

Systemes de gestion de bases de données

Année universitaire 2015-2016

Objectifs

- Approfondir les connaissances en SGBD relationnels
- Maîtriser le langage SQL dans son niveau avancé
- Maîtriser la programmation côté serveur avec le langage le langage PL/SQL d'ORACLE et savoir le contexte d'application.
- Développer des procédures et fonctions stockées ,des déclencheurs et découvrir les packages des fonctions offertes par ORACLE

Plan du module

Partie I SQL Avancé	Chapitre1: Langage de définition de données
	Chapitre 2: Langage de manipulation de données
	Chapitre 3: Langage d'interrogation de données
Partie II Programmation PL/SQL	Chapitre 1: Structure d'un bloc anonyme PL/SQL
	Chapitre 2: Les curseurs
	Chapitre 3 : Les sous programmes
	Chapitre 4: les exceptions
	Chapitre 5: Les déclencheurs
	Chapitre 6: les packages Pl/SQL

Chapitre 1: Le langage de définition de données

Objectifs

- Manipulation des tables
- Manipulation des vues
- Manipulation des synonymes
- Manipulation des index
- Manipulation des séquences

Structured Query Language (SQL)

- ❑ Un langage de gestion de base de données à base de requêtes qui permet la manipulation et l'extraction des données d'une base de données.
- ❑ SQL est un langage de type « déclaratif » non procédural: simple interrogation directe possible par les utilisateurs et réponses rapides à des questions non prévues par l'application.
- ❑ SQL offre 4 sous langages:
 - LDD (langage de définition de données)
 - LMD (langage de manipulation de données)
 - LCD (langage de contrôle de données)
 - LID (langage d'interrogation de données)

Types de données

Data Type	Description
VARCHAR2(size)	Données de type caractère de longueur variable
CHAR(size)	Données de type caractère de longueur fixe
NUMBER(p,s)	Données numériques de longueur variable
DATE	Valeur de date et d'heure
LONG	Données de type caractère de longueur variable (jusqu'à 2 GB)
INTEGER	Variante du type NUMBER qui s'écrit sur 4 octets

Création de table

Syntaxe

```
CREATE TABLE NomTable (  
    NomColonne1 TypeDonnée1,  
    NomColonne2 TypeDonnée2,  
    ...  
);
```

Exemple:

```
CREATE TABLE Etudiant ( Netudiant number,  
    NomEtud varchar2(20),  
    PrenomEtud varchar2(50)  
);
```

```
Desc[ribe] Etudiant;
```


Création de table à partir de table

Syntaxe

```
CREATE TABLE NomTable AS  
                SELECT * FROM Nom_Table1  
                ;
```

Exemple:

```
CREATE TABLE Etud AS  
                SELECT * From Etudiant;
```

```
Desc[ribe] Etud;
```

Gestion des contraintes

- ❑ Il s'agit de règles qui doivent être vérifiées en permanence.
- ❑ Les types de contrainte suivants sont pris en charge:
 - NOT NULL
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK
- ❑ les contraintes peuvent être créées au moment de la création des tables ou bien après leur création

Création des contraintes

▪ Contrainte de colonne

```
Create Table NomTable  
( colonne1 type1[CONSTRAINT nom_contrainte] Type_de_contrainte,  
  colonne2 type2  
);
```

▪ Contrainte de table

```
Create Table NomTable  
( colonne1 type1,  
  colonne2 type2,  
  CONSTRAINT nom_contrainte Type_de_contrainte nom_colonne  
);
```

Création des contraintes

Exemple 1:

```
CREATE TABLE Etudiant
( Netudiant number CONSTRAINT pk_Netudiant PRIMARY KEY,
  NomEtud varchar2(20),
  PrenomEtud varchar2(50));
```

Exemple 2:

```
CREATE TABLE Etudiant
( Netudiant number,
  NomEtud varchar2(20),
  PrenomEtud varchar2(50),
  CONSTRAINT pk_Netudiant PRIMARY KEY(Netudiant)
);
```

La contrainte NOT NULL

- ❑ Interdit l'insertion de valeur NULL pour l'attribut correspondant

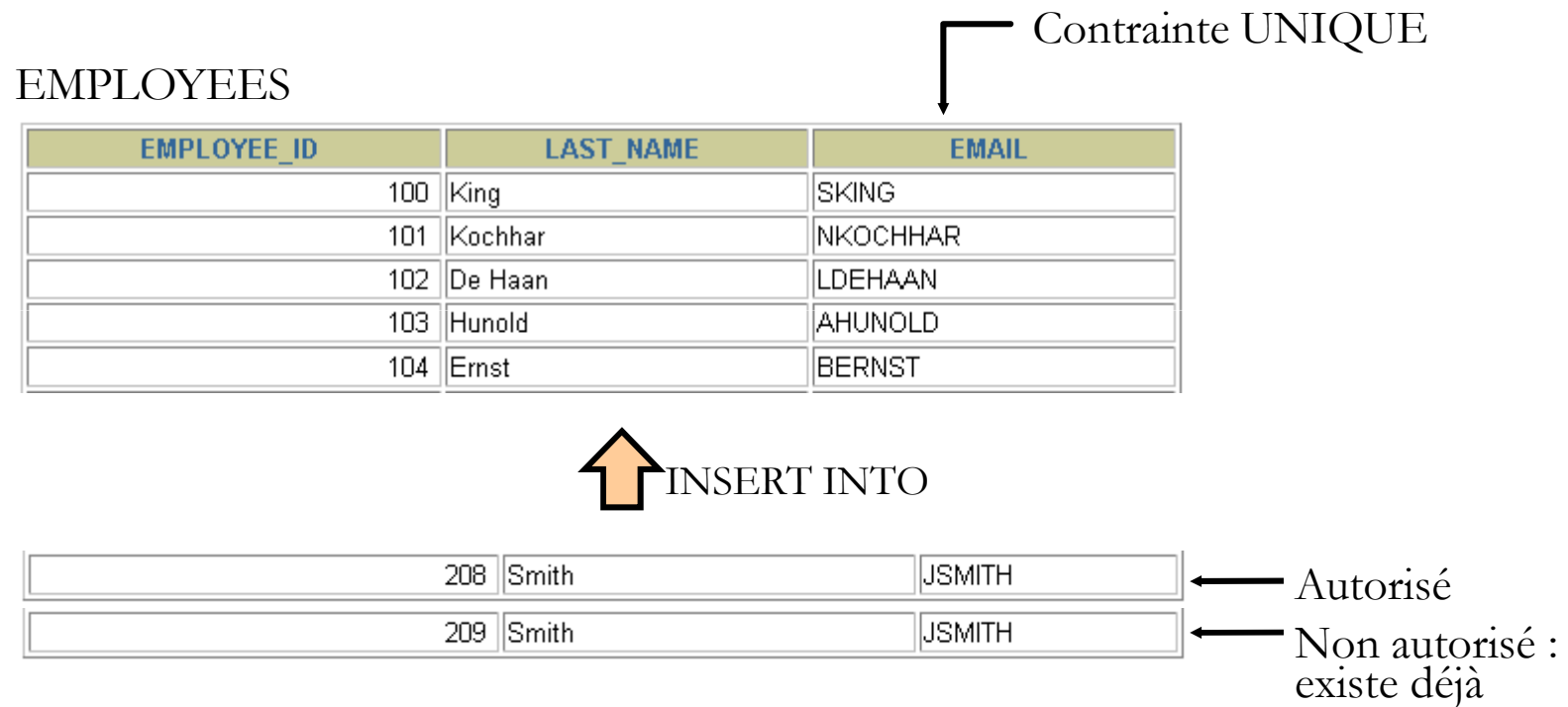
EMPLOYEE_ID	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	DEPARTMENT_ID
100	King	SKING	515.123.4567	17-JUN-87	AD_PRES	24000	90
101	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	17000	90
102	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	17000	90
103	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	9000	60
104	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	6000	60
178	Grant	KGRANT	011.44.1644.429263	24-MAY-99	SA_REP	7000	
200	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	4400	10

...

20 rows selected.

Absence de NOT NULL
contrainte (n'importe quelle
ligne peut contenir une valeur
NULL pour cette colonne)

La contrainte UNIQUE



La contrainte PRIMARY KEY

PRIMARY KEY

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

...

Non autorisé
(valeur null)

↑ INSERT INTO

	Public Accounting		1400
50	Finance	124	1500

Non autorisé
(50 existe déjà)

La contrainte FOREIGN KEY

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500

...

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
102	De Haan	90
103	Hunold	60
104	Ernst	60
107	Lorentz	60

...



INSERT INTO

200	Ford	9
201	Ford	60

Non autorisé
(9 n'existe pas)

Autorisé

Définition FOREIGN KEY

CONSTRAINT nom_contrainte **FOREIGN KEY** (nom_colonne)

REFERENCES *Nom_Table* (nom_colonne)

[**ON DELETE** ***CASCADE/SET NULL***]

- **ON DELETE CASCADE** : supprime les lignes dépendantes dans la table enfant lorsqu'une ligne de la table parent est supprimée.
- **ON DELETE SET NULL**: convertit les valeurs des clés étrangères dépendantes en valeurs NULL. Cette contrainte ne peut être exécutée que si toutes les colonnes de clé étrangère de la table cible acceptent des valeurs NULL.

La contrainte CHECK

- ❑ Définit une condition à laquelle chaque ligne doit répondre.

```
..., salary    NUMBER(2)  
      CONSTRAINT emp_salary_min  
      CHECK (salary > 0),...
```

- ❑ Dans le cas où on doit comparer 2 attributs

Exple: salaire_min < salaire_max

→ On définit la contrainte au niveau table

```
..., salaire_min    NUMBER(2),  
      salaire_max    NUMBER(2),  
      CONSTRAINT check_salary CHECK (salaire_min < salaire_max)
```

Modifier la structure d'une table

- Utilisez l'instruction ALTER TABLE pour,
 - Ajouter une colonne
 - Modifier le type d'une colonne existante
 - Supprimer une colonne
 - Ajouter une contrainte
 - Supprimer une contrainte

Mise à jour des colonnes

- Ajouter une colonne

```
ALTER TABLE nom_table ADD  
    (column1 datatype1,  
     column2 datatype2...);
```

- Modifier le type d'une colonne

```
ALTER TABLE nom_table MODIFY  
    (column1 datatype1,  
     column2 datatype2...);
```

- Supprimer une colonne

```
ALTER TABLE nom_table DROP  
    (nom_colonne);
```

Exemples

```
ALTER TABLE Etudiant ADD  
    (Age Number(2));
```

```
ALTER TABLE Etudiant MODIFY  
    (PrenomEtud varchar2(100));
```

```
ALTER TABLE Etudiant DROP(Age);
```

Ajouter une contrainte

- ❑ La commande ALTER TABLE est aussi utilisée pour :
 - Ajouter une contrainte
 - Supprimer une contrainte
 - Activer/ désactiver une contrainte
- ❑ On ne peut pas modifier une contrainte

```
ALTER TABLE  Nom_Table  
ADD CONSTRAINT nom_de_contrainte  
type (nom_de_colonne);
```

Supprimer une contrainte

```
ALTER TABLE  Nom_Table  
DROP CONSTRAINT nom_de_contrainte ;
```

Exemples

- Ajouter une contrainte à la table ETUDIANT

```
ALTER TABLE  Etudiant  
ADD CONSTRAINT pk_Netudiant PRIMARY KEY (Netudiant) ;
```

- Supprimer la contrainte

```
ALTER TABLE  Etudiant  
DROP CONSTRAINT pk_Netudiant ;
```

Supprimer une table

```
DROP TABLE    Nom_Table [CASCADE CONSTRAINT] ;
```

- la commande TRUNCATE TABLE Nom_Table;

Permet de vider la table et conserve sa structure contrairement à DROP qui supprime totalement la table

Les vues

- ❑ Une vue est une table virtuelle constituée par des sous-ensembles de données d'une ou plusieurs tables.
- ❑ Elle est définie par une requête prédéfinie **stockée** dans la base de données.
- ❑ Elle est exploitée comme une table mais elle **ne nécessite pas d'espace physique de stockage**.
- ❑ Utilité:
 - ✓ Pour des raisons de sécurité : restreindre l'accès aux données.
 - ✓ Conserver des données de synthèses afin d'éviter les requêtes très complexes.
 - ✓ Présenter des différentes vues pour les mêmes données...

Création de vue

- Cas d'une vue simple

```
CREATE VIEW  Nom_Vue AS SELECT colonne1,colonne2...  
                                FROM  Nom_Table  
                                WHERE Conditions  
  
WITH { READ ONLY | CHECK OPTION }  
[ CONSTRAINT constraint ];
```

- Exemple

```
CREATE VIEW Emp500 AS SELECT * FROM Employees  
                        where salary >500;
```

Création de vue

- Cas d'une vue complexe

```
CREATE VIEW  Nom_Vue AS SELECT colonne1,colonne2...  
                                FROM  Nom_Table1, Nom_Table2  
                                WHERE Conditions  
  
WITH { READ ONLY | CHECK OPTION }  
[ CONSTRAINT constraint ];
```

- **Exemple:** Créer une vue qui contient pour chaque service, son nom ainsi que le nombre de ces employés

```
CREATE VIEW  Services AS  
    Select  snom Nom, count(Empno) Nombre_Employé,  
    FROM Employees, Dept  
    WHERE  EmpDeptno = Deptno  
    GROUP BY Deptnom ;
```

Les séquences

- Objet de la base de données qui permet de générer une suite d'entiers uniques.

```
CREATE SEQUENCE <nom-séquence>  
    [INCREMENT BY <valeur-pas>]           (par défaut: 1)  
    [START WITH <valeur-début>]  
    [MAXVALUE <valeur-max> | NOMAXVALUE]  
    [MINVALUE <valeur-min> | NOMINVALUE]  
    [CYCLE | NOCYCLE]                       (par défaut: NOCYCLE)  
    [CACHE <nb-valeurs> | NOCACHE];        (par défaut: NOCACHE)
```

Les séquences

- ❑ L'option [NO]CYCLE précise si le compteur doit revenir à la valeur initiale après avoir atteint la valeur maximale (MAXVALUE) ou non. Dans le cas de NOCYCLE, une erreur est retournée si le compteur est appelé après avoir atteint la valeur maximale.
- ❑ L'option CACHE permet d'optimiser l'utilisation des séquences en plaçant en mémoire un certain nombre de valeurs générées par la séquence.

Interrogation d'une séquence

- ❑ L'interrogation d'une séquence se fait par l'utilisation des pseudo colonnes :
 - NEXTVAL : incrémente la séquence et retourne la nouvelle valeur. Si c'est le premier appel, alors elle initialise la séquence.
 - CURRVAL : retourne la valeur courante de la séquence. Elle ne peut être appelée que si NEXTVAL a été appelée au moins une fois (pour l'initialisation).
- ❑ Utilisation:

```
INSERT INTO nom_table VALUES(  
    nom_sequence.NEXTVAL, listes_colonnes ...);
```
- ❑ Interrogation de CURRVAL :

```
SELECT nom_sequence.CURRVAL FROM SYS.dual;
```

Modification d'une séquence

- Il est possible de modifier tous les paramètres d'une séquence (à part le paramètre START WITH).

```
ALTER SEQUENCE nom_séquence  
[ INCREMENT BY <valeur-pas> ]  
[ MAXVALUE <valeur-max> | NOMAXVALUE ]  
[ CYCLE | NOCYCLE ]  
[ CACHE <nb-valeurs> | NOCACHE ] ;
```

Remarques:

- ➔ La modification des paramètres d'une séquence n'affecte pas sa valeur courante (elle reste inchangée).
- ➔ La séquence doit être supprimée puis recrée pour la faire repartir d'un numéro différent.

Les synonymes

- ❑ C'est un *Alias* sur un Objet de la base ou Schéma, une sorte de raccourcis.
- ❑ Utilité
 - Masquer le vrai nom des objets et leur localisations.
 - Simplifier les noms des objets.
 - Éviter le pré-fixage dans les requêtes avec le nom de son propriétaire.

```
CREATE [OR REPLACE][PUBLIC] SYNONYM NomSynonym  
FOR [schéma.]nomObjet;
```


Les indexes

- ❑ Un index est un objet complémentaire de la base de données permettant d'indexer certaines colonnes des tables dans le but d'améliorer (accélérer) l'accès aux données (même rôle qu'un index de livre).
- ❑ BD sans index → balayage complet de la table (\approx feuilleter le livre page par page)
- ❑ BD avec index → consultation de l'index ensuite accès direct à l'information (\approx recherche du numéro de page contenant l'information recherchée dans l'index).
- ❑ L'index est stocké séparément de la table pour la quelle il a été créé.

Création d'un index

a. Implicitement

Un index est créé automatiquement sur une colonne définie comme clé primaire (PRIMARY KEY) ou avec une contrainte d'unicité (UNIQUE).

b. Explicitement

```
CREATE [UNIQUE] INDEX Nom_Index  
ON nom_Table(colonne1, colonne2 ,...);
```

Exple: Créer un index pour accélérer les recherches avec les noms des personnes dans la table employees

```
CREATE INDEX nom_idx ON employees(pnom);
```

Les indexes

- ❑ Un index peut porter sur plusieurs colonnes : la clé d'accès sera la concaténation des différentes colonnes.
- ❑ On peut créer plusieurs index indépendants sur une même table.
- ❑ Suppression d'un index

```
DROP INDEX nom_index [ON table]
```

Exemple:

```
DROP INDEX nom_idx ;
```

Chapitre 2: Le langage de manipulation de données

Objectifs

- Insertion des lignes dans les tables
 - Insertion implicite,
 - Insertion explicite,
 - Insertion par lots
- Mettre à jour les lignes d'une table
- Supprimer des données d'une table

Insertion de tuples dans une table

- On parle de 3 types d'insertion de tuples dans une tables
 - Insertion implicite
 - Insertion explicite
 - Insertion par lots

- **Insertion implicite**

```
INSERT INTO nom_table VALUES  
    (valeur1,valeur2 ,..., valeurN);
```

Exemple

```
INSERT INTO departments VALUES  
    ( 50 , 'DEVELOPMENT' , 'WASHINGTON' );
```

Insertion de tuples dans une table

□ Insertion explicite

```
INSERT INTO nom_table(colonne1, colonne2, ... colonneN)  
                VALUES(valeur1,valeur2 ,..., valeurN);
```

Exemple

```
INSERT INTO departments(deptno,dname,loc)  
                VALUES (50,'DEVELOPMENT', WASHINGTON');
```

```
INSERT INTO departments(loc,dname,deptno)  
                VALUES ('WASHINGTON', 'DEVELOPMENT', 50);
```

Remarque:

L'ordre des colonnes et des valeurs doivent être le même

Insertion de tuples avec des valeurs NULL

❑ Méthode implicite

```
INSERT INTO departments(deptno,dname)  
                VALUES (50, 'DEVELOPMENT');
```

→ Loc aura implicitement la valeur NULL

❑ Méthode explicite

```
INSERT INTO departments(deptno,dname,loc)  
                VALUES (50, 'DEVELOPMENT', NULL);
```


Insertion à partir d'une autre table

```
INSERT INTO nom_table(colonne1, colonne2, ... colonneN)
        SELECT colonne11, colonne12, ..., colonne1N
FROM nomtable1;
```

Exemple

```
INSERT INTO dept(deptno,dname,loc)
        SELECT      department_id,department_name,location
FROM departments where location='TUNIS');
```

Insertion des valeurs de type DATE

□ Sysdate

```
INSERT INTO EMP(empno, ename, job, mgr ,hiredate, sal,  
comm, deptno) VALUES (7196, 'GREEN', 'SALESMAN', 7782,  
SYSDATE, 2000, NULL, 10)
```

□ INSERTION DE DATE DANS UN FORMAT SPECIFIQUE

```
INSERT INTO EMP(empno, ename, job, mgr, hiredate, sal,  
comm, deptno) VALUES (2296, 'AROMANO', 'SALESMAN', 7782,  
TO_DATE('3 Févr 2007', 'DD,MON,YYYY'), 1300, NULL, 10)
```

Mise à jour des lignes

- ❑ La clause Update permet de modifier les lignes d'une table

```
UPDATE nom_table SET colonne=valeur [,colonne = valeur]  
[WHERE conditions]
```

Exemple

```
UPDATE Employees SET department_id=20  
                WHERE employee_id =7782;
```

```
UPDATE Employees SET department_id=20
```

Mise à jour des lignes avec des sous-interrogations

```
UPDATE    employees
SET       job_id  = (SELECT  job_id
                        FROM    employees
                        WHERE    employee_id = 205),
           salary  = (SELECT  salary
                        FROM    employees
                        WHERE    employee_id = 205)
WHERE      employee_id    = 114;
```

Mise à jour des lignes à partir d'une autre table

```
UPDATE    employees
SET       department_id = (SELECT department_id
                             FROM    departments
                             WHERE   department_name = 'DEVELOPMENT' ),
WHERE     employee_id    = 114;
```

Suppression de lignes

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

Exemple

➤ *DELETE FROM dept WHERE dname='DEVELOPMENT';*

→ Le département DEVELOPMENT est supprimé

➤ *DELETE FROM emp*

WHERE hiredate > to_date('01.01.97', 'DD.MM.YY');

→ Tous les employés embauchés après le 01.01.97 seront supprimés)

➤ *DELETE FROM dept;*

→ Tous les départements sont supprimés)

Suppression de lignes en fonction d'une autre table

```
DELETE FROM employees  
WHERE department_id = (SELECT department_id  
                        FROM departments  
                        WHERE department_name = 'SALES' );
```

→ Les employés du département 'SALES' seront supprimés

Chapitre 3:

Le langage d'interrogation de données

Objectifs

- Requêtes simples
- Restriction et tri des données
- Les fonctions monolignes
- Les fonctions multilignes (les fonctions de groupe)
- Les jointures
- Les sous-interrogations
- Les opérateurs ensemblistes

Partie1

Requête SELECT simple

Requête SELECT simple

```
SELECT*|{[DISTINCT] column|expression[alias],...}  
FROM table;
```

- **SELECT** : indique les colonnes à récupérer
- **DISTINCT** : supprime les doublons
- **FROM** : indique les tables recherchées

Exemple:

```
SELECT * FROM Departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400

Requête SELECT simple

Exemple:

```
SELECT department_id, department_name FROM  
Departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales
90	Executive
100	Finance
More than 10 rows available. Increase rows selector to view more rows.	

Expressions Arithmétiques

- ❑ Les expressions de la clause select peuvent être créées avec des données de type date ou number en utilisant les opérateurs arithmétiques.

Opérateur	Description
+	Addition
-	Soustraction
/	Division
*	Multiplication

Exemple:

```
SELECT employee_id, salary, salary*2 FROM Employees;
```

EMPLOYEE_ID	SALARY	SALARY*2
100	24010	48020
101	17000	34000
102	17000	34000
103	9000	18000
104	6000	12000

Valeur NULL

- ❑ La valeur NULL est différente de la valeur Zéro, ou d'un espace.
- ❑ Elle signifie la non existence d'une valeur pour la ligne en question

Exemple:

```
SELECT employee_id, last_name, commission_pct FROM  
Employees;
```

EMPLOYEE_ID	LAST_NAME	COMMISSION_PCT
142	Davies	-
143	Matos	-
144	Vargas	-
145	Russell	,4
146	Partners	,3
147	Errazuriz	,3

Valeur NULL

- ❑ Le résultat des expressions arithmétiques contenant une valeur Null renvoie une valeur NULL.

Exemple:

```
SELECT employee_id, last_name, salary*(1+commission_pct)
FROM Employees;
```

EMPLOYEE_ID	LAST_NAME	SALARY*(1+COMMISSION_PCT)
142	Davies	-
143	Matos	-
144	Vargas	-
145	Russell	19600
146	Partners	17550
147	Errazuriz	15600

Alias de colonne

- ❑ Permet de renommer un entête de colonne, est utile lors de l'utilisation des opérateurs arithmétiques.

```
SELECT*|[DISTINCT] column|expression [AS][alias],...}  
FROM table;
```

- ❑ Il est nécessaire d'insérer les guillemets si l'alias choisi contient des espaces ou des caractères spéciaux (#, \$,...), ou si on veut qu'il y en ai une distinction entre minuscule et majuscule.

Alias de colonne: Exemple

```
SELECT employee_id Identifiant , last_name AS name ,  
salary*12 "Annual Salary" FROM Employees;
```

IDENTIFIANT	NAME	Annual Salary
100	King	288120
101	Kochhar	204000
102	De Haan	204000
103	Hunold	108000
104	Ernst	72000
105	Austin	57600
106	Pataballa	57600
107	Lorentz	50400
108	Greenberg	144000

Concaténation

- ❑ L'opérateur de concaténation est représenté par le symbole **||**
- ❑ Permet la concaténation de chaînes de caractères et de colonnes
- ❑ Le résultat de concaténation sera une expression de type caractère

```
SELECT last_name || ' is a ' || job_id AS "Employees"  
FROM employees;
```

Employees
King is a AD_PRES
Kochhar is a AD_VP
De Haan is a AD_VP
Hunold is a IT_PROG
Ernst is a IT_PROG
Austin is a IT_PROG
Pataballa is a IT_PROG

Partie 2

Restriction et Tri des données

Clause Where

- ▣ La clause where est utilisée pour restreindre les lignes renvoyés par une requête SQL.
- ▣ La clause WHERE suit la clause FROM

```
SELECT* | {[DISTINCT] column|expression[alias],...}  
FROM table Where Conditions;
```

Clause Where

```
SELECT employee_id, last_name, job_id, department_id
FROM   employees
WHERE  department_id = 90 ;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	DEPARTMENT_ID
100	King	AD_PRES	90
101	Kochhar	AD_VP	90
102	De Haan	AD_VP	90

Utilisation des chaînes de caractères et Date

- ❑ Les chaînes de caractères et les dates sont incluses entre apostrophes.
- ❑ Les valeurs de type caractère distinguent les majuscules des minuscules et les valeurs de date sont sensibles au format.
- ❑ Le format de date par défaut est DD-MM-RR

```
SELECT last_name, job_id, department_id  
FROM employees  
WHERE last_name = 'Whalen' ;
```

Condition de comparaison

Opérateur	Signification
=	Egal à
>	Supérieur à
>=	Supérieur ou égal à
<	Inférieur à
<=	Inférieur ou égal à
<>	Non égal à
BETWEEN ...AND...	Entre deux valeurs (incluses)
IN(set)	Correspond à une valeur quelconque d'une liste
LIKE	Correspond à un modèle de caractère
IS NULL	Est une valeur NULL

Condition de comparaison

```
SELECT last_name, salary FROM employees  
WHERE salary <= 3000 ;
```

LAST_NAME	SALARY
Matos	2600
Vargas	2500

```
SELECT last_name, salary FROM employees  
WHERE salary BETWEEN 2500 AND 3500 ;
```

LAST_NAME	SALARY
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500

Condition de comparaison

```
SELECT employee_id, last_name, salary, manager_id
FROM employees
WHERE manager_id IN (100, 101, 201);
```

EMPLOYEE_ID	LAST_NAME	SALARY	MANAGER_ID
202	Fay	6000	201
200	Whalen	4400	101
205	Higgins	12000	101
101	Kochhar	17000	100
102	De Haan	17000	100
124	Mourgos	5800	100
149	Zlotkey	10500	100
201	Hartstein	13000	100

Condition de comparaison

- ❑ Utiliser la condition LIKE pour effectuer des recherches de chaînes de caractères valides via l'utilisation de caractères génériques.
- ❑ Les conditions de recherche peuvent contenir soit des caractères littéraux, soit des nombres :
 - % indique un nombre quelconque de caractères (zéro ou plus)
 - _ indique un caractère unique

```
SELECT last_name  
FROM employees  
WHERE last_name LIKE '_o%';
```

LAST_NAME
Kochhar
Lorentz
Mourgos

Condition de comparaison

- ▣ Tester la présence de valeurs NULL avec l'opérateur IS NULL.

```
SELECT last_name, manager_id  
FROM employees  
WHERE manager_id IS NULL ;
```

LAST_NAME	MANAGER_ID
King	

Conditions logiques

Opérateur	Signification
AND	Renvoie TRUE si les deux conditions sont vraies.
OR	Renvoie TRUE si l'une des deux conditions est vraie.
NOT	Renvoie TRUE si la condition qui suit est fausse.

Conditions logiques

SELECT employee_id, last_name, job_id, salary **FROM** employees
WHERE salary >=10000
AND job_id LIKE '%MAN%' ;

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
149	Zlotkey	SA_MAN	10500
201	Hartstein	MK_MAN	13000

SELECT employee_id, last_name, job_id, salary **FROM** employees
WHERE salary >=10000 OR job_id LIKE '%MAN%' ;

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
124	Mourgos	ST_MAN	5800
149	Zlotkey	SA_MAN	10500
174	Abel	SA_REP	11000
201	Hartstein	MK_MAN	13000
205	Higgins	AC_MGR	12000

Conditions logiques

```
SELECT last_name, job_id
FROM employees
WHERE job_id
      NOT IN ('IT_PROG', 'ST_CLERK', 'SA_REP') ;
```

LAST_NAME	JOB_ID
King	AD_PRES
Kochhar	AD_VP
De Haan	AD_VP
Mourgos	ST_MAN
Zlotkey	SA_MAN
Whalen	AD_ASST
Hartstein	MK_MAN
Fay	MK_REP
Higgins	AC_MGR
Gietz	AC_ACCOUNT

Règles de priorité

Priorité	Opérateur
1	Arithmetic operators
2	Concatenation operator
3	Comparison conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not equal to
7	NOT logical condition
8	AND logical condition
9	OR logical condition

Règles de priorité

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id = 'SA_REP'
OR job_id = 'AD_PRES'
AND salary > 15000;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000
Abel	SA_REP	11000
Taylor	SA_REP	8600
Grant	SA_REP	7000

```
SELECT last_name, job_id, salary
FROM employees
WHERE (job_id = 'SA_REP'
OR job_id = 'AD_PRES')
AND salary > 15000;
```

LAST_NAME	JOB_ID	SALARY
King	AD_PRES	24000

La clause ORDER BY

- ❑ Trier les lignes extraites à l'aide de la clause ORDER BY:
 - ASC : ordre croissant (par défaut)
 - DESC : ordre décroissant
- ❑ La clause ORDER BY vient en dernier dans l'instruction SELECT

```
SELECT last_name, job_id, department_id, hire_date
FROM employees ORDER BY hire_date ;
```

LAST_NAME	JOB_ID	DEPARTMENT_ID	HIRE_DATE
King	AD_PRES	90	17-JUN-87
Whalen	AD_ASST	10	17-SEP-87
Kochhar	AD_VP	90	21-SEP-89
Hunold	IT_PROG	60	03-JAN-90
Ernst	IT_PROG	60	21-MAY-91

...

..... 20 rows selected.

La clause ORDER BY

- Trier par ordre décroissant :

```
SELECT last_name, job_id, department_id, hire_date  
FROM employees ORDER BY hire_date DESC ;
```

- Trier par alias de colonnes:

```
SELECT employee_id, last_name, salary*12 annsal  
FROM employees ORDER BY annsal ;
```

- Trier selon plusieurs colonnes:

```
SELECT last_name, department_id, salary  
FROM employees ORDER BY department_id, salary DESC;
```

Les fonctions mono lignes

Les fonctions mono lignes

- ❑ Les fonctions mono lignes sont des fonctions qui opèrent sur chaque ligne
- ❑ Ils peuvent modifier le type de données
- ❑ Il existe 5 familles des fonctions mono lignes:
 - Les fonctions de type caractère
 - Les fonctions numériques
 - Les fonctions date
 - Les fonctions de conversion
 - Les fonctions générales

Les fonctions de type caractère

- ❑ 1. les fonctions de manipulation de la casse
 - **LOWER(arg)** : convertit les caractères majuscules en minuscules
 - **UPPER(arg)** : convertit les caractères minuscules en majuscules
 - **INITCAP(arg)** : convertit l'initiale de chaque mot en majuscule et les caractères suivants en minuscules

```
SELECT last_name, lower(last_name) , upper(last_name),  
initcap(last_name) FROM EMPLOYEES;
```

LAST_NAME	LOWER(LAST_NAME)	UPPER(LAST_NAME)	INITCAP(LAST_NAME)
Abel	abel	ABEL	Abel
Ande	ande	ANDE	Ande
Atkinson	atkinson	ATKINSON	Atkinson
Austin	austin	AUSTIN	Austin
Baer	baer	BAER	Baer
Baida	baida	BAIDA	Baida

Les fonctions de type caractère

- ❑ 2. les fonctions de manipulation de caractères
 - **CONCAT(arg1,arg2)** : concatène la première chaîne avec la seconde = ||
 - **SUBSTR(arg,m[,n])** : extrait une sous chaîne d'une autre chaîne
 - **LENGTH(arg)** : taille d'une chaîne en caractères
 - **INSTR(arg,cc)** : retourne la position de cc dans l'argument.

```
select concat(last_name, first_name), substr(last_name,2,4),  
length(last_name), instr(last_name,'a') from employees;
```

CONCAT(LAST_NAME,FIRST_NAME)	SUBSTR(LAST_NAME,2,4)	LENGTH(LAST_NAME)	INSTR(LAST_NAME,'A')
AbelEllen	bel	4	0
AndeSundar	nde	4	0
AtkinsonMozhe	tkin	8	0
AustinDavid	usti	6	0
BaerHermann	aer	4	2
BaidaShelli	aida	5	2
BandaAmit	anda	5	2
BatesElizabeth	ates	5	2

Les fonctions de type caractère

- **LPAD(arg,m,cc)** : insérer à gauche de l'argument le caractère cc jusqu'à ce que la taille de la chaîne retournée soit égale à m.
- **RPAD(arg,m,cc)** insérer à droite de l'argument le caractère cc jusqu'à ce que la taille de la chaîne retournée soit égale à m.

```
select rpad(last_name,15,'-'), lpad(last_name,15,'-') from employees;
```

RPAD(LAST_NAME,15,'-')	LPAD(LAST_NAME,15,'-')
Abel-----	-----Abel
Ande-----	-----Ande
Atkinson-----	-----Atkinson
Austin-----	-----Austin
Baer-----	-----Baer
Baida-----	-----Baida
Banda-----	-----Banda

Les fonctions de type caractère

- **LTRIM (arg,cc)** : renvoie l'argument dont elle enlève chaque caractère de cc si ce caractère se trouve à son extrémité gauche.
- **RTRIM(arg,cc)** : renvoie l'argument duquel elle enlève chaque caractère de cc si ce caractère se trouve à son extrémité droite.
- **TRIM(cc from arg)** : le dimensionnement se fait des deux cotés

Select trim('n' from 'nation'), ltrim('hello','h'), rtrim('operation','on')
from dual;

TRIM('N'FROM'NATION')	LTRIM('HELLO','H')	RTRIM('OPERATION','ON')
atio	ello	operati

Les fonctions de type caractère

- **REPLACE** : remplace toutes les occurrences de la chaîne recherchée par une autre

```
select 'JACK and JUE' , replace('JACK and JUE','J','BL') from dual;
```

'JACKANDJUE'	REPLACE('JACKANDJUE','J','BL')
JACK and JUE	BLACK and BLUE

TRANSLATE : remplace chaque caractère recherché par son correspondant dans la chaîne principale

```
select 'MONDAY', translate('MONDAY','MON','FRI') from dual;
```

'MONDAY'	TRANSLATE('MONDAY','MON','FRI')
MONDAY	FRIDAY

Les fonctions numérique

- ▣ **ROUND**: arrondit la valeur à une décimale donnée.

$\text{ROUND}(46.8) \rightarrow 47$

$\text{ROUND}(46.862,1) \rightarrow 46.9$

$\text{ROUND}(46.862,2) \rightarrow 46.86$

$\text{ROUND}(46.862,-1) \rightarrow 50$

$\text{ROUND}(51.342,-2) \rightarrow 100$

Il est à noter que $\text{ROUND}(5.5) \rightarrow 6$

Les fonctions numérique

- ▣ **TRUNC** : tronque la valeur à une décimale donnée.

$\text{TRUNC}(46.862,1) \rightarrow 46.8$

$\text{TRUNC}(46.862,2) \rightarrow 46.86$

$\text{TRUNC}(46.862,-1) \rightarrow 40$

$\text{TRUNC}(51.342,-2) \rightarrow 0$

- ▣ **MOD** : renvoie le reste de la division.

$\text{MOD}(5,2) \rightarrow 1$

Les fonctions numérique

- **CEIL(arg)** : retourne le plus petit entier supérieur à l'argument.

CEIL(99.9) → 100

CEIL(-11.1) → -11

- **FLOOR(arg)** : retourne le plus grand entier inférieur à l'argument.

FLOOR(99.9) → 99

FLOOR(-11.1) → -12

Les fonctions de manipulation de dates

- On peut faire des calculs arithmétiques sur les DATE :
 - **DATE1 + n**: ajoute ou soustrait n jours de DATE1 (selon le signe de n) et retourne la date résultat.

Remarque : l'ajout d'un nombre d'heures à une date se fait en divisant le nombre d'heures par 24

- **DATE1 - DATE2** : retourne la différence entre les deux dates en nombre réel de jours, car l'opération de soustraction tient compte des heures des deux dates.

```
SELECT sysdate, sysdate+2, to_date('20/09/1998') - 10,  
       Trunc(sysdate - to_date('10/10/2008'),0)  
FROM dual ;
```

SYSDATE	SYSDATE+2	TO_DATE('20/09/1998')-10	TRUNC(SYSDATE-TO_DATE('10/10/2008'),0)
17/02/14	19/02/14	10/09/98	1956

Les fonctions de manipulation de dates

- ❑ **MONTHS_BETWEEN (date1,date2)** : retourne en nombre réel la différence en mois entre les deux dates.
- ❑ **ADD_MONTHS(date1,n)** : retourne la date qui vient n mois après date1.
- ❑ **NEXT_DAY(date1,char/n)** : Elle retourne le prochain jour char dans la semaine de date1.

$\text{char} \in \{\text{'Lundi'}, \text{'Mardi'}, \dots, \text{'Dimanche'}\}, n \in \{1, 2, \dots, 7\}.$

- ❑ **LAST_DAY(date1)** : retourne la date du dernier jour du mois de date1.
- ❑ **Extract** : extraction du jour, mois, année, heure, minute et seconde

Les fonctions de manipulation de dates

```
select trunc(MONTHS_BETWEEN('09/01/95','01/11/94'))  
nbr_mois,  
ADD_MONTHS ('31/01/96',1), NEXT_DAY('09/01/95','JEUDI'),  
LAST_DAY('01/02/95')  
from dual ;
```

NBR_MOIS	ADD_MONTHS('31/01/96',1)	NEXT_DAY('09/01/95','JEUDI')	LAST_DAY('01/02/95')
2	29/02/96	12/01/95	28/02/95

Les fonctions de manipulation de dates

```
SELECT systimestamp,  
       extract(day from systimestamp) as Jour,  
       extract(month from systimestamp) as Mois,  
       extract(year from systimestamp) as Année,  
       extract(hour from systimestamp)+1 as heure,  
       extract(minute from systimestamp) as Minute,  
       trunc(extract(second from systimestamp)) as Second  
FROM dual;
```

SYSTIMESTAMP	JOUR	MOIS	ANNÉE	HEURE	MINUTE	SECOND
17-FÉVR.-14 10.55.55,347000 AM +01:00	17	2	2014	10	55	55

Les fonctions de manipulation de dates

- ❑ **ROUND**(date1[,fmt]): retourne date1 arrondie selon le format *fmt*
- ❑ **TRUNC**(date1[,fmt]): tronque une date

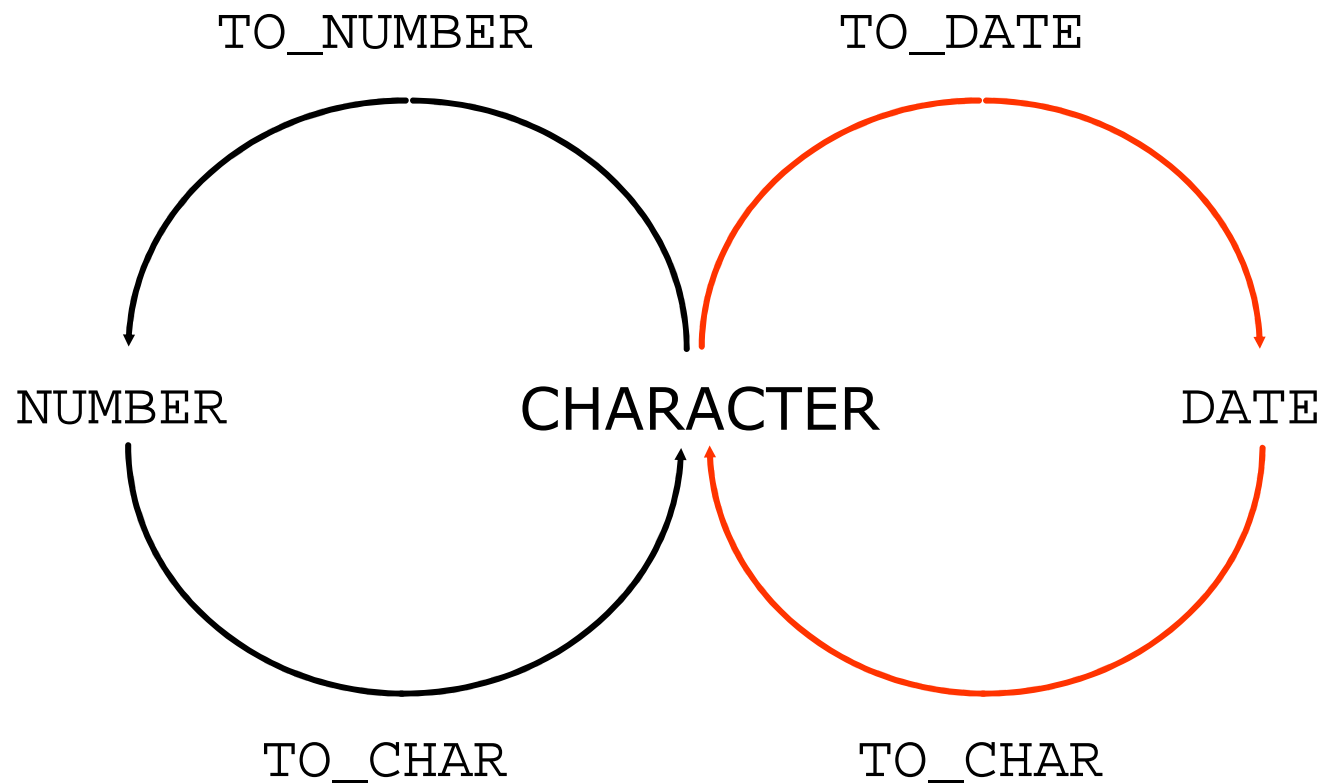
select **ROUND**(SYSDATE,'DD'), **ROUND**(SYSDATE,'MM'),
ROUND(SYSDATE,'YY') from dual;

Select **TRUNC**(SYSDATE , 'DD'), **TRUNC**(SYSDATE , 'MM'),
TRUNC(SYSDATE , 'YY') from dual ;

ROUND(SYSDATE,'DD')	ROUND(SYSDATE,'MM')	ROUND(SYSDATE,'YY')
17/02/14	01/03/14	01/01/14

TRUNC(SYSDATE,'DD')	TRUNC(SYSDATE,'MM')	TRUNC(SYSDATE,'YY')
17/02/14	01/02/14	01/01/14

Les fonctions de conversion



Les fonctions de conversion

- ❑ **TO_CHAR(Date1,'fmt')** : convertit une date en chaîne de caractères et l'affiche dans le format 'fmt' indiqué.

```
SELECT TO_CHAR(SYSDATE,'DD/MM/YYYY') FROM DUAL;
```

```
SELECT TO_CHAR(SYSDATE,'DAY') FROM DUAL;
```

- ❑ **TO_CHAR(NOMBRE,'fmt')** : convertit un nombre en chaîne de caractère dans le format spécifié.

```
Select TO_CHAR (900.00,'$999.999') from dual ;
```

Les fonctions de conversion

- ▣ **TO_NUMBER(char,'fmt')** : convertit une chaîne de caractère en nombre.

```
select TO_NUMBER('$1500','$9999') from dual ;
```

- ▣ **TO_DATE(char,'fmt')** : convertit char en date selon le format de date mentionnée.

```
select TO_DATE('23/10/2004','DD/MM/YYYY') from dual ;
```

Eléments du modèle de format date

Element	Result
YYYY	Année complète en 4 chiffres
YY	Année en deux chiffres
YEAR	Année en lettres (en Anglais)
MM	Valeur à deux chiffres du mois
MONTH	Nom complet du mois
MON	Abréviation à trois lettres du mois
DY	Abréviation à trois lettres du jour de la semaine
DAY	Nom complet du jour de la semaine
DD	Valeur numérique du jour du mois
W	Numéro de la semaine dans le mois (la semaine 1 commence le 1er jour du mois et dure 7 jours)
WW	Numéro de la semaine dans l'année (la semaine 1 commence le 1er jour de l'année et dure 7 jours).
Spth	Suffixe demandant l'affichage en toutes lettres suffixé de TH.
Sp	Suffixe demandant l'affichage en toutes lettres(DDSP;FOUR)

Les fonctions de conversion

```
SELECT last_name,  
       TO_CHAR(hire_date, 'fmDD Month YYYY') AS HIREDATE  
FROM   employees;
```

LAST_NAME	HIREDATE
King	17 June 1987
Kochhar	21 September 1989
De Haan	13 January 1993
Hunold	3 January 1990
Ernst	21 May 1991
Lorentz	7 February 1999
Mourgos	16 November 1999

Eléments du modèle de format Numérique

Element	Result
9	Représente un nombre
0	Force l'affichage d'un zéro
\$	Insère un signe dollar
L	Utilise le symbole monétaire local
.	Affiche un point en tant que séparateur décimal
,	Affiche une virgule en tant que séparateur de milliers

Les fonctions de conversion

```
SELECT TO_CHAR(salary, '$99,999.00') SALARY  
FROM   employees  
WHERE  last_name = 'Ernst';
```

SALARY
\$6,000.00

Autres Fonctions

- ❑ **NVL(arg,val)** : retourne *val* si *arg* est **NULL**.
- ❑ **NVL2(arg,val1,val2)**: retourne *val2* si *arg* est **NULL** sinon retourne *val1*.
- ❑ **Coalesce (exp1,expr2,expr3,...)** : retourne la première valeur non nulle

```
SELECT employee_id, last_name, salary, commission_pct,  
       NVL(commission_pct,0)  comm,  NVL(to_char(commission_pct),  
       'Pas de commission') comm1,  
       NVL2(commission_pct,commission_pct/2,0)    comm2    FROM  
employees;
```

Autres Fonctions

❑ **NULLIF (val1, val2)** : si val1 = val2 la valeur NULL est retournée sinon val1

❑ **Decode (expr, val1, val11, val2, val21, Valn, valn1, default)** :

retourne valn1 si expr = valn sinon default

❑ **Case** : évalue une liste de conditions et retourne un résultat parmi les cas possibles

```
SELECT NULLIF(last_name,  
last_name) last_name,  
NULLIF(100, 200),  
trunc(salary /  
NULLIF(commission_pct, 0), 2)  
"sal/comm"  
FROM employees;
```

LAST_NAME	NULLIF(100,200)	Sal/Comm
-	100	-
-	100	-
-	100	35000
-	100	45000
-	100	40000
-	100	36666,66
-	100	52500

Autres Fonctions

```
❑ SELECT last_name, job_id, salary,  
      CASE job_id WHEN 'IT_PROG' THEN 1.10*salary  
        WHEN 'ST_CLERK' THEN 1.15*salary  
        WHEN 'SA_REP' THEN 1.20*salary  
        ELSE salary END "REVISED_SALARY"  
FROM employees;
```

```
❑ SELECT last_name, job_id, salary,  
      DECODE(job_id, 'IT_PROG' , 1.10*salary,  
        'ST_CLERK', 1.15*salary,  
        'SA_REP' , 1.20*salary,  
        salary)      REVISED_SALARY  
FROM employees;
```

Autres fonctions

- ❑ **Rank**: retourne le rang de chaque ligne au sein de la partition d'un ensemble de résultats.
- ❑ **Dense_Rank**: retourne le rang des lignes à l'intérieur de la partition d'un ensemble de résultats, sans aucun vide dans le classement.
- ❑ **Row_number** : retourne le numéro séquentiel d'une ligne d'une partition d'un ensemble de résultats, en commençant à 1 pour la première ligne de chaque partition

Autres fonctions

La différence entre RANK et DENSE_RANK:

- la fonction DENSE_RANK ne laisse aucun trous (intervalle) dans une série de rangée lorsque celles-ci sont très proches.
- Si on utilise DENSE_RANK pour classer deux personnes dans une compétition qui sont très proches au niveau du temps, on dira qu'ils occupent les deux la même place (ex. les deux seront classés 3ème), la personne suivante occupera la 4ème place, malgré qu'elle soit la 5ème personne.
- RANK agira de même, sauf que la personne suivante sera classée en 5ème position.

Autres fonctions: Row_number

```
select row_number() over (partition by department_id order  
by department_id) No,department_id, last_name || ' ' ||  
first_name "nom et prenom",salary from employees
```

```
select row_number() over (partition by department_id  
order by salary desc) No,department_id, last_name || ' ' ||  
first_name "nom et prenom",salary from employees
```

NO	DEPARTMENT_ID	Nom Et Prenom	SALARY
1	10	Whalen Jennifer	4400
1	20	Fay Pat	6000
2	20	Hartstein Michael	13000
1	30	Raphaely Den	11000
2	30	Khoo Alexander	3100
3	30	Colmenares Karen	2500
4	30	Tobias Sigal	2800
5	30	Himuro Guy	2600
6	30	Baida Shelli	2900
1	40	Mavris Susan	6500
1	50	Weiss Matthew	8000
2	50	Grant Douglas	2600
3	50	Kaufling Payam	7900
4	50	Vollman Shanta	6500
5	50	Mourgos Kevin	5800

NO	DEPARTMENT_ID	Nom Et Prenom	SALARY
1	10	Whalen Jennifer	4400
1	20	Hartstein Michael	13000
2	20	Fay Pat	6000
1	30	Raphaely Den	11000
2	30	Khoo Alexander	3100
3	30	Baida Shelli	2900
4	30	Tobias Sigal	2800
5	30	Himuro Guy	2600
6	30	Colmenares Karen	2500
1	40	Mavris Susan	6500
1	50	Fripp Adam	8200
2	50	Weiss Matthew	8000
3	50	Kaufling Payam	7900
4	50	Vollman Shanta	6500
5	50	Mourgos Kevin	5800
6	50	Sarchand Nandita	4200

Autres fonctions: rank

```
select rank() over (partition by department_id  
order by salary desc) No, last_name, first_name, salary,  
department_id from employees order by department_id desc
```

5	Abel	Ellen	11000	80
5	Cambraut	Gerald	11000	80
7	Zlotkey	Eleni	10500	80
7	Vishney	Clara	10500	80
9	Bloom	Harrison	10000	80
9	Tucker	Peter	10000	80
9	King	Janette	10000	80
12	Fox	Tayler	9600	80
13	Bernstein	David	9500	80
13	Sully	Patrick	9500	80
13	Greene	Danielle	9500	80
16	Hall	Peter	9000	80
16	McEwen	Allan	9000	80
18	Hutton	Alyssa	8800	80

Autres fonctions: dense_rank

```
select dense_rank() over (partition by department_id
order by salary desc) No, last_name, first_name,
salary, department_id from employees
order by department_id
```

8	Bell	Sarah	4000	50
9	Everett	Britney	3900	50
10	Chung	Kelly	3800	50
11	Dilly	Jennifer	3600	50
11	Ladwig	Renske	3600	50
12	Rajs	Trenna	3500	50
13	Dellinger	Julia	3400	50
14	Mallin	Jason	3300	50
14	Bissot	Laura	3300	50
15	Taylor	Winston	3200	50
15	Nayer	Julia	3200	50
15	Stiles	Stephen	3200	50
15	McCain	Samuel	3200	50
16	Davies	Curtis	3100	50
16	Walsh	Alana	3100	50
16	Fleur	Jean	3100	50

Les fonctions de groupe

Les fonctions de groupe

- ❑ Les fonctions d'agrégation ou fonction de groupes utilisés pour manipuler des ensembles de valeurs dans le but de fournir des informations de synthèse
- ❑ Aussi appelées les fonctions multi-lignes puisqu'ils opèrent sur un ensemble de lignes et renvoie une seule valeur
 - `AVG([distinct | all]expr)` : valeur moyenne en ignorant les valeurs NULL
 - `COUNT ([* | distinct | all]expr)` : nombre de lignes où expr est différente de NULL. Le caractère * comptabilise toutes les lignes sélectionnées.
 - `MAX ([distinct | all]expr)` : valeur maximale en ignorant les valeurs NULL

Les fonctions de groupe

- *MIN([distinct | all]expr)* : valeur minimale en en ignorant les valeurs NULL
- *STDDEV([distinct | all]expr)* : ecart-type en en ignorant les valeurs NULL
- *SUM([distinct | all]expr)* : somme en en ignorant les valeurs NULL
- *VARIANCE([distinct | all]expr)* : variance en ignorant les valeurs NULL

Les fonctions de groupe

- ❑ Les fonction MAX et MIN peuvent être utilisés sur des colonnes de type numérique, date ou de type caractère

- ❑ *Exemple*

```
select max(salary), min(hire_date), max(hire_date),  
min(last_name), max(last_name) from employees;
```

MAX(SALARY)	MIN(HIRE_DATE)	MAX(HIRE_DATE)	MIN(LAST_NAME)	MAX(LAST_NAME)
24010	17/06/87	19/02/13	Abel	Zlotkey

- ❑ les fonctions SUM et AVG sont utilisées sur des colonnes de type numérique uniquement

```
select round(avg(salary)), sum(salary) from employees;
```

ROUND(AVG(SALARY))	SUM(SALARY)
6413	692620

Les fonctions de groupe

- ❑ Le **count(*|expr)** renvoi le nombre de lignes d'une table ou le nombre de lignes avec des valeurs **NON NULL** pour expr
- ❑ *Exemple*

```
Select count(*), count(commission_pct) from  
employees where department_id=50;
```

COUNT(*)	COUNT(COMMISSION_PCT)
45	0

➔ les valeurs nulles ne sont pas prises en compte

Les fonctions de groupe

- ❑ **COUNT(DISTINCT expr)** renvoie le nombre de valeurs non NULL distinctes de expr.

```
SELECT COUNT(department_id)
FROM   employees;
```

COUNT(DEPARTMENT_ID)
107

```
SELECT COUNT(DISTINCT department_id)
FROM   employees;
```

COUNT(DISTINCTDEPARTMENT_ID)
11

Créer des groupes de données

```
SELECT      [column,] group_function(column), ...
FROM        table
[WHERE      condition]
[GROUP BY   column]
[ORDER BY   column];
```

- ❑ Les lignes d'une table peuvent être divisées en groupes plus petits à l'aide de la clause GROUP BY.
- ❑ Toutes les colonnes de la liste SELECT qui ne sont pas incluses dans des fonctions de groupe doivent figurer dans la clause GROUP BY

Exemple

```
SELECT      department_id, AVG(salary)
FROM        employees
GROUP BY    department_id ;
```

Créer des groupes de données

- Afficher pour chaque département, son numéro, le nombre total des employés et le salaire le plus élevé.

```
Select department_id department, count(*) "Nbre employees",  
max(salary) "Salaire Max" from employees  
group by department_id  
order by department_id;
```

DEPARTMENT	Nbre Employees	Salaire Max
10	1	4400
20	3	13000
30	6	11000
40	1	6500
50	45	8200

Créer des groupes de données

- Utiliser la clause GROUP BY sur plusieurs colonnes

```
SELECT    department_id dept_id, job_id,  
SUM(salary)  
FROM      employees  
GROUP BY  department_id, job_id ;
```

DEPT	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
20	AD_VP	1200
20	MK_MAN	13000
20	MK_REP	6000
30	PU_CLERK	13900
30	PU_MAN	11000
40	HR_REP	6500
50	SH_CLERK	64300
50	ST_CLERK	55700
50	ST_MAN	36400

Requêtes illégales avec les fonctions de groupe

```
SELECT department_id, COUNT(last_name)
FROM employees;
```



ORA-00937: la fonction de groupe ne porte pas sur un groupe simple

➔ Toute colonne ou expression de la liste SELECT qui ne constitue pas une fonction d'agrégation doit figurer dans la clause GROUP BY:

```
SELECT department_id, COUNT(last_name)
FROM employees GROUP BY department_id;
```

DEPARTMENT_ID	COUNT(LAST_NAME)
100	6
30	6
-	1
90	3
20	3
70	1

Requêtes illégales avec les fonctions de groupe

- ❑ La clause WHERE ne doit pas contenir des fonctions de groupe

```
SELECT    department_id, AVG(salary)
FROM      employees
WHERE     AVG(salary) > 8000
GROUP BY department_id;
```



ORA-00934: fonction de groupe non autorisée ici

- ➔ Pour restreindre les résultats des groupes on introduit la clause **HAVING**

```
SELECT    column, group_function
FROM      table
[WHERE    condition]
[GROUP BY group_by_expression]
[HAVING   group_condition]
[ORDER BY column];
```

Restriction des résultats de groupe

→ Pour avoir les département dont le salaire moyen est $>$ à 8000

```
SELECT    department_id, AVG(salary)
FROM      employees
GROUP BY  department_id
HAVING    AVG(salary) > 8000;
```

DEPARTMENT_ID	ROUND(AVG(SALARY))
70	10000
80	8956
90	19337
100	8602
110	10150

→ Lorsque vous utilisez la clause HAVING, le serveur Oracle restreint les groupes de la façon suivante:

1. Les lignes sont regroupées
2. La fonction de groupe est appliquée
3. Les groupes qui correspondent à la clause HAVING s'affichent.

Restriction des résultats de groupe

□ *Exemple*

```
SELECT    job_id, SUM(salary) PAYROLL
FROM      employees
WHERE     job_id NOT LIKE '%REP%'
GROUP BY  job_id
HAVING    SUM(salary) > 13000
ORDER BY  SUM(salary);
```

JOB_ID	PAYROLL
PU_CLERK	13900
AD_PRES	24010
IT_PROG	28800
AD_VP	35200
ST_MAN	36400
FL_ACCOUNT	39610
ST_CLERK	55700
SA_MAN	61000
SH_CLERK	64300

Fonctions de groupe imbriquées

Les fonctions de groupes peuvent être imbriquées

Exemple

```
SELECT    MAX(AVG(salary)) "Salaire moyen le plus élevé"  
FROM      employees  
GROUP BY department_id;
```

Salaire Moyen Le Plus Elevé
19337

Les jointures

Les opérateurs ensemblistes

Les sous-interrogations

Définition de Jointure

- Une jointure est une opération relationnelle qui sert à extraire des données de plusieurs tables
- La condition de jointure peut être exprimée dans la clause :

Where T1.C1=T2.C1

On T1.C1=T2.C1

- Précédez le nom de la colonne par le nom de la table lorsque celui-ci figure dans plusieurs tables

Produit cartésien

- ❑ On obtient un produit cartésien lorsque :
 - Une condition de jointure est omise
 - Une condition de jointure est incorrecte
- ❑ A chaque ligne de la table 1 sont jointes toutes les lignes de la table 2.
- ❑ Le nombre de lignes renvoyées est égal $n1*n2$ où
 - $n1$ est le nombre de lignes de la table 1
 - $n2$ est le nombre de lignes de la table 2
- ❑ `SELECT last_name, department_name`
`FROM employees`
`CROSS JOIN departments ;`

Jointure interne

- ❑ Une **jointure interne** ou **équijointure** est une jointure avec une condition de jointure contenant un opérateur d'égalité.

C'est la plus répandue, elle combine les lignes qui ont des valeurs équivalentes pour les colonnes de la jointure.

- ❑ Généralement dans ce type de jointure, ce sont les Primary Key et Foreign Key qui sont utilisées.

Jointure interne

```
SELECT  T1.COLONNE1, ...T1.COLONNEn,  
        T2.COLONNE1, ...T2.COLONNEm  
FROM T1 INNER JOIN T2  
      ON T1.C1=T2.C1  
WHERE CONDITION(S)
```

==

```
SELECT T1.COLONNE1, ...T1.COLONNEn,  
        T2.COLONNE1, ...T2.COLONNEm  
FROM T1 ,T2  
WHERE  T1.C1=T2.C1 AND CONDITION(S)
```

- ▣ La directive INNER devant JOIN est optionnelle.

Jointure interne

```
SELECT empno, ename, emp.deptno, dname, loc  
FROM emp INNER JOIN dept  
ON emp.deptno=dept.deptno  
order by deptno, ename;
```

EMPNO	ENAME	DEPTNO	DNAME	LOC
7782	CLARK	10	ACCOUNTING	NEW YORK
7839	KING	10	ACCOUNTING	NEW YORK
7934	MILLER	10	ACCOUNTING	NEW YORK
7876	ADAMS	20	RESEARCH	DALLAS
7902	FORD	20	RESEARCH	DALLAS
7566	JONES	20	RESEARCH	DALLAS
7788	SCOTT	20	RESEARCH	DALLAS
7369	SMITH	20	RESEARCH	DALLAS
7499	ALLEN	30	SALES	CHICAGO

Jointure interne

- On peut ajouter des conditions supplémentaires à une condition de jointure

```
SELECT e.employee_id, e.last_name, e.department_id,  
d.department_id, d.location_id
```

```
FROM employees e JOIN departments d
```

```
ON (e.department_id = d.department_id)
```

```
AND e.manager_id = 149 ;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
174	Abel	80	80	2500
176	Taylor	80	80	2500

Jointure interne

- ❑ On peut aussi créer des jointures avec trois ou plusieurs tables avec la clause ON.

```
SELECT employee_id, city, 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;
```

EMPLOYEE_ID	CITY	DEPARTMENT_NAME
103	Southlake	IT
104	Southlake	IT
107	Southlake	IT
124	South San Francisco	Shipping
141	South San Francisco	Shipping
142	South San Francisco	Shipping
143	South San Francisco	Shipping
144	South San Francisco	Shipping

Jointure externe

- ❑ Une jointure externe **OUTER JOIN** élargie le résultat d'une jointure simple (**INNER JOIN**) et permet en plus d'extraire des enregistrements qui ne répondent pas aux critères de jointure.
- ❑ Une jointure externe renvoie toutes les lignes qui satisfont la condition de jointure et retourne également une partie de ces lignes de la table pour laquelle aucune des lignes de l'autre ne satisfait la condition de jointure.

```
SELECT T1.Colonne1, ...T1.Colonnen,  
       T2.Colonne1, ...T2.Colonnem  
FROM T1 LEFT | RIGHT | FULL OUTER  
     JOIN T2  
     ON T1.C1=T2.C1
```

Jointure externe

- ❑ Le sens de la jointure externe LEFT ou RIGHT de la clause OUTER JOIN désigne la table dominante.
- ❑ FROM table1 t1 **LEFT OUTER JOIN** table2 t2 ON (t1.column_name = t2.column_name);
- ❑ FROM table1 t1 **RIGHT OUTER JOIN** table2 t2
ON (t1.column_name = t2.column_name);
- ❑ **EXEMPLE / JOINTURE EXTERNE GAUCHE**
SELECT employee_id, last_name, department_name
from **employees E LEFT OUTER JOIN departments D**
on E.department_ID = D.department_ID
order by employee_id;

Auto-jointure

- L'auto-jointure est une Liaison d'une table à elle-même (relation reflexive)

```
SELECT a.employee_id empno ,a.last_name ename, a.job_id  
job, b.last_name as MGR
```

```
FROM employees a INNER JOIN employees b
```

```
ON a. manager_id=b.employee_id
```

```
ORDER BY empno;
```


EMPNO	ENAME	JOB	MGR
7876	ADAMS	CLERK	SCOTT
7499	ALLEN	SALESMAN	BLAKE
7698	BLAKE	MANAGER	KING
7782	CLARK	MANAGER	KING
7902	FORD	ANALYST	JONES
7900	JAMES	CLERK	BLAKE
7566	JONES	MANAGER	KING
7654	MARTIN	SALESMAN	BLAKE
7934	MILLER	CLERK	CLARK

La Non-Equijointure

- Une **non équijointure** est une jointure avec une condition de jointure contenant un **opérateur d'inégalité**. La non-équi-jointure vous permet de joindre deux tables dans lesquelles il n'existe pas de correspondance directe entre les colonnes.

EMP		
EMPNO	ENAME	SAL
7839	KING	5000
7698	BLAKE	2850
7782	CLARK	2450
7566	JONES	2975
7654	MARTIN	1250
7499	ALLEN	1600
7844	TURNER	1500
7900	JAMES	950

SALGRADE		
GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999



La Non-Equijointure

```
SELECT empno,ename, job, sal, losal,hisal,grade
FROM emp INNER JOIN salgrade
ON sal BETWEEN losal and hisal
ORDER BY sal;
```

GRADE	LOSAL	HISAL
1	700	1200
2	1201	1400
3	1401	2000
4	2001	3000
5	3001	9999

EMPNO	ENAME	JOB	SAL	LOSAL	HISAL	GRADE
7369	SMITH	CLERK	800	700	1200	1
7900	JAMES	CLERK	950	700	1200	1
7876	ADAMS	CLERK	1100	700	1200	1
7521	WARD	SALESMAN	1250	1201	1400	2
7654	MARTIN	SALESMAN	1250	1201	1400	2
7934	MILLER	CLERK	1300	1201	1400	2
7844	TURNER	SALESMAN	1500	1401	2000	3
7499	ALLEN	SALESMAN	1600	1401	2000	3
7782	CLARK	MANAGER	2450	2001	3000	4

La jointure naturelle

- ❑ La clause **NATURAL JOIN** est basée sur toutes les colonnes des deux tables portant le même nom.
- ❑ Elle sélectionne les lignes des deux tables dont les valeurs sont identiques dans toutes les colonnes qui correspondent.
- ❑ Si les colonnes portant le même nom présentent des types de données différents, une erreur est renvoyée.

Jointure naturelle

```
SELECT department_id, department_name,  
location_id, city  
FROM departments
```

NATURAL JOIN locations ;

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
60	IT	1400	Southlake
50	Shipping	1500	South San Francisco
10	Administration	1700	Seattle
90	Executive	1700	Seattle
110	Accounting	1700	Seattle
190	Contracting	1700	Seattle
20	Marketing	1800	Toronto
80	Sales	2500	Oxford

Opérateurs ensemblistes

OPERATEUR	DESCRIPTION
INTERSECT	Ramène toutes les lignes communes aux deux requêtes
UNION	Toutes les lignes distinctes ramenées par les deux requêtes
UNION ALL	Toutes les lignes ramenées par les deux requêtes y compris les doublons
MINUS	Toutes les lignes ramenées par la première requête sauf les lignes ramenées par la seconde requête

Union

- ❑ Le nombre de colonnes et le type des colonnes doivent être identiques dans les 2 ordres Select
- ❑ L'opérateur UNION intervient sur toutes les colonnes

```
SELECT ename, job, sal FROM  
  emp  
WHERE job ='SALESMAN'  
ORDER BY ename;
```

ENAME	JOB	SAL
ALLEN	SALESMAN	1600
MARTIN	SALESMAN	1250
TURNER	SALESMAN	1500
WARD	SALESMAN	1250

```
SELECT ename, job, sal from  
  emp  
where sal >= 1500 ORDER BY  
  ename;
```

ENAME	JOB	SAL
ALLEN	SALESMAN	1600
BLAKE	MANAGER	2850
CLARK	MANAGER	2450
FORD	ANALYST	3000
JONES	MANAGER	2975
KING	PRESIDENT	5000
SCOTT	ANALYST	3000
TURNER	SALESMAN	1500

Union

```
SELECT ENAME, job, sal FROM emp  
WHERE job ='SALESMAN'
```

UNION

```
SELECT ename, job, sal FROM Emp  
WHERE sal >= 1500  
ORDER BY ename
```

❑ L'opérateur **UNION ALL** ne supprime pas les doublons.

ENAME	JOB	SAL
ALLEN	SALESMAN	1600
BLAKE	MANAGER	2850
CLARK	MANAGER	2450
FORD	ANALYST	3000
JONES	MANAGER	2975
KING	PRESIDENT	5000
MARTIN	SALESMAN	1250
SCOTT	ANALYST	3000
TURNER	SALESMAN	1500
WARD	SALESMAN	1250

Intersection

```
SELECT ENAME, job, sal FROM emp  
WHERE job ='SALESMAN'
```

INTERSECT

```
SELECT ename, job, sal FROM Emp  
WHERE sal >= 1500  
ORDER BY ename;
```

ENAME	JOB	SAL
ALLEN	SALESMAN	1600
TURNER	SALESMAN	1500

MINUS

```
SELECT ENAME, job, sal FROM emp  
WHERE job ='SALESMAN'
```

MINUS

```
SELECT ename, job, sal FROM Emp  
WHERE sal >= 1500  
ORDER BY ename;
```

```
SELECT ename, job, sal FROM Emp  
WHERE sal >= 1500
```

MINUS

```
SELECT ENAME, job, sal FROM emp  
WHERE job ='SALESMAN'  
ORDER BY ename
```

ENAME	JOB	SAL
MARTIN	SALESMAN	1250
WARD	SALESMAN	1250

ENAME	JOB	SAL
BLAKE	MANAGER	2850
CLARK	MANAGER	2450
FORD	ANALYST	3000
JONES	MANAGER	2975
KING	PRESIDENT	5000
SCOTT	ANALYST	3000

Les Sous-interrogations

SELECT select_liste

FROM table

WHERE **expr operateur (SELECT select_liste FROM table)**


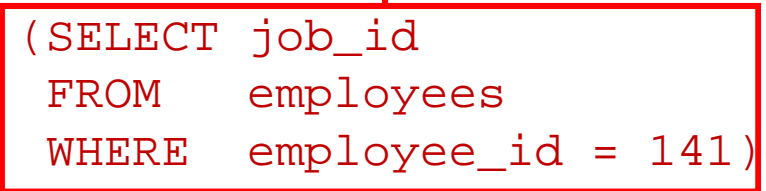


- La sous-interrogation (requête interne) est exécutée une seule fois avant la requête principale
- Le résultat de la sous-interrogation est utilisé par la requête principale (requête externe)
- Une sous-interrogation est utilisée dans les clauses (WHERE, HAVING et FROM)
- Les opérateurs de comparaison mono-ligne (>, >=, <, <=, ...)
- Les opérateurs de comparaison multi-ligne (IN, ALL, ANY)

Les Sous-interrogations mono ligne

- ❑ Renvoient une seule ligne
- ❑ Utilisent des opérateurs de comparaison monolignes

Opérateur	Signification
=	Egal à
>	Supérieur à
>=	Supérieur ou égal à
<	Inférieur à
<=	Inférieur ou égal à
<>	Non égal à

Les Sous-interrogations mono ligne

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =  ST_CLERK
        
AND    salary >  2600
        
```

LAST_NAME	JOB_ID	SALARY
Rajs	ST_CLERK	3500
Davies	ST_CLERK	3100

Les Sous-interrogations multiligne

- ❑ Renvoient plusieurs lignes
- ❑ Utilisent des opérateurs de comparaison multilignes

Opérateur	Signification
IN	Egal à un nombre quelconque de la liste
ANY	Compare la valeur à chaque valeur renvoyée par la sous-interrogation
ALL	Compare la valeur à toutes les valeurs renvoyées par la sous-interrogation

Les Sous-interrogations multiligne

```
SELECT employee_id, last_name, job_id, salary
FROM   employees          9000, 6000, 4200
WHERE  salary < ANY
      (SELECT salary
       FROM   employees
       WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	ST_MAN	5800
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500

...
10 rows selected.

Les Sous-interrogations multi ligne

```
SELECT employee_id, last_name, job_id, salary
FROM   employees      9000, 6000, 4200
WHERE  salary < ALL
      (SELECT salary
       FROM   employees
       WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	ST_CLERK	3500
142	Davies	ST_CLERK	3100
143	Matos	ST_CLERK	2600
144	Vargas	ST_CLERK	2500