



Base de données II

Concepts et programmation procédurale

Mise en œuvre avec

Oracle SQL

Agenda

Introduction

Consultation de données

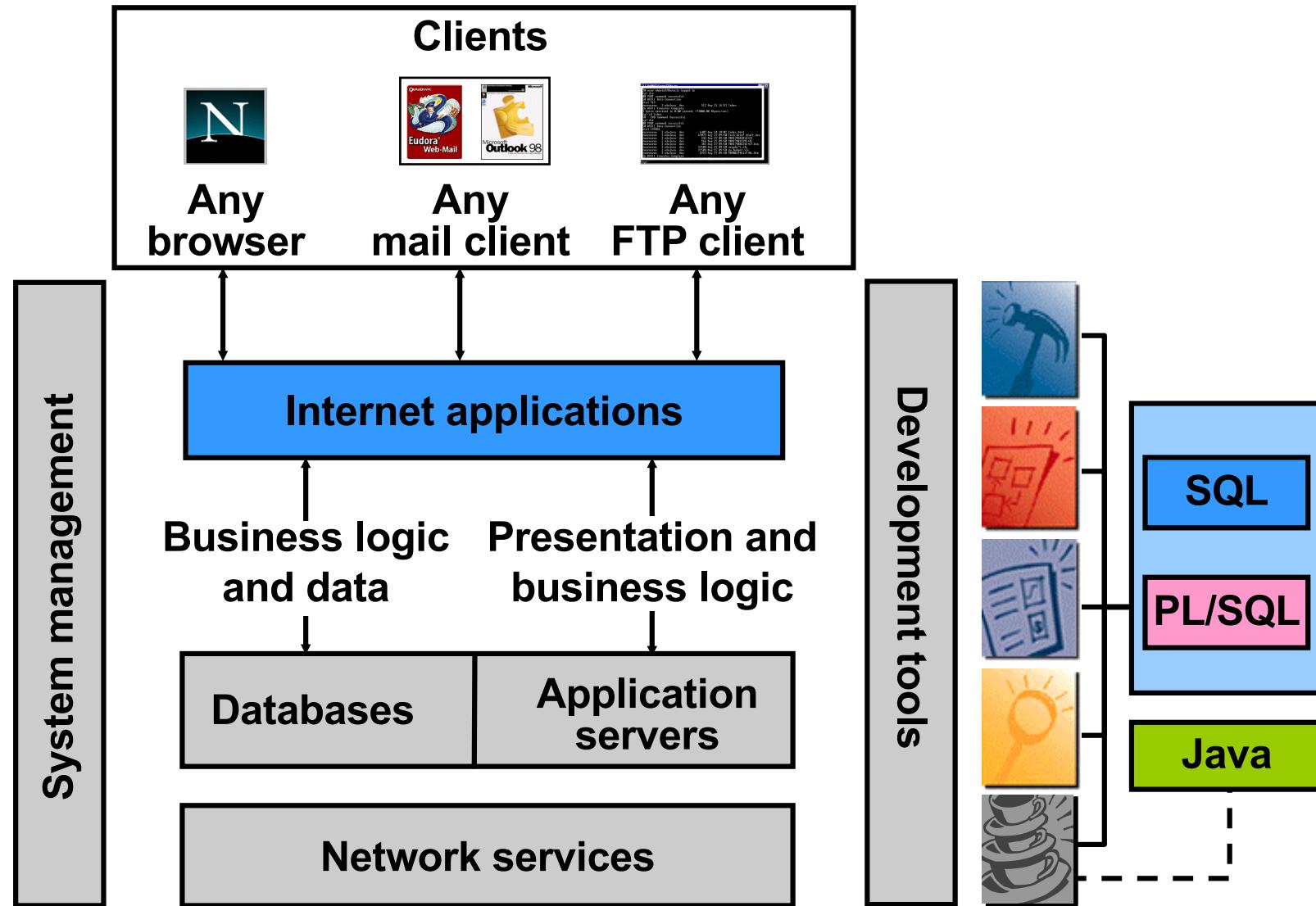
Manipulation de données (DML)

Définition de données

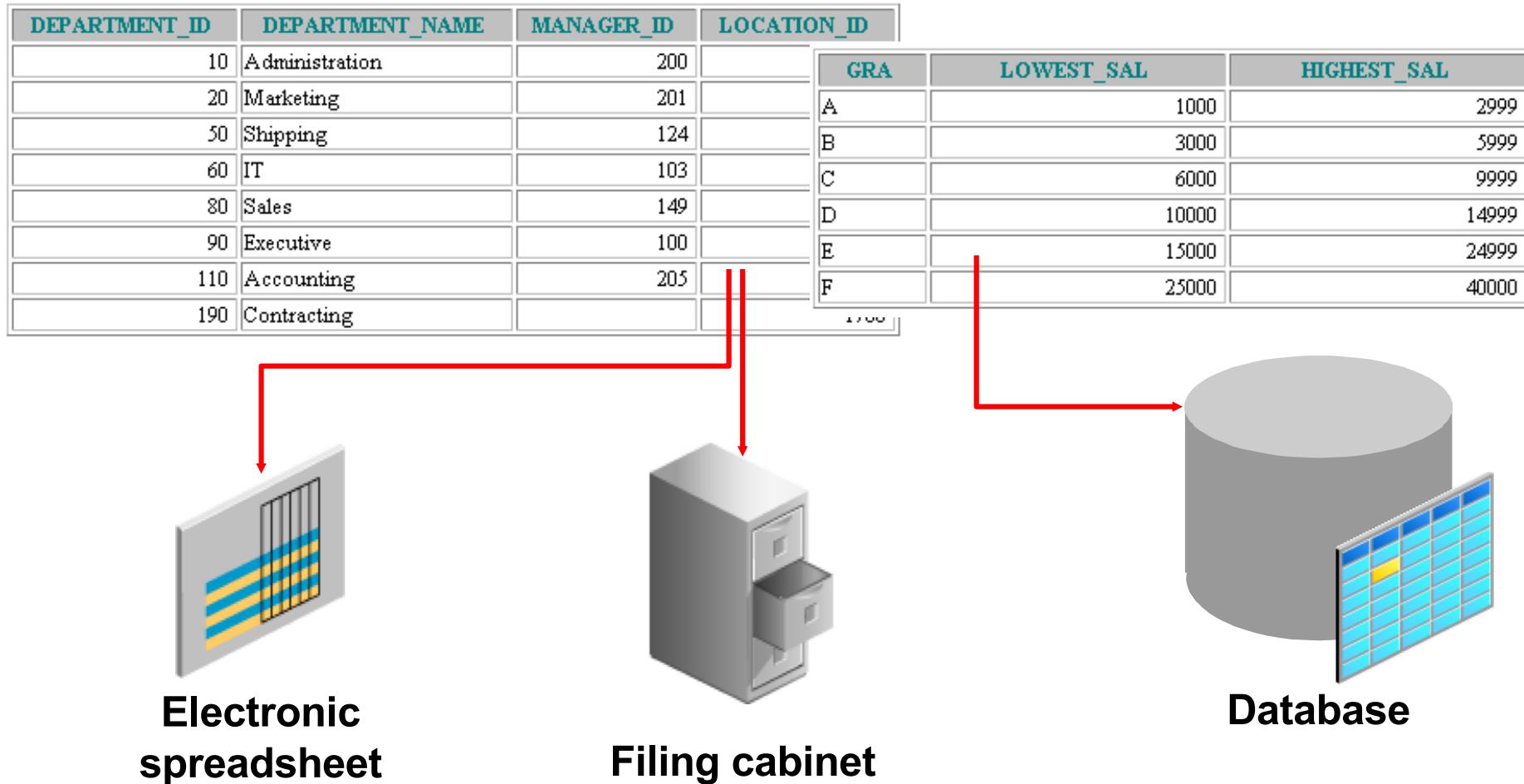
INTRODUCTION

ORACLE®

Oracle Internet Platform



Stockage de données sur différents supports

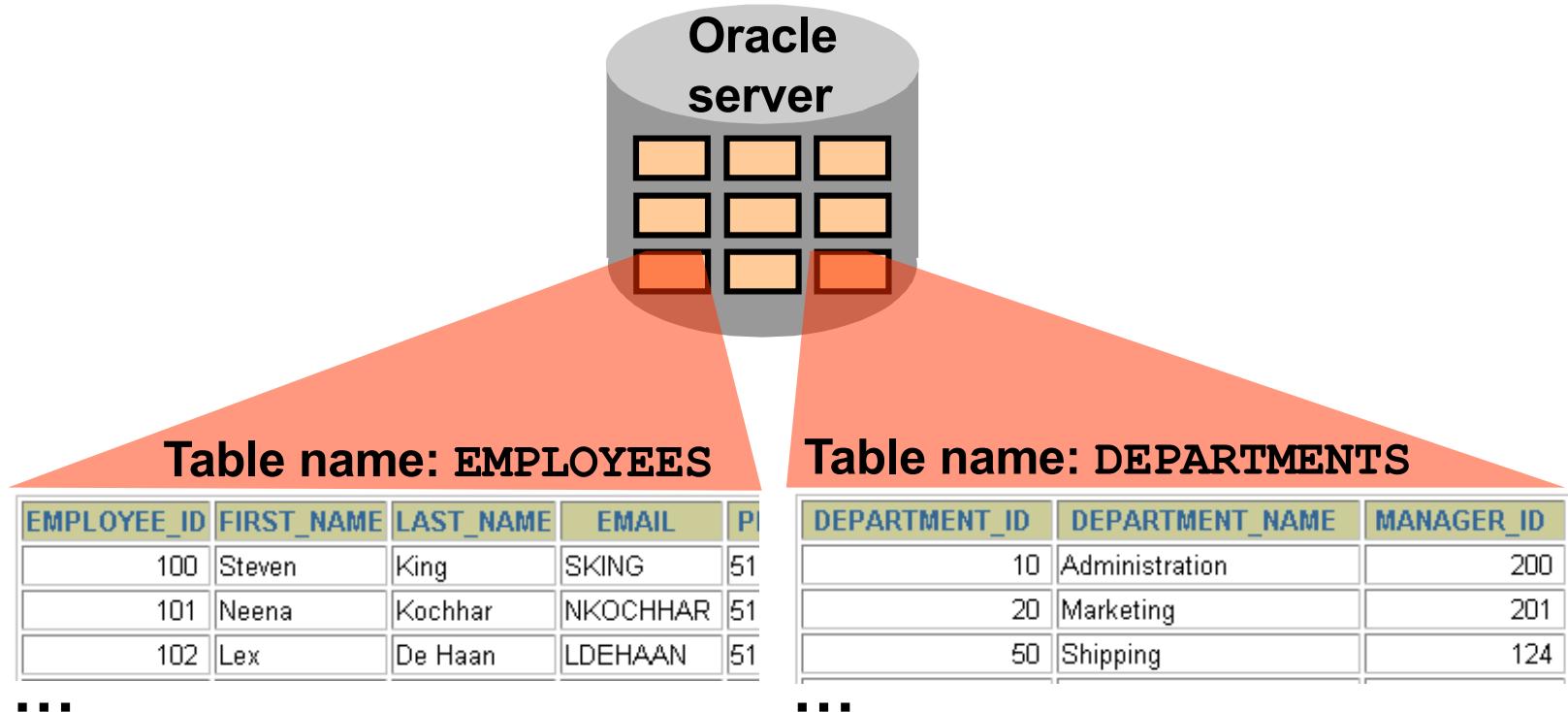


Base de données Relationelle: Concept

- Dr. E. F. Codd a proposé le model relationel pour les systemes de gestion de base de données en 1970.
- Il est la base de *relational database management system (RDBMS)*.
- Le modèle relationnel se compose des éléments suivants
 - Collection d'objets ou relations
 - Ensemble d'opérateurs qui agissent sur les relations
 - Intégrité des données pour l'exactitude et la cohérence

Définition d'une base de données relationnelle

Une base de données relationnelle est un ensemble de relations ou de table à deux dimensions.



Lier plusieurs Tables

- Chaque ligne de données dans une table est identifiée par une clé primaire (PK).
- Vous pouvez relier logiquement de données de plusieurs tables à l'aide de clés étrangères (FK).

Table name: EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	DEPARTMENT_ID
174	Ellen	Abel	80
142	Curtis	Davies	50
102	Lex	De Haan	90
104	Bruce	Ernst	60
202	Pat	Fay	20
206	William	Gietz	110

...

Primary key

Foreign key

Table name: 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
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

Primary key

Relational Database Terminology

A column as PK

EMPLOYEE_ID	LAST_NAME	FIRST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
100	King	Steven	24000		90
101	Kochhar	Neena	17000		90
102	De Haan	Lex	17000		90
103	Hunold	Alexander	9000		60
104	Ernst	Bruce	6000		60
107	Lorentz	Diana	4200		60
124	Mourgos	Kevin	5800		50
141	Rajs	Trenna	3500		50
142	Davies	Curtis	3100		50
143	Matos	Randall	2600		50
144	Vargas	Peter	2500		50
149	Zlotkey	Eleni	10500	.2	80
174	Abel	Ellen	11000	.3	80
176	Taylor	Jonathon	8600	.2	80
178	Grant	Kimberely	7000	.15	
200	Whalen	Jennifer	4400		10
201	Hartstein	Michael	13000		20
202	Fay	Pat	6000		20
205	Higgins	Shelley	12000		110
206	Gietz	William	8300		110

A single row

2

A column not a PK

3

null value

4

a foreign key.

5

A field

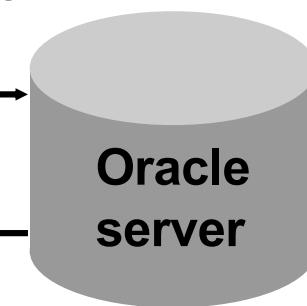
6

Communiquer avec RDBMS en utilisant SQL

SQL statement is entered.

```
SELECT department_name  
FROM departments;
```

**Statement is sent to
Oracle server.**



SQL Statements

SELECT
INSERT
UPDATE
DELETE
MERGE

Data manipulation language (DML)

CREATE
ALTER
DROP
RENAME
TRUNCATE
COMMENT

Data definition language (DDL)

GRANT
REVOKE

Data control language (DCL)

COMMIT
ROLLBACK
SAVEPOINT

Transaction control

Les Tables Utilisées dans le cours

EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	King	SKING	515.123.4567	17-JUN-87	AD_PRES	240
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	170
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	170
103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	90
104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-91	IT_PROG	60
107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-99	IT_PROG	42
124	Kevin	Mourgos	KMOURGOS	650.123.5234	16-NOV-99	ST_MAN	58
141	Trenna	Rajs	TRAJS	650.121.8009	17-OCT-95	ST_CLERK	35
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	31

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
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

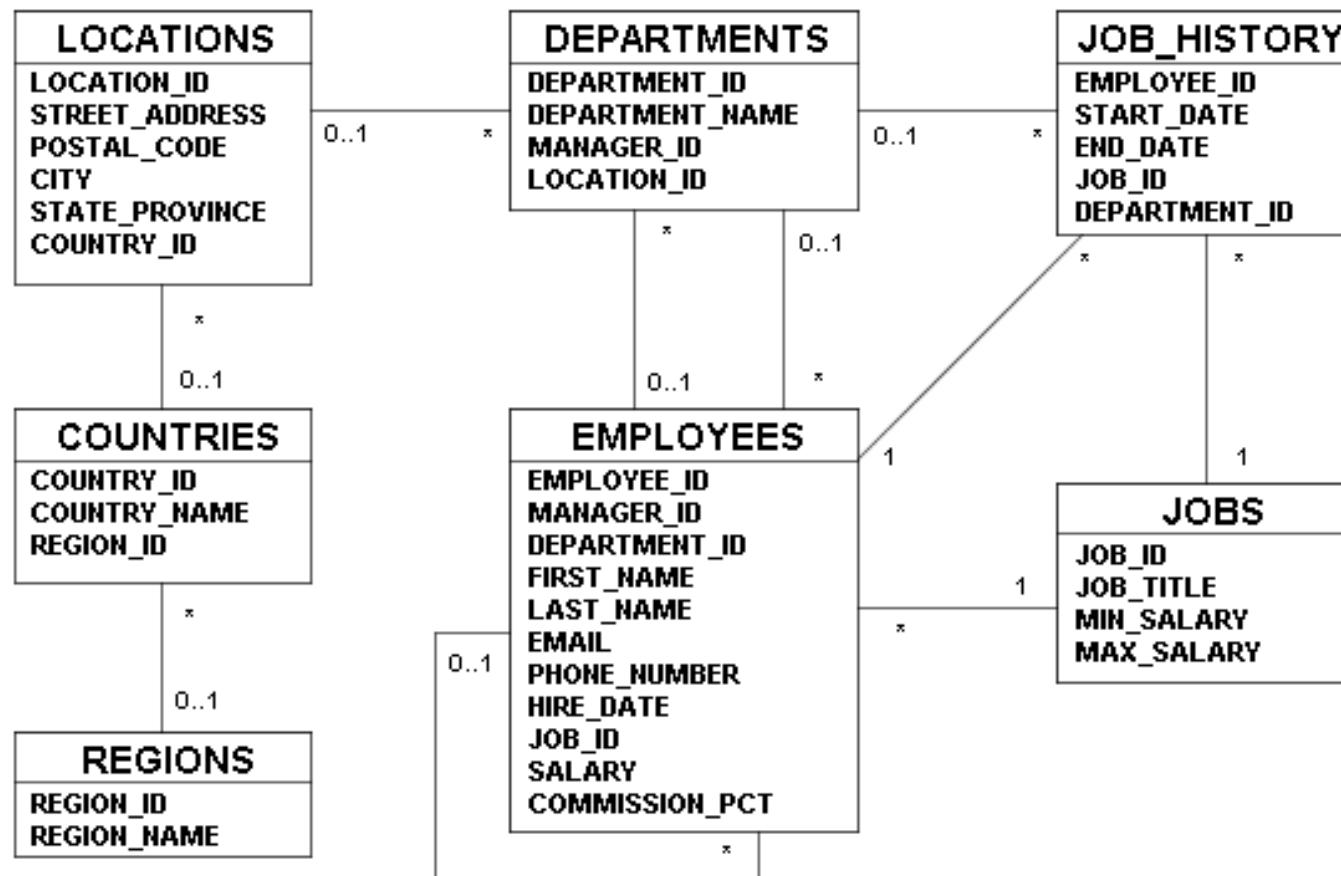
DEPARTMENTS

GRADE	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

JOB_GRADES

ORACLE®

Human Resources (hr) Data Set



Récupération de données avec l'instruction SELECT

SQL SELECT

Projection

Table 1

Selection

Table 1

Join

Table 1

Table 2

La requête SELECT

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

- **SELECT identifie les colonnes à afficher.**
- **FROM identifie la table contenant ces colonnes.**

Selectionner toutes les colonnes

```
SELECT *
FROM 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
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

8 rows selected.

Selectionner des Colonnes Specifiques

```
SELECT department_id, location_id  
FROM departments;
```

DEPARTMENT_ID	LOCATION_ID
10	1700
20	1800
50	1500
60	1400
80	2500
90	1700
110	1700
190	1700

8 rows selected.

Utilisation des Operateurs

```
SELECT last_name, salary, salary + 300  
FROM employees;
```

LAST_NAME	SALARY	SALARY+300
King	24000	24300
Kochhar	17000	17300
De Haan	17000	17300
Hunold	9000	9300
Ernst	6000	6300
...		

20 rows selected.

La valeur Null

- Une valeur null est une valeur qui n'est pas disponible, non attribuées, inconnu ou inapplicable.
- Une valeur null n'est pas identique à un zéro ou un espace blanc.

```
SELECT last_name, job_id, salary, commission_pct  
FROM employees;
```

LAST_NAME	JOB_ID	SALARY	COMMISSION_PCT
King	AD_PRES	24000	
Kochhar	AD_VP	17000	
...			
Zlotkey	SA_MAN	10500	.2
Abel	SA_REP	11000	.3
Taylor	SA_REP	8600	.2
...			
Gietz	AC_ACCOUNT	8300	

20 rows selected.

Valeurs null dans les Expressions arithmétiques

Les expressions arithmétiques qui contient une valeur nulle sont évaluées à null.

```
SELECT last_name, 12*salary*commission_pct  
FROM employees;
```

Kochhar	
Bana	
Manzal	TOTALMISSIONS*RAJAS=51
...	
Zlotkey	25200
Abel	39600
Taylor	20640
...	
Gietz	

20 rows selected.

Utilisation des alias de colonnes

```
SELECT last_name AS name, commission_pct comm  
FROM employees;
```

NAME	COMM
King	
Kochhar	
De Haan	
...	

20 rows selected.

```
SELECT last_name "Name" , salary*12 "Annual Salary"  
FROM employees;
```

Name	Annual Salary
King	288000
Kochhar	204000
De Haan	204000
...	

20 rows selected.

Opérateur de Concaténation

L'opérateur de concaténation :

- concatène des colonnes ou des strings
- Est représenté par (||)

```
SELECT      last_name || job_id AS "Employees"  
FROM        employees;
```

Employees
KingAD_PRES
KochharAD_VP
De HaanAD_VP
...

20 rows selected.

Utilisation de la clause WHERE

- Limiter les lignes rentrées à l'aide de la clause WHERE :

```
SELECT * | { [DISTINCT] column|expression [alias],... }  
FROM   table  
[WHERE condition(s) ] ;
```

Utiliser la condition NULL

Tester si null avec l'opérateur IS NULL.

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

LAST_NAME	MANAGER_ID
King	

Utiliser la fonction NVL

```
SELECT last_name, salary, NVL(commission_pct, 0),  
       (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL  
FROM employees;
```

LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
King	24000	0	288000
Kochhar	17000	0	204000
De Haan	17000	0	204000
Hunold	9000	0	108000
Ernst	6000	0	72000
Lorentz	4200	0	50400
Mourgos	5800	0	69600
Rajs	3500	0	42000
...			

20 rows selected.

Expressions Conditionnelles

- Permet l'utilisation de la logique IF-THEN-ELSE dans une requête SQL
- Utilise deux méthodes:
 - L'expression CASE
 - La fonction DECODE

CASE Expression

Facilite les requêtes conditionnelles en faisant le travail
d'une Instruction IF-THEN-ELSE :

```
CASE expr WHEN comparison_expr1 THEN return_expr1
            [WHEN comparison_expr2 THEN return_expr2
            WHEN comparison_exprn THEN return_exprn
            ELSE else_expr]
END
```

Utiliser une expression CASE

Exemple

```
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;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

Utiliser la Fonction DECODE

```
DECODE(col|expression, search1, result1  
      [, search2, result2,...,]  
      [, default])
```

```
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;
```

LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...			
Lorentz	IT_PROG	4200	4620
Mourgos	ST_MAN	5800	5800
Rajs	ST_CLERK	3500	4025
...			
Gietz	AC_ACCOUNT	8300	8300

20 rows selected.

ORACLE®

Exercices

- 1. Produire un rapport qui affiche le nom et le salaire des employés qui gagnent plus de 12 000 \$. Le salaire de l'employé King doit être masqué avec des ****.**
- 2. Modifier la requête pour afficher le nom et le salaire des employés qui gagnent entre 5 000 \$ et 12 000 \$ et sont dans le service 20 ou 50. Étiqueter les colonnes par employé et salaire mensuel, respectivement.**

Corrigé

1. Produire un rapport qui affiche le nom et le salaire des employés qui gagnent plus de 12 000 \$. Le salaire de l'employé King doit être masqué avec des ****.

```
SELECT last_name, job_id,
       DECODE(last_name, 'King', '*****', salary)
          SALARY
     FROM employees
    Where salary>12000;
```

Exercices

3. Ecrire une requête qui retourne le nom d'un employé et le niveau de son salaire.

- Le niveau du salaire est
 - 'Low' si le salary<5000
 - 'Medium' si 5000<=salary<10000
 - 'Good' si 10000<=salary<20000 THEN '
sinon, il est 'Excellent'

Corrigé

3. Ecrire une requête qui retourne le nom d'un employé et le niveau de son salaire.

- Le niveau du salaire est
 - 'Low' si le salary<5000
 - 'Medium' si 5000<=salary<10000
 - 'Good' si 10000<=salary<20000 THEN '
sinon, il est 'Excellent'

```
SELECT last_name,salary,  
(CASE WHEN salary<5000 THEN 'Low'  
WHEN salary<10000 THEN 'Medium'  
WHEN salary<20000 THEN 'Good'  
ELSE 'Excellent'  
END) qualified_salary  
FROM employees;
```

Utiliser la clause ORDER BY

- Trier le résultat avec la clause ORDER BY :
 - ASC: avec un ordre ascendant, par default
 - DESC: avec un ordre descendant

```
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.

ORACLE®

Utiliser la clause ORDER BY

- Tri descendant:

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

1

- Tri d'un alias:

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

2

- Tri de multiples colonnes:

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

3

Exercices

- 1. Créer une requête pour afficher le nom et l'emploi de tous les employés qui n'ont pas de manager.**
- 2. Créer un rapport pour afficher le nom, le salaire et la commission de tous les employés qui gagnent des commissions. Trier les données par ordre décroissant de salaires et de commissions**

Utiliser Group By

La fonction Group By opèrent sur des ensembles de lignes pour donner un résultat par groupe

EMPLOYEES

DEPARTMENT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
50	5800
50	3500
50	3100
50	2600
50	2500
80	10500
80	11000
80	8600
	7000
10	4400

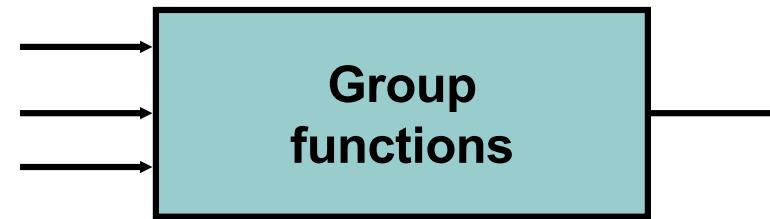
20 rows selected.

Maximum salary in
EMPLOYEES table

MAX(SALARY)
24000

Types de Group Functions

- **AVG**
- **COUNT**
- **MAX**
- **MIN**
- **STDDEV**
- **SUM**
- **VARIANCE**



Group Functions: Syntax

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

Utiliser les fonctions AVG, SUM, MIN et MAX

Vous pouvez utiliser AVG et SUM pour des données numériques.

```
SELECT AVG(salary), MAX(salary),  
       MIN(salary), SUM(salary)  
FROM   employees  
WHERE  job_id LIKE '%REP%';
```

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8150	11000	6000	32600

Vous pouvez utiliser MIN et MAX pour les types de données date, numérique ou chaîne caractère.

```
SELECT MIN(hire date), MAX(hire date)  
FROM   employees;
```

MIN(HIRE_	MAX(HIRE_
17-JUN-87	29-JAN-00

Utiliser la fonction COUNT

COUNT (*) retourne le nombre de lignes dans une table

:

1

```
SELECT COUNT(*)  
FROM employees  
WHERE department_id = 50;
```

COUNT()

5

COUNT (expr) retourne le nombre de lignes avec des valeurs non nulles pour l'expr:

2

```
SELECT COUNT(commission_pct)  
FROM employees  
WHERE department_id = 80;
```

COUNT(COMMISSION_PCT)

3

Utiliser le mot clé DISTINCT

- COUNT(DISTINCT expr) retourne le nombre de valeurs non null distinctes de l'expr.
- afficher le nombre de valeurs distinctes de département_id dans la table EMPLOYEES

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

COUNT(DISTINCTDEPARTMENT_ID)
7

La Function Group by et les Valeurs Null

Les functions Group by ignorent les valeurs null:

1

```
SELECT AVG(commission_pct)  
FROM employees;
```

AVG(COMMISSION_PCT)

.2125

La fonction NVL force group functions à inclure les null values:

2

```
SELECT AVG(NVL(commission_pct, 0))  
FROM employees;
```

AVG(NVL(COMMISSION_PCT,0))

.0425

La clause GROUP BY : Syntaxe

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

Vous pouvez diviser les lignes d'une table en petits groupes en utilisant la clause GROUP BY.

La clause GROUP BY : Exemple

Toutes les colonnes dans la liste de SELECT et qui ne sont pas dans les fonctions de groupe doivent être dans la clause GROUP BY..

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

DEPARTMENT_ID	AVG(SALARY)
10	4400
20	9500
50	3500
60	6400
80	10033.3333
90	19333.3333
110	10150
	7000

8 rows selected.

Restreindre les Résultats de Group by avec la Clause HAVING

Lorsque vous utilisez la clause HAVING, le serveur Oracle restreint les groupes comme suit :

- 1. Les lignes sont regroupées.**
- 2. La function group by est appliquée.**
- 3. Les groupes vérifiant la clause HAVING sont affichés.**

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

Restreindre les Résultats de Group by avec la Clause HAVING

```
SELECT      department_id, MAX(salary)
FROM        employees
GROUP BY    department_id
HAVING      MAX(salary)>10000 ;
```

DEPARTMENT_ID	MAX(SALARY)
20	13000
80	11000
90	24000
110	12000

Restreindre les Résultats de Group by avec la Clause HAVING

```
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
IT_PROG	19200
AD_PRES	24000
AD_VP	34000

Exercices

Le département HR a besoin des rapports suivants:

- 1.** Trouver le plus haut, le plus bas, la somme et la moyenne des salaires de tous les employés. Étiqueter les colonnes respectivement par Maximum, Minimum, Sum et Average,
- 2.** Modifier la requête pour afficher le minimum, maximum, sum, et average des salaires pour chaque types d'emploi.
- 3.** Écrire une requête pour afficher le nombre de personnes ayant le même travail.

Exercices

- 1. Créer une requête pour afficher le nombre total d'employés et, pour ce total, le nombre d'employés embauchés en 1995, 1996, 1997 et 1998. Créer des en-têtes de colonnes appropriés.**
- 2. Créer une requête pour afficher les emplois (job), les salaires correspondants basés sur le numéro de département et les sommes de salaires pour ces emplois, pour les départements 20, 50, 80 et 90. Créer des en-têtes de colonnes appropriés.**

Excercise

```
SELECT COUNT(*) total,  
      SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1995,1,0))"1995",  
      SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1996,1,0))"1996",  
      SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1997,1,0))"1997",  
      SUM(DECODE(TO_CHAR(hire_date, 'YYYY'),1998,1,0))"1998"  
FROM employees;
```

HIERARCHICAL RETRIEVAL

Echantillons de la Table EMPLOYEES

EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
100	King	AD_PRES	
101	Kochhar	AD_VP	100
102	De Haan	AD_VP	100
103	Hunold	IT_PROG	102
104	Ernst	IT_PROG	103
105	Austin	IT_PROG	103
106	Pataballa	IT_PROG	103
107	Lorentz	IT_PROG	103
108	Greenberg	FI_MGR	101

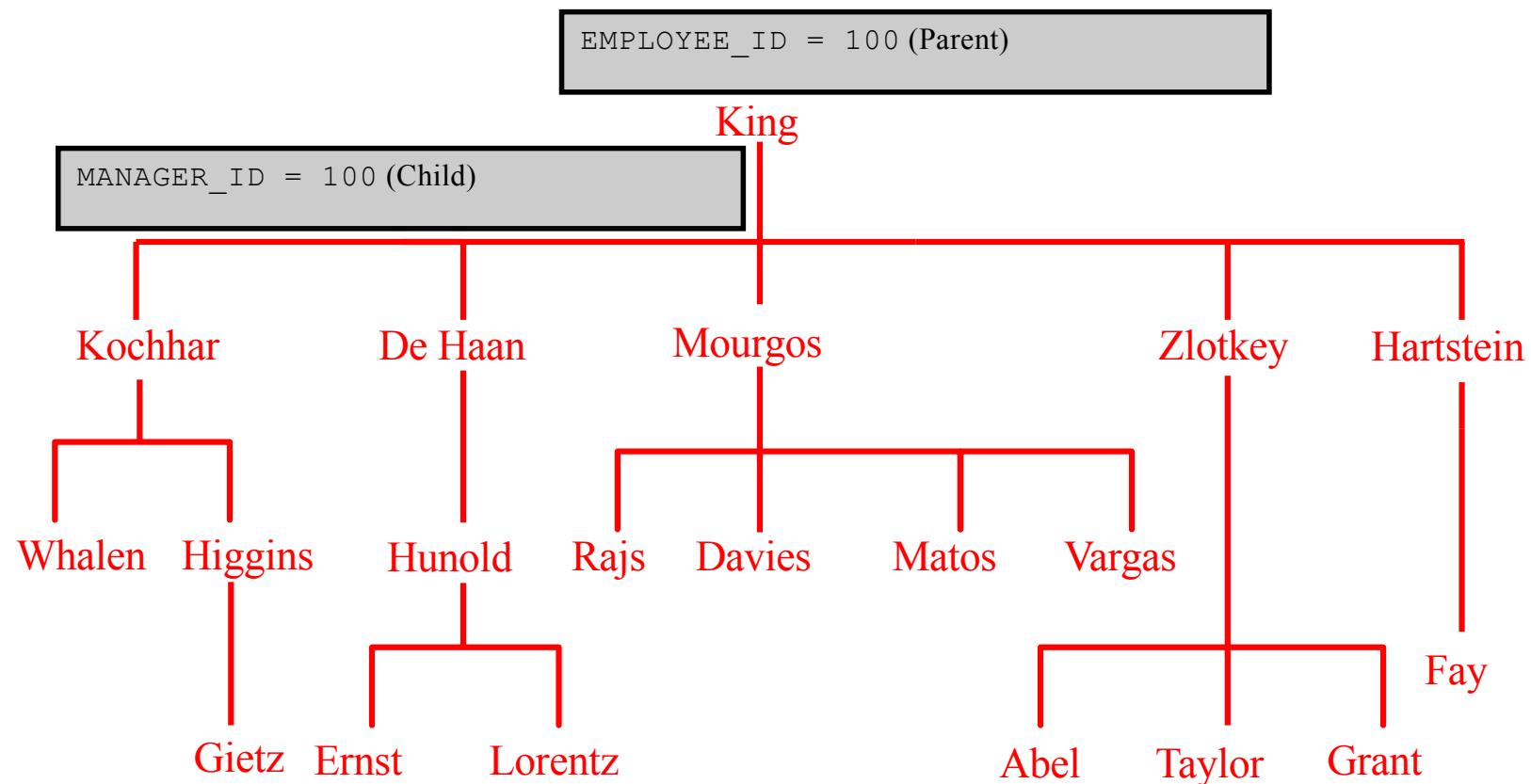
...

EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
196	Walsh	SH_CLERK	124
197	Feeney	SH_CLERK	124
198	OConnell	SH_CLERK	124
199	Grant	SH_CLERK	124
200	Whalen	AD_ASST	101
201	Hartstein	MK_MAN	100
202	Fay	MK_REP	201
203	Mavris	HR_REP	101
204	Baer	PR_REP	101
205	Higgins	AC_MGR	101
206	Gietz	AC_ACCOUNT	205

107 rows selected.

ORACLE®

Organigramme



Requêtes hiérarchiques

```
SELECT [LEVEL], column, expr...
  FROM table
 [WHERE condition(s)]
 [START WITH condition(s)]
 [CONNECT BY PRIOR condition(s)] ;
```

WHERE condition:

```
expr comparison_operator expr
```

Note: ce genre de requête SELECT ne peut pas contenir de jointure

Parcourir l'arbre

Starting Point

- Spécifie la condition à remplir
- Accepte n'importe quelle condition valable

```
START WITH column1 = value
```

Pour la table EMPLOYEES, commencer par l'employé Kochhar.

```
...START WITH last_name = 'Kochhar'
```

Note: Les clauses CONNECT BY PRIOR et START WITH ne sont pas des standards ANSI SQL.

Parcourir l'arbre

```
CONNECT BY PRIOR column1 = column2
```

Parcourir de haut en bas

```
... CONNECT BY PRIOR employee_id = manager_id
```

Direction

Top down → Column1 = Parent Key
Column2 = Child Key

Bottom up → Column1 = Child Key
Column2 = Parent Key

Ou encore

```
... CONNECT BY employee_id = PRIOR manager_id
```

Parcourir l'arbre : De bas en haut

```
SELECT employee_id, last_name, job_id, manager_id  
FROM   employees  
START  WITH employee_id = 101  
CONNECT BY PRIOR manager_id = employee_id ;
```

EMPLOYEE_ID	LAST_NAME	JOB_ID	MANAGER_ID
101	Kochhar	AD_VP	100
100	King	AD_PRES	

Note: La clause CONNECT BY ne peut pas contenir de sous-requêtes!

Parcourir l'arbre : De haut en bas

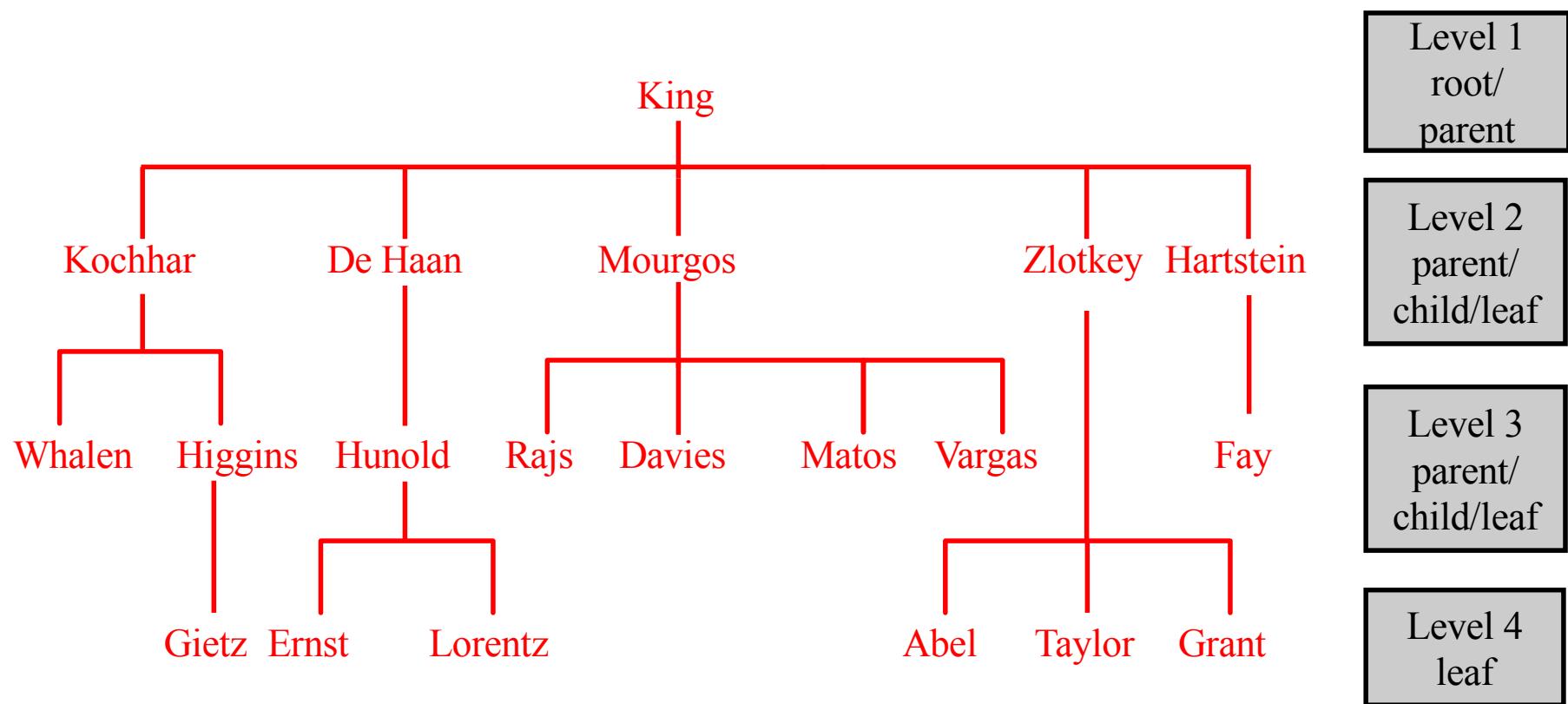
```
SELECT  last_name||' reports to '||  
PRIOR  last_name "Walk Top Down"  
FROM    employees  
START   WITH last_name = 'King'  
CONNECT BY PRIOR employee_id = manager_id ;
```

Walk Top Down

King reports to
King reports to
Kochhar reports to King
Greenberg reports to Kochhar
Faviet reports to Greenberg
Chen reports to Greenberg
...

108 rows selected.

Ranger les lignes avec LEVEL Pseudocolumn



Mise en forme des rapports hiérarchiques à l'aide LEVEL and LPAD

Afficher la hiérarchie avec une mise en retrait des différents niveaux.

```
COLUMN org_chart FORMAT A12
SELECT LPAD(last_name, LENGTH(last_name)+(LEVEL*2)-2, '_')
       AS org_chart
FROM   employees
START WITH first_name='Steven' AND last_name='King'
CONNECT BY PRIOR employee_id=manager_id
```

ORG_CHART
King
Kochhar
Greenberg
Faviet
Chen
Sciarrino
Urman
Popp
Whalen
Mavris
Baer
Higgins
Gietz

ORACLE®

Exercices

- Produire un rapport affichant un organigramme du département de Mr Mourgos. Afficher les noms, les salaires et les département IDs de ces employés
- Produire un organigramme de la société.
Commencez par la personne du plus haut niveau.

Consulter plusieurs tables

Obtention de données à partir de plusieurs Tables

EMPLOYEES

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
100	King	90
101	Kochhar	90
...		
202	Fay	20
205	Higgins	110
206	Gietz	110

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
10	Administration	1700
20	Marketing	1800
50	Shipping	1500
60	IT	1400
80	Sales	2500
90	Executive	1700
110	Accounting	1700
190	Contracting	1700



EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
200	10	Administration
201	20	Marketing
202	20	Marketing
...		
102	90	Executive
205	110	Accounting
206	110	Accounting

Types de Jointures

Tous les types de jointures compatibles avec le standard SQL:1999 incluant se qui suit:

- **Cross joins**
- **Natural joins**
- **USING clause**
- **Full (or two-sided) outer joins**
- **Arbitrary join conditions for outer joins**

Jointure de Tables à l'aide SQL:1999 Syntaxe

Utiliser les jointures pour interroger les données de plusieurs tables :

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

Créer Natural Joins

- La clause NATURAL JOIN se base sur toutes les colonnes dans les deux tables ayant le même nom.
- elle sélectionne les lignes des deux tables qui ont des valeurs égales pour toutes les colonnes ayant les mêmes noms.
- Si les colonnes ayant le même nom ont des types de données différents, une erreur est renvoyée.

Récupération des enregistrements avec Natural Joins

```
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

8 rows selected.

Récupération des Enregistrements avec Natural Joins et la Clause WHERE

```
SELECT department_id, department_name,  
       location_id, city  
  FROM departments  
NATURAL JOIN locations  
 WHERE department_id IN (20, 50);
```

Des restrictions supplémentaires sur une jointure naturelle sont implémentées à l'aide d'une clause WHERE.

Récupération des Enregistrements avec la Clause USING

- Si plusieurs colonnes portent le même nom, ayant des types de données différents, la clause NATURAL JOIN peut être remplacée par une clause USING pour spécifier les colonnes qui doivent être utilisées pour cette equijoin.
- La clause USING peut être utilisée pour spécifier uniquement les colonnes qui doivent être utilisées pour une equijoin.
- NATURAL JOIN et USING sont mutuellement exclusive.

Creating Joins with the USING Clause

EMPLOYEES

EMPLOYEE_ID	DEPARTMENT_ID
200	10
201	20
202	20
124	50
141	50
142	50
143	50
144	50
103	60
104	60
107	60
149	80
174	80
176	80

DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
20	Marketing
50	Shipping
60	IT
60	IT
60	IT
80	Sales
80	Sales
80	Sales

Foreign key

Primary key

Récupération des enregistrements avec la clause USING

```
SELECT employees.employee_id, employees.last_name,  
       departments.location_id, department_id  
  FROM employees JOIN departments  
USING (department_id);
```

EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
200	Whalen	1700	10
201	Hartstein	1800	20
202	Fay	1800	20
124	Mourgos	1500	50
141	Rajs	1500	50
142	Davies	1500	50
144	Vargas	1500	50
143	Matos	1500	50

...
19 rows selected.

Récupération des Enregistrements avec la Clause ON

```
SELECT e.employee_id, e.last_name, e.department_id,  
       d.department_id, d.location_id  
  FROM employees e JOIN departments d  
 WHERE e.department_id = d.department_id;
```

EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID	LOCATION_ID
200	Whalen	10	10	1700
201	Hartstein	20	20	1800
202	Fay	20	20	1800
124	Mourgos	50	50	1500
141	Rajs	50	50	1500
142	Davies	50	50	1500
143	Matos	50	50	1500
...				

19 rows selected.

Self-Joins Using the ON Clause

EMPLOYEES (WORKER)

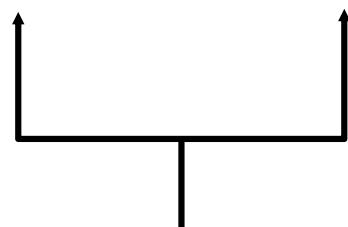
EMPLOYEE_ID	LAST_NAME	MANAGER_ID
100	King	
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103
107	Lorentz	103
124	Mourgos	100

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst
107	Lorentz
124	Mourgos

...



**MANAGER_ID in the WORKER table is equal to
EMPLOYEE_ID in the MANAGER table.**

Self-Joins Using the ON Clause

```
SELECT e.last_name emp, m.last_name mgr
FROM   employees e JOIN employees m
ON     (e.manager_id = m.employee_id);
```

EMP	MGR
Hartstein	King
Zlotkey	King
Mourgos	King
De Haan	King
Kochhar	King
...	

19 rows selected.

Nonequi joins

EMPLOYEES

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hunold	9000
Ernst	6000
Lorentz	4200
Mourgos	5800
Rajs	3500
Davies	3100
Matos	2600
Vargas	2500
Zlotkey	10500
Abel	11000
Taylor	8600
...	

20 rows selected.

JOB_GRADES

GRA	LOWEST_SAL	HIGHEST_SAL
A	1000	2999
B	3000	5999
C	6000	9999
D	10000	14999
E	15000	24999
F	25000	40000

← Salary in the EMPLOYEES table must be between lowest salary and highest salary in the JOB_GRADES table.

Récupération des Enregistrements avec Nonequi joins

```
SELECT e.last_name, e.salary, j.grade_level  
FROM employees e JOIN job_grades j  
ON e.salary  
BETWEEN j.lowest_sal AND j.highest_sal;
```

LAST_NAME	SALARY	GRA
Matos	2600	A
Vargas	2500	A
Lorentz	4200	B
Mourgos	5800	B
Rajs	3500	B
Davies	3100	B
Whalen	4400	B
Hunold	9000	C
Ernst	6000	C
...		

20 rows selected.

Outer Joins

DEPARTMENTS

DEPARTMENT_NAME	DEPARTMENT_ID
Administration	10
Marketing	20
Shipping	50
IT	60
Sales	80
Executive	90
Accounting	110
Contracting	190

8 rows selected.

EMPLOYEES

DEPARTMENT_ID	LAST_NAME
90	King
90	Kochhar
90	De Haan
60	Hunold
60	Ernst
60	Lorentz
50	Mourgos
50	Rajs
50	Davies
50	Matos
50	Vargas
80	Zlotkey

20 rows selected.

Aucun employé pour le département 190;
Une INNER JOIN (NATURAL JOIN, USING, ou ON) n'affichera pas ce département

LEFT OUTER JOIN

Cette requête récupère toutes les lignes de la table EMPLOYEES, qui est la table de gauche, même s'il n'a aucune correspondance dans la table DEPARTMENTS.

```
SELECT e.last_name, e.department_id, d.department_name  
FROM employees e LEFT OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing

...

De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		

20 rows selected.

L'employé Grant figure bien parmi les résultats grâce à une *OUTER JOIN*

RIGHT OUTER JOIN

Cette requête récupère toutes les lignes de la table DEPARTMENTS, qui est la table de droite, même s'il n'a aucune correspondance dans la table EMPLOYEES.

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

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
Davies	50	Shipping
...		
Kochhar	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
	190	Contracting

20 rows selected.

le département 190 figure bien parmi les résultats grâce à une *OUTER JOIN*

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name  
FROM employees e FULL OUTER JOIN departments d  
ON (e.department_id = d.department_id) ;
```

LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
Whalen	10	Administration
Fay	20	Marketing
Hartstein	20	Marketing
...		
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant		
	190	Contracting

21 rows selected.

Récupération des Enregistrements avec Cross Joins

- CROSS JOIN produit le produit cartésien de deux tables

```
SELECT last_name, department_name  
FROM employees  
CROSS JOIN departments ;
```

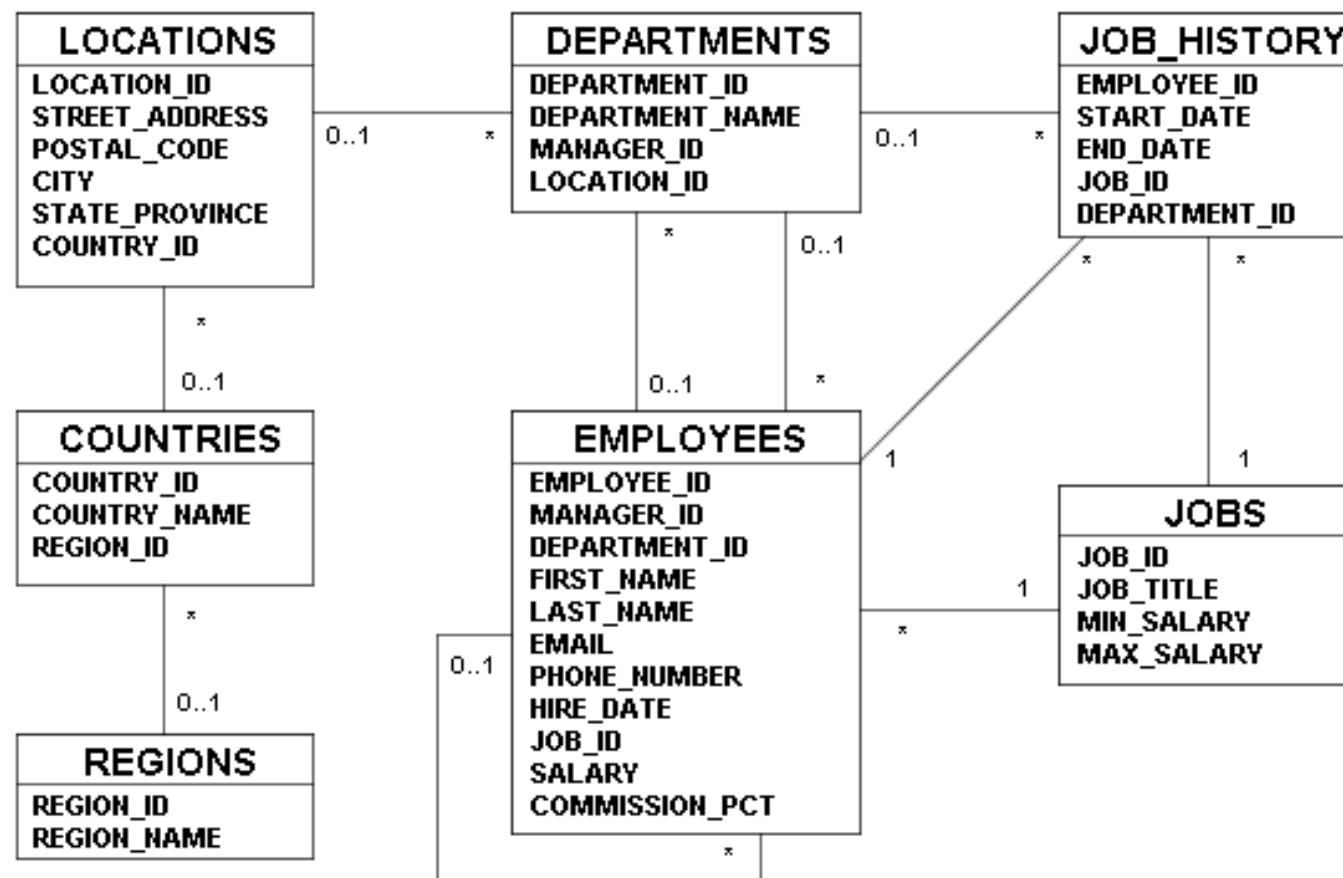
LAST_NAME	DEPARTMENT_NAME
King	Administration
Kochhar	Administration
De Haan	Administration
Hunold	Administration
■ ■ ■	

160 rows selected.

Exercices

- 1. Écrire une requête pour produire les adresses de tous les départements. Utiliser les tables LOCATIONS et COUNTRIES. Afficher la location ID, street address, city, state or province, et country dans le résultat. Utiliser une NATURAL JOIN pour produire ces résultats. tout d'abord afficher la structure de la table LOCATION**
- 2. créer une requête qui affiche le nom, le job, le nom du département, le salaire et le nom du grade de tous les employés.**

Human Resources (hr) Data Set



Exercices

- 3.** Produire un rapport des salariés de Toronto. Afficher le nom, le job, numéro de département et le nom du département pour tous les employés qui travaillent à Toronto.
- 4.** Produire un rapport qui affiche le nom d'un employé son Num ainsi que le nom de son manager et le Num de ce dernier. Étiqueter les colonnes par Employee, Emp#, Manager, and Mgr#, respectivement.

Exercices

- 5. Afficher les noms et les dates d'embauche pour tous les employés qui ont été embauchés avant leurs managers, ainsi que les noms et les dates d'embauche de leurs managers.**

Sous-requête : Syntaxe

```
SELECT      select_list
FROM        table
WHERE       expr operator
            (SELECT      select_list
             FROM       table) ;
```

- La sous-requête (requête interne) s'exécute avant la requête principale (requête externe).
- Le résultat de la sous-requête est utilisé par la requête principale.

Sous-requête : Exemple

```
SELECT last_name, salary  
FROM   employees    11000  
WHERE  salary >  
       (SELECT salary  
        FROM   employees  
        WHERE  last_name = 'Abel');
```

LAST_NAME	SALARY
King	24000
Kochhar	17000
De Haan	17000
Hartstein	13000
Higgins	12000

Manipulation de données

Data Manipulation Language: DML

- **Une instruction DML est utilisée lorsque vous :**
 - Ajoutez des lignes dans une table
 - Modifiez des lignes existantes dans une table
 - Supprimez les lignes existantes d'une table
- **Une *transaction* se compose d'une collection d'instructions DML qui forment une unité logique de travail.**

Ajouter une nouvelle ligne à une Table

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
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

70 Public Relations

100

1700

Nouvelle
ligne

Insérer la nouvelle ligne
dans la table
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
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting		1700

70 Public Relations

100

1700

ORACLE®

INSERT : Synataxe

- **INSERT statement:**

```
INSERT INTO    table [(column [, column...])]  
VALUES        (value [, value...]);
```

- **Avec cette syntaxe, une seule ligne est insérée à la fois.**

Insérer une nouvelle ligne

- Insérer une nouvelle ligne contenant les valeurs pour chaque colonne.
- Lister les colonnes et puis les valeurs correspondantes
- Par défaut, toutes les colonnes d'une table sont considérées

```
INSERT INTO departments(department_id,  
                      department_name, manager_id, location_id)  
VALUES (70, 'Public Relations', 100, 1700);  
1 row created.
```

- Encadrer les valeurs de caractère et de la date entre apostrophes.

Copie des lignes d'une autre Table

```
INSERT INTO sales_reps(id, name, salary, commission_pct)
SELECT employee_id, last_name, salary, commission_pct
FROM employees
WHERE job_id LIKE '%REP%';
```

4 rows created.

- Ne pas utiliser la clause VALUES.

UPDATE : Syntaxe

- **Modifier des données existantes avec UPDATE**

```
UPDATE      table
SET         column = value [, column = value, ...]
[WHERE      condition];
```

- **Peut modifier plusieurs lignes à la fois.**

Mettre à jour les lignes d'une Table

```
UPDATE employees  
SET department_id = 70  
WHERE employee_id = 113;  
1 row updated.
```

- Toutes les lignes de la table sont modifiées si vous omettez la clause WHERE :

```
UPDATE copy_emp  
SET department_id = 110;  
22 rows updated.
```

Mise à jour des lignes en à partir d'une autre Table

```
UPDATE copy_emp
SET department_id = (SELECT department_id
                      FROM employees
                      WHERE employee_id = 100)
WHERE job_id          = (SELECT job_id
                         FROM employees
                         WHERE employee_id = 200);
1 row updated.
```

DELETE

Syntaxe:

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

- **Exemple**

```
DELETE FROM departments
WHERE department_name = 'Finance';
1 row deleted.
```

- **Toutes les lignes dans la table sont supprimées si vous omettez la clause WHERE :**

```
DELETE FROM copy_emp;
22 rows deleted.
```

TRUNCATE

- Supprime toutes les lignes d'une table, laissant la structure de la table intacte.

Attention!

- est une instruction de langage (DDL) de définition de données plutôt qu'une instruction DML ; ne peut pas être facilement annulée par un ROLLBACK
- Ne déclenche pas les delete triggers de la table.
- Syntaxe:

```
TRUNCATE TABLE table_name;
```

- Exemple:

```
TRUNCATE TABLE copy_emp;
```

LES TRANSACTIONS

Les Transactions

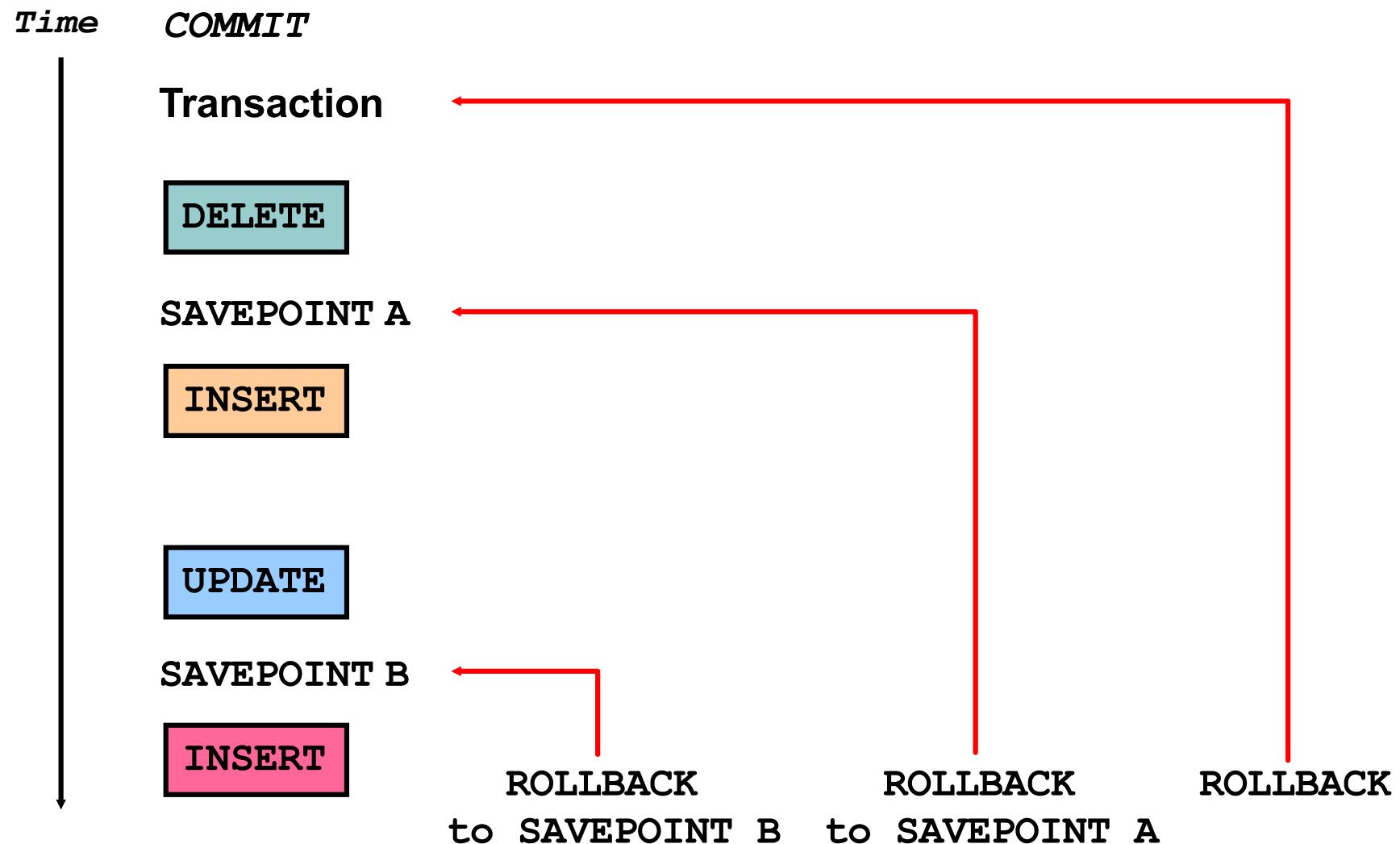
Une transaction de base de données est constituée de l'une des opérations suivantes

- **Un ensemble d'instructions DML qui constituent une logique de MAJ de la BD**
- **Une instruction DDL pour la définition de données**
- **Une instruction DCL pour le contrôle des données**

Les Transactions

- **Commence lors de la première instruction DML**
- **Se termine par l'un des événements suivants**
 - Un COMMIT ou ROLLBACK
 - Une instruction DDL ou DCL (Attention! Dans ce cas un commit automatique est effectué).
 - L'utilisateur quitte le système
 - Le système sort inopinément

Contrôler une Transactions



Note: SAVEPOINT is not ANSI standard SQL.

Annuler les changements avec ROLLBACK

- Créer un marqueur dans une transaction en cours en utilisant l'instruction SAVEPOINT.
- Annuler les chagements depuis ce marqueur à l'aide de l'instruction ROLLBACK TO SAVEPOINT.

```
UPDATE...  
SAVEPOINT update_done;  
Savepoint created.  
INSERT...  
ROLLBACK TO update_done;  
Rollback complete.
```

Validation des données avec COMMIT

- Apportez les modifications

```
DELETE FROM employees  
WHERE employee_id = 99999;  
1 row deleted.
```

```
INSERT INTO departments  
VALUES (290, 'Corporate Tax', NULL, 1700);  
1 row created.
```

- Validez les modifications :

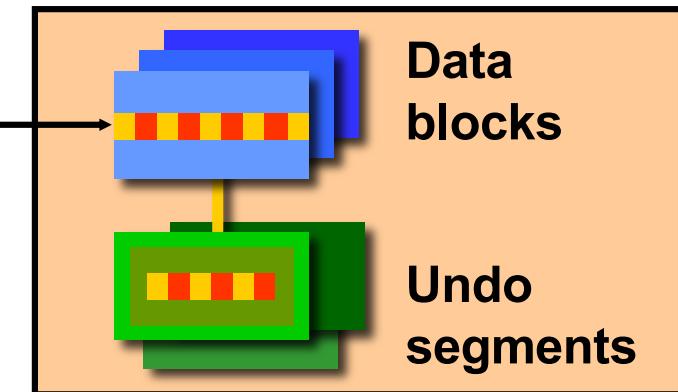
```
COMMIT;  
Commit complete.
```

Lecture cohérente

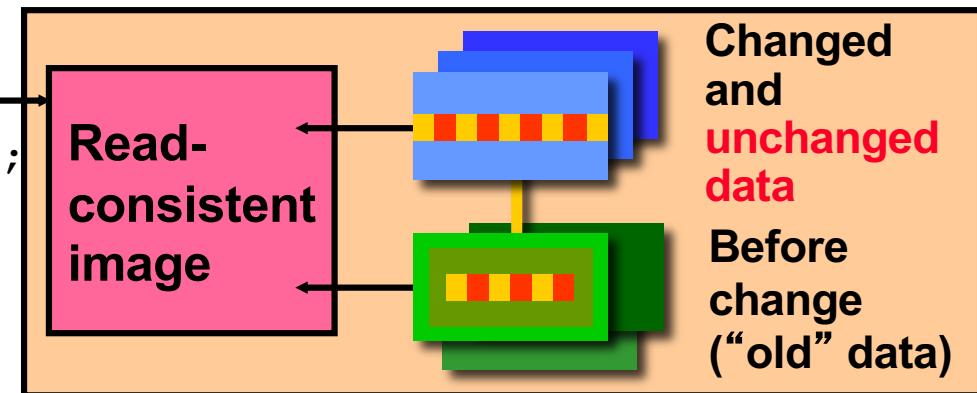
User A



```
UPDATE employees  
SET salary = 7000  
WHERE last_name = 'Grant';
```



```
SELECT *  
FROM userA.employees;
```



User B

Utilisation de DDL pour la définition de données

Database Objects

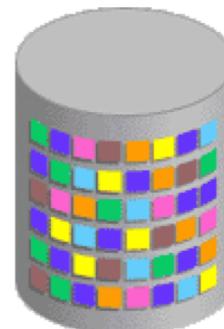
Object	Description
Table	L'unité basique de stockage. Elle est composée de lignes de données
View	Représente logiquement un sous-ensemble de données d'une ou de plusieurs tables.
Sequence	Génère une séquence de valeurs numériques
Index	Améliore la performance des requêtes.
Synonym	Donne des alternatives à des noms d'objets.

CREATE TABLE

- Vous devez avoir :
 - Le privilège CREATE TABLE
 - Un espace de stockage

```
CREATE TABLE [schema.]table  
    (column datatype [DEFAULT expr] [, . . .]);
```

- Vous spécifiez :
 - Le nom de la Table
 - Column name, column data type, and column size



Creation d'une Tables

- Créer une table.

```
CREATE TABLE dept
  (deptno      NUMBER(2),
   dname       VARCHAR2(14),
   loc         VARCHAR2(13),
   create_date DATE DEFAULT SYSDATE);
```

Table created.

- Vérifier la création

```
DESCRIBE dept
```

Name	Null?	Type
DEPTNO		NUMBER(2)
DNAME		VARCHAR2(14)
LOC		VARCHAR2(13)
CREATE_DATE		DATE

Inclure des Contraintes

- Les contraintes forcent l'application des règles de gestion au niveau de la table.
- Types de contraintes:
 - NOT NULL
 - UNIQUE
 - PRIMARY KEY
 - FOREIGN KEY
 - CHECK



Definir les Contraintes

- **Syntaxe:**

```
CREATE TABLE [schema.]table  
    (column datatype [DEFAULT expr]  
     [column_constraint],  
      ...  
     [table_constraint][,...]);
```

- **Au niveau d'une colonne**

```
column [CONSTRAINT constraint_name] constraint_type,
```

- Au niveau de la table**

```
column,...  
[CONSTRAINT constraint_name] constraint_type  
(column, ...),
```

Definir les Contraintes

- Au niveau d'une colonne

```
CREATE TABLE employees (
    employee_id  NUMBER(6)
        CONSTRAINT emp_emp_id_pk PRIMARY KEY,
    first_name    VARCHAR2(20),
    ...);
```

1

- Au niveau de la table

```
CREATE TABLE employees (
    employee_id  NUMBER(6),
    first_name    VARCHAR2(20),
    ...
    job_id        VARCHAR2(10) NOT NULL,
    CONSTRAINT emp_emp_id_pk
        PRIMARY KEY (EMPLOYEE_ID));
```

2

La Contrainte FOREIGN KEY

Définie au niveau de la table

```
CREATE TABLE employees (
    employee_id      NUMBER(6),
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25),
    salary           NUMBER(8,2),
    commission_pct   NUMBER(2,2),
    hire_date        DATE NOT NULL,
    department_id    NUMBER(4),
    .
    .
    .
    CONSTRAINT emp_dept_fk FOREIGN KEY (department_id)
        REFERENCES departments(department_id),
    CONSTRAINT emp_email_uk UNIQUE(email));
```

La Contrainte FOREIGN KEY

Définie au niveau de la colonne

```
CREATE TABLE employees (
    employee_id      NUMBER(6) ,
    last_name        VARCHAR2(25) NOT NULL,
    email            VARCHAR2(25) ,
    salary           NUMBER(8,2) ,
    commission_pct   NUMBER(2,2) ,
    hire_date        DATE NOT NULL,
    department_id    NUMBER(4) CONSTRAINT emp_dept_fk
                      REFERENCES departments(department_id),
    .
    .
    .
    CONSTRAINT emp_email_uk UNIQUE(email));
```

CREATE TABLE: Exemple

```
CREATE TABLE employees
( employee_id      NUMBER(6)
  CONSTRAINT emp_employee_id PRIMARY KEY
, first_name        VARCHAR2(20)
, last_name         VARCHAR2(25)
  CONSTRAINT emp_last_name_nn NOT NULL
, email             VARCHAR2(25)
  CONSTRAINT emp_email_nn    NOT NULL
  CONSTRAINT emp_email_uk    UNIQUE
, phone_number      VARCHAR2(20)
, hire_date         DATE
  CONSTRAINT emp_hire_date_nn NOT NULL
, job_id            VARCHAR2(10)
  CONSTRAINT emp_job_nn     NOT NULL
, salary            NUMBER(8,2)
  CONSTRAINT emp_salary_ck   CHECK (salary>0)
, commission_pct    NUMBER(2,2)
, manager_id        NUMBER(6)
, department_id     NUMBER(4)
  CONSTRAINT emp_dept_fk    REFERENCES
                           departments (department_id));
```

Création d'une Table à l'aide d'une sous-requête

```
CREATE TABLE dept80
AS
SELECT employee_id, last_name,
       salary*12 ANNSAL,
       hire_date
  FROM employees
 WHERE department_id = 80;
```

Table created.

```
DESCRIBE dept80
```

Name	Null?	Type
EMPLOYEE_ID		NUMBER(6)
LAST_NAME	NOT NULL	VARCHAR2(25)
ANNSAL		NUMBER
HIRE_DATE	NOT NULL	DATE

Créer d'autres Objets de la BD

Database Objects

Object	Description
Table	L'unité basique de stockage. Elle est composée de lignes de données
View	Représente logiquement un sous-ensemble de données d'une ou de plusieurs tables.
Sequence	Génère une séquence de valeurs numériques
Index	Améliore la performance des requêtes.
Synonym	Donne des alternatives à des noms d'objets.

C'est quoi une vue

EMPLOYEES table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY
100	Steven	Kirg	SKING	515.123.4567	17-JUN-87	AD_FRES	2400
101	Neena	Kochhar	NKOCHHAR	515.123.4568	21-SEP-89	AD_VP	1700
102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-93	AD_VP	1700
103	Alexander	Humold	AHUNOLD	590.423.4567	03-JAN-90	IT_PROG	900
104	Bruce	Emart	BERNST	590.423.4668	21-MAY-91	IT_PROG	600
107	Diana	Lorentz	DLORENTZ	590.423.5667	07-FEB-99	IT_PROG	420
124	Kavar	Mourgos	INMOURGOS	650.123.5234	16-NOV-99	ST_MAN	580
141	Trenna	Rac	TRAIS	650.121.8009	17-OCT-95	ST_CLERK	350
142	Curtis	Davies	CDAVIES	650.121.2994	29-JAN-97	ST_CLERK	310
143	Randall	Matos	RMATOD	650.121.3074	16-MAR-90	ST_CLERK	200
EMPLOYEE_ID	LAST_NAME			SALARY	JUL-96	ST_CLERK	250
	149	Zlotkey		10500	JAN-00	SA_MAN	1050
	174	Abel		11000	MAY-96	SA_REP	1100
	176	Taylor		0600	MAR-98	SA_REP	860
170	Kimberly	Gore	KURANI	011.44.1044.423200	24-MAY-99	SA_REP	700
200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-87	AD_ASST	440
201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-96	MK_MAN	1300
202	Pat	Fay	PFAY	603.123.6666	17-AUG-97	MK_REP	600
205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-94	AC_MGR	1200
206	William	Gietz	WGIETZ	515.123.8181	07-JUN-94	AC_ACCOUNT	830

20 rows selected.

ORACLE®

Créer une Vue

Syntaxe

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW view
  [(alias[, alias]...)]
  AS subquery
  [WITH CHECK OPTION [CONSTRAINT constraint]]
  [WITH READ ONLY [CONSTRAINT constraint]];
```

- **Exemple:** Créez la vue EMPVU80, qui contient les détails des employés dans le département 80 :

```
CREATE VIEW empvu80
  AS SELECT employee_id, last_name, salary
    FROM employees
   WHERE department_id = 80;
```

View created.

- Décrire la structure de la vue

```
DESCRIBE empvu80
```

Consulter une vue

```
SELECT *  
FROM salvu50;
```

ID_NUMBER	NAME	ANN_SALARY
124	Mourgos	69600
141	Rajs	42000
142	Davies	37200
143	Matos	31200
144	Vargas	30000

Simple Views and Complex Views

Feature	Simple Views	Complex Views
Number of tables	One	One or more
Contain functions	No	Yes
Contain groups of data	No	Yes
DML operations through a view	Yes	Not always

Modifier une vue

- Avec l'utilisation de CREATE OR REPLACE VIEW.

```
CREATE OR REPLACE VIEW empvu80
  (id_number, name, sal, department_id)
AS SELECT employee_id, first_name || ' '
          || last_name, salary, department_id
     FROM employees
    WHERE department_id = 80;
```

View created.

Création d'une vue complexe

Créer une vue complexe qui contient des fonctions de groupe et un jointure:

```
CREATE OR REPLACE VIEW dept_sum_vu
  (name, minsal, maxsal, avgsal)
AS SELECT      d.department_name, MIN(e.salary) ,
                MAX(e.salary), AVG(e.salary)
  FROM        employees e JOIN departments d
  ON            (e.department_id = d.department_id)
 GROUP BY    d.department_name;
```

View created.

Règles pour l'exécution des Opérations DML sur une vue

- Vous pouvez généralement effectuer des opérations DML sur des vues simples. 
- Vous ne pouvez pas supprimer des lignes si la vue contient les éléments suivants :
 - Group functions
 - A GROUP BY clause
 - The DISTINCT keyword
 - The pseudocolumn ROWNUM keyword

Règles pour l'exécution des Opérations DML sur une vue

Vous ne pouvez pas modifier les données dans une vue si elle contient :

- **Group functions**
- **A GROUP BY clause**
- **The DISTINCT keyword**
- **The pseudocolumn ROWNUM keyword**
- **Columns defined by expressions**

Règles pour l'exécution des Opérations DML sur une vue

Vous ne pouvez pas ajouter de données via une vue si elle inclut :

- **Group functions**
- **A GROUP BY clause**
- **The DISTINCT keyword**
- **The pseudocolumn ROWNUM keyword**
- **Columns defined by expressions**
- **NOT NULL columns in the base tables that are not selected by the view**

Using the WITH CHECK OPTION Clause

- Vous pouvez assurer que les opérations DML effectuées sur une de vue restent dans le domaine de la vue en utilisant la clause WITH CHECK OPTION clause:

```
CREATE OR REPLACE VIEW empvu20
AS SELECT *
  FROM employees
 WHERE department_id = 20
  WITH CHECK OPTION CONSTRAINT empvu20_ck ;
```

View created.

- Toute tentative visant à modifier le numéro de département pour toute ligne de la vue échoue parce qu'il viole contrainte la WITH CHECK OPTION.

Suppression d'une vue

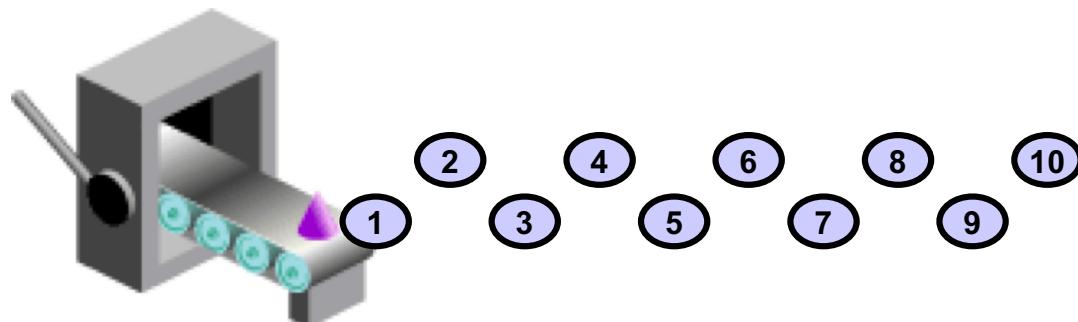
Vous pouvez supprimer une vue sans perte de données parce qu'une vue est basée sur des tables sous-jacentes dans la base de données.

```
DROP VIEW view;
```

```
DROP VIEW empvu80;  
View dropped.
```

Les Sequences

- Générer automatiquement des numéros uniques
- Est un objet partageable
- Peut être utilisé pour créer une valeur de clé primaire
- Représente des codes artificiels et non des codes métiers



CREATE SEQUENCE :

Syntaxe

```
CREATE SEQUENCE sequence
    [INCREMENT BY n]
    [START WITH n]
    [ {MAXVALUE n | NOMAXVALUE} ]
    [ {MINVALUE n | NOMINVALUE} ]
    [ {CYCLE | NOCYCLE} ]
    [ {CACHE n | NOCACHE} ] ;
```

Exemple

```
CREATE SEQUENCE dept_deptid_seq
    INCREMENT BY 10
    START WITH 120
    MAXVALUE 9999
    NOCACHE
    NOCYCLE;
```

Sequence created.

NEXTVAL et CURRVAL Pseudocolumns

- **NEXTVAL** retourne la valeur suivante de la séquence disponible. Elle retourne une valeur unique, chaque fois qu'il est référencé, même pour des utilisateurs différents.
- **CURRVAL** Obtient la valeur actuelle de la séquence.

Utiliser une Sequence

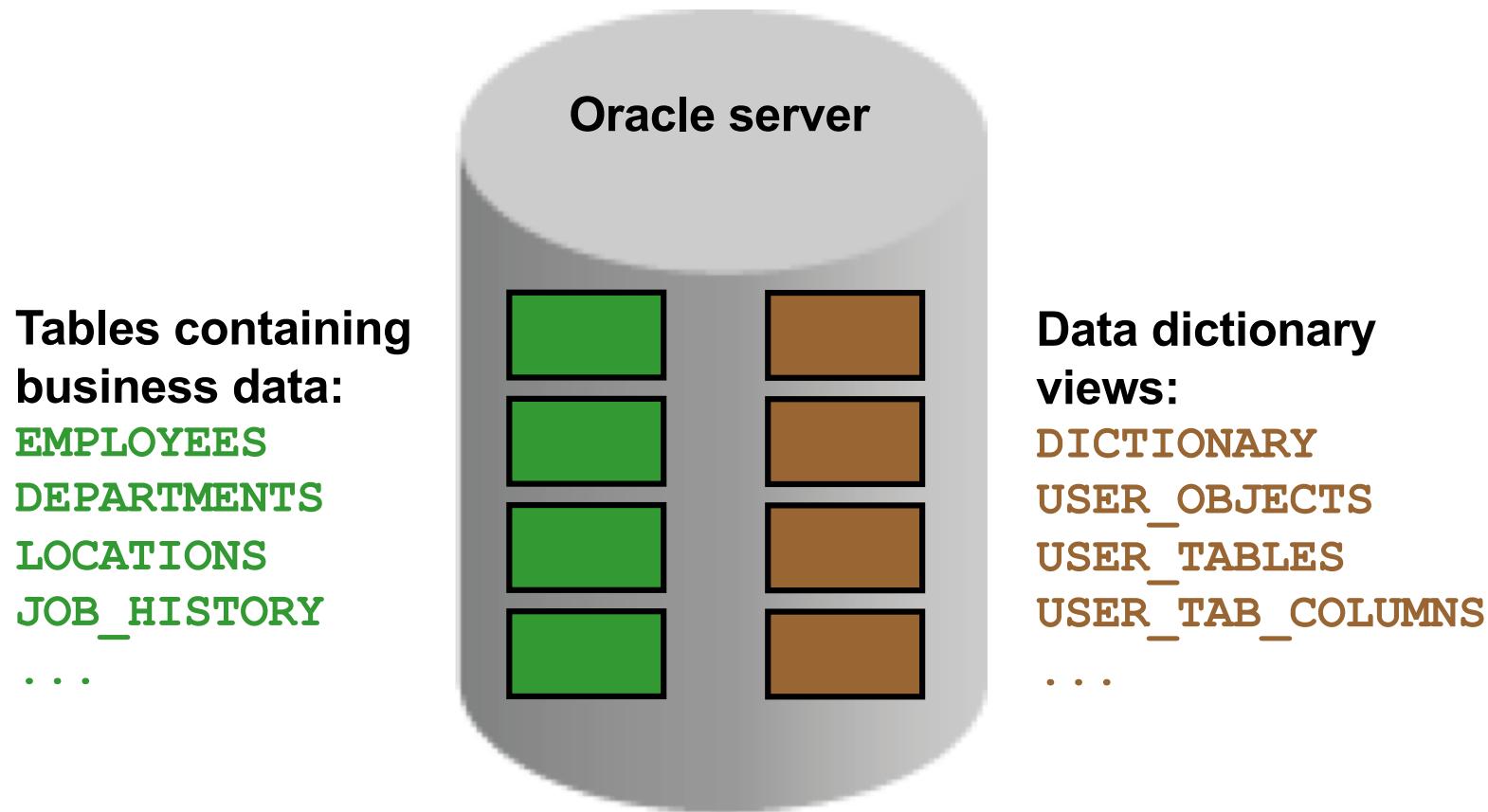
```
INSERT INTO departments(department_id,  
                      department_name, location_id)  
VALUES      (dept_deptid_seq.NEXTVAL,  
                      'Support', 2500);  
  
1 row created.
```

- Afficher la valeur actuelle de la séquence
DEPT_DEPTID_SEQ :

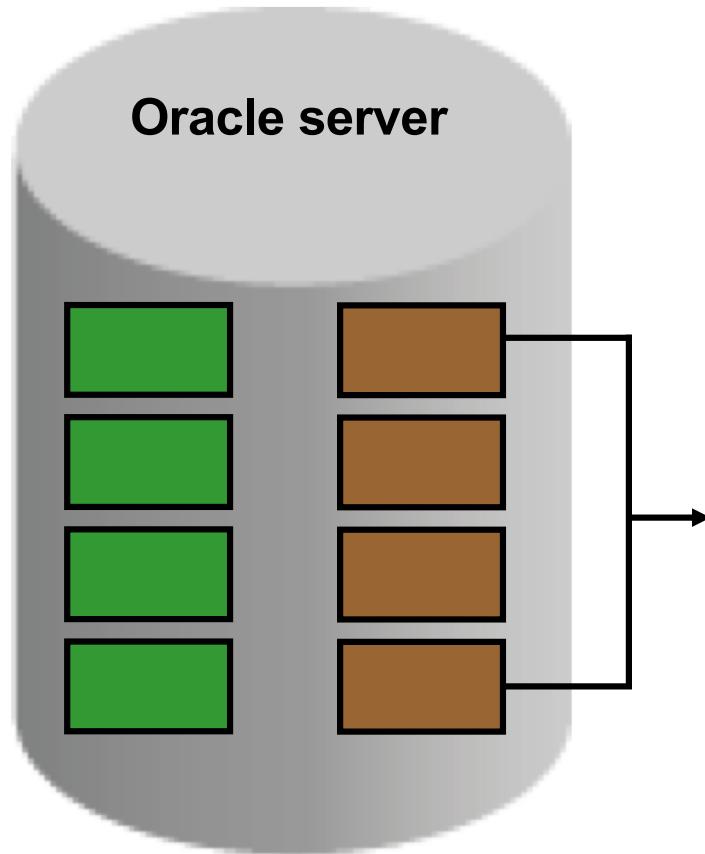
```
SELECT      dept_deptid_seq.CURRVAL  
FROM        dual;
```

Utilisation du dictionnaire de données

The Data Dictionary



Data Dictionary Structure



Consists of:

- **Base tables**
- **User-accessible views**

Data Dictionary Structure

View naming convention:

View Prefix	Purpose
USER	User's view (what is in your schema; what you own)
ALL	Expanded user's view (what you can access)
DBA	Database administrator's view (what is in everyone's schemas)
V\$	Performance-related data

Comment utiliser les vues du dictionnaire

Commencez par DICTIONARY. Il contient les noms et les descriptions des vues du dictionnaire.

DESCRIBE DICTIONARY

Name	Null?	Type
TABLE_NAME		VARCHAR2(30)
COMMENTS		VARCHAR2(4000)

```
SELECT *
FROM   dictionary
WHERE  table_name = 'USER_OBJECTS';
```

TABLE_NAME	COMMENTS
USER_OBJECTS	Objects owned by the user

USER_OBJECTS View

```
SELECT object_name, object_type, created, status  
FROM user_objects  
ORDER BY object_type;
```

OBJECT_NAME	OBJECT_TYPE	CREATED	STATUS
REG_ID_PK	INDEX	10-DEC-03	VALID
...			
DEPARTMENTS_SEQ	SEQUENCE	10-DEC-03	VALID
REGIONS	TABLE	10-DEC-03	VALID
LOCATIONS	TABLE	10-DEC-03	VALID
DEPARTMENTS	TABLE	10-DEC-03	VALID
JOB_HISTORY	TABLE	10-DEC-03	VALID
JOB_GRADES	TABLE	10-DEC-03	VALID
EMPLOYEES	TABLE	10-DEC-03	VALID
JOBS	TABLE	10-DEC-03	VALID
COUNTRIES	TABLE	10-DEC-03	VALID
EMP_DETAILS_VIEW	VIEW	10-DEC-03	VALID

Table Information

USER_TABLES:

```
DESCRIBE user_tables
```

Name	Null?	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
TABLESPACE_NAME		VARCHAR2(30)
CLUSTER_NAME		VARCHAR2(30)
IOT_NAME		VARCHAR2(30)

```
SELECT table_name  
FROM user_tables;
```

TABLE_NAME
JOB_GRADES
REGIONS
COUNTRIES
LOCATIONS
DEPARTMENTS
...

Column Information

USER_TAB_COLUMNS:

```
DESCRIBE user_tab_columns
```

Name	Null?	Type
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME	NOT NULL	VARCHAR2(30)
DATA_TYPE		VARCHAR2(106)
DATA_TYPE_MOD		VARCHAR2(3)
DATA_TYPE_OWNER		VARCHAR2(30)
DATA_LENGTH	NOT NULL	NUMBER
DATA_PRECISION		NUMBER
DATA_SCALE		NUMBER
NULLABLE		VARCHAR2(1)
COLUMN_ID		NUMBER
DEFAULT_LENGTH		NUMBER
DATA_DEFAULT		LONG

...

Column Information

```
SELECT column_name, data_type, data_length,  
       data_precision, data_scale, nullable  
  FROM user_tab_columns  
 WHERE table_name = 'EMPLOYEES';
```

COLUMN_NAME	DATA_TYPE	DATA_LENGTH	DATA_PRECISION	DATA_SCALE	NUL
EMPLOYEE_ID	NUMBER	22	6	0	N
FIRST_NAME	VARCHAR2	20			Y
LAST_NAME	VARCHAR2	25			N
EMAIL	VARCHAR2	25			N
PHONE_NUMBER	VARCHAR2	20			Y
HIRE_DATE	DATE	7			N
JOB_ID	VARCHAR2	10			N
SALARY	NUMBER	22	8	2	Y
COMMISSION_PCT	NUMBER	22	2	2	Y
MANAGER_ID	NUMBER	22	6	0	Y
DEPARTMENT_ID	NUMBER	22	4	0	Y

Constraint Information

- **USER_CONSTRAINTS** describes the constraint definitions on your tables.
- **USER_CONS_COLUMNS** describes columns that are owned by you and that are specified in constraints.

```
DESCRIBE user_constraints
```

Name	Null?	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
CONSTRAINT_TYPE		VARCHAR2(1)
TABLE_NAME	NOT NULL	VARCHAR2(30)
SEARCH_CONDITION		LONG
R_OWNER		VARCHAR2(30)
R_CONSTRAINT_NAME		VARCHAR2(30)
DELETE_RULE		VARCHAR2(9)
STATUS		VARCHAR2(8)
...		

Constraint Information

```
SELECT constraint_name, constraint_type,  
       search_condition, r_constraint_name,  
       delete_rule, status  
FROM   user_constraints  
WHERE  table_name = 'EMPLOYEES';
```

CONSTRAINT_NAME	CON	SEARCH_CONDITION	R_CONSTRAINT_NAME	DELETE_RULE	STATUS
EMP_LAST_NAME_NN	C	"LAST_NAME" IS NOT NULL			ENABLED
EMP_EMAIL_NN	C	"EMAIL" IS NOT NULL			ENABLED
EMP_HIRE_DATE_NN	C	"HIRE_DATE" IS NOT NULL			ENABLED
EMP_JOB_NN	C	"JOB_ID" IS NOT NULL			ENABLED
EMP_SALARY_MIN	C	salary > 0			ENABLED
EMP_EMAIL_UK	U				ENABLED
EMP_EMP_ID_PK	P				ENABLED
EMP_DEPT_FK	R		DEPT_ID_PK	NO ACTION	ENABLED
EMP_JOB_FK	R		JOB_ID_PK	NO ACTION	ENABLED
EMP_MANAGER_FK	R		EMP_EMP_ID_PK	NO ACTION	ENABLED

Constraint Information

```
DESCRIBE user_cons_columns
```

Name	Null?	Type
OWNER	NOT NULL	VARCHAR2(30)
CONSTRAINT_NAME	NOT NULL	VARCHAR2(30)
TABLE_NAME	NOT NULL	VARCHAR2(30)
COLUMN_NAME		VARCHAR2(4000)
POSITION		NUMBER

```
SELECT constraint_name, column_name  
FROM user_cons_columns  
WHERE table_name = 'EMPLOYEES' ;
```

CONSTRAINT_NAME	COLUMN_NAME
EMP_EMAIL_UK	EMAIL
EMP_SALARY_MIN	SALARY
EMP_JOB_NN	JOB_ID
EMP_HIRE_DATE_NN	HIRE_DATE

...

ORACLE®

View Information

1

DESCRIBE user_views

Name	Null?	Type
VIEW_NAME	NOT NULL	VARCHAR2(30)
TEXT_LENGTH		NUMBER
TEXT		LONG

2

SELECT DISTINCT view_name FROM user_views;

VIEW_NAME
EMP_DETAILS_VIEW

3

**SELECT text FROM user_views
WHERE view_name = 'EMP DETAILS VIEW' ;**

TEXT
SELECT e.employee_id, e.job_id, e.manager_id, e.department_id, d.location_id, l.country_id, e.first_name, e.last_name, e.salary, e.commission_pct, d.department_name, j.job_title, l.city, l.state_province, c.country_name, r.region_name FROM employees e, departments d, jobs j, locations l, countries c, regions r WHERE e.department_id = d.department_id AND d.location_id = l.location_id AND l.country_id = c.country_id AND c.region_id = r.region_id AND j.job_id = e.job_id WITH READ ONLY

Sequence Information

```
DESCRIBE user_sequences
```

Name	Null?	Type
SEQUENCE_NAME	NOT NULL	VARCHAR2(30)
MIN_VALUE		NUMBER
MAX_VALUE		NUMBER
INCREMENT_BY	NOT NULL	NUMBER
CYCLE_FLAG		VARCHAR2(1)
ORDER_FLAG		VARCHAR2(1)
CACHE_SIZE	NOT NULL	NUMBER
LAST_NUMBER	NOT NULL	NUMBER

Sequence Information

- Verify your sequence values in the USER_SEQUENCES data dictionary table.

```
SELECT    sequence_name, min_value, max_value,  
          increment_by, last_number  
FROM      user_sequences;
```

SEQUENCE_NAME	MIN_VALUE	MAX_VALUE	INCREMENT_BY	LAST_NUMBER
LOCATIONS_SEQ	1	9900	100	3300
DEPARTMENTS_SEQ	1	9990	10	280
EMPLOYEES_SEQ	1	1.0000E+27	1	207

- The LAST_NUMBER column displays the next available sequence number if NOCACHE is specified.