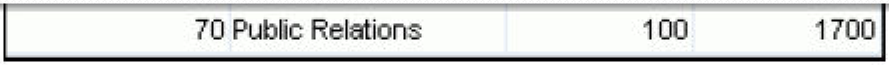
Database DML

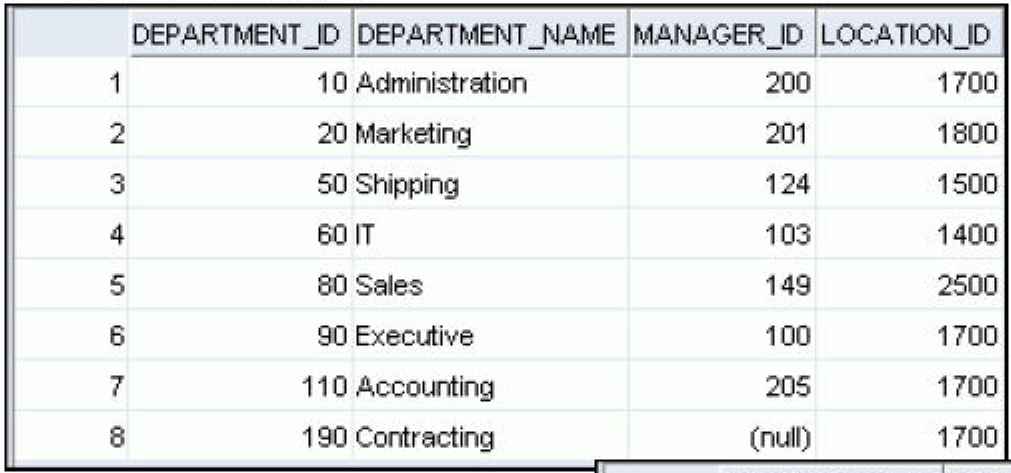
Insert Data



	70	Public Relations	100	1700

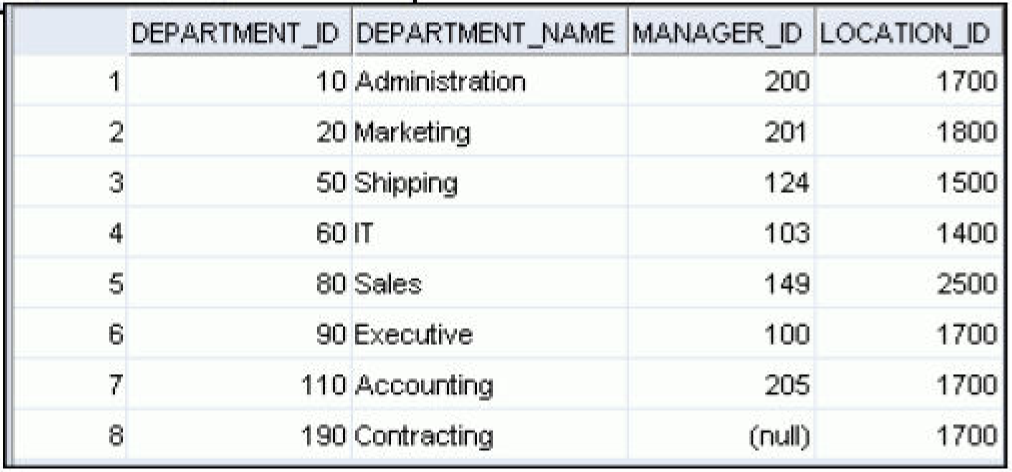
New row

DEPARTMENTS

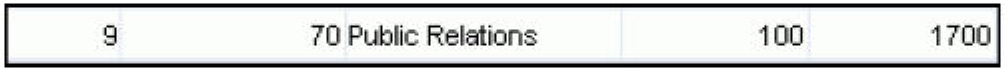


	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

Insert new row into the `DEPARTMENTS` **table.**



	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

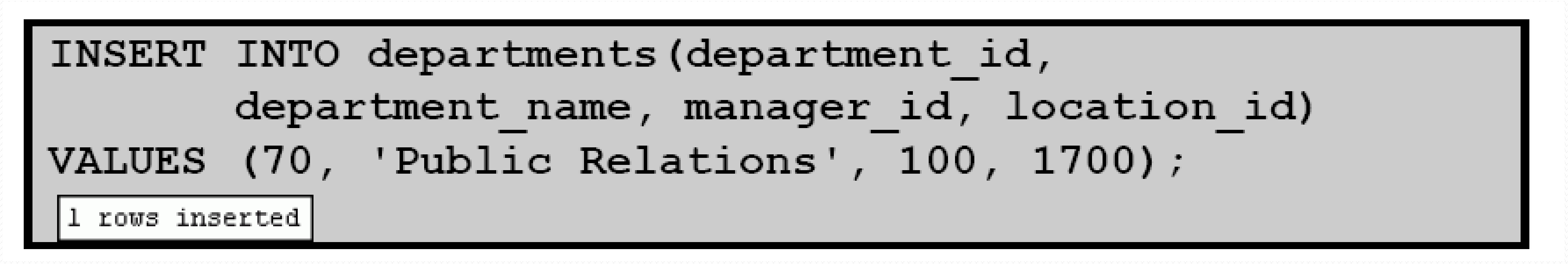


9	70	Public Relations	100	1700

Insert Data

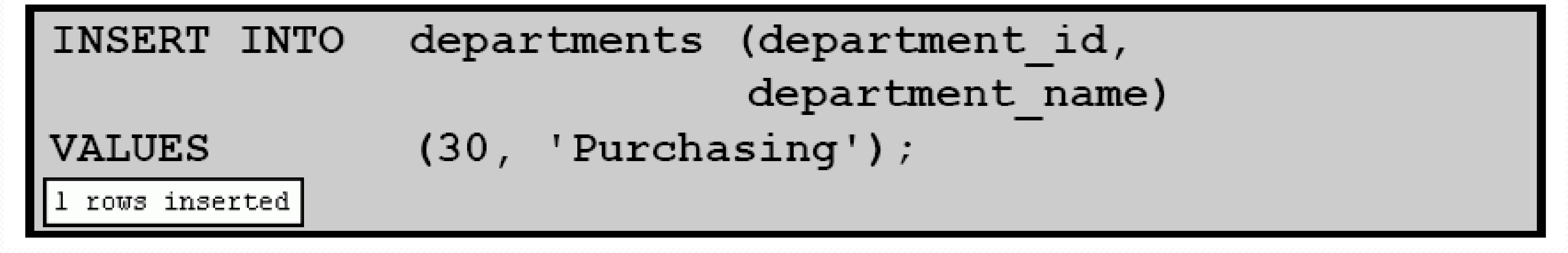
```
INSERT INTO   table [(column [, column...])]
VALUES        (value [, value...]);
```

Example



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

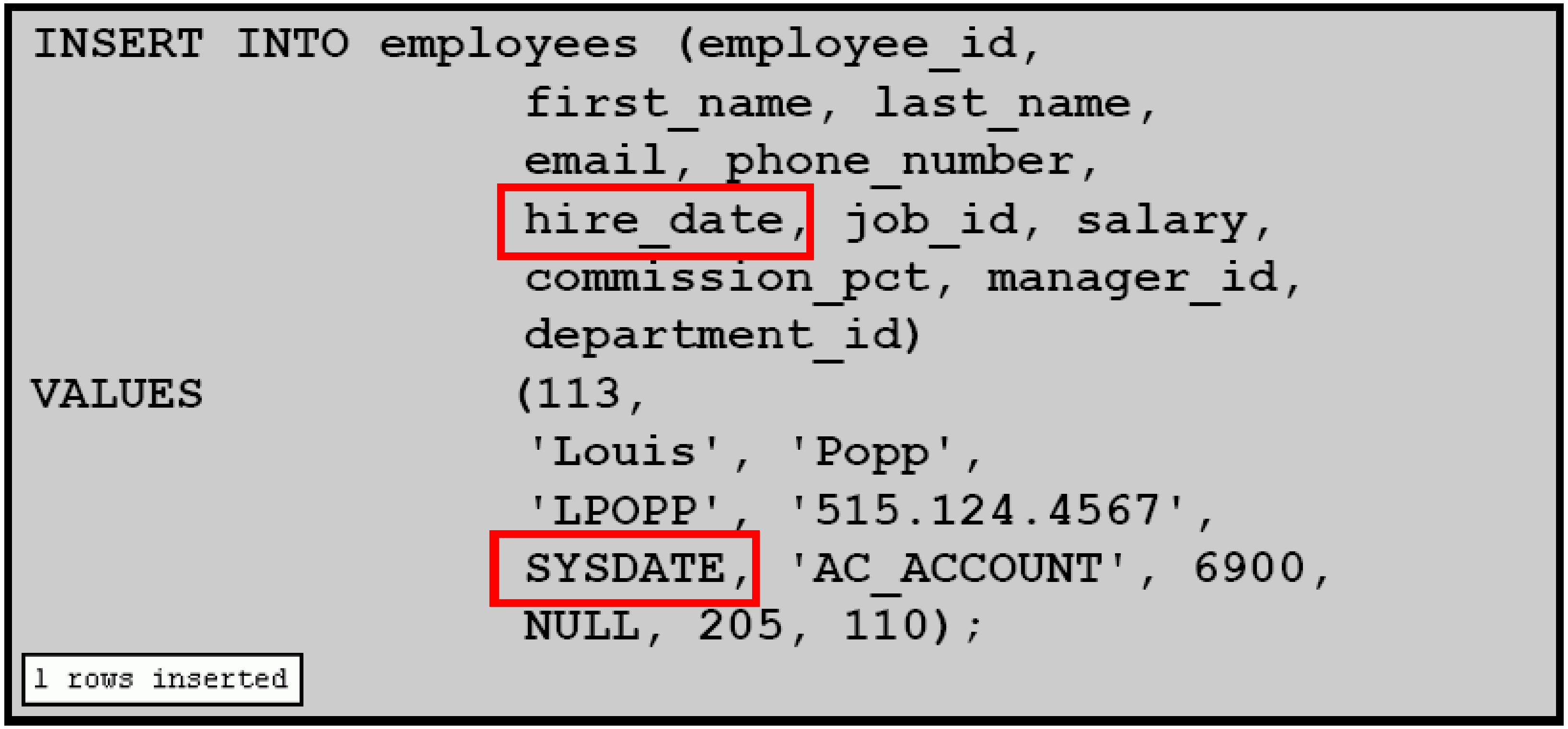
```
1 rows inserted
```



```
INSERT INTO   departments (department_id,
                           department_name)
VALUES         (30, 'Purchasing');
```

```
1 rows inserted
```

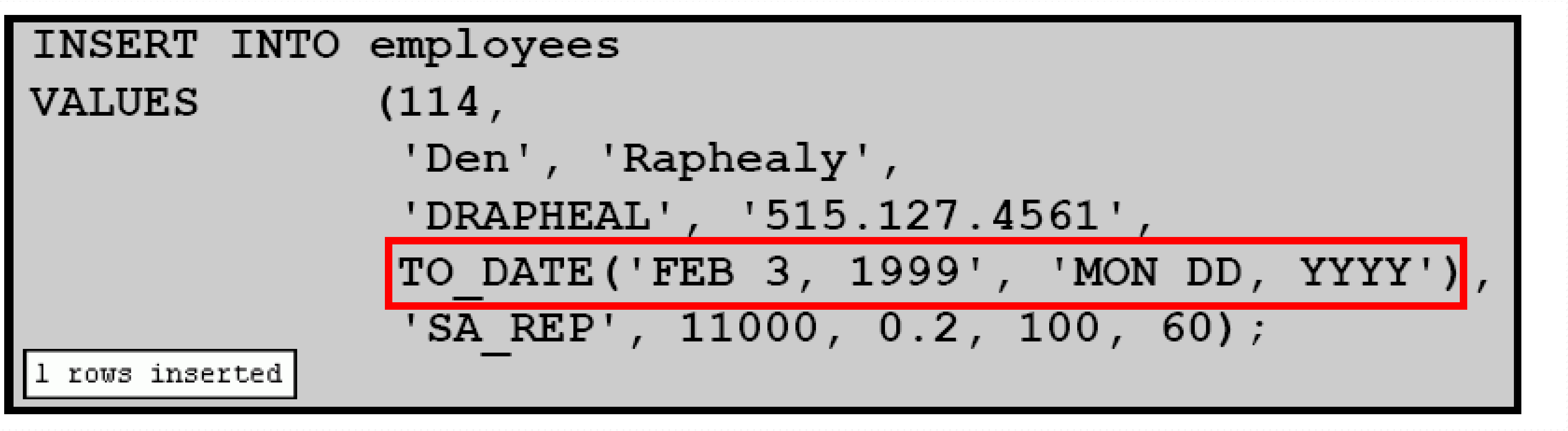
Example



```
INSERT INTO employees (employee_id,
                 first_name, last_name,
                 email, phone_number,
                 hire_date, job_id, salary,
                 commission_pct, manager_id,
                 department_id)
VALUES          (113,
                 'Louis', 'Popp',
                 'LPOPP', '515.124.4567',
                 SYSDATE, 'AC_ACCOUNT', 6900,
                 NULL, 205, 110);
```

```
1 rows inserted
```

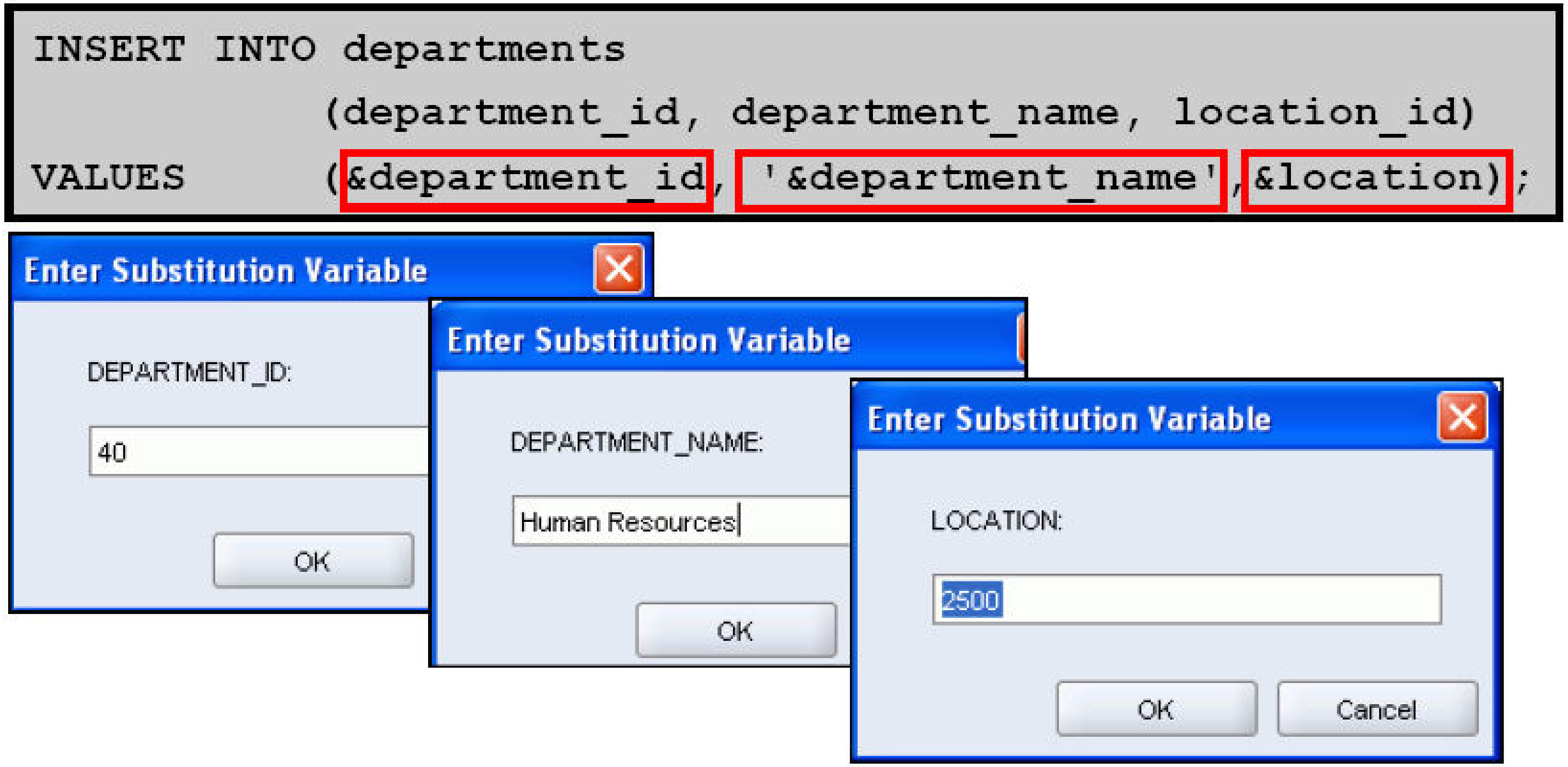
Example



```
INSERT INTO employees
VALUES      (114,
             'Den', 'Raphealy',
             'DRAPHEAL', '515.127.4561',
            TO_DATE('FEB 3, 1999', 'MON DD, YYYY'),
             'SA_REP', 11000, 0.2, 100, 60);
```

```
1 rows inserted
```

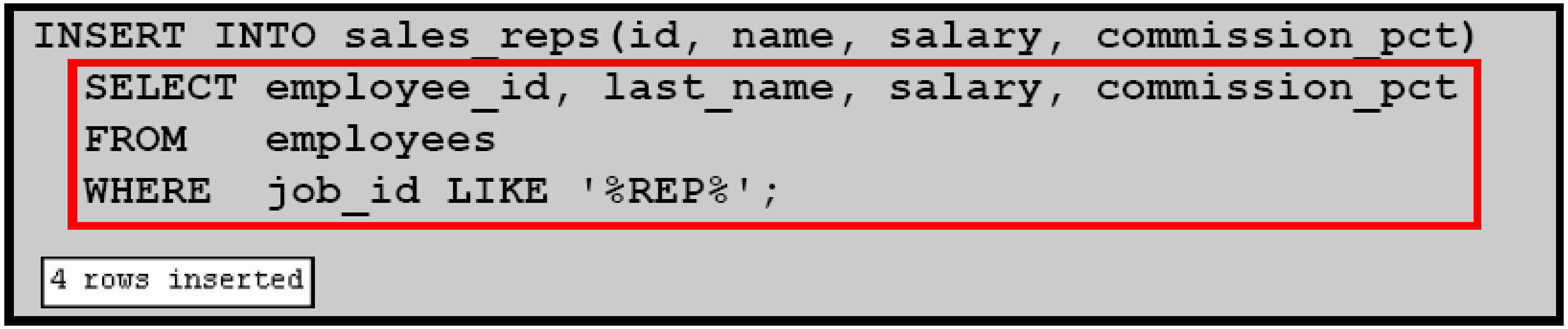
Creating script



```
INSERT INTO departments
            (department_id, department_name, location_id)
VALUES      (&department_id, '&department_name',&location);
```

Enter Substitution Variable
DEPARTMENT_ID:
40
OK
Enter Substitution Variable
DEPARTMENT_NAME:
Human Resources
OK
Enter Substitution Variable
LOCATION:
2500
OK
Cancel

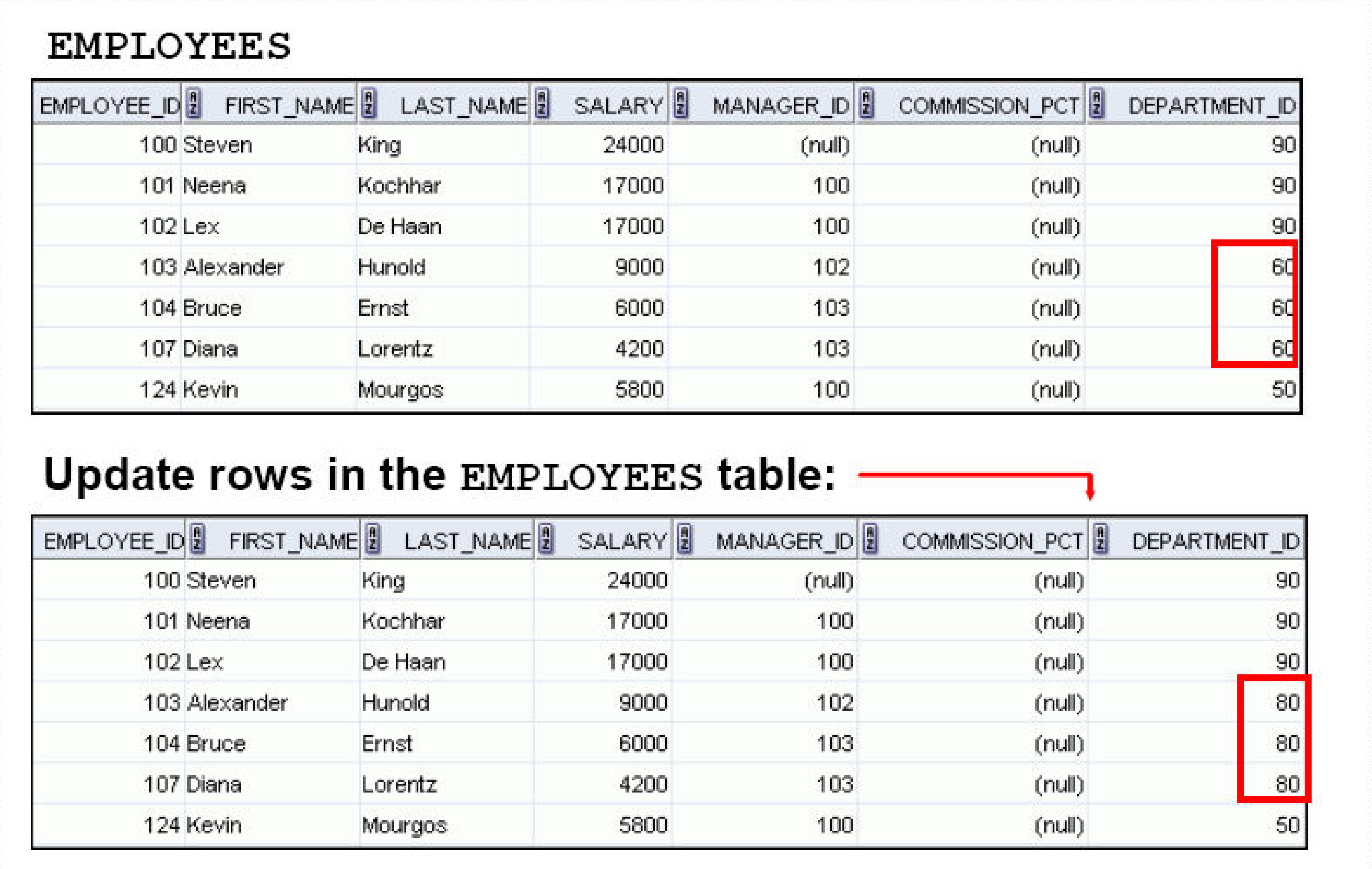
Copying rows from another table



```
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 inserted
```

Update



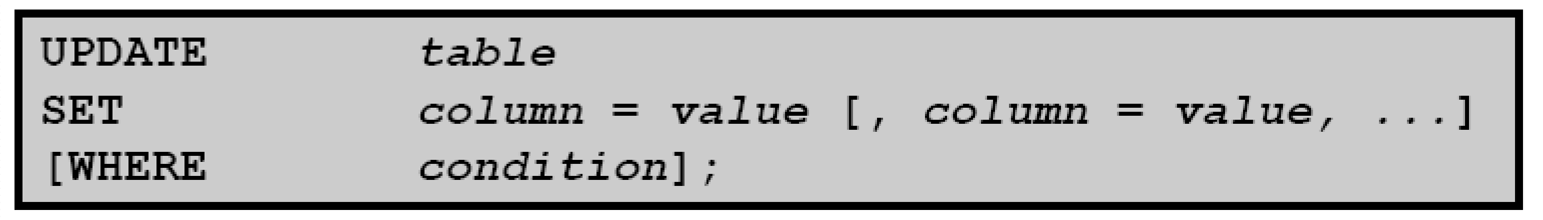
EMPLOYEES

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	60
104	Bruce	Ernst	6000	103	(null)	60
107	Diana	Lorentz	4200	103	(null)	60
124	Kevin	Mourgos	5800	100	(null)	50

Update rows in the `EMPLOYEES` table:

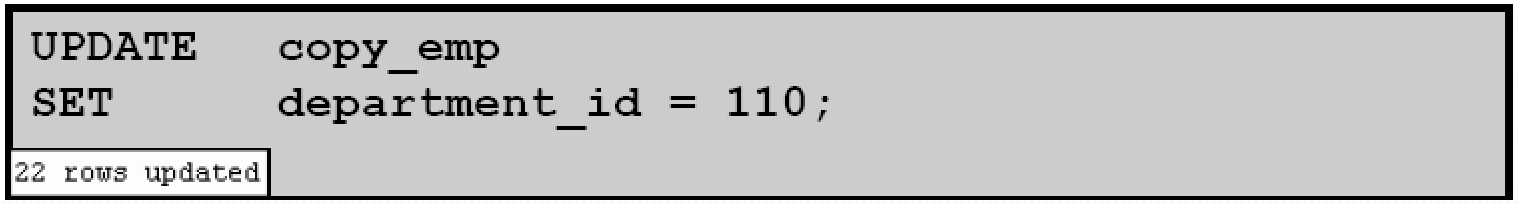
EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	MANAGER_ID	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000	(null)	(null)	90
101	Neena	Kochhar	17000	100	(null)	90
102	Lex	De Haan	17000	100	(null)	90
103	Alexander	Hunold	9000	102	(null)	80
104	Bruce	Ernst	6000	103	(null)	80
107	Diana	Lorentz	4200	103	(null)	80
124	Kevin	Mourgos	5800	100	(null)	50

Update

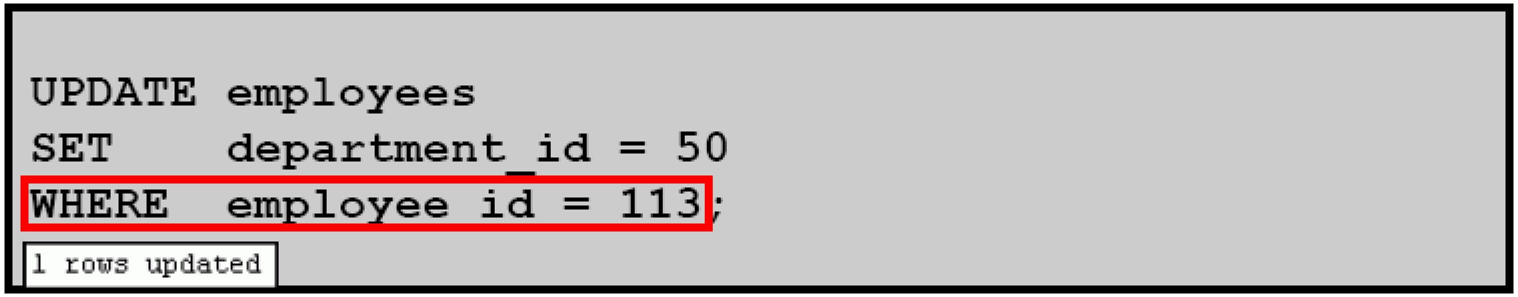


```
UPDATE        table
SET           column = value [, column = value, ...]
[WHERE        condition];
```

Example

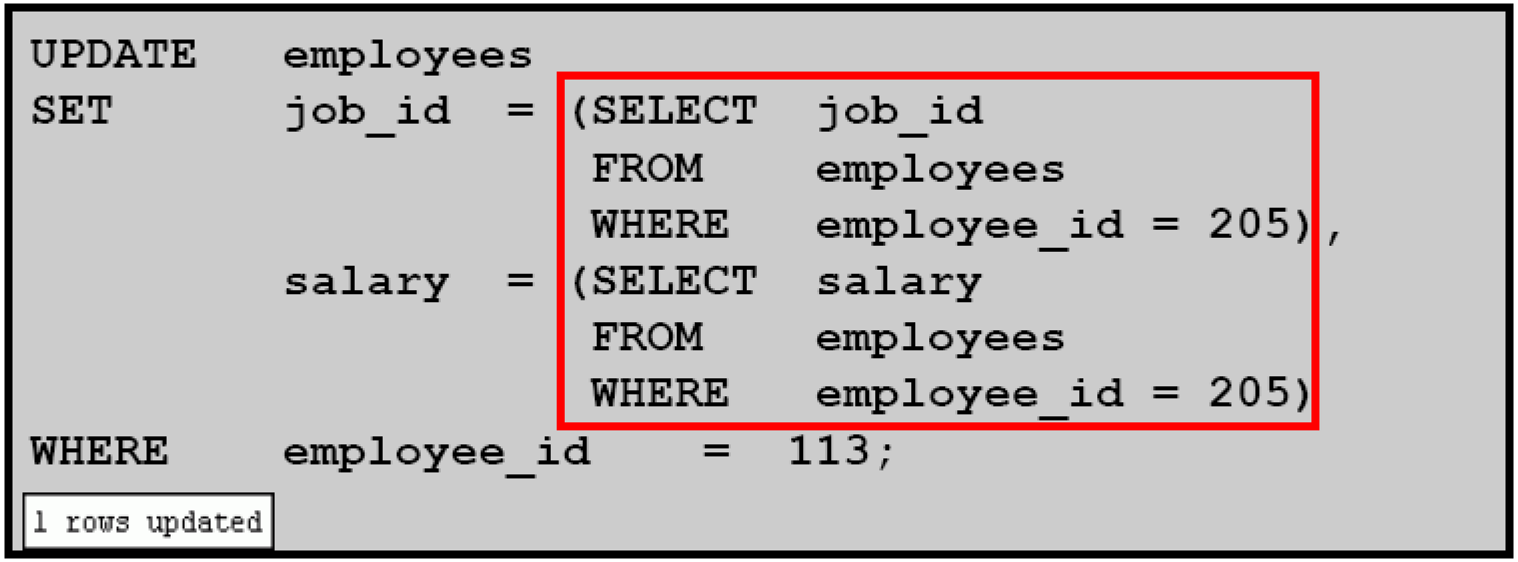


```
UPDATE    copy_emp
SET       department_id = 110;
22 rows updated
```



```
UPDATE employees
SET    department_id = 50
WHERE  employee_id = 113;
1 rows updated
```

Example

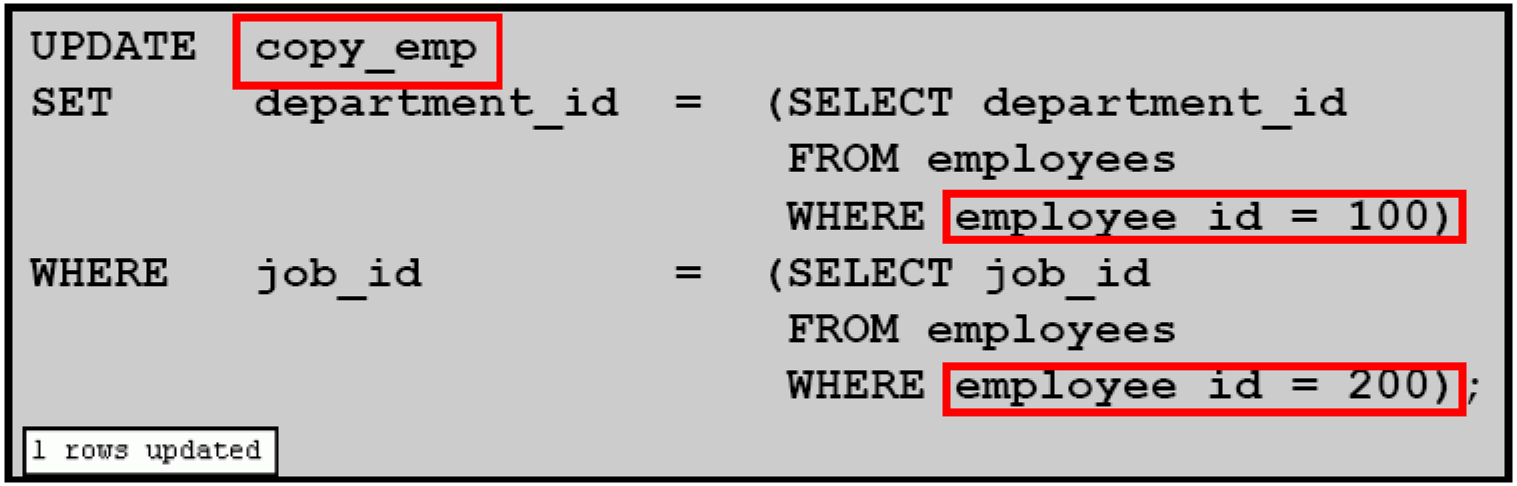


```
UPDATE   employees
SET      job_id   = (SELECT  job_id
                     FROM    employees
                     WHERE   employee_id = 205),
         salary   = (SELECT  salary
                     FROM    employees
                     WHERE   employee_id = 205)
WHERE    employee_id    =  113;
```

```
1 rows updated
```

```
UPDATE employees
SET (job_id, salary)  = (SELECT  job_id, salary
                        FROM    employees
                        WHERE   employee_id = 205)
WHERE    employee_id    =  113;
```

Update based on another table

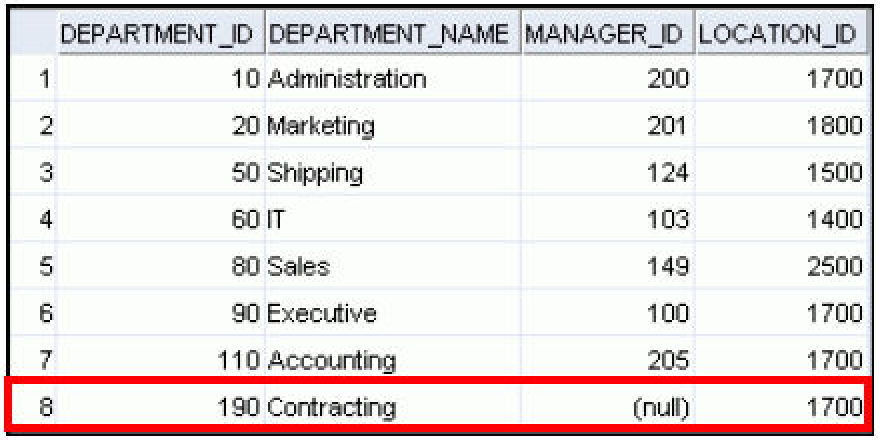


```
UPDATE   copy_emp
SET      department_id  =  (SELECT department_id
                            FROM employees
                            WHERE employee_id = 100)
WHERE    job_id         =  (SELECT job_id
                            FROM employees
                            WHERE employee_id = 200);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

```
1 rows updated
```

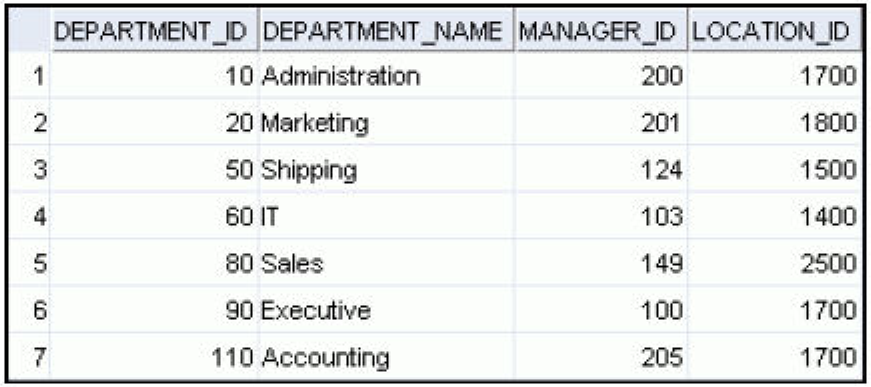
Delete

DEPARTMENTS



	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700
8	190	Contracting	(null)	1700

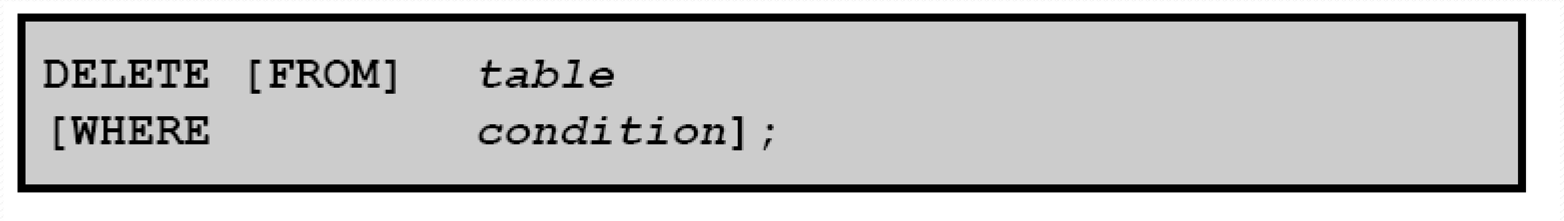
Delete a row from the `DEPARTMENTS` table:



	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	50	Shipping	124	1500
4	60	IT	103	1400
5	80	Sales	149	2500
6	90	Executive	100	1700
7	110	Accounting	205	1700

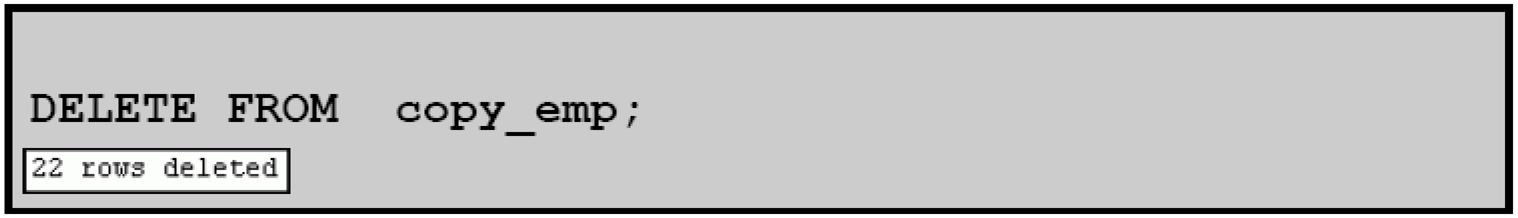
Delete table

- **Delete**



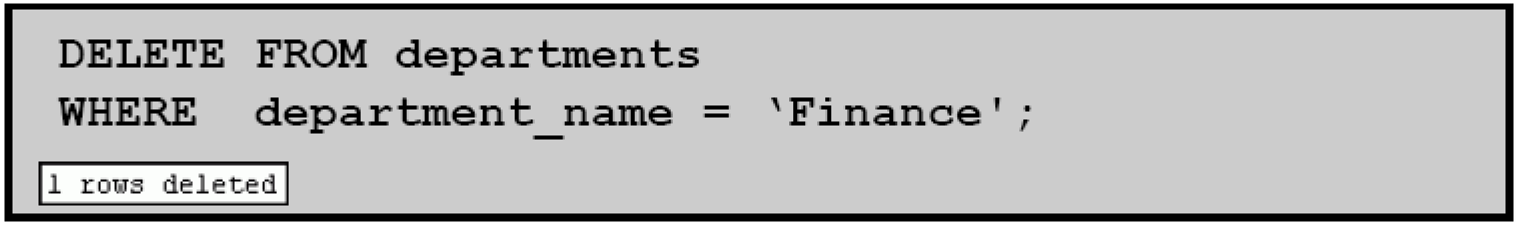
```
DELETE [FROM]   table
[WHERE          condition];
```

Example



```
DELETE FROM  copy_emp;
```

```
22 rows deleted
```



```
DELETE FROM departments
WHERE  department_name = 'Finance';
```

```
1 rows deleted
```

Truncate table

- **Truncate (DDL): Deletes all data in a table.**

```
truncate table tablename
```

Example:

```
truncate table publisher
```

Delete vs Truncate

Comparison Key	Delete	Truncate	Drop
Statement Type	DML	DDL	DDL
Basic	It is used to delete specific data of the table	It is used to delete entire data of the table	It is used to delete the whole table along with its data
Where clause	We can use a where clause	No where clause	No where clause
Locking	It locks the table row before deleting the row	It locks the entire table	No locking
Rollback	We can rollback the changes	We cannot rollback the changes	We cannot rollback the changes
Commit	You have to explicitly commit the changes	Implicitly committed	Implicitly committed
Performance	Slower than truncate	Faster than delete	-

Sequences

Sequences

- Automatically generates unique numbers
- Is typically used to create a primary key

Create Sequence Syntax

```
CREATE SEQUENCE sequence_name
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}];
```

Example of Creating a Sequence

```
CREATE SEQUENCE deptid_seq
INCREMENT BY 10
START WITH 5
MAXVALUE 9999;
```

NEXTVAL and CURRVAL

- **NEXTVAL** returns the next available sequence value. It returns a unique value every time it is referenced, even for different users
- **CURRVAL** obtains the current sequence value.

- **NEXTVAL** must be issued for that sequence before **CURRVAL** contains a value

Using a sequence in INSERT

```
INSERT INTO dept(deptno, dname, loc)
VALUES (deptid_seq.NEXTVAL,'Support',   ' HONG
KONG' );
```

View the Current Value

```
SELECT deptid_seq.CURRVAL
FROM dual;
```

Removing a Sequence

```
DROP SEQUENCE deptid_seq;
```

Views

Views

Views are relations, except that they are not physically stored.

For presenting different information to different users

Employee(ssn, name, department, project, salary)

```
CREATE VIEW  Developers AS
   SELECT name, project
   FROM  Employee
   WHERE department = “Development”
```

View

Person(name, city)
Purchase(buyer, seller, product, store)
Product(name, maker, category)

```
CREATE VIEW  Seattle-view  AS

        SELECT  buyer, seller, product, store
        FROM     Person, Purchase
        WHERE   Person.city = “Seattle”    AND
                   Person.name = Purchase.buyer
```

We have a new virtual table:
Seattle-view(buyer, seller, product, store)

Creating a View

- You embed a subquery within the CREATE VIEW statement.

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

- The subquery can contain complex SELECT syntax.
- The subquery cannot contain an ORDER BY clause.

Creating a View

Create a view, EMPVU10, that contains details of employees in department 10.

```
SQL> CREATE VIEW     empvu10
  2  AS SELECT       empno, ename, job
  3  FROM                  emp
  4  WHERE                 deptno = 10;
View created.
```

Describe View :

```
SQL> DESCRIBE empvu10
```

Modifying a View

- Modify the EMPVU10 view by using CREATE OR REPLACE VIEW clause. Add an alias for each column name.

```
SQL> CREATE OR REPLACE VIEW empvu10
  2           (employee_number, employee_name,
job_title)
  3  AS SELECT      empno, ename, job
  4  FROM                        emp
  5  WHERE                       deptno = 10;
View created.
```

- Column aliases in the CREATE VIEW clause are listed in the same order as the columns in the subquery.

Using the WITH CHECK OPTION Clause

You can ensure that DML on the view stays within the domain of the view by using the WITH CHECK OPTION clause.

```
SQL> CREATE OR REPLACE VIEW empvu20
  2  AS SELECT      *
  3  FROM             emp
  4  WHERE            deptno = 20
  5  WITH CHECK OPTION CONSTRAINT empvu20_ck;
View created.
```

Any attempt to change the department number for any row in the view will fail because it violates the WITH CHECK OPTION constraint.

Denying DML Operations

You can ensure that no DML operations occur by adding the WITH READ ONLY option to your view definition.

```
SQL> CREATE OR REPLACE VIEW empvu10
  2         (employee_number, employee_name,
job_title)
  3   AS SELECT      empno, ename, job
  4   FROM                        emp
  5   WHERE                       deptno = 10
  6   WITH READ ONLY;
View created.
```

Any attempt to perform a DML on any row in the view will result in Oracle Server error.

Removing a View

- Remove a view without losing data because a view is based on underlying tables in the database.

```
DROP VIEW view;
```

```
SQL> DROP VIEW empvu10;
View dropped.
```

Simple Views vs 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 through view	Yes	Not always