (à l'époque *Relational Software Inc.*) commercialise la première version de SQL.

La première norme SQL date de 1987 et la dernière norme aujourd'hui est **SQL:2011** (SQL ISO/CEI 9075:2011).

Un SGBD/R est un logiciel permettant la gestion des données en utilisant le modèle relationnel.

Ce logiciel est généralement complété par un ensemble d'outils de conception, de développent, d'administration, des pré-compilateurs, des outils de génération de formulaires et d'états et d'analyse des données.

Le SGBD permet aujourd'hui de stocker des données non structurées du type Large Object (LOB).

Plusieurs acteurs sont présents sur ce marché[2] :

- Oracle
- IBM : DB2
- Microsoft : SQL server
- Mysql / MariaDB
- PostGres
- FireBird
- Berkeley DB
- Informix

[1] 1923-2003

[2] Le site https://db-engines.com/ propose un classement mensuel des SGBDR.

- Teradata
- Sybase

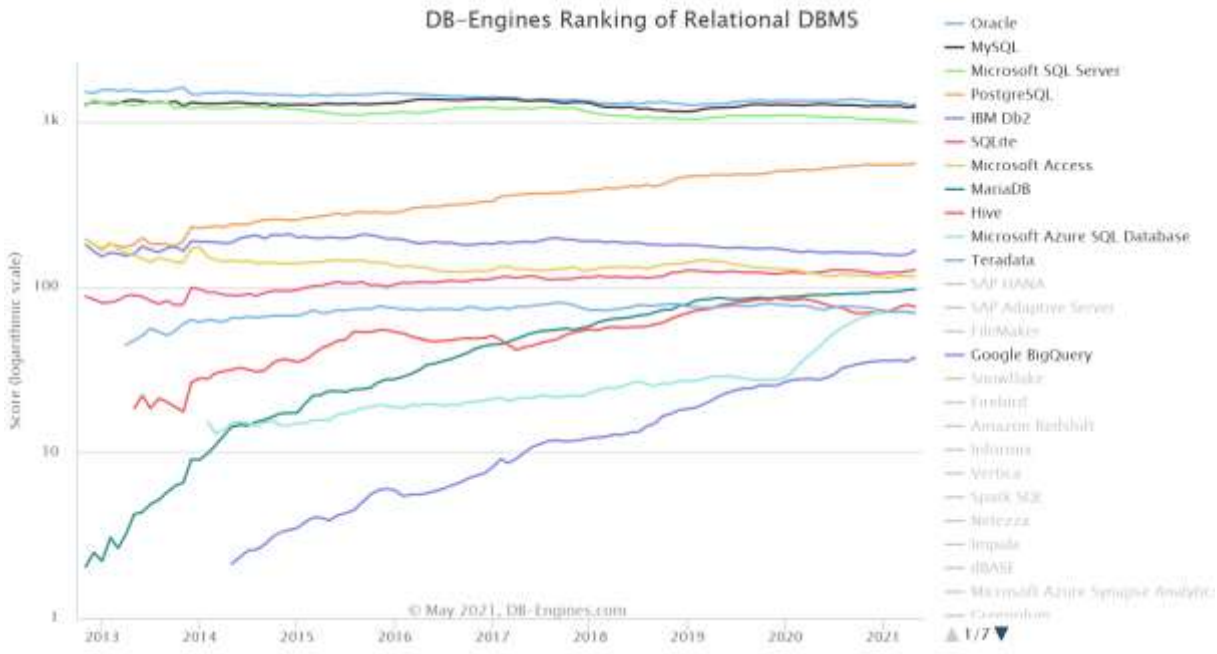


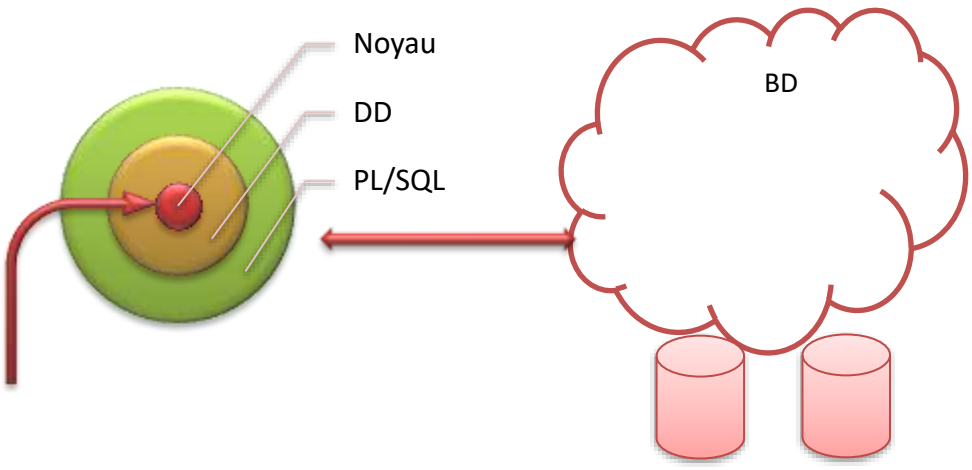

**Figure 1 : Classement des moteurs de bases de données relationnelles**

Aujourd'hui, le SQL est un langage essentiel des *Data Analysts*.

Le SGBD doit permettre :

- De concevoir et de mettre à jour le Modèle Physique de Données (MPD)
- De créer, modifier, supprimer des données de la base.
- Assurer l'indépendance entre données et traitements,
- Tout en garantissant l'ensembles des contraintes et la logique interne du modèle de données
- Contrôler l'accès aux données (identification, autorisations, concurrence des accès, …)
- Garantir les meilleures performances
- …

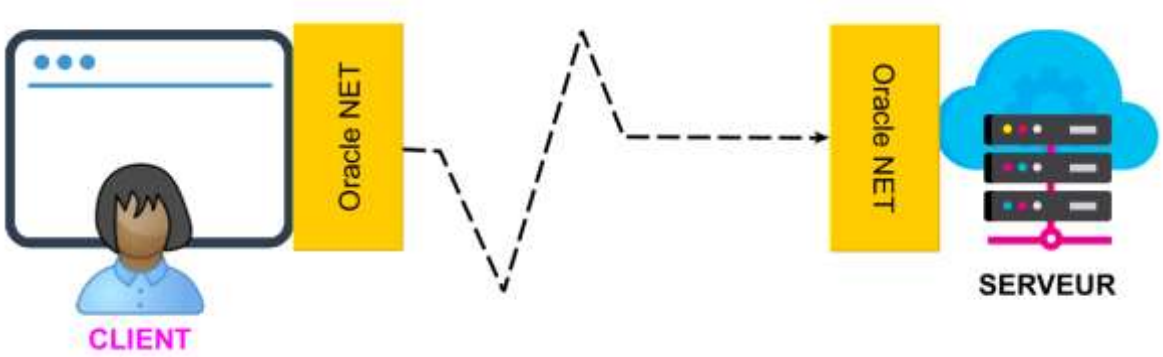
Schéma du SG BD / R Oracle



Rôle du noyau :

- Centralisation des accès
- Mise à disposition
- Mise en partage
- Concurrence des accès
- Gestion de la sécurité
- Identification / Authentification
- Autorisations d'accès
- Intégrité de la base $\begin{cases} logique\ ou\ interne \\ physique\ ou\ externe\ (journalisation) \end{cases}$
- Maintien du Dictionnaire de Données (DD)
- Optimisation des plans d'exécution

## I.2 Environnement de travail




L'application cliente, SQL*Plus par exemple, transmet une requête au serveur.

Pour établir la connexion réseau, le client devra renseigner plusieurs informations au serveur :

- Nom du serveur (ou son adresse IP)
- Port réseau
- Protocol
- Service

L'ensemble des informations transitera à travers la couche réseau d'Oracle nommé ORACLE NET.

Après l'exécution de la requête par le serveur, il enverra le résultat éventuel au client à sa demande.

## I.3 Le dictionnaire de données

C'est le méta-modèle : un ensemble de tables dynamiques décrivant l'ensemble de la structure de la base de données :

- Tables, colonnes, vues, index, synonymes, …
- Utilisateurs,
- Droits et privilèges,
- Programmes stockés,

- etc ...

Les utilisateurs s'intéresseront à leurs données tandis que les exploitants et DBA (*DataBase Administrator*) s'intéresseront au DD.

Le contenu du DD sert à décrire les structures du modèle application.

SQL est un langage :

- Non procédural
- Déclaratif, permet de décrire le résultat attendu sans décrire le moyen de l'obtenir.
- Non conçus pour une logique de traitement (*if, for, …* )

Dans certains cas, SQL pourra faire appel à un langage externe « EMBEDDED SQL »

Après une demande du DoD (*Depatment Of Defense*) à Oracle, ce dernier développera le langage PL/SQL inspiré d'ADA[3] lui-même inspiré du langage *Pascal*.

Les interfaces d'accès au noyau :

- SQL*Plus (ouvert, peu convivial)
- SQL*Developer
- J Developer
- …

[3] Langage de programmation développé par le français Jean Ichbiah à la demande du département de la Défense des États-Unis (DOD). Le nom "Ada" a été choisi en l'honneur d'Ada Lovelace (°10/12/1815 †27/11/1852) considérée comme la première informaticienne.

# 2 Le modèle relationnel

Le modèle relationnel est encore appelé :

Entité-Relation
Entité-Association
Objet-Relation

## 2.1 Définitions

- Caractéristiques :
  Simple et naturel : compris par tous.
  Rigoureux et non ambigu.

- Type :
  Concept abstrait générique exprimant l'essence d'une classe d'objet.

- Domaine :
  Ensemble, fini ou infini, des valeurs possibles que peut prendre un attribut.

- Propriété:
  Plus petit élément d'information manipulé par le système d'information **qui a un sens pour l'organisme**.

    - Simple ou composée
      *Exemples : le nom d'un salarié (simple) et son adresse (composée).*

- Occurrence :
  Élément individualisé et appartenant à un type.

- Individu :
  Ou entité est un regroupement de propriétés.

- Clé candidate :
  Ensemble minimal de propriétés d'un individu, telle qu'il n'existe pas deux occurrences de cet individu pour lesquelles cette propriété ou cet ensemble de propriétés puisse prendre une même valeur. Une clé candidate peut être un attribut artificiel utilisé à cette fin.

- Degré :
  Nombre d'attributs d'une relation (entité) et donc, nombre de colonnes d'une table.

## 2.2 Le Modèle Conceptuel de Données (MCD)

Le modèle conceptuel de données est une représentation graphique des informations manipulées par le système d'information (SI).

On partira du **dictionnaire de données** et des **règles de gestion** pour en faire une représentation schématique indépendante de l'organisation et des solutions techniques.

Après un recueil rigoureux d'information sur le terrain auprès des utilisateurs et bien sûr, des nouvelles demandes ; on construira le dictionnaire des données.

Typiquement, ce dictionnaire contient :

- Les données retenues non redondantes
- Leur type de données
- Des commentaires

*Exemple : le nom, prénom, adresse, salaire d'un employé. Pour les 3 premières de type alphanumérique et le salaire de type numérique.*

| Données | Type | Commentaire | Synonymes |
|---|---|---|---|
| **Nom employé** | Alphabétique | Le nom d'un employé | Patronyme |
| **Prénom employé** | Alphabétique | Le prénom d'un employé | |
| **Adresse** | Alphanumérique | L'adresse postale d'un employé | |
| **Salaire employé** | Numérique | Le salaire annuel brut d'un employé | |

*Exemple : le nom, prénom, adresse, salaire d'un employé. Pour les 3 premières de type alphanumérique et le salaire de type numérique.*

Il « suffira » de dégager à partir de ces informations (données et règles) des **ENTITÉS** et des **RELATIONS** entre-elles.

| EMPLOYE |
|---|
| Matricule |
| Nom |
| Prénom |
| Adresse |
| Salaire |

| DEPARTEMENT |
|---|
| Id_dept |
| Nom_dept |

← Nom de l'entité

← Propriétés

L'entité est un objet composé de propriétés dont l'une sera son identifiant. Ce sont des objets de gestion comme par exemple : un article, un client, un employé, une facture, …

L'identifiant est une ou plusieurs propriétés permettant d'identifier de manière unique une occurrence de l'objet. L'identifiant sera représenté souligné et apparaitra en premier dans la liste des propriétés.

*Exemple : un matricule, un numéro de facture, un code service, …*

Puis, on déterminera les relations entre les entités. Une relation exprime une association entre des entités. Ces relations seront :

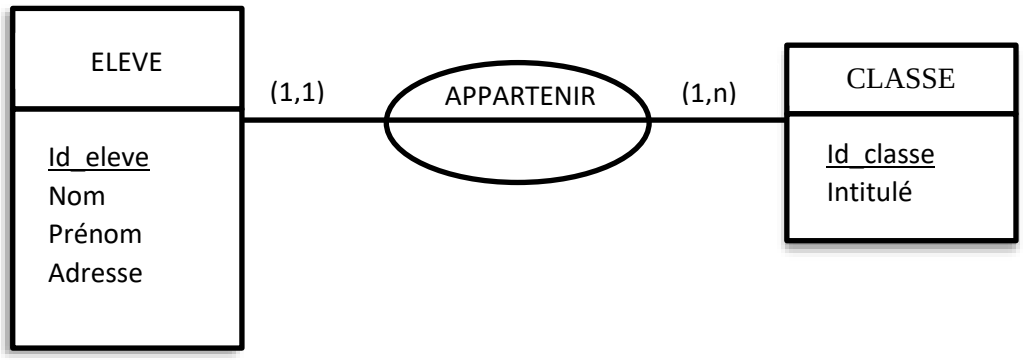
- Définies par un verbe
- De dimension binaire, ternaire ou n-aire
- Porteuses d'information (auront leurs propres propriétés)
- Renseignées par des cardinalités

La **dimension** d'une relation définit le nombre d'entité qui participe à cette relation.

La **cardinalité** exprime le nombre d'occurrences d'un objet qui participe à la relation. On s'intéressera uniquement aux cardinalités minimales et maximales.

- 0,1 : une occurrence de l'objet participe au plus une fois à la relation.
- 1,1 : une occurrence de l'objet participe une fois et une seule à la relation.
- 1,n : une occurrence de l'objet participe au moins une fois à plusieurs à la relation.
- 0,n : aucune précision quant à la participation des occurrences de l'objet.

*Exemple :*



## 2.3 Exemples de tables

| Nom | 1er trimestre | 2ème trimestre | 3ème trimestre |
|---|---|---|---|
| Boltzmann | 12 | 8 | 15 |
| De Broglie | 7 | 14 | 18 |
| Dirac | 8 | 13 | 16 |
| Einstein | 9 | 12 | 17 |
| Kaluza | 15 | 17 | 16 |

| Unité | Laboratoire | Directeur |
|---|---|---|
| UFR 12 | Quantique | Planck |
| UFR 12 | Nucléaire | Curie |
| UFR 16 | Optique | Bohr |
| UFR 7 | Atomique | Minkowski |
| UFR 8 | Mathématique | Klein |

| Unité | Ville |
|---|---|
| UFR 12 | Kiel |
| UFR 16 | Copenhague |
| UFR 7 | Alexotas |
| UFR 8 | Stockholm |
| | |

| Id_emp | Nom | Salaire | Ville |
|---|---|---|---|
| 10 | Boltzmann | 32000 | Kiel |
| 8 | De Broglie | 37000 | Kiel |
| 5 | Dirac | 42000 | Copenhague |
| 23 | Einstein | 41000 | Alexotas |
| 14 | Kaluza | 39000 | Stockholm |

# 3 SQL, les différents langages

Le langage SQL[4] sera utilisé pour interroger la base de données ou pour effectuer des opérations sur les objets des bases de données relationnelles.

SQL est un langage :

- Non procédural
- Assertionnel ou déclaratif, permet de décrire le résultat attendu sans décrire le moyen de l'obtenir.
- Non conçus pour une logique de traitement (if, for, … )

Il est généralement diviser en sous-langages composé d'instructions pour effectuer des tâches différentes.

| | |
|---|---|
| Langage de manipulation de données (DML) | SELECT<br>INSERT<br>UPDATE<br>DELETE<br>MERGE |
| **Langage de définition de données (DDL)** | CREATE<br>ALTER<br>DROP<br>RENAME<br>TRUNCATE<br>COMMENT |
| Langage de contrôle de données (DCL) | GRANT<br>REVOKE |
| **Langage de contrôle des transactions (TCL)** | COMMIT<br>ROLLBACK<br>SAVEPOINT |

[4] Donald D. Chamberlin (1944) est le co-inventeur avec Raymond Boyce du langage SQL

# 4 L'ordre SELECT

La commande SELECT permet d'extraire, des tables de la base, un ensemble de lignes.

Dans sa version triviale, cette commande effectue ce qu'on appelle une ***projection*** en algèbre relationnelle.

La projection permet d'extraire un sous-ensemble de la table en ne sélectionnant qu'une partie des attributs de l'entité (ou la relation). Cette opération porte sur une seule relation (ou table).

A donc pour effet de réduire la table aux seules colonnes énoncées par la projection.

Elle s'écrit : $T = \sigma_{cond}(R)$

L'écriture de cette commande peut être très simple à très complexe et peut faire appel à des niveaux d'imbrications multiples.

## 4.1 Syntaxe

La syntaxe la plus simple est la suivante :

```
SELECT  [DISTINCT] {* |  NomCol1 | ExprSQL [AS etiq1] [,Nomcol2 |
ExprSQL [etiq2] … ] }
FROM NomTable ;
```

*ExprSQL* ::= < opérande1 [opérateur opérande] >

Opérateur ::= < + - / * || >

Opérande ::= < NomCol | Constante | FonctionSQL | ExprSQL >

La clause **DISTINCT** permet de supprimer les doublons à l'affichage. Elle porte sur l'ensemble des colonnes de la clause SELECT.

La clause DISTINCT provoque un tri implicite à utiliser avec précautions pour des raisons de performances.

```
SELECT DISTINCT job_id
FROM employees;
```

L'exemple suivant illustre l'opérateur de projection :

```
SELECT last_name, salary, commission_pct,
     salary* commission_pct AS "Montant Commission"
FROM employees;
```

## 4.2 L'opérateur concat ( || )

L'opérateur « || » (double barre verticale) est une écriture simplifiée de la fonction SQL `concat()`

```
SELECT 'Le salaire de ' || last_name || ' est de ' || salary || '€'
     AS phrase
FROM employees;
```

# 5 Restreindre et trier les données

## 5.1 L'opérateur WHERE

La clause **WHERE** (ou l'opérateur WHERE) permet de restreindre le nombre de lignes d'une requête.

La **restriction** permet d'extraire un sous-ensemble de la table par une condition sur un ou plusieurs des attributs de la relation. Elle a pour conséquence d'extraire qu'une partie de ses tuples. Une restriction peut porter sur plusieurs attributs.

Elle s'écrit : $T = \prod_{attributs}(R)$

Cette clause s'appuie sur une condition (ou prédicat) qui est une expression logique qui sera évaluée pour chaque ligne. Les lignes ramenées seront celles qui satisfont à la condition.

**SELECT** [**DISTINCT**] { * | NomCol1 | ExprSQL [AS etiq1] [,Nomcol2 | ExprSQL [etiq2] … ] }
**FROM** NomTable
**WHERE** << condition de restriction >> ;

<< condition de restriction >> ::=

| Nomcol | Opérateur de comparaison | Valeur de comparaison |
|---|---|---|
| **colonne** | = | Constante, colonne |
| | < | Constante, colonne |
| | > | Constante, colonne |
| | <= | Constante, colonne |
| | >= | Constante, colonne |
| | != ou <> | Constante, colonne |
| | [NOT] BETWEEN val_inf AND val_sup[5] | |
| | [NOT] IN (val1, val2,…) | |
| | [NOT] LIKE motif[6] | |
| | IS [NOT] NULL | |

Les opérateurs logiques AND, OR et NOT[7] permettent de combiner les conditions.

Les salariés gagnants plus de 40 000

```
SELECT nom, salaire
FROM employes
WHERE salaire > 40000;
```

[5] Relation d'ordre numérique, lexicographique ou temporel.
[6] Le motif est une chaîne de caractère ou une date
[7] L'ordre de priorité est le suivant : plus prioritaire NOT, AND, OR moins prioritaire.

Les salariés dont le service et la fonction correspondent à l'un des couples de valeurs indiquées

```
SELECT nom, fonction, salaire
WHERE (id_dept, fonction) IN ( (20, 'EMPLOYE'), (10, 'DIRECTEUR') );
```

Le motif utilise les **jokers** % et _ (souligné).

| | |
|---|---|
| % | Représente une suite quelconque de caractères (y compris aucun). |
| _ (souligné) | Représente un caractère quelconque. |

Il est possible de définir un caractère d'échappement pour désactiver les jokers.

```
SELECT *
FROM employees
WHERE last_name LIKE '% \_%' ESCAPE '\' ;
```

## 5.2 Variable de substitution

Une variable de substitution permet d'écrire des requêtes interactives ou avec des arguments.

```
SELECT last_name, hire_date FROM employees
WHERE job_id = '&categorie' ;
```

Avec esperluette double (&&)

```
SELECT last_name, hire_date, &&Nomcol FROM employees
ORDER BY &NomCol  ;
```

La double esperluette indique à l'application de conserver la valeur de la variable à chaque référence de la variable dans le script.

Définir une variable de substitution : l'instruction `DEFINE`

```
DEFINE categorie = 15
SELECT last_name, hire_date FROM employees
WHERE job_id = '&categorie' ;
UNDEFINE categorie
```

Les salariés embauchés depuis 1995 gagnant moins de 9000. Ces salariés seront classés par ordre alphabétique croissant de leur nom.

Liste de TOUS les salariés, classés par catégorie professionnelle, à l'exception de ceux qui ne sont pas affectés au service (DEPARTMENT_ID) 50.

```
SELECT * FROM employees
WHERE department_id != 50
```

```
      OR department_id IS NULL
ORDER BY job_id ;
```

## 5.3 La clause ORDER BY

Par défaut, l'ordre des lignes ramenées par l'instruction SELECT est indéterminé.

La clause ORDER BY permet de classer à l'affichage les lignes extraites de la commande SELECT.

**SELECT** [**DISTINCT**] {* | NomCol1 | ExprSQL [AS etiq1] [,Nomcol2 | ExprSQL [etiq2] … ] }
**FROM** NomTable
**WHERE** << condition de restriction >>
**ORDER BY** {NomCol | ExprSQL | etiq | Pos [ASC |DESC]} [,NomCol2 [ASC |DESC]];

| | |
|---|---|
| Expression | Les lignes seront triées sur la valeur de l'expression. |
| Étiquette | Référence à l'expression associée à l'étiquette indiquée dans le SELECT. Les lignes seront triées sur la valeur de l'expression et permet d'éviter de reformuler complètement l'expression lorsque celle-ci est complexe. |
| Position | Tri sur l'expression repérée par sa position dans la clause SELECT. Cette formulation est nécessaire en cas de recours à des opérateurs ensemblistes (voir plus loin). |

### 5.3.1 Cas de l'absence de valeur (NULL)

Par défaut, les valeurs NULL seront affichées au-delà des valeurs renseignées.

Pour modifier ce comportement utiliser NULLS FIRST (ou NULLS LAST dans l'ordre décroissant).

```
SELECT *
FROM employees
ORDER BY commission_pct NULLS FIRST ;
```

Toutes expressions arithmétiques contenant la valeur NULL renvoie NULL. On dit que, la valeur NULL est l'élément absorbant[8].

## 1.1 Variable de substitution

Une variable de substitution permet d'écrire des requêtes interactives ou avec des arguments.

[8] En mathématiques, un élément absorbant d'un ensemble pour une loi de composition interne est un élément de cet ensemble qui transforme tous les autres éléments en l'élément absorbant lorsqu'il est combiné avec eux par cette loi. $\forall x, aTx = a$

```
SELECT last_name, hire_date FROM employees
WHERE job_id = '&categorie' ;
```

Avec esperluette double (&&)

```
SELECT last_name, hire_date, &&Nomcol FROM employees
ORDER BY &NomCol  ;
```

La double esperluette indique à l'application de conserver la valeur de la variable à chaque référence de la variable dans le script.

Définir une variable de substitution : l'instruction `DEFINE`

```
DEFINE categorie = 15
SELECT last_name, hire_date FROM employees
WHERE job_id = '&categorie' ;
UNDEFINE categorie
```

Les salariés embauchés depuis 1995 gagnant moins de 9000. Ces salariés seront classés par ordre alphabétique croissant de leur nom.

Liste de TOUS les salariés, classés par catégorie professionnelle, à l'exception de ceux qui ne sont pas affectés au service (DEPARTMENT_ID) 50.

```
SELECT * FROM employees
WHERE department_id != 50
     OR department_id IS NULL
ORDER BY job_id ;
```

# 6 Les jointures

## 6.1 Les jointures

La jointure $R \bowtie S$ est une opération binaire avec 2 entités qui utilise des attributs de même type et de même sémantique avec ses 2 entités pour créer un tuple unique.

Autrement dit, c'est l'association de 2 tables à l'aide de colonnes comparables.

Généralement, la jointure traduit la notion de clé étrangère qui reflète une dépendance entre ces deux tables.

La jointure peut être considérée comme la combinaison d'un produit cartésien et d'une restriction !

Une jointure naturelle est une jointure mettant en jeu des colonnes de même nom et dans le cadre d'une opération d'équijointure (opérateur d'égalité). Inversement on parlera none équijointure.

Seules les lignes qui vérifient la condition seront affichées, on parle d'*inner join*.

Les lignes qui seront situées dans l'une ou l'autre table qui ne vérifient pas la condition de jointure seront affichées dans le cas d'une *outer join*.

Note

Un produit cartésien (A x B) résulte généralement, d'un oubli de la condition de jointure (CROSS JOIN).

```
SELECT table1.col, table2.col
FROM table1
[ NATURAL [INNER ]JOIN table2 ] |
[ [INNER ]JOIN table2 USING (autre_col) ] |
[ [INNER ]JOIN table2      ON (table1.col = table2.col) ] |
[ LEFT | RIGHT | FULL OUTER JOIN table2 ON (table1.col = table2.col)
] |
[ CROSS JOIN table2 ] ;
```

Opération de jointure.

| JOIN | Opération de jointure |
|---|---|
| **NATURAL** | En présence de cette option, l'opération de jointure (équi-jointure) sera effectuée sur l'ensemble des colonnes, des deux tables, qui portent le même nom. Attention : si les tables ne disposent pas de colonnes portant un nom identique, un produit cartésien est généré. Remarque : si une étoile est indiquée derrière l'ordre SELECT, la colonne de jointure sera affichée en première position (et présentée une seule fois) |

| | |
|---|---|
| INNER | La clause facultative INNER (valeur par défaut) permet de qualifier explicitement la jointure en tant que jointure interne classique (seules les lignes vérifiant la condition de jointure sont ramenées). |
| **{ LEFT \| RIGHT \| FULL } OUTER** | Qualifie la jointure en tant que jointure externe. |
| ON <condition de jointure> | Permet d'indiquer explicitement le nom des colonnes et la condition de jointure entre les deux sources. Clause indiquée en substitution de la clause NATURAL. |
| **USING ( colonne(s) )** | Permet d'indiquer explicitement le nom commun de(s) seule(s) colonne(s) sur lesquelles effectuer la jointure (équi-jointure) entre les deux sources.<br><br>Clause indiquée en substitution de la clause NATURAL. |

Exemple de jointure naturelle :

```
SELECT department_id, department_name, location_id, city
FROM departments
NATURAL JOIN locations ;
```

Dans ce cas, toutes les colonnes de même nom dans chaque table seront utilisées pour établir la jointure. Les colonnes doivent être de même type et de même sémantique.

### 6.1.1 Alias de table ou synonyme local

Il est parfois utile ou nécessaire de fournir un alias à une table lors de l'écriture d'un ordre SQL.

Dès lors, il est possible de préfixer les noms des colonnes. On y gagnera en termes de lisibilité. Cette règle peut faire partie d'une démarche qualité.

Exemple de jointure entre 3 tables :

```
SELECT     e.employee_id, l.city, d.department_name
FROM  employees e  JOIN     departments d
      ON    d.department_id = e.department_id
      JOIN  locations l
            ON    d.location_id = l.location_id ;
```

Exemple avec une condition WHERE :

```
SELECT e.last_name, e.salary, d.department_name
FROM employees e JOIN departments d
  ON (e.department_id = d.department_id)
  WHERE e.salary BETWEEN 8000 AND 15000;
```

Le nom, la date d'embauche et le grade salarial (grade_level) des employés embauchés avant 2000.

```
SELECT e.last_name, e.hire_date, g.grade_level
FROM employees e INNER JOIN job_grades g
ON (e.salary BETWEEN g.lowest_sal AND g.highest_sal)
WHERE e.hire_date < TO_DATE('01/01/2000' , 'DD/MM/YYYY') ;
```

### 6.1.2 Auto-Jointure

C'est l'opération de joindre une table à elle-même ; on parle de réflexivité.

Elle nécessite de désigner deux fois le même nom de table derrière la clause FROM en utilisant des alias de noms de tables.

Dans ce cas, on utilisera un alias différent pour chaque occurrence de la table.

```
SELECT worker.last_name emp, manager.last_name mgr
FROM employees worker JOIN employees manager
ON (worker.manager_id = manager.employee_id ) ;
EMP                       MGR
------------------------- -------------------------
Hunold                    De Haan
Fay                       Hartstein
Gietz                     Higgins
Lorentz                   Hunold
Ernst                     Hunold
Zlotkey                   King
Mourgos                   King
Kochhar                   King
Hartstein                 King
De Haan                   King
Whalen                    Kochhar
Higgins                   Kochhar
Vargas                    Mourgos
Rajs                      Mourgos
Matos                     Mourgos
Davies                    Mourgos
Taylor                    Zlotkey
Grant                     Zlotkey
Abel                      Zlotkey

19 rows selected
```

Q : Quels sont les employés qui gagnent plus que leur chef ?

### 6.1.3 Les jointures externes

La jointure externe (OUTER JOIN) traduit une extension de la jointure interne classique (INNER JOIN) en préservant dans le résultat (partiellement ou totalement) les lignes ne vérifiant pas la correspondance de jointure.

La jointure externe exprimera une relation non obligatoire entre 2 tables. Autrement dit, elle traduit l'existence de la cardinalité minimale zéro.

De fait, une jointure externe entre des tables dont les cardinalités valent soient (1,1) ou (1,n) n'a pas de sens !

Les noms, salaires et si possible services (department_name) de TOUS les employés gagnant entre 6000 et 14000.

```
SELECT e.last_name, e.salary, d.department_name
FROM employees e LEFT OUTER JOIN departments d
  ON (e.department_id = d.department_id)
  WHERE e.salary BETWEEN 6000 AND 14000;
```

La direction de la jointure externe (LEFT ou RIGHT) précise la table depuis laquelle on veut afficher toutes les lignes. Ici on affichera les employés qui ne sont pas affectés à un département.

```
SELECT d.department_name, e.last_name
FROM employees e RIGHT OUTER JOIN departments d
  ON (e.department_id = d.department_id)
ORDER BY 1, 2 ;
```

Ci-dessus, les départements où aucun salarié affecté, seront affichés

# 7 Les fonctions de groupe

Les fonctions de groupe (ou d'agrégat) effectuent des opérations sur un ensemble de lignes pour retourner un seul résultat.

Les fonctions de groupes peuvent apparaitre dans les clauses `SELECT`, `ORDER BY` ou `HAVING`.

```
SELECT  [DISTINCT] {* |  NomCol1 | ExprSQL [AS etiq1] | Fg1
[,Nomcol2 | ExprSQL [etiq2] … ] }
FROM NomTable
GROUP BY Nomcol1 [,NomCol2,…] ;
```

| Fonction | Description |
|---|---|
| **COUNT ( [ *\| ALL \| DISTINCT ]** expression **)** | Nombre de ligne |
| **SUM ( [ ALL \| DISTINCT ]** expression **)** | Retourne la somme des valeurs. |
| **MIN ( [ ALL \| DISTINCT ]** expression **)** | Retourne la valeur la plus petite. |
| **MAX ( [ ALL \| DISTINCT ]** expression **)** | Retourne la valeur la plus grande. |
| **AVG ( [ ALL \| DISTINCT ]** expression **)** | Retourne la moyenne |
| **MEDIAN ( expression )** | Retourne la valeur médiane d'une distribution supposée continue. |
| **STDDEV**[9] **( [ ALL \| DISTINCT ]** expression **)** | Retourne l'écart type d'une distribution (noté σ) |
| **VARIANCE**[10] **( [ ALL \| DISTINCT ]** expression **)** | Retourne la variance d'une distribution |
| **LISTAGG** | Construit une liste concaténée de valeurs |

- `ALL` Par défaut toutes les valeurs (renseignées) sont prises en compte.
- `DISTINCT` Ne prend seulement en compte que les valeurs différentes.

La plupart des fonctions de groupes disposent de clauses supplémentaires permettant leur usage dans le cadre de fonctions analytiques (contexte *data wharehouse*).

## 7.1.1 Regroupement de lignes : clause GROUP BY


On pourra regrouper un ensemble en sous-ensembles ayant une valeur commune.

Exemple : calculer le salaire moyen par catégorie professionnelle.

[9] Ecart-type (*standard deviation*) : l'écart type est une mesure de la dispersion d'une variable aléatoire réelle. Il est défini comme la racine carrée de la variance.

[10] Variance : peut être définie comme la moyenne des carrés des écarts à la moyenne.

Le partitionnement a pour objectif de **regrouper** les lignes qui possèdent des attributs partageant la même valeur. On utilisera pour ce faire, la clause **GROUP BY**.

Note

La projection (SELECT) fera apparaitre uniquement les fonctions de groupes et/ou les colonnes qui apparaissent dans la clause GROUP BY.

### 7.1.2 Restrictions sur groupes : Clause HAVING

De façon similaire à la clause WHERE sur les lignes, la clause **HAVING** filtrera des groupes de lignes. Autrement dit, la clause HAVING est à la fonction de groupe ce qu'est la clause WHERE à la colonne.

```
SELECT  [DISTINCT] {* |  NomCol1 | ExprSQL [AS etiq1] | Fg1
[,Nomcol2 | ExprSQL [etiq2] … ] }
FROM NomTable
GROUP BY Nomcol1 [,NomCol2,…]
HAVING <<conditions de restriction du groupe>> ;
```

Conditions de restriction du groupe ::= Fonction_de_groupe op_groupe { Constante | ExprSQL }

### 7.1.3 Les expressions ROLLUP et CUBE

Il existe des expressions supplémentaires à l'instruction GROUP BY pour la réalisation de tableaux croisés à des fins analytiques.

```
SELECT c.raisonsociale, co.numcom , SUM(a.prixunit*l.qtecom) Montant
FROM client c JOIN commande co
        ON (c.idclient=co.idclient)
                JOIN ligne_com l
                ON (co.numcom=l.numcom)
                        JOIN article a
                        ON(l.idarticle=a.idarticle)
GROUP BY ROLLUP (c.raisonsociale, co.numcom)
```

### 7.1.4 La fonction analytique LISTAGG

Cette fonction analytique a pour but de construire une liste de valeur (*measure_column*) regroupée par la clause WITHIN GROUP.

Syntaxe :

```
LISTAGG (measure_column [, 'delimiter'])
  WITHIN GROUP (order_by_clause) [OVER (query_partition_clause)]
```

Exemple :

Afficher la liste (nom) de tous les salariés par département.

```
SQL> SELECT department_id,
LISTAGG(last_name, ',') WITHIN GROUP (ORDER BY last_name) AS
```

```
salaries
FROM   employees
GROUP  BY department_id
```

```
DEPARTMENT_ID SALARIES
------------- ------------------------------------------------------
           10 Whalen
           20 Fay, Hartstein
           30 Baida, Colmenares, Himuro, Khoo, Raphaely, Tobias
           40 Mavris
           50 Atkinson, Be_ll, Bissot, Bull, Cabrio, Chung, Davies, Dellinger,
Dilly, Everett, Feeney, Fleaur, Fripp, Gates, Gee, Geoni, Grant, Jones, Kaufling,
Ladwig, Landry, Mallin, Markle, Marlow, Matos, McCain, Mikkilineni, Mourgos, Nayer,
OConnell, Olson, Patel, Perkins, Philtanker, Rajs, Rogers, Sarchand, Seo, Stiles,
Sullivan, Taylor, Vargas, Vollman, Walsh, Weiss

           60 Austin, Ernst, Hunold, Lorentz, Pataballa
           70 Baer
           80 Abel, Ande, Banda, Bates, Bernstein, Bloom, Cambrault, Cambrault,
Doran, Errazuriz, Fox, Greene, Hall, Hutton, Johnson, King, Kumar, Lee, Livingston,
Marvins, McEwen, Olsen, Ozer, Partners, Russell, Sewall, Smith, Smith, Sully,
Taylor, Tucker, Tuvault, Vishney, Zlotkey

           90 De Haan, King, Kochhar
          100 Chen, Faviet, Greenberg, Popp, Sciarra, Urman
          110 Gietz, Higgins
              Grant

12 rows selected.
```

# 8 Les sous-interrogations

En lieu et place d'une expression SQL, on pourra utiliser un ordre SELECT.

Vous pouvez utiliser une sous requête dans les clauses **FROM**, **WHERE** et **HAVING**.

Une sous-requête dans la clause FROM est déconseillée pour des raisons de lisibilité de l'instruction. Dans ce cas, préférez l'utilisation de vues.

Les requêtes imbriquées, ou sous-requêtes, se placent entre parenthèses.

Il est possible d'imbriquer 255 niveaux de sous-requêtes !

```
SELECT select_list
FROM table
WHERE expression opérateur
                (SELECT select_list
                FROM table) ;
```

Les sous-requêtes seront classées en 3 catégories.

- Les instructions dont la sous-requête ramène **une et une seule** ligne
- Les instructions dont la sous-requête est susceptible de ramener **n** lignes (**n>1**)
- Les requêtes corrélées

## 8.1 Sous-interrogations mono-ligne

Les opérateurs suivants pourront être utilisés : =, >, >=, <, <=, <>.

La sous-requête devra soit utiliser un prédicat WHERE qui ne ramènera à coup sûr et dans le temps, une seule ligne ou bien en utilisant une fonction de groupe.

Exemple : Les salariés payés en-dessous de la moyenne générale des salaires.

```
SELECT last_name, salary
FROM employees
WHERE salary < (SELECT AVG(salary) FROM employees) ;
```

## 8.2 Sous-interrogations multilignes

Si l'opérateur de comparaison qui introduit la sous-requête est un opérateur d'égalité ou de différence, il s'attend à recevoir une donnée scalaire.

Dans le cas contraire, sur une liste de valeurs, on utilisera

- IN remplace =
- NOT IN remplace <>

Si votre sous-requête comporte un opérateur d'inégalité, c'est que celle-ci contient une ambiguïté.

Il faudra réécrire la sous-requête afin qu'elle soit de type mono-ligne.

Exemple : Les salariés payés en-dessus du directeur des ventes (SA_MAN).

```
SELECT last_name
FROM employees
WHERE salary < (SELECT salary FROM employees
           WHERE job_id = 'SA-MAN') ;
```

Attention aux sous-requêtes multi-ligne

Même question pour les ST_CLERK !

```
SELECT last_name
FROM employees
WHERE salary < (SELECT salary FROM employees
           WHERE job_id = 'ST-CLERK') ;
```

Cette requête échoue car la sous-requête ramène plus d'une ligne !

Correction possible :

```
SELECT last_name
FROM employees
WHERE salary > (SELECT MAX(salary) FROM employees
           WHERE job_id = 'ST-CLERK') ;
```

Ici, les employés gagnant plus que le gros salaire des ST_CLERK.

### 8.2.1 Les opérateurs de comparaison multilignes

Dans le cas où la sous-requête est susceptible de ramener plusieurs lignes, il sera nécessaire d'utiliser les opérateurs multilignes suivants :

- IN
- ANY (ou SOME)
- ALL
- ANY et ALL devront être précédé par un des opérateurs suivants : =, !=, >, <, <=, >=.

Il est bon de noter que :

- <ANY signifie moins que le maximum
- >ANY signifie plus que le minimum
- =ANY équivaut à IN.
- !=ALL équivaut à NOT IN

### 8.2.2 L'opérateur existentiel EXISTS

Est-il vrai ou faux qu'il existe au moins une ligne vérifiant le critère.

```
SELECT * FROM departments
WHERE NOT EXISTS
(SELECT * FROM employees
WHERE employees.department_id = departments.department_id) ;
```

En pratique, cette requête sera plus performante puisqu'elle vérifie seulement l'existence ou non d'une ligne sélectionnée.

Exemple : Liste des employés qui occupent un poste de manager ?

```
SELECT * FROM employees
WHERE employee_id IN (SELECT manager_id FROM employees) ;

SELECT * FROM employees
WHERE employee_id  NOT IN (SELECT manager_id FROM employees) ;
```

Attention cette 2ème requête, la sous-requête doit retourner des données renseignées.

## 8.3 Les requêtes corrélées

Dans une sous-requête corrélée (ou synchronisée), la requête interne attend des valeurs de la requête externe.

Une ou plusieurs colonnes de la requête externe seront référencées dans la requête interne.

Autrement dit, la valeur de la colonne de la requête externe sera un paramètre de la requête interne.

```
SELECT col1, col2, … FROM Table1 T1
WHERE expr oper ( SELECT col1, col2,… FROM Table2 T2
                    WHERE T1.col1 oper T2.col2 ) ;
```

Cette corrélation se traduit par l'utilisation de l'alias de la table externe dans la requête interne.

# 9 Opérateurs ensemblistes

On définit les opérateurs ensemblistes par l'union, l'intersection et la différence.

L'union et l'intersection sont des opérations commutatives :

$$T1 \cup T2 = T2 \cup T1$$

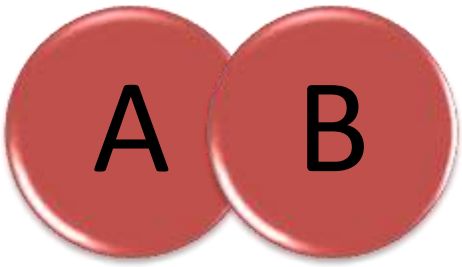
$$T1 \cap T2 = T2 \cap T1$$

Par contre, la différence n'est pas commutative :

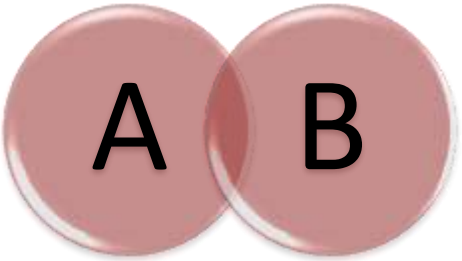
$$T1 - T2 \neq T2 - T1$$

*En algèbre relationnelle* $A \cap B$ *peut aussi s'écrire :* $A - (A - B)$

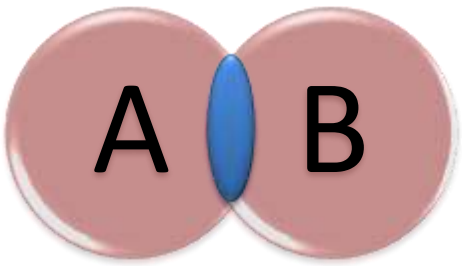
```
SELECT expr1, expr2, … FROM table1
OPERATEUR ENSEMBLISTE
SELECT expr1, expr2, … FROM table2 ;
```



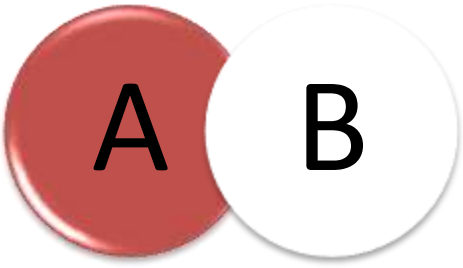

UNION




UNION ALL




INTERSECT




MINUS

Les opérateurs ensemblistes sont :

| Opérateurs | Définition |
|---|---|
| **UNION** | Lignes issues des 2 requêtes sans les doublons |
| **UNION ALL** | Lignes issues des 2 requêtes avec les doublons |
| **INTERSECT** | Lignes communes aux 2 requêtes |
| **MINUS** | Lignes de la 1ère requête réduit de celles qui sont présentes dans la 2ème. |

Les 2 ensembles doivent être uni-compatibles tant au niveau nombre de colonne et type de colonne.

Dans la table résultat, le nom des colonnes hérite du nom des colonnes de la 1ère table.

De même la clause ORDER BY ne peut apparaitre qu'à l'endroit du 2ème SELECT

# 10 Manipulations des données

## 10.1 Ajout de ligne : l'ordre INSERT

L'instruction INSERT permet d'ajouter une ou plusieurs lignes à une table.

```
INSERT INTO table [ (col1 [, col2…] ) ]
 { VALUES (valeur1 | DEFAULT [, valeur2…] )
      | Ordre_SELECT };
```

Dans le cas de l'écriture suivante :

```
INSERT INTO table
VALUES (valeur1 [, valeur2…] ) ;
```

Il est nécessaire d'énumérer les valeurs dans l'ordre de définition des colonnes et toutes les colonnes doivent être renseignées.

```
INSERT INTO employees
VALUES (152 , 'DUPONT', 'René', NULL, NULL,… ) ;
```

## 10.2 Copier des lignes d'une autre table

```
INSERT INTO table2 (col1 [,col2,…] )
SELECT col1 [, col2,… ] FROM table1
WHERE [condition] ;
```

## 10.3 Mise à jour de lignes : l'ordre UPDATE

La mise à jour (modification) de données s'effectue par l'ordre **UPDATE**.

Cet ordre permet de modifier la valeur d'une ou de plusieurs colonnes, de ligne(s) actuellement présente(s) dans une table.

```
UPDATE cible [ alias ]
SET colonne1 = { expression | sous-requête | DEFAULT }
     [ , colonne2 = expression ... ]
[ WHERE condition ]
```

- La cible peut désigner une table, une vue ou une sous-requête.
- En l'absence de clause WHERE toutes les lignes sont mises à jour.
- La nouvelle valeur peut être une expression SQL, le résultat d'une sous-requête ou bien la valeur par DEFAUT telle que définie pour la colonne.
- Fournir un alias en cas de recours à des sous-requêtes corrélées.
- La clause SET peut désigner une liste de colonnes (énumérées entre parenthèses).

Augmenter de 5% le salaire de tous les salariés !

```
UPDATE employees
SET salary = salary * 1.05 ;
```

## 10.4 Suppression de lignes : l'ordre DELETE

L'instruction DELETE permet de supprimer une ou plusieurs lignes d'une table.

```
DELETE [FROM] cible [ alias ]
[WHERE condition ] ;
```

- La cible peut désigner une table, une vue ou une sous-requête.
- En l'absence de clause WHERE, toutes les lignes sont supprimées.
- Fournir un alias en cas de recours à des sous-requêtes corrélées dans l'expression des conditions.

Élimine le contenu de la table mais ne libère pas l'espace physique.

## 10.5 L'ordre TRUNCATE

```
TRUNCATE TABLE nom_table ;
```

Élimine tout le contenu d'une table ET libère l'espace physique comme le DROP TABLE.

## 10.6 Insertion multi-tables

Placement conditionnel, dans des tables différentes, d'un ensemble de lignes résultant de l'exécution d'un ordre SELECT.

```
INSERT { ALL | FIRST }
WHEN condition1 THEN INTO cible1 [ ( colonne1 [, colonne2 ... ] ) ]
                       [ VALUES ( valeur1 [, valeur2 ... ] ) ]
[ WHEN condition2 THEN INTO cible2 ... ]
[ ELSE INTO cibleN ... ]
ordre_SELECT ;
```

| | |
|---|---|
| WHEN condition | Condition(s) déterminant la cible choisie pour l'insertion de lignes (THEN INTO cible). Les conditions doivent référencer les colonnes retournées par l'ordre SELECT. |
| colonne | Nom de(s) colonne(s) à renseigner (la totalité des colonnes si leur nom n'est pas précisé). |
| VALUES ( ... ) | Désigne le nom des seules colonnes de l'ordre SELECT dont il s'agit de récupérer la valeur (la totalité des colonnes si leur nom n'est pas précisé). |
| ALL \| FIRST | **ALL** : toutes les conditions sont évaluées pour chaque ligne. De ce fait une copie de la même ligne peut être insérée dans plusieurs cibles.<br>**FIRST** : la première condition vérifiée détermine le choix de l'unique cible pour l'insertion de la ligne. |
| Ordre_SELECT | Ordre SELECT désignant les lignes à répartir dans les différentes cibles. |

Exemple

Insertion des lignes récupérées dans trois tables différentes selon le montant du salaire des lignes sélectionnées. Récupération du contenu de colonnes particulières dans le 3ème cas.

```
INSERT
     WHEN sal < 4000 THEN INTO salarie_01
```

```
     WHEN sal BETWEEN 2000 AND 5000 THEN INTO salarie_02
     ELSE INTO salarie_03 (code, nom, categorie, salaire)
          VALUES (matric, nom, fonction, sal)
SELECT * FROM salarie;
```

## I.4 Fusion de lignes : l'instruction MERGE

La fusion de ligne correspond à l'ajout ou mise à jour simultané de lignes dans une table et s'effectue par l'ordre **MERGE**.

Cette instruction combine deux opérations (INSERT et UPDATE) effectuées sur une table à partir de lignes prélevées d'autre(s) table(s).

L'indication d'une condition permettra de distinguer le choix entre l'ajout de nouvelles lignes ou la mise à jour de lignes déjà existantes.

Privilèges nécessaires :

- Table source : SELECT
- Table cible : INSERT et UPDATE

### I.4.1 syntaxe

```
MERGE INTO cible
USING source
ON ( condition(s) )
WHEN MATCHED THEN UPDATE
     SET colonne1 = { expression | DEFAULT }
       [,colonne2 = { expression2 | DEFAULT }]…
     [ WHERE condition(s) ]
WHEN NOT MATCHED THEN INSERT ( colonne1 [, colonne2 ... ] )
     VALUES ( { expression | DEFAULT } [ , ... ] )
     [ WHERE condition(s) ]
```

| | |
|---|---|
| **USING source** | Origine des lignes à récupérer : table, vue ou sous-requête. |
| **ON ( condition(s) )** | Condition(s) déterminant le choix entre la mise à jour (UPDATE) et l'ajout de ligne (INSERT). |
| **WHEN MATCHED THEN UPDATE** | Exécution d'un UPDATE lorsque la condition est vérifiée. |
| **WHEN NO MATCHED THEN INSERT** | Exécution d'un INSERT lorsque la condition exprimée derrière la clause ON n'est pas vérifiée. |

*Exemple*

Mise à jour, ou récupération, de lignes comptables à partir d'une table de mouvement selon la présence ou non des lignes concernées dans le livre comptable.

```
MERGE INTO livre_compta lc
USING mouv_compta mc
ON ( lc.numlig = mc.numlig )
WHEN MATCHED THEN UPDATE
     SET lc.monttc = mc.monttc
WHEN NOT MATCHED THEN INSERT ( lc.numlig, lc.dtecri, lc.monttc,
lc.pointe )
VALUES (mc.numlig, mc.dtecri, mc.monttc, 'O');
```

# 11 Les ordres LDD

## 11.1 Le CREATE TABLE

Les tables

La table est la représentation physique d'une entité ou d'une relation du MCD[11].

C'est l'unité élémentaire de stockage dans une base Oracle.

Une table :

- Est créée par `CREATE TABLE`
- Modifiée par `ALTER TABLE`
- Supprimée par `DROP TABLE`

```
CREATE TABLE [schéma.]NomTable
(col1 type_donnée [ DEFAULT expr ] [NOT NULL] [,…]
[CONSTRAINT NomContrainte typeContrainte] [,…]) ;
```

- *Schéma* : désigne le nom du propriétaire de la table, sinon l'utilisateur courant.
- *NomTable* : Chaîne de 30 caractères maximum. Le nom n'est pas sensible à la casse sauf s'il est encadré de guillemets (").

Il est d'usage (mais non obligatoire évidemment) de mettre les noms de table au singulier

- col1 type_donnée : Nom de la colonne et son type (NUMBER, CHAR, VARCHAR2, DATE,…). DEFAULT renseigne une valeur si la colonne n'est pas renseignée (NULL).
- *NomContrainte typeContrainte* : Nom de la contrainte et son type (PRIMARY KEY, NOT NULL,…)

Lors de la création de la table, vous pouvez spécifier le tablespace qui servira à stocker l'index lié à la contrainte de clé primaire.

```
CREATE TABLE employe (
     Id_emp NUMBER(5) CONSTRAINT pk_employe PRIMARY KEY,
     nom VARCHAR2(30)
)
TABLESPACE data
     ENABLE PRIMARY KEY USING INDEX
     TABLESPACE TBS_ndx;
```

[11] MCD Modèle Conceptuel de Données

### 11.1.1 Les vues du dictionnaire

```
[DBA|ALL|USER]_TABLES
```

### 11.1.2 Privilèges

```
CREATE TABLE
CREATE ANY TABLE
```

## 11.2 Les types de données

Les types de données qui définissent les colonnes d'une table varient d'un SGBD à l'autre.

*Les principaux types Oracle :*

| Type de données | Description |
|---|---|
| **VARCHAR2 (*n[octet, car]*)**<br>**VARCHAR (*n[octet, car]*)** | Chaîne de caractères de longueur variable d'une taille maximale n (max=4000 /32767[12])<br>4000 octets ou caractères si `MAX_STRING_SIZE = STANDARD`<br>32767 octets ou caractères si `MAX_STRING_SIZE = EXTENDED` |
| **CHAR(*n*)** | Chaîne de caractères de longueur fixe d'une taille *n.* (max=2000) |
| **NVARCHAR2, NVARCHAR** | Chaîne de caractères de longueur variable. Un caractère peut être codé sur plusieurs octets |
| **NUMBER [ (p [, s]) ]** | Numérique de *p* chiffres maximal dont *s* décimales.<br>p ∈ [1, 38] et s ∈ [-84, 127]<br>Un *number* à une taille de 1 à 22 octets.<br>± 999…(38x) x$10^{125}$<br>Synonymes : NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL. |
| **FLOAT [(p)]** | Sous-type de *NUMBER* |
| **DATE** | Information comprenant une date et une heure (année, mois, jour, heure, minute, seconde). Valeur comprise entre 01/01/-4712 au 31/12/9999. Taille constante de 7 octets. |
| **TIMESTAMP [(précision)]** | Année, mois, jour, heure, minute, seconde, où *précision* elle la partie fractionnaire de la seconde. Précision=[0,9] |
| **LONG (obsolète)** | Chaîne de caractères de 2Gio. Utilisez le type LOB |
| **CLOB** | |
| **RAW(taille)** | Type binaire de 2000/32767 octets maximum.<br>2000 octets si `MAX_STRING_SIZE = STANDARD`<br>32767 octets si `MAX_STRING_SIZE = EXTENDED` |
| **LONG RAW** | Type binaire de 2Gio maximum. |
| **BLOB, CLOB, NCLOB** | Type binaire ou chaîne de caractères de grande taille. Taille maximale = (4 Gio - 1) x (database block size). |
| **BFILE** | Pointeur vers un fichier externe (4Gio maxi) |
| **ROWID** | Adresse physique d'une ligne sur 6 octets |

[12] Depuis Oracle 12c avec le paramètre `MAX_STRING_SIZE = EXTENDED`

## 11.3 Les contraintes

Les contraintes d'intégrité vérifient les règles de validité des données conformes aux règles de gestion. Ce contrôle est effectué par le noyau avant chaque modification des données (ajout, modification ou suppression).

Les contraintes seront de nature :

- Type conforme au domaine et à l'usage de la propriété.
- Obligatoire (NOT NULL), la colonne doit avoir une valeur.
- Unicité (UNIQUE), la colonne ne doit pas avoir de doublon.
- Clé primaire ou identifiant (PRIMARY KEY), la colonne est renseignée et est n'a pas de doublon.
- Clé étrangère ou contrainte référentielle (FOREIGN KEY), la colonne fait référence à une autre colonne d'une autre table.
- De vérification (CHECK), la colonne doit vérifier une condition.

*Exemple d'écriture d'une contrainte CHECK :*

```
ALTER TABLE Employe
ADD CONSTRAINT emp_salary_min
     CHECK (salary > 32000) ;
```

Autre exemple :

```
ALTER TABLE Employe
ADD CONSTRAINT ck_emp_courriel
     CHECK (courriel LIKE '%@%.%') ;
```

## 11.4 Modification de la structure d'une table : ALTER TABLE

Il est possible de modifier la définition d'une table existante sans remettre en cause son contenu lorsque le contexte applicatif évolue.

L'ordre **`ALTER TABLE`** fournit un très grand nombre de possibilités, parmi lesquelles :

- Ajout ou suppression de colonnes ou de contraintes d'intégrité.
- Modification de définition de colonnes existantes (sur le typage et la valeur par défaut).
- Et bien d'autres choses encore…

```
ALTER TABLE NomTable
{
ADD ( colonne_définition [ , colonne_définition ... ] )
| MODIFY ( colonne_définition [ , colonne_définition ... ] )
| DROP ( colonne )
| RENAME COLUMN ancien_NomColonne TO nouveau_NomColonne
| ADD CONSTRAINT contrainte_de_table
| DROP CONSTRAINT contrainte
| RENAME CONSTRAINT NomContrainte TO nouveau_NomContrainte
| DISABLE CONSTRAINT contrainte [ CASCADE ]
| ENABLE [ VALIDATE | NOVALIDATE ] CONSTRAINT contrainte
| RENAME TO nouveau_NomTable
}
```

| | |
|---|---|
| **ADD** | Ajout de colonne(s). Lors de l'ajout d'une colonne, l'indication (facultative) de la clause DEFAULT appliquera cette valeur à l'ensemble des lignes actuellement présentes dans la table. |
| **MODIFY** | Modification de la définition actuelle de la colonne. (Caractéristiques de type, de longueur et de valeur par défaut) |
| **DROP** | Suppression de colonne. |
| **RENAME COLUMN … TO** | Renommer une colonne |
| **ADD CONSTRAINT** | Ajout de contrainte(s) d'intégrité. |
| **DROP CONSTRAINT** | Suppression de contrainte d'intégrité |
| **RENAME CONSTRAINT … TO** | Renommer une contrainte d'intégrité. |
| **DISABLE CONSTRAINT** | Désactivation de la contrainte. CASCADE : propagation sur la clé extérieure. |
| **ENABLE CONSTRAINT** | Réactivation de la contrainte. VALIDATE : contrôle préalable du contenu actuel (par défaut). NO VALIDATE : sans contrôle préalable du contenu actuel. |
| **RENAME TO** | Renommer une table |

Exemples

Ajout d'une colonne à la table désignée

```
ALTER TABLE service ADD ( adresse VARCHAR2(60) )
```

Ajout d'une colonne virtuelle.

```
ALTER TABLE salarie ADD ( revenu AS ( sal + nvl(comm, 0) ) )
```

Renommer une colonne

```
ALTER TABLE salarie RENAME COLUMN addr TO adresse ;
```

## 11.5 Les commentaires

Il est possible d'ajouter des commentaires aux tables (ou aux vues) et aux colonnes afin de fournir une documentation au modèle physique.

Ces commentaires seront stockés dans le dictionnaire et consultables dans les vues USER_TAB_COMMENTS et USER_COL_COMMENTS.

```
COMMENT ON { TABLE NomTable | COLUMN NomTable.NomCol }
IS 'commentaire' ;
```

Pour supprimer un commentaire, il suffit de fournir une chaîne vide.

Exemple :

```
COMMENT ON COLUMN employe.salaire IS 'salaire brut annuel' ;
```

## 11.6 Les vues

Les vues sont des tables logiques ; seul est stocké dans le dictionnaire le texte de la requête servant à définir la vue. En d'autres termes, elles n'utilisent aucun segment de données.

- Indépendance Logique
- Commodité
- Confidentialité
- Intégrité

### 11.6.1 Création

```
CREATE [ OR REPLACE ] [FORCE | NOFORCE ]VIEW [schema.]nom_vue
[(alias [,alias]…]
AS interrogation
[WITH {READ ONLY | CHECK OPTION [CONSTRAINT nom_contrainte ]}];
```

| | |
|---|---|
| **REPLACE** | Recréer la vue si elle existe déjà |
| **Alias** | Nom de colonnes dans la vue |
| **Interrogation** | Requête sur laquelle est construite la vue |
| **FORCE\|NOFORCE** | L'option FORCE permet de créer la vue même si la requête (interrogation) comporte des erreurs |
| **WITH CHECK OPTION** | Interdit la création ou la modification de lignes qui ne pourraient être elles-mêmes sélectionnées à travers la vue |
| **WITH READ ONLY** | Seule la consultation des lignes est autorisée |

```
CREATE OR REPLACE VIEW vue_emp
AS SELECT last_name, first_name, salary FROM hr.employees ;
```

### 11.6.2 Utilisation des vues

Les vues améliorent la sécurité et la confidentialité des données.

Elles garantissent une meilleure lisibilité et qualité du code.

Il est possible, sous certaines conditions, de mettre à jour des données à travers les vues. Dans ce cas, la vue ne doit pas être basée sur :

- Une clause DISTINCT
- Une fonction de groupe ou la clause GROUP BY
- Une jointure (sauf rare exception)
- Un opérateur ensembliste

- Ou toute fonction modifiant les valeurs des colonnes

### 11.6.3 Suppression d'une vue

```
DROP VIEW [schema.]nom_vue ;
```

La suppression de la vue implique :

- Suppression de sa définition dans le dictionnaire
- Les procédures PL/SQL basées sur la vue sont conservées mais auront le statut INVALID
- Les synonymes sont conservés également

## 11.7 Les synonymes

Les synonymes permettent de rendre transparents le propriétaire de l'objet (table, vue, séquence ou morceau de code) et la localisation de l'objet (DB_LINK). Les deux principaux objectifs du synonyme sont d'améliorer la sécurité en masquant le propriétaire ou la localisation de l'objet, et de simplifier l'écriture des ordres SQL.

Par ailleurs, le synonyme permet de conserver le même code même si le nom de la table change, il suffit dans ce cas de redéfinir seulement le synonyme.

### 11.7.1 Syntaxe

```
CREATE [PUBLIC] SYNONYM [schema.]nom_synonyme
          FOR [schema.]nom_objet[@dblink] ;
```

```
CREATE SYNONYM emp
FOR hr.employees@BD_formation;
```

### 11.7.2 Privilèges

CREATE SYNONYM

CREATE ANY SYNONYM

CREATE PUBLIC SYNONYM

### 11.7.3 Suppression des synonymes

```
DROP [PUBLIC] SYNONYM [schema.]nom_synonyme ;
```

### 11.7.4 Vues du dictionnaire

DBA|ALL|USER_SYNONYMS

## 11.8 Les séquences

Les séquences permettent de générer des valeurs numériques entières séquentielles.

Elles sont l'objet idéal pour la création de clés primaires. Oracle gère l'accès concurrent et libère la ressource même si la transaction n'est pas terminée (par un *Commit* ou un *Rollback*).

### 11.8.1 Syntaxe

```
CREATE SEQUENCE nom_sequence
           [INCREMENT BY valeur1]
           [START WITH valeur2]
           [MAXVALUE valeur3/NOMAXVALUE]
           [MINVALUE valeur4/NOMINVALUE]
           [CYCLE/NOCYCLE]
           [CACHE valeur5/NOCACHE] ;
```

MAXVALUE <= $10^{28}$.

### 11.8.2 Privilèges

CREATE SEQUENCE

CREATE ANY SEQUENCE

### 11.8.3 Utilisation

L'appel à la valeur courante de la séquence se fait par : nom_sequence.*currval*. L'accès à cette valeur ne peut se faire que si un appel à *nextval* préalable ait été fait lors de la session.

L'accès à la valeur suivante se fait par : nom_sequence.*nextval*

La séquence peut être utilisée :

- Avec une commande INSERT après la clause VALUES
- Avec un SELECT
- Avec une commande UPDATE après la clause SET.

### 11.8.4 Modification

```
ALTER SEQUENCE nom_sequence
           [INCREMENT BY valeur1]
           [MAXVALUE valeur3/NOMAXVALUE]
           [MINVALUE valeur4/NOMINVALUE]
           [CYCLE/NOCYCLE]
           [CACHE valeur5/NOCACHE]
           [ORDER | NOORDER ] ;
```

Les valeurs sont comprises entre $-(10^{-27}-1)$ et $10^{28}-1$.

### 11.8.5 Suppression

```
DROP SEQUENCE nom_sequence ;
```

### 11.8.6 Vues du dictionnaire

```
USER|ALL|DBA_SEQUENCES
```

## 11.9 Les index

Les index servent à réduire les coûts de requêtes ramenant un échantillon restreint des lignes d'une table.

Il existe différents types d'index :

- Index B-TREE (Balanced Tree)
- Index BITMAP
- Les tables organisées en index (IOT)
- Les clusters indexés ou hash

### 11.9.1 Index B-TREE

Un index B-TREE est un arbre équilibré.

Il pourra être efficace dans les conditions suivantes :

- Clé comportant suffisamment de valeurs distinctes **> 200**
- Colonnes contenant peu de valeurs NULL (les valeurs NULL ne sont pas indexées)
- Colonnes utilisées dans la clause WHERE
- Colonnes utilisées dans la clause JOIN
- Si le pourcentage de ligne ramené est faible
- Clés multiples

Un index ne pourra pas être utilisé avec les opérateurs suivants :

- != ou <>
- IS NULL
- IS NOT NULL
- NOT IN
- LIKE valeur numérique ou type date
- LIKE '%chaine'

Et si la colonne est un argument d'une fonction ou fait partie d'une expression SQL.

Les mises à jour de la table sont ralenties par la mise à jour des index associés.

Pour créer un index, utiliser la commande suivante :

```
CREATE INDEX nom_index ON nom_table(nom_col1, nom_col2,…) ;
```

La commande suivante :

```
ALTER INDEX nom_index REBUILD [ TABLESPACE nom_tablespace ]
```

Recrée un nouvel index et à la fin de la création l'ancien index est détruit.

# 12 ANNEXES

## 12.1 FONCTIONS PREDEFINIES

Les fonctions SQL sont de différents types : mono-ligne, multi-lignes, analytiques ou de référencement de types d'objet Oracle.

Ces fonctions peuvent être classées par les types des arguments qu'elles utilisent :

- Les fonctions sur les numériques
- Les fonctions sur les chaînes de caractères
- Les fonctions sur les dates
- Les fonctions de conversions

Voici une liste non exhaustive des fonctions disponibles :

### 12.1.1 FONCTIONS NUMERIQUES

**ABS** (n : NUMBER) : NUMBER
Retourne la valeur absolue de *n*

```
Ex. : ABS(-15) = 15
```

**CEIL** (n : NUMBER) : NUMBER
Retourne le plus petit entier immédiatement ≥ *n*

```
Ex. :      CEIL(2850.12) = 2851
```

**FLOOR** (n : NUMBER) : NUMBER
Retourne le plus grand entier immédiatement ≤ *n*

```
Ex. :      FLOOR(2850.12) = 2850
```

**MOD** (m : NUMBER, n : NUMBER) : NUMBER
Reste de la division euclidienne $\frac{m}{n}$

Autre syntaxe : m MOD n.

```
Ex. :      MOD(10,3) = 1
reste := 10 MOD 3;
```

**POWER** (n : NUMBER, m : NUMBER) : NUMBER
Retourne $n^m$

```
Ex. : POWER(3,4) = 81
```

**ROUND** (n : NUMBER[,m : NUMBER]) : NUMBER
Retourne *n* arrondi à la décimale *m*. Si *m* n'est pas spécifié, *n* est arrondi à 0 décimale. Si *m* < 0, *n* est arrondi à gauche du point décimal.

```
Ex. :      ROUND(129.687,1) = 129.7
     ROUND(129.687) = 130
     ROUND(129.687,-2) = 100
```

**SIGN** (n : NUMBER) : NUMBER
Retourne -1 si *n* < 0, 0 si *n* = 0, 1 si *n* > 0

```
Ex. : SIGN(-15) = -1
```

**SQRT** (n : NUMBER) : NUMBER
Retourne la racine carrée de *n* ($\sqrt{n}$) ou NULL si *n* < 0

```
Ex. : SQRT(2307) = 48.0312
```

**TRUNC** (n : NUMBER[,m : NUMBER]) : NUMBER
Retourne *n* tronqué à la décimale *m*. Si *m* n'est pas spécifié, *n* est tronqué à 0 décimales. Si *m* < 0, *n* est tronqué à gauche du point décimal.

```
Ex. :     TRUNC(129.687,1) = 129.6
      TRUNC(129.687) = 129
      TRUNC(129.687,-2) = 100
```

**WIDTH_BUCKET** ( *expression, valeur_mini, valeur_maxi, nombre_classes* )
Retourne la position de *l'expression* au sein d'un pseudo-histogramme de classes pour lequel les extrémités sont définies par *valeur_mini* et *valeur_maxi* et comportant une quantité d'intervalles réguliers définie par *nombre*_classes.

Exemple :

```
SELECT last_name, salary,
   WIDTH_BUCKET(salary, 0,25000, 25) "Tranche"
   FROM Employees
   ORDER BY last_name, "Tranche";
```

### 12.1.2 FONCTIONS SUR CHAINES

**ASCII** (ch : CHAR) : NUMBER
Retourne le code ASCII du premier caractère de la chaîne *ch*

```
Ex. :     ASCII ('Bonjour') = 66
```

**CHR** (n : NUMBER) : CHAR
Retourne le caractère de code ASCII *n*

```
Ex. :     CHR (83) = 'S'
```

**INITCAP** (ch : CHAR) : CHAR
Retourne la chaîne *ch*, avec la première lettre de chaque mot en majuscule

```
Ex. :     INITCAP ('salut max') = 'Salut Max'
```

**INSTR** (ch1 : CHAR, ch2 : CHAR[,n : NUMBER[,m : NUMBER]]) : NUMBER
Retourne la position de l'occurrence *m* de la chaîne *ch2* dans la chaîne *ch1*, à partir de la position *n* (*n*,*m* : 1 par défaut).

```
Ex. : INSTR ('quoi de neuf docteur', 'eu', 1, 2) = 18
```

**LENGTH** (ch : CHAR) : NUMBER
Retourne le nombre de caractères de la chaîne *ch*

```
Ex. :     LENGTH ('anticonstitutionnel') = 19
```

**LOWER** (ch : CHAR) : CHAR
Retourne la chaîne *ch* en minuscules

```
Ex. : LOWER ('PAS VU')  = 'pas vu'
```

**LPAD** (ch1 : CHAR, lg : NUMBER [,ch2 : CHAR]) : CHAR
Retourne la chaîne *ch1* complétée à gauche sur *lg* caractères par la chaîne *ch2* (blanc par défaut)

```
Ex.: LPAD ('PAS PRIS', 15, '*=+') = '*=+*=+*PAS PRIS'
```

**LTRIM** (ch1 : CHAR [,ch2 : CHAR]) : CHAR
Enlève les caractères de gauche de *ch1* jusqu'au premier n'appartenant pas à *ch2* (blanc par défaut)

```
Ex. :     LTRIM('RIRI FIFI', 'RI') = 'FIFI'
          LTRIM('LOULOU') = 'LOULOU'
```

**REPLACE** (ch1 : CHAR, ch2 : CHAR [,ch3 : CHAR]) : CHAR
Retourne la chaîne *ch1*, chaque occurrence de *ch2* ayant été remplacée par *ch3*. Si *ch3* n'est pas spécifiée, enlève toutes les occurrences de *ch2* dans *ch1*.

```
Ex. :     REPLACE('SA SOUPE','S','L') = 'LA LOUPE'
     REPLACE('EPRISE','E') = 'PRIS'
```

**RPAD** (ch1 : CHAR, lg : NUMBER[,ch2 : CHAR]) : CHAR
Retourne la chaîne *ch1* complétée à droite sur *lg* caractères par la chaîne *ch2* (blanc par défaut)

```
Ex. : RPAD('TATARA',12,'TA') = 'TATARATATATA'
```

**RTRIM** (ch1 : CHAR [,ch2 : CHAR]) : CHAR
Enlève les caractères de fin de *ch1* après le dernier n'appartenant pas à *ch2* (blanc par défaut)

```
Ex. :     RTRIM ('SCROGNEUGNEU','ENGU') = 'SCRO'
     RTRIM ('IDEFIX') = 'IDEFIX'
```

**SOUNDEX** (ch : CHAR) : CHAR
Retourne la chaîne représentant le "son" (représentation phonétique) de *ch1*

```
Ex. :     SOUNDEX('RHUM') = SOUNDEX('ROME')
```

**SUBSTR** (ch : CHAR ,n : NUMBER[, lg : NUMBER]) : CHAR
Retourne l'extrait de la chaîne *ch* à partir de la position *n*, sur *lg* caractères (ou jusqu'à la fin par défaut)

```
Ex. :     SUBSTR('DESIDERATA',7) = 'RATA'
```

**TRANSLATE** (ch1 : CHAR,ch2 : char,ch3 : CHAR) : CHAR
Retourne la chaîne *ch1* dont les caractères appartenant à *ch2* ont été remplacés par les caractères correspondants de *ch3*

```
Ex. : TRANSLATE('SACREBLEU!','!BL','RCO') = 'SACRECOEUR'
```

**UPPER** (ch : CHAR) : CHAR
Retourne la chaîne *ch* en majuscules

```
Ex. :     UPPER('minus') = 'MINUS'
```

### 12.1.3 FONCTIONS SUR DATE

Formats possibles pour une date :

| Format | Description |
|---|---|
| **CC, SCC** | Siècle, Siècle signé |
| **YY, YYYY** | Année sur 2 ou 4 chiffres |
| **Q** | Trimestre |
| **MONTH, MON, MM** | Mois |
| **WW** | Numéro de semaine de l'année |
| **W** | Numéro de semaine du mois |
| **DD** | Jour du mois |
| **DDD** | Nombre de jours écoulés depuis le début de l'année (1-366) |
| **DAY** | jour de la semaine |
| **DS** | Format court : '*30/08/22*' |
| **DL** | Format long : *'mardi 30 août 2022'* |
| **HH, HH24** | Heure |
| **MI** | Minute |
| **SS** | Seconde |
| **SSSSS** | Nombre de seconde écoulées depuis minuit |
| **TS** | Heure format court : *'22:33:59'* |

**ADD_MONTHS** (d : DATE, n : NUMBER) : DATE
Retourne : la date *d* ajoutée de *n* mois

```
Ex. : ADD_MONTHS('21/12/12',2) = '21/02/13'
```

**LAST_DAY** (d : DATE) : DATE
Retourne la date du dernier jour du mois contenant *d*

```
Ex. : LAST_DAY('15-FEB-91') = '28-FEB-91'
```

**MONTHS_BETWEEN** (d1 : DATE, d2 : DATE) : NUMBER
Retourne le nombre de mois entre *d1* et *d2*

```
Ex. : MONTHS_BETWEEN('17-JUL-91','25-DEC-92') = -17.258
```

**NEXT_DAY** (d : DATE, ch : CHAR) : DATE
Retourne la date du jour *ch* suivant la date d

```
Ex. : NEXT_DAY('07-NOV-91','Sunday')= '10-NOV-91'
```

**EXTRACT** (year FROM *date*)
Extrait et affiche la valeur spécifiée par le 1er champ de la date (dans cet exemple, l'année).

**ROUND** (d : DATE [, ch : CHAR]) : DATE
Retourne la date d arrondi comme spécifié par le format *ch*. Si *ch* est omis, *d* est arrondie selon le format 'DD'

**TRUNC** (d : DATE [, ch : CHAR]) : DATE
Retourne la date d tronquée comme spécifié par le format *ch*. Si *ch* est omis, *d* est tronqué selon le format 'DD'

### 12.1.4 FONCTIONS DE CONVERSION

**TO_CHAR** (d : DATE [,ch CHAR]) : CHAR
Retourne la chaîne représentant la date *d* au format *ch*

```
Ex. : TO_CHAR(d, 'DD/MM/YY  HH24:MI')= '24/11/91 12:48'
```

**TO_CHAR** (n : NUMBER [,ch CHAR]) : CHAR
Retourne la chaîne représentant le nombre *n* au format *ch*

```
Ex. : TO_CHAR(12.456,'00000.99')= '00012.46'
```

**TO_DATE** (ch1 : CHAR [,ch2 : CHAR]) : DATE
Retourne la date correspondant à la chaîne *ch1* représentée selon le format *ch2*

```
Ex. :TO_DATE('01/06/92','DD/MM/YY')= '01-JUN-92'
```

**TO_NUMBER** (ch CHAR) : NUMBER
Retourne le nombre correspondant à la chaîne *ch*, qui doit représenter un nombre valide

```
Ex. :TO_NUMBER('12E2') = 1200
```

### 12.1.5 FONCTIONS PARTICULIERES

**DECODE** (a : CHAR, val1 : CHAR, expr1: CHAR[,val2 : CHAR,expr2 : CHAR ,…],a :CHAR) : CHAR
Retourne : si *a* = *val1* retourne *expr1*,
si *a* = *val2* retourne *expr2,…*
sinon retourne *a*.

**GREATEST** (e1, e2, ...)
Retourne la valeur la plus grande de la liste (*e1, e2, …* )
Tous les éléments doivent être de même type.

```
Ex. :GREATEST ('a', 'z', 'm') = 'z'
```

**LEAST** (e1, e2, ...)
Retourne la valeur la plus petite de la liste (*e1, e2, …* )
Tous les éléments doivent être de même type.

```
Ex. : LEAST (5, 9, 2, 3) = 2
```

**USERENV** (ch : CHAR) : CHAR
Retourne l'informations sur l'utilisateur connecté et sa session. *ch* doit appartenir à : ENTRYID, SESSIONID, TERMINAL ou LANGUAGE

```
Ex. :USERENV ('SESSIONID') = 18
```

### 12.1.6 Fonctions relatives à la valeur *NULL*.

**NVL** (a1 : CHAR, a2 : CHAR) : CHAR

**NVL** (a1 : NUMBER, a2 : NUMBER) : NUMBER

**NVL** (a1 : DATE, a2 : DATE) : DATE

**NVL** (a1 : BOOLEAN, a2 : BOOLEAN) : BOOLEAN

Retourne : si *a1* n'est pas NULL, retourne *a1*, sinon *a2*.

```
Ex. : NVL (a, 'chaîne') retourne 'chaîne',si a = NULL.
```

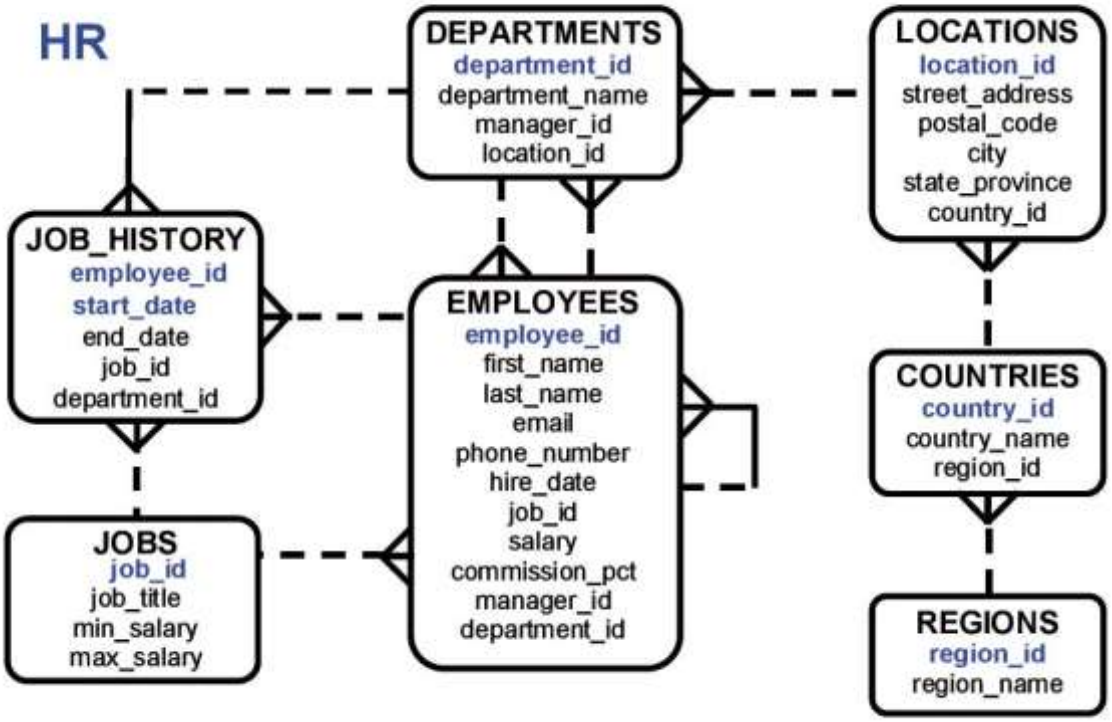
**NULLIF** (*expr1, expr2*) : renvoie *NULL* si `expr1=expr2`, expr1 sinon.

### 12.1.7 Les fonctions de groupe

| Fonction | Définition |
|---|---|
| **AVG (ExprSQL)** | Retourne la moyenne arithmétique d'un ensemble de valeurs. |
| **COUNT( )** | nombre de ligne |
| **SUM( )** | le total arithmétique |
| **MIN( )** | le minimum d'une série |
| **MAX( )** | le maximum d'une série |
| **STDDEV ( )** | l'écart type d'une distribution |
| **VARIANCE ( )** | la variance d'une distribution |
| **LISTAGG** | Construit une liste concaténée de valeurs |

## 12.2 Modèle HR

**Diagramme entité/relation HR**



## 12.4 Commandes SQL*Plus

**Tableau 1 : Commandes du buffer SQL*Plus**

| Commande | Description |
|---|---|
| **r ou /** | Exécute (run) le contenu du buffer |
| **l** | Liste le contenu du buffer |
| **l*n*** | Liste la *nième* ligne |
| **i** | Insère une ligne après la ligne courante |
| **a *texte*** | Ajoute *texte* à la fin de la ligne courante |
| **del** | Supprime la ligne courante |
| **c/*texte1*/*texte2*** | Substitue la 1ère occurrence de *texte1* par *texte2* |
| **clear** | Efface le buffer |
| **quit ou exit** | Quitte SQL*Plus |
| **connect *user/password@descripteur*** | Nouvelle connexion |
| **get *fichier*** | Charge le contenu du fichier dans le buffer |
| **save *fichier*** | Sauvegarde le contenu du buffer dans le *fichier* |
| **start *fichier* ou @ *fichier*** | Exécute le *fichier.sql* |
| **spool *fichier*** | Créer le *fichier.lst* dans le répertoire courant qui contiendra la trace des entrées/sorties jusqu'à spool off. |

**Tableau 2 : Variables d'environnement**

| Commande | Description |
|---|---|
| **set autocommit {on \| off \| immediate \| n}** | Validation automatique après une ou *n* commandes |
| **set echo {on \| off }** | Affichage des commandes avant exécution |
| **set linesize *n*** | Nombre de colonnes (caractères) par ligne |
| **set pagesize *n*** | Nombre de lignes d'une page de résultat |
| **set serverout [PUT] {on \| off}** | Activation de l'affichage pour tracer les exécutions |
| **set termout {on \| off}** | Affichage des résultats |
| **set time { on \| off }** | Affichage de l'heure dans le prompt |
| **set head { on \| off }** | Affichage le nom des colonnes dans le résultat |

**Tableau 3 : paramètres de la commande SHOW**

| Paramètre | Description |
|---|---|
| ***variable*** | Variable d'environnement (autocommit, echo, …) |
| **ALL** | Affiche toutes les variables d'environnement |
| **Errors** | Erreur de compilation d'un bloc PL/SQL |
| **Release** | Version du SGBDR |
| **User** | Nom de l'utilisateur connecté |

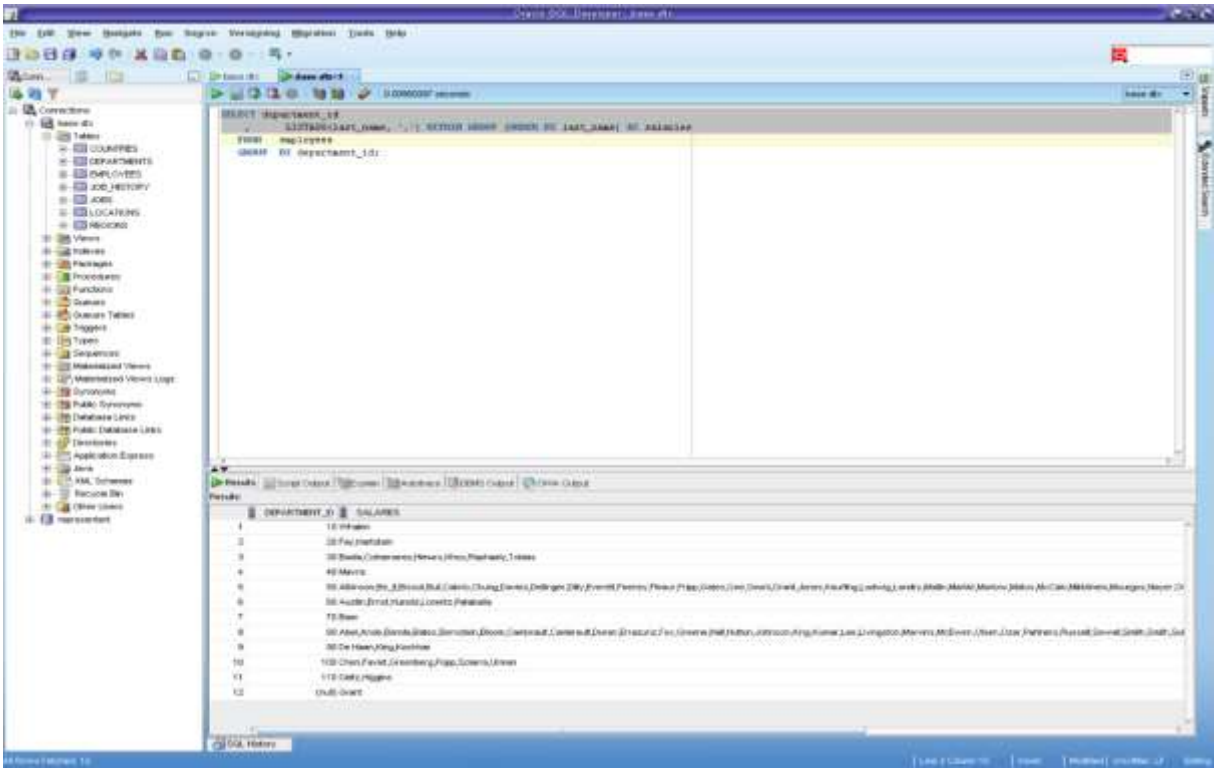
### *12.4.1 Le fichier login.sql*

Ce fichier contient les paramètres sql*plus qui seront lus à chaque démarrage de sql*plus.

Ce fichier doit se trouver dans un répertoire référencé par la variable d'environnement `ORACLEPATH` de l'utilisateur.

```
$ export ORACLEPATH=$HOME
```

## 12.5 SQLDeveloper



# 13 SQLcl

SQLcl est la rencontre entre SQL*Plus et SQL*Developer.

C'est un outil en ligne de commande permettant d'enrichir les fonctionnalités de SQL*Plus. En particulier, on appréciera la gestion de l'historique des commandes !

## 13.1 Quelques commandes

```
info
ddl
ctas
```

## 13.2 Quelques exemples de format :

```
set sqlformat ansiconsole
set sqlformat csv
set sqlformat insert
set sqlformat xml
```

## 13.3 Easter eggs

```
show sqldev
show sqldev2
```

# 14 Bibliographie / Sitographie


- *Introduction aux bases de données 8ème édition* Chris J. Date (Vuibert), décembre 2004. ISBN-13: 978-2711748389
- *SQL pour Oracle 7ème édition* Christian Soutou (Eyrolles), 2015
- https://db-engines.com : classement mensuel des SGBDR
- https://livesql.oraclecom : plateforme pour apprendre le SQL et le PL/SQL
- https://oracle.com/goto/oll : Oracle Learning Library

# II. Les privilèges

**12**

CREATE TABLE ..... 48

# III. Vues du dictionnaire

## Vues

USER_SEQUENCES ..... 50
USER_SYNONYMS ..... 49

# IV. Acronymes

## Acronymes

ANSI : *American National Standards Institute* ..... 10
DBA : *DataBase Administrator* ..... 13
SEQUEL : *Structured English QUEry Language* ..... 10
SGBDR : *systèmes de gestion de base de données relationnelle* ..... 8
SQL : *Structured Query Language* ..... 10

# V. INDEX


**&**

*& esperluette* ..... 23
*&& double esperluette* ..... 23

**|**

*|| Voir concat*

**A**

*ACID* ..... 42
*ADD_MONTHS* ..... 25
*ALL* ..... 37
*ANY* ..... 37
*atomique* ..... 42

**B**

*BD* ..... 10
*Berkeley DB* ..... 10
*BFILE* ..... 46
*BLOB* ..... 46

**C**

*Cardinalité* ..... 16
*CASE* ..... 26
*CHAR* ..... 46
*Clé candidate* ..... 14
*clé étrangère* ..... 32
*CLOB* ..... 46
*Cluster indexé* ..... 50
*COALESCE* ..... 26
*cohérente* ..... 42
`concat` ..... 20
*Contrainte*
- CHECK ..... 47
- FOREIGN KEY ..... 47
- NOT NULL ..... 47
- PRIMARY KEY ..... 47
- UNIQUE ..... 47

*COUNT* ..... 28, 57
*CROSS JOIN* ..... *Voir Produit cartésien*

**D**

*Data Analysts* ..... 11
*DATE* ..... 46
*DB2* ..... 10
*DCL* ..... 18
*DDL* ..... 18
*DECODE* ..... *Voir CASE*
*DEFINE* ..... 23
*Degré* ..... 14
*DELETE* ..... 41
*Dimension* ..... 16
*DML* ..... 18
*Domaine* ..... 14
*DROP TABLE* ..... 41
*durable* ..... 42

**E**

*élément absorbant* ..... 26
*Entité* ..... 15
*EXISTS* ..... 37
*ExprSQL* ..... 19

**F**

*FireBird* ..... 10
*fonctions* ..... 24

**G**

*GROUP BY* ..... 28

**H**

*hash* ..... *Voir Cluster indexé*
*HAVING* ..... 29

## I

*IBM* ..... *10*
*Identifiant* ..... *15*
*IN* *37*
*Index BITMAP* ..... *50*
*Index B-TREE* ..... *50*
*individu* ..... *14*
*Informix* ..... *10*
*inner join* ..... *32*
*INSERT* ..... *41*
*IOT* ..... *50*
*isolée* ..... *42*

## J

*jointures*
jointures naturelles ..... 32

## L

*LAST_DAY* ..... *25*
*Les fonctions de groupe*
AVG ..... 28
*LISTAGG* ..... *28, 57*
*LOB* ..... *10, Voir Large Object*
*LONG* ..... *46*
*LONG RAW* ..... *Voir RAW*

## M

*MAX* ..... *57*
*MERGE* ..... *43*
*MIN* ..... *57*
*MONTHS_BETWEEN* ..... *25*
*Mysql* ..... *10*

## N

*NATURAL JOIN* ..... *32*
*NEXT_DAY* ..... *25*
*NULL* ..... *47*
*NULLIF* ..... *26*
*NULLS FIRST* ..... *22*
*NULLS LAST* ..... *Voir NULLS FIRST*
*NVL* ..... *26*
*NVL2* ..... *26*

## O

*Occurrence* ..... *14*
*ORDER BY* ..... *22*
*outer join* ..... *32*
*OUTER JOIN* ..... *32*

## P

*PostGres* ..... *10*
*pré-compilateurs* ..... *10*
*produit cartésien* ..... *32*
*projection* ..... *19*
*Propriété* ..... *14*

## R

*restriction* ..... *21*

## S

*SEQUEL* ..... *10*
*SOME* ..... *Voir ANY*
*sous-interrogations* ..... *36*
*SQL server* ..... *10*
*SQL:2011* ..... *10*
*STDDEV* ..... *28, 57*
*SUM* ..... *57*
*Sybase* ..... *11*
*synonyme local* ..... *33*
*System R* ..... *10*

## T

*Tables organisées en index* ..... *50, Voir IOT*
*Teradata* ..... *11*
*Transaction* ..... *42*
*TRUNCATE* ..... *41*
*Type* ..... *14*

## U

*UNDEFINE* ..... *Voir DEFINE*
*UPDATE* ..... *41*

## V

*VARCHAR2* ..... *46*

*VARIANCE* ..... *28, 57*