Le Langage SQL
Le **L**angage de **M**anipulation des **D**onnées
L'instruction **SELECT**

Table des Matières

Introduction

SQL est un langage non procédural. Vous l'utilisez pour indiquer au système quelles données rechercher ou modifier sans lui indiquer comment réaliser ce travail.

SQL ne dispose pas d'instructions pour contrôler le flux d'exécution du programme, pour définir une fonction ou exécuter une boucle, ni d'expressions conditionnelle.

A un niveau théorique, les instructions SQL peuvent être réparties en trois catégories:

- Le langage de définition de données, ou LDD, qui définit la structure des données.
- Le langage de manipulation de données, ou LMD, qui recherche et modifie les données.
- Le langage de contrôle de données, ou LCD, qui définit les privilèges d'accès accordés aux utilisateurs de la base de données.

La catégorie LMD comprend quatre instructions de base :

- SELECT, qui recherche les données d'une table.
- INSERT, qui ajoute des données à une table.
- UPDATE, qui modifie les lignes existantes d'une table.
- DELETE, qui supprime les lignes d'une table.

L'instruction SELECT

La forme simplifiée d'une requête SQL SELECT est la suivante :

```
SELECT [DISTINCT] {colonne1, colonne2, …, colonneN | *}     -- Projection
   FROM {table~~(s)~~}  [JOIN table]                        -- Jointure, …
   [WHERE {condition}]                                      -- Restriction
   [ORDER BY {colonne(s) [ASC|DESC] [NULLS FIRST|LAST]}]    -- Tri
```

Clauses d'une instruction SELECT

Clause SELECT

- Permet de spécifier la liste des colonnes à retourner : séparer chaque colonne par une virgule
- AS "Nom de l'alias" permet de renommer la colonne
- Mot clé DISTINCT : Recherche de lignes distinctes : les doublons sont éliminés

Clause FROM

- spécifie **la** table (principale) sur laquelle porte la requête
- dans le cas où la requête doit porter sur plusieurs tables, il FAUT utiliser JOIN

Clause WHERE

- spécifie un critère de restriction (ou de sélection) à utiliser pour la requête
- contient une ou plusieurs conditions qui doivent être satisfaites par une ligne pour qu'elle soit extraite par la requête
- Vous pouvez utiliser les opérateurs AND, OR et NOT pour combiner plusieurs conditions à satisfaire dans une requête. Ces opérateurs peuvent apparaître plusieurs fois mais il faut prendre garde à leurs priorités et/ou utiliser des parenthèses

  Évitez de mélanger le AND et le OR ! Si toutes les conditions doivent être vraies, utilisez AND. Si au moins une condition doit être vraie, utilisez OR.

Clause ORDER BY

- Permet de trier les tuples (enregistrements) retournés par la requête selon l'ordre (ASC/DESC) spécifié pour les colonnes à trier.
- Il est possible de spécifier si les valeurs `NULL` doivent se trouver au début ou à la fin de la liste (`NULLS FIRST` ou `NULLS LAST`).

Restreindre le nombre de résultats : `FETCH FIRST n ROWS` ou `WHERE ROWNUM`

- *Certains sgbd ont une clause `LIMIT` permettant de restreindre le nombre de résultats.*
- Oracle n'utilise pas `LIMIT`, mais `FETCH NEXT n ROWS` *(à partir de la version Oracle 12c)*. Il est possible de définir un `OFFSET` *(départ)*, un pourcentage, ainsi qu'une restriction `ONLY` ou `WITH TIES`.
- `WHERE ROWNUM` permet de ne récupérer qu'un certain nombre d'enregistrements :

  ```
  SELECT emp_nom, emp_prenom FROM exe_employe   WHERE ROWNUM <= 5;
  ```

  Attention lorsqu'il y a une clause `ORDER BY` : la condition `WHERE` étant faite AVANT l'`ORDER BY`, seuls les n enregistrements récupérés seront triés ; par conséquent, le résultat ne sera pas le même selon qu'Oracle utilise ou non un index :

  ```
  SELECT emp_nom, emp_prenom FROM exe_employe
     WHERE ROWNUM <= 5
     ORDER BY emp_nom;
  ```

  5 enregistrements sont d'abord récupérés, puis ceux-ci seront triés ensuite par nom.

Opérateurs

Opérateurs logiques

Opération	Opérateur
Egal à	=
Différent de	<>
Inférieur à	<
Inférieur ou égal à	<=
Supérieur à	>
Supérieur ou égal à	>=

Opérateurs arithmétiques

Opération	Opérateur
Addition	+
Soustraction	-
Multiplication	*
Division	/

Opérateur de concaténation

Opération	Opérateur
Concaténation	||

Autres opérateurs

LIKE

Lorsqu'on n'est pas certains de l'orthographe exacte d'un élément on peut se servir de l'opérateur LIKE pour rechercher des lignes. Le caractère "%" sert de caractère générique (appelé aussi **joker** ou **wildcard**) dans ce contexte. Il est équivalent à zéro ou plusieurs caractères. Le caractère "_" sert à remplacer n'importe quel caractère unique.

Ex : `… WHERE emp_nom LIKE 'T%'`

BETWEEN

Cet opérateur fonctionne avec des valeurs numériques, des chaînes et des dates. L'opérateur BETWEEN est l'équivalent de deux expressions reliées par l'opérateur logique AND. Lorsqu'il est utilisé de façon appropriée, il simplifie la requête.

Ex : `… WHERE emp_salaire BETWEEN 5000 AND 6000;`

IN

Permet de comparer la valeur d'une colonne ou d'une expression avec une liste de valeurs possibles. L'opérateur IN retourne une valeur booléenne qui est VRAI ou FAUX.

- VRAI si l'expression est égale à une des valeurs de la liste.
- FAUX si l'expression n'est égale à aucune des valeurs de la liste.

Une alternative à l'opérateur IN est la combinaison des différentes conditions avec l'opérateur logique OR.

Fonctions texte

Les fonctions permettent d'obtenir un résultat particulier basé sur les paramètres qui lui sont passés.

Ex : UPPER(*chaîne_de_caractères*) -> retourne la chaîne en majuscule

LOWER(*chaîne_de_caractères*) -> retourne la chaîne en minuscule

INITCAP(*chaîne_de_caractères*) -> retourne la chaîne de caractères avec l'initiale en majuscule

LENGTH(*chaîne_de_caractères*) -> retourne la longueur de la chaîne

Expressions

Une expression est une combinaison d'une ou plusieurs valeurs, opérateurs et fonctions qui retourne une valeur.

Ex :
```
exe_employe.emp_nom
'Nom : ' || emp_nom
INITCAP(emp_nom)
```

Valeur NULL

La valeur NULL dans une colonne a plusieurs significations :

- Aucune autre valeur dans cette colonne n'est applicable pour la ligne en question
- La colonne n'a pas encore reçu de valeur

Si vous souhaitez rechercher les enregistrements d'une table dans lesquels une colonne particulière n'a pas encore reçu de valeur, vous pouvez le spécifier au moyen de l'expression "**IS NULL**" dans une clause WHERE. En effet, une valeur **peut être** NULL, mais elle ne sera jamais **égale à** NULL parce que NULL est une valeur indéfinie.

Ex :
```
… WHERE emp_salaire IS NULL;
… WHERE emp_salaire IS NOT NULL;
```

Analogie : NULL peut être vu comme le contenu d'une boîte qui n'a jamais été ouverte. Il pourrait y avoir n'importe quoi à l'intérieur. Donc, il n'est pas possible de comparer une boîte qui n'a pas été ouverte à une autre parce que vous ne savez pas ce qu'il y aura à l'intérieur de chacune d'elle.

Fonctions statistiques principales

```
AVG([DISTINCT|ALL] expr)          -- moyenne des valeurs de l'expression
                                  -- numérique expr pour tous les
                                  -- enregistrements de la requête
MIN([DISTINCT|ALL] expr)          -- valeur minimum de l'expression expr
MAX([DISTINCT|ALL] expr)          -- valeur maximum de l'expression expr
SUM([DISTINCT|ALL] expr)          -- somme des valeurs de l'expression
                                  -- numérique expr
COUNT({* | [DISTINCT|ALL] expr}) -- compte le nombre de valeurs de
                                  -- l'expression expr pour tous les
                                  -- enregistrements de la requête
```

Quelques fonctions spéciales

NVL

SYNTAXE :

NVL(expr1, expr2)

EMPLOI :

Si expr1 contient la valeur NULL, alors retourne expr2, sinon retourne expr1.

EXEMPLES :

```
SELECT emp_prenom, emp_nom, nvl(emp_email,'PAS DE MAIL') AS "Mail"
   FROM exe_employe;

   EMP_PRENOM EMP_NOM  Mail
   ---------- -------  -----------
   Jean       BON      jb@heg.ch
   Yves       REMORD   PAS DE MAIL
   Alex       TERIEUR  at@heg.ch
```

Les employés touchent une commission de 5%, ou de 150.- si leur salaire n'est pas défini :

```
SELECT emp_nom, nvl(emp_salaire*5/100,150) FROM exe_employe;
```

Pour mettre du texte au lieu d'un nombre, il faut tout convertir en texte :

```
SELECT emp_nom, nvl(TO_CHAR(emp_salaire*5/100),'Pas de commission')
   FROM exe_employe;
```

DECODE

SYNTAX:

DECODE(expr, valeur1, resultat1 [,valeur2, resultat2] ... [, default])

EMPLOI :

En fonction de la valeur de expr, retourne pour valeur1 le resultat1, pour valeur2 le resultat2, ainsi de suite, et dans les autres cas la valeur par défaut.

EXEMPLE :

```
SELECT emp_nom,
         DECODE(emp_etatCivil, 1, 'Célibataire',
                               2, 'Marié',
                               3, 'Divorcé',
                               4, 'Veuf',
                                  'Inconnu') AS "État civil"  FROM exe_employe;
```

CASE VS DECODE

SYNTAX:

```
CASE { simple_case_expression
     | searched_case_expression
     }
     [ else_clause ]
END
```

EMPLOI :

L'expression CASE permet de représenter un IF … THEN … ELSE tout en évitant d'invoquer une fonction (par ex. : DECODE).
La différence réside dans le fait que CASE respecte mieux les standards de codage et est plus simple à utiliser. Il a été introduit pour remplacer DECODE.

EXEMPLES :

```
SELECT emp_nom, CASE emp_etatCivil
                        WHEN 1 THEN 'Célibataire'
                        WHEN 2 THEN 'Marié'
                        WHEN 3 THEN 'Divorcé'
                        WHEN 4 THEN 'Veuf'
                        ELSE        'Inconnu'
                END AS "État civil"
   FROM exe_employe;
```

```
SELECT emp_nom, CASE
                  WHEN emp_salaire < 4444 THEN 'Petit salaire'
                  WHEN emp_salaire > 7777 THEN 'Grand salaire'
                  ELSE                         'Salaire moyen'
                END AS "Salaire"
   FROM exe_employe;
```

Voir la documentation Oracle pour les autres fonctions !

Expressions de type date/temps

Oracle fournit un type de données spécial, DATE, pour gérer les données de type date et heure. Ce type possède son propre format interne (il ne requiert que 7 octets pour le stockage) et fournit assez d'espace pour le stockage du siècle, de l'année, du mois, du jour, de l'heure, des minutes et des secondes.

- Recherche de la date et de l'heure courantes avec **SYSDATE**

  Exemple : `SELECT SYSDATE FROM dual;`

Formats de date :

Chacun des éléments (siècle, année, mois, jour, heures, minutes et secondes) de type DATE, peut être extrait et formaté indépendamment.

Le format utilisé par défaut est défini dans les paramètres de la base de données (`NLS_DATABASE_PARAMETERS`) ainsi que dans les paramètres de la session courante (`NLS_SESSION_PARAMETERS`). Le format de la date peut être changé dans la session courante à l'aide de l'instruction suivante :

```
ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YYYY';
```

Conversion d'une date en chaîne de caractères :

La syntaxe d'utilisation de cette fonction est : TO_CHAR(valeur date, format date)

Les variables sont définies ainsi :

- valeur date est une constante de type date, une valeur de date d'une colonne ou une valeur de date retournée par une fonction intégrée.
- format date est un format de date valide.

Vous avez aussi la possibilité de produire un élément de date ou d'heure dans différents formats. Vous devez utiliser la fonction TO CHAR avec un format de date/heure pour convertir la valeur de date interne en une valeur de type caractère.

Une fois qu'une valeur de date a été convertie en une chaîne au moyen de la fonction TO_CHAR, vous pouvez l'utiliser comme argument dans d'autres fonctions de manipulation de chaînes.

Exemple :

```
SELECT SUBSTR(TO_CHAR(SYSDATE,'dd/month/yyyy'),4,5) AS "5 premières
lettres du mois" FROM dual;
```

retourne les 5 premières lettres du mois courant.

Conversion d'une chaîne de caractères en date :

La syntaxe d'utilisation de cette fonction est : TO_DATE (valeur chaîne, format date)

Les variables sont définies ainsi :

- valeur chaîne est une valeur littérale de type chaîne, une chaîne d'une colonne ou une valeur chaîne retournée par une fonction intégrée.
- format date est un format de date valide.

Exemple :

```
SELECT TO_DATE('1 janvier 2020') FROM dual;
```

Type unique pour les dates et les heures :

Exemple :

```
SELECT TO_CHAR(com_date,'dd mm yyyy') FROM exe_commande WHERE com_no=123;
```

retourne 12 03 2020 mais il est également possible de connaître l'heure de la commande.

```
SELECT TO_CHAR(com_date,'hh mi ss') FROM exe_commande WHERE com_no=123;
```

retourne 17 35 42

Calcul de la différence entre deux dates

Un autre avantage du type DATE est qu'il permet les opérations arithmétiques sur les dates. Vous pouvez ajouter ou soustraire les jours d'une date existante.

Utilisation de fonctions avec les dates

LAST_DAY(date)	Retourne le dernier jour du mois.
ADD_MONTHS(date, nombre)	Retourne la date issue de l'addition du nombre de mois de la date passée en paramètre.
MONTHS_BETWEEN(date1, date2)	Retourne la différence en mois entre les deux dates passées en paramètres.
NEXT_DAY(date,jour)	Retourne la date du premier jour spécifié postérieur à la date donnée en premier argument.
GREATEST(date1,date2)	Retourne la plus récente des dates entre date1 et date2.
LEAST(date1,date2)	retourne la plus ancienne des dates entre date1 et date2.

Tableau des différents formats du type date

Format	Description	**Plage de valeurs**
ss	Seconde	0 – 59
sssss	Secondes après minuit	0 – 86399
MI	Minute	0 – 59
HH	Heure	0 – 12
HH12		
HH24	Heure militaire	0 – 23
DAY	Jour de la semaine en entier	DIMANCHE – SAMEDI
DY	Jour de la semaine abrégé	DI – SA
D	Jour de la semaine	1 – 7
DD	Jour du mois	01 – 31 (selon le mois)
FMDD	Jour du mois	1 – 31 (selon le mois)
DDD	Jour de l'année	1 – 366 (selon l'année)
DDTH	Jour du mois (rang)	Par exemple : 5ème
DDSP	Jour du mois en lettres	Par exemple : dix sept
DDSPTH	Combinaison des deux précédents	Par exemple : cinquième
MM	Numéro du mois	1 – 12
MON	Mois abrégé	JAN-DEC
MONTH	Mois en entier (arrondi à partir du 16ème jour)	JANVIER – DECEMBRE
Y	Dernier chiffre de l'année	0 – 9
YY	Deux derniers chiffres de l'année	00 – 99
YYY	Trois derniers chiffres de l'année	000 – 999
YYYY	Année en entier numérique	Par exemple 2000
SYYYY	Année (arrondie à partir 1er juillet)	
YEAR	Année complète en lettres	Par exemple : DEUX MILLE
CC	Siècle	Par exemple
Q	Trimestre (arrondi après le 16ème jour du second mois du trimestre)	1 – 4
J	Date julienne	Par exemple : 2448000
W	Semaine du mois	15
WW	Semaine de l'année	1 – 52
AM	Avant midi	AM
PM	Après midi	PM

Remarque : certains formats peuvent être mentionnés en minuscules ou en majuscules (DAY, Day ou day). Le résultat affiché respectera la casse employée pour le format.

La clause Group by

```
SELECT [DISTINCT | ALL] { *
                        | { [schema.]{table | view | snapshot}.*
                          | expr }  [ [AS] c_alias ]
                       [, { [schema.]{table | view | snapshot}.*
                          | expr } [ [AS] c_alias ]  ] ... }
    FROM [schema.]{table | view | snapshot}[@dblink] [t_alias]

      [, [schema.]{table | view | snapshot}[@dblink] [t_alias] ] ...
    [WHERE condition ]
    [GROUP BY expr [, expr] ... [HAVING condition] ]
    [{UNION | UNION ALL | INTERSECT | MINUS} SELECT command ]
    [ORDER BY {expr|position} [ASC | DESC]
           [, {expr|position} [ASC | DESC]] ...]
    [FOR UPDATE [OF [[schema.]{table | view}.]column

                  [, [[schema.]{table | view}.]column] ...] [NOWAIT] ]
```

Utile pour obtenir un seul résultat par valeur du groupe définit (sur une expression) dans une requête.

Exemple :

Pour obtenir la somme et la moyenne des salaires par département, on peut écrire la requête suivante :

```
SELECT emp_dep_no, SUM(emp_salaire), AVG(emp_salaire)
   FROM exe_employe
   GROUP BY emp_dep_no;
```

Le résultat sera :

	EMP_DEP_NO	SUM(EMP_SALAIRE)	AVG(EMP_SALAIRE)
1	1	5000	5000
2	3	11000	5500
3	4	13000	6500
4	5	11000	5500
5		6000	6000

GROUP BY : La clause HAVING

La clause HAVING permet d'effectuer des restrictions portant sur les groupes résultant d'un GROUP BY.

En effet, si la clause WHERE effectue la restriction au niveau des enregistrements des tables impliquées dans la requête, la clause HAVING s'applique au résultat du regroupement après l'application du WHERE.

Ainsi, ces deux clauses peuvent parfaitement cohabiter dans une requête SQL :

- le « WHERE » limite le nombre d'enregistrements faisant partie des groupes
- le « HAVING » limite le nombre de groupe obtenus dans le résultat final

Exemple :

Si on désire obtenir le salaire moyen par département, et ce uniquement pour les employés gagnant plus de 4'000.-, et pour les départements dont le salaire moyen est inférieur à 7'000.-, il faut utiliser à la fois le WHERE et le HAVING :

1) il faut sélectionner uniquement les employés dont le salaire dépasse 4'000.- (ici on emploie le WHERE)
2) il faut sélectionner les groupes dont la moyenne est inférieure à 7'000.- (ici on emploie le HAVING)

Ainsi, si on décompose la requête, on aura d'abord une restriction des enregistrements équivalente à la requête (employés dont le salaire dépasse 4'000.-) :

```
SELECT emp_dep_no, emp_salaire
   FROM exe_employe
   WHERE emp_salaire > 4000;
```

	EMP_DEP_NO	EMP_SALAIRE
1	1	5000.00
2		6000.00
3	4	8000.00
4	5	5000.00
5	4	5000.00
6	3	7000.00
7	5	6000.00

Puis un regroupement de ces données par rapport au département :

```
SELECT emp_dep_no, AVG(emp_salaire)
   FROM exe_employe
   WHERE emp_salaire > 4000
   GROUP BY emp_dep_no;
```

	EMP_DEP_NO	AVG(EMP_SALAIRE)
1	1	5000
2	3	7000
3	4	6500
4	5	5500
5		6000

Et enfin, une sélection des départements correspondant à la condition portant sur les groupes (départements dont la moyenne est inférieure à 7'000.-) :

```
SELECT emp_dep_no, AVG(emp_salaire)
   FROM exe_employe
   WHERE emp_salaire > 4000
   GROUP BY emp_dep_no
   HAVING AVG(emp_salaire) < 7000;
```

	EMP_DEP_NO	AVG(EMP_SALAIRE)
1	1	5000
2	4	6500
3	5	5500
4		6000

Les jointures

Théoriquement, le concept de jointure correspond à la construction d'une nouvelle relation basée sur deux autres relations (tables ou sous-requêtes SQL).

Une requête SQL peut contenir plusieurs jointures, à chaque fois entre deux relations. Ce principe permet de combiner les données provenant de nombreuses tables (jusqu'à 256…) dans une seule et même requête. Une jointure permet donc de combiner les colonnes de plusieurs tables. Il n'existe aucune limitation quant à l'ordre des colonnes sélectionnées. Il n'est même pas nécessaire de sélectionner des colonnes dans chaque table de la jointure.

En principe, deux relations peuvent être combinées à l'aide d'une jointure si elles contiennent des colonnes qui peuvent être mises en correspondance. Cela se fait le plus souvent en utilisant la correspondance entre une colonne de clé étrangère et la clé primaire référencée par celle-ci.

> ***Définition***
>
> La jointure de deux relations R1 et R2 selon un critère généralisé C est l'ensemble des tuples du produit cartésien R1xR2 satisfaisant le critère C.

Une jointure est donc un sous ensemble du produit cartésien de deux relations.

La norme SQL comprend plusieurs types de jointures normalisées que l'on peut regrouper de la façon suivante :

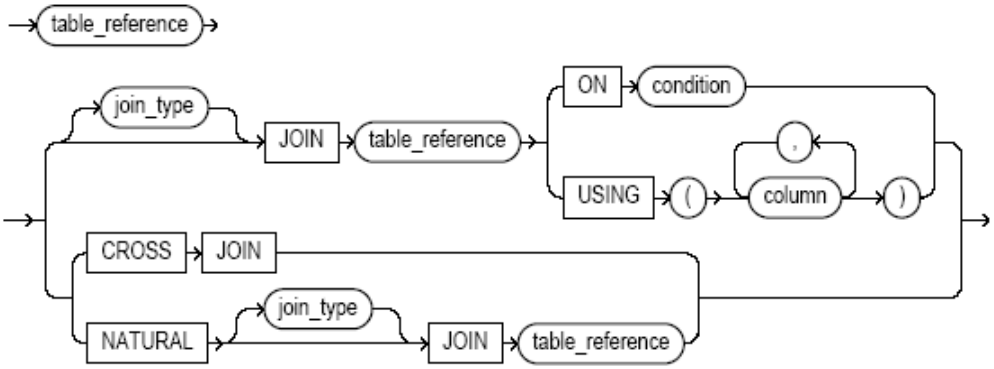
- La Jointure Croisée ou « cross join » ou Produit cartésien
- La Jointure Naturelle ou équi-jointure ou « inner join »
  (par opposition, il existe aussi la non équi-jointure,
  et par extension l'autojointure)
- La Jointure Externe ou « outer join »
- La Jointure Union

Définitions

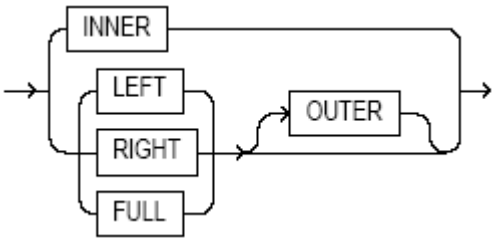
Equi-jointure	R1 et R2 sur les attributs CR1, CR2 est la jointure selon le critère CR1 = CR2 (égalité des valeurs des attributs)
Jointure naturelle	C'est une équi-jointure à laquelle on a enlevé les attributs redondants
Non equi-jointure	R1 et R2 sur les attributs CR1, CR2 est la jointure selon le critère CR1 [>, <, >=, <=, #] CR2 (non-égalité des valeurs des attributs)
Autojointure	R selon Ci est la jointure de R avec elle-même selon le critère : CR1 = CR2
Semi-jointure	Ne conserve que les attributs d'une des deux relations (avec seulement les tuples participant à la jointure)
Jointure externe	Pour ne pas perdre de l'information rajoute, du côté de la première relation, les tuples ne participant pas à la jointure en rajoutant des valeurs « null » pour les attributs correspondant de l'autre relation.
Jointure union	Il s'agit simplement de l'opération ensembliste d'union

Syntaxe Oracle de la clause de jointure :

Dans la clause FROM :



Types de jointures (join_type):



Quelques conseils importants

Dans la mesure du possible, utilisez toujours un opérateur de jointure normalisé (mot clef JOIN).

En effet :

- Les jointures faites dans la clause WHERE (ancienne syntaxe de 1986 !) ne permettent pas de faire la distinction de prime abord entre ce qui relève de la restriction et ce qui relève de la jointure.
- Il est à priori absurde de vouloir filtrer dans le WHERE (ce qui restreint les données du résultat) et de vouloir "élargir" ce résultat par une jointure dans la même clause WHERE.
- La lisibilité des requêtes est plus grande en utilisant la syntaxe à base de JOIN, en isolant ce qui concerne la jointure, mais aussi en isolant avec clarté chaque condition de jointures entre chaque couple de table
- L'optimisation d'exécution de la requête est souvent plus pointue du fait de l'utilisation du JOIN.
- Lorsque l'on utilise l'ancienne syntaxe et que l'on supprime la clause WHERE a des fins de tests, le moteur SQL réalise le produit cartésien des tables ce qui revient la plupart du temps à mettre à genoux le serveur !

Jointure croisée ou Produit cartésien

Notation : X=R1 x R2

Définition

Le produit cartésien de deux relations R1 et R2 est une relation ayant pour attributs tous les attributs de R1 et de R2 et dont les tuples sont constitués de toutes les concaténations possibles d'un tuple de R1 à un tuple de R2.

Synthèse

- Les relations peuvent ne pas être compatibles
- Concaténation des tuples de R1 avec ceux de R2
    - peut être utilisée comme étape intermédiaire pour jointure

Exemple

	EMP_NO	EMP_NOM	EMP_DEP_NO
1	1	BON	1
2	2	REMORD	3
3	3	TERIEUR	
4	4	PROVISTE	4
5	6	DISSOIRE	4

	DEP_NO	DEP_NOM
1	1	RH
2	2	Achat
3	3	Vente
4	4	Marketing

X = exe_employe x exe_dept

	EMP_NO	EMP_NOM	EMP_DEP_NO	DEP_NO	DEP_NOM
1	1	BON	1	1	RH
2	2	REMORD	3	1	RH
3	3	TERIEUR		1	RH
4	4	PROVISTE	4	1	RH
5	6	DISSOIRE	4	1	RH
6	1	BON	1	2	Achat
7	2	REMORD	3	2	Achat
8	3	TERIEUR		2	Achat
9	4	PROVISTE	4	2	Achat
10	6	DISSOIRE	4	2	Achat
11	1	BON	1	3	Vente
12	2	REMORD	3	3	Vente
13	3	TERIEUR		3	Vente
14	4	PROVISTE	4	3	Vente
15	6	DISSOIRE	4	3	Vente
16	1	BON	1	4	Marketing
17	2	REMORD	3	4	Marketing
18	3	TERIEUR		4	Marketing
19	4	PROVISTE	4	4	Marketing
20	6	DISSOIRE	4	4	Marketing

Syntaxe SQL :

Commande :

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

Exemple :

```
SELECT emp_no, emp_nom, dep_no, dep_nom
  FROM exe_employe
  CROSS JOIN exe_dept ;
```

Jointure naturelle ou équi-jointure

Exemple

	EMP_NO	EMP_NOM	EMP_DEP_NO
1	1	BON	1
2	2	REMORD	3
3	3	TERIEUR	
4	4	PROVISTE	4
5	6	DISSOIRE	4

	DEP_NO	DEP_NOM
1	1	RH
2	2	Achat
3	3	Vente
4	4	Marketing

X = exe_employe x exe_dept

	EMP	EMP_NOM	EMP	DEP	DEP_NOM
1	1	BON	1	1	RH
2	2	REMORD	3	1	RH
3	3	TERIEUR		1	RH
4	4	PROVISTE	4	1	RH
5	6	DISSOIRE	4	1	RH
6	1	BON	1	2	Achat
7	2	REMORD	3	2	Achat
8	3	TERIEUR		2	Achat
9	4	PROVISTE	4	2	Achat
10	6	DISSOIRE	4	2	Achat
11	1	BON	1	3	Vente
12	2	REMORD	3	3	Vente
13	3	TERIEUR		3	Vente
14	4	PROVISTE	4	3	Vente
15	6	DISSOIRE	4	3	Vente
16	1	BON	1	4	Marketing
17	2	REMORD	3	4	Marketing
18	3	TERIEUR		4	Marketing
19	4	PROVISTE	4	4	Marketing
20	6	DISSOIRE	4	4	Marketing

X = Jointure (exe_employe, exe_dept)
/ dep_no = emp_dep_no

	EMP_NO	EMP_NOM	EMP_DEP_NO	DEP_NO	DEP_NOM
1	1	BON	1	1	RH
2	2	REMORD	3	3	Vente
3	4	PROVISTE	4	4	Marketing
4	6	DISSOIRE	4	4	Marketing

Syntaxe SQL :

Commande :

```
SELECT ...
  FROM <table gauche>
  [INNER] JOIN <table droite> ON condition_de_jointure
```

Exemple :

```
SELECT emp_no, emp_nom, dep_no, dep_nom
  FROM exe_employe
  INNER JOIN exe_dept ON dep_no = emp_dep_no
```

Non-équijointure

Exemple

	EMP_NO	EMP_NOM	EMP_DEP_NO
1	1	BON	1
2	2	REMORD	3
3	3	TERIEUR	
4	4	PROVISTE	4
5	6	DISSOIRE	4

	DEP_NO	DEP_NOM
1	1	RH
2	2	Achat
3	3	Vente
4	4	Marketing

X = exe_employe x exe_dept

	EMP	EMP_NOM	EMP	DEP	DEP_NOM
1	1	BON	1	1	RH
2	2	REMORD	3	1	RH
3	3	TERIEUR		1	RH
4	4	PROVISTE	4	1	RH
5	6	DISSOIRE	4	1	RH
6	1	BON	1	2	Achat
7	2	REMORD	3	2	Achat
8	3	TERIEUR		2	Achat
9	4	PROVISTE	4	2	Achat
10	6	DISSOIRE	4	2	Achat
11	1	BON	1	3	Vente
12	2	REMORD	3	3	Vente
13	3	TERIEUR		3	Vente
14	4	PROVISTE	4	3	Vente
15	6	DISSOIRE	4	3	Vente
16	1	BON	1	4	Marketing
17	2	REMORD	3	4	Marketing
18	3	TERIEUR		4	Marketing
19	4	PROVISTE	4	4	Marketing
20	6	DISSOIRE	4	4	Marketing

X = Jointure (exe_employe, exe_dept)
/ dep_no **<>** emp_dep_no

	EMP_NO	EMP_NOM	EMP_DEP_NO	DEP_NO	DEP_NOM
1	2	REMORD	3	1	RH
2	4	PROVISTE	4	1	RH
3	6	DISSOIRE	4	1	RH
4	1	BON	1	2	Achat
5	2	REMORD	3	2	Achat
6	4	PROVISTE	4	2	Achat
7	6	DISSOIRE	4	2	Achat
8	1	BON	1	3	Vente
9	4	PROVISTE	4	3	Vente
10	6	DISSOIRE	4	3	Vente
11	1	BON	1	4	Marketing
12	2	REMORD	3	4	Marketing

Syntaxe SQL :

Commande :

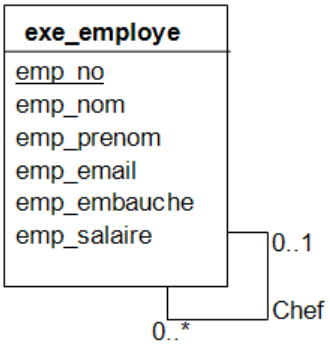
```
SELECT ...
  FROM <table gauche>
  [INNER] JOIN <table droite> ON condition_de_jointure
```

Exemple :

```
SELECT emp_no, emp_nom, dep_no, dep_nom
  FROM exe_employe
  INNER JOIN exe_dept ON dep_no <> emp_dep_no
```

Autojointure

L'autojointure est fréquemment utilisée sur des tables qui possèdent des associations réflexives.



Exemple autojointure sur la table exe_employe

	EMP_NO	EMP_NOM	EMP_EMP_NO
1	1	BON	
2	2	REMORD	1
3	3	TERIEUR	1
4	4	PROVISTE	1
5	6	DISSOIRE	2

X = AutoJointure (chef.emp_no = empl.emp_emp_no)

	EMP_NO	EMP_NOM	EMP_EMP_NO	CHEF_NO	CHEF
1	2	REMORD	1	1	BON
2	3	TERIEUR	1	1	BON
3	4	PROVISTE	1	1	BON
4	6	DISSOIRE	2	2	REMORD

Syntaxe SQL :

Commande :

```
SELECT ...
  FROM <table gauche>
  [INNER] JOIN <table droite>
    ON condition_de_jointure
```

Exemple :

```
SELECT empl.emp_no, empl.emp_nom, empl.emp_emp_no,
       chef.emp_no AS "CHEF_NO", chef.emp_nom  AS "CHEF"
   FROM exe_employe empl
   INNER JOIN exe_employe chef ON chef.emp_no = empl.emp_emp_no;
```

Jointure Externe

Définition

La jointure externe de deux relations R1 et R2 est obtenue en deux étapes de la manière suivante :

1. On effectue une jointure de R1 et de R2
2. On ajoute à la relation obtenue en (1) les tuples de R1 et de R2 qui ne participent pas à la jointure complétés avec des valeurs nulles pour les attributs de l'autre relation.

Exemple

	EMP_NO	EMP_NOM	EMP_DEP_NO
1	1	BON	1
2	2	REMORD	3
3	3	TERIEUR	
4	4	PROVISTE	4
5	6	DISSOIRE	4

	DEP_NO	DEP_NOM
1	1	RH
2	2	Achat
3	3	Vente
4	4	Marketing

Jointure Naturelle (exe_employe, exe_dept)
/ dep_no = emp_dep_no

	EMP_	EMP_NOM	EMP_DE	DEP_	DEP_NOM
1	1	BON	1	1	RH
2	2	REMORD	3	3	Vente
3	4	PROVISTE	4	4	Marketing
4	6	DISSOIRE	4	4	Marketing

Joint-**Ext-FULL** (exe_employe, exe_dept)
/ dep_no = emp_dep_no

	EMP_	EMP_NOM	EMP_DEI	DEP_	DEP_NOM
1	1	BON	1	1	RH
2	2	REMORD	3	3	Vente
3	4	PROVISTE	4	4	Marketing
4	6	DISSOIRE	4	4	Marketing
5	3	TERIEUR			
6				2	Achat

Joint-**Ext-LEFT** (exe_employe, exe_dept)
/ dep_no = emp_dep_no

	EMP_	EMP_NOM	EMP_DEI	DEP_	DEP_NOM
1	1	BON	1	1	RH
2	2	REMORD	3	3	Vente
3	4	PROVISTE	4	4	Marketing
4	6	DISSOIRE	4	4	Marketing
5	3	TERIEUR			

Joint-**Ext-RIGHT** (exe_employe, exe_dept)
/ dep_no = emp_dep_no

	EMP_	EMP_NOM	EMP_DEI	DEP_	DEP_NOM
1	1	BON	1	1	RH
2	2	REMORD	3	3	Vente
3	4	PROVISTE	4	4	Marketing
4	6	DISSOIRE	4	4	Marketing
5				2	Achat

Syntaxe SQL :

Commande :

```
SELECT ...
  FROM <table gauche>
  {FULL | LEFT | RIGHT} [OUTER] JOIN <table droite>
    ON condition_de_jointure
```

Exemple :

```
SELECT emp_no, emp_nom, emp_dep_no, dep_no, dep_nom
   FROM exe_employe
   FULL OUTER JOIN exe_dept ON dep_no = emp_dep_no;
```

Opérateurs ensemblistes

Pour l'exemple utilisons les tables exe_employe et exe_client suivantes :

EMP_NO	EMP_NOM
1	BON
2	REMORD
3	TERIEUR
4	PROVISTE
6	DISSOIRE

CLI_NO	CLI_NOM
111	DISSOIRE
222	REMORD
333	DORSA

Opérateurs ensemblistes

Opération	Opérateur	Exemple	Résultat

Union	UNION	`SELECT emp_nom FROM exe_employe`
		`  UNION`
		`SELECT cli_nom FROM exe_client;`
			EMP_NOM
			BON
			DISSOIRE
			DORSA
			PROVISTE
			REMORD
			TERIEUR

Union complète (avec enregistrements à double)	UNION ALL	`SELECT emp_nom FROM exe_employe`
		`  UNION ALL`
		`SELECT cli_nom FROM exe_client;`
			EMP_NOM
			BON
			REMORD
			TERIEUR
			PROVISTE
			DISSOIRE
			DISSOIRE
			REMORD
			DORSA

Intersection	INTERSECT	`SELECT emp_nom FROM exe_employe`
		`  INTERSECT`
		`SELECT cli_nom FROM exe_client;`
			EMP_NOM
			DISSOIRE
			REMORD

Différence	MINUS	`SELECT emp_nom FROM exe_employe`
		`  MINUS`
		`SELECT cli_nom FROM exe_client;`
		
		`SELECT cli_nom FROM exe_client`
		`  MINUS`
		`SELECT emp_nom FROM exe_employe;`
			EMP_NOM
			BON
			PROVISTE
			TERIEUR
			
			CLI_NOM
			DORSA

SOUS-REQUETES SQL

Une sous-requête est une instruction SELECT imbriquée dans une instruction SELECT ou dans une autre sous-requête. La sous-requête s'écrit entre parenthèses ( ).

a) Une sous-requête pour trouver une valeur

```
Exemple : Obtenir la liste des employés travaillant dans le même département
          que l'employé nommé 'BON' :
```

Étant donné que « `SELECT emp_dep_no FROM exe_employe WHERE emp_nom = 'BON'` » nous retournera le numéro de département de l'employé 'Bon', c'est-à-dire la valeur 1, il suffit de remplacer cette valeur 1 du « `SELECT emp_nom, emp_dep_no FROM exe_employe WHERE emp_dep_no=`**1** » par la sous-requête écrite plus haut, en la mettant entre parenthèse :

```
SELECT emp_nom, emp_dep_no FROM exe_employe
   WHERE emp_dep_no = (SELECT emp_dep_no FROM exe_employe WHERE emp_nom = 'BON');
```

b) Autres opérateurs utilisables avec une sous-requête

Vous pouvez utiliser trois formes de syntaxe pour créer une sous-requête :

- `expression [NOT] IN (instruction_sql)`
- `expression et opérateur [ANY | ALL | SOME] (instruction_sql)`
- `[NOT] EXISTS (instruction_sql)`

`expression`	Une expression recherchée dans le jeu d'enregistrements résultant de la sous-requête.
`expression et opérateur`	Une expression et un opérateur de comparaison (<, >, …) qui compare l'expression avec les résultats de la sous-requête. *N.B : Si une sous-requête retourne UN seul tuple on peut alors utiliser les opérateurs de comparaison : =, <, >, >=, <=, <>*
`instruction_sql`	Une instruction SELECT respectant le même format et les mêmes règles que toute autre instruction SELECT. Elle doit figurer entre parenthèses.

Vous pouvez utiliser une sous-requête à la place d'une expression dans la liste de champs d'une instruction **SELECT**, dans une clause **FROM**, une clause **WHERE** ou dans une clause **HAVING**.

Dans une sous-requête, vous utilisez une instruction SELECT pour fournir un jeu d'une ou plusieurs valeurs spécifiques à évaluer dans l'expression de la clause WHERE ou HAVING.

[NOT] IN - Utilisez l'opérateur IN pour n'extraire de la requête principale que les enregistrements pour lesquels un des enregistrements de la sous-requête contient une valeur identique. Inversement, vous pouvez utiliser NOT IN pour n'extraire que les enregistrements de la requête principale pour lesquels aucun enregistrement de la sous-requête ne contient de valeur identique.

```
Exemple : Obtenir la liste des employés habitant dans une ville qui existe
comme ville d'un département :

SELECT * FROM exe_employe
  WHERE emp_ville IN (SELECT dep_ville FROM exe_dept);
```

[ANY | ALL | SOME] - Utilisez l'opérateur ANY ou SOME, qui sont synonymes, pour extraire de la requête principale des enregistrements qui répondent à la comparaison avec au moins un enregistrement extrait de la sous-requête. Utilisez l'attribut ALL pour n'extraire de la requête principale que les enregistrements qui répondent à la comparaison avec tous les enregistrements extraits de la sous-requête.

```
Exemple : Obtenir la liste des employés qui gagnent plus que tous les
employés du département 3 :

SELECT * FROM exe_employe
  WHERE emp_salaire > ALL (SELECT emp_salaire FROM exe_employe
                                                  WHERE emp_dep_no=3);
```

[NOT] EXISTS - Utilisez l'attribut EXISTS (avec le mot réservé facultatif NOT) dans des comparaisons vrai/faux pour déterminer si la sous-requête retourne des enregistrements. L'opérateur EXISTS retourne TRUE si la sous-requête retourne au moins une ligne, FALSE si aucune ligne n'est renvoyée. L'opérateur EXISTS est utile dans les situations où l'on n'est pas directement intéressé par les valeurs de colonnes retournées par la sous-requête.

Vous pouvez faire référence dans la sous-requête à des colonnes de la requête principale. On parle alors de sous-requête corrélée (c'est un moyen de simuler une boucle). En effet, pour chaque enregistrement parcouru par la requête principale, la sous-requête est évaluée différemment en fonction des valeurs actuelles de la requête principale.

```
Par exemple, on pourrait afficher ainsi la liste des départements dans
lesquels il y a au moins un employé :

SELECT * FROM exe_dept
  WHERE EXISTS (SELECT * FROM exe_employe WHERE emp_dep_no = dep_no);
```

Quelques précisions :

- La sous-requête (requête interne) est exécutée avant la requête principale.
- Le résultat de la sous-requête est utilisé par la requête principale
- Les sous-requêtes sont incluses entre parenthèses
- Il n'y a pas de clause ORDER BY dans la requête interne
- Si une sous-requête retourne un seul tuple
  → on peut alors utiliser les opérateurs de comparaison : =, <, >, >=, <=, <>
- Les sous-requêtes sont moins performantes qu'une requête avec jointure par exemple. Il convient donc de les utiliser uniquement lorsque le problème le nécessite.

Pour plus d'information, c.f. chapitre 9-7 de la documentation Oracle (Using subqueries)