Statements

Statements – they help us to create the table and insert the data.

There are 3 types of statements,

- **DDL** – Data Definition Language – the various commands in DDL are :- Create, Drop, Truncate, Alter, Rename
- **DML** – Data Manipulation Language – the various commands in DML are :- Insert, Update, Delete
- **TCL** – Transaction Control Language – the various commands in TCL are :- Rollback, Commit, Savepoint

DDL

CREATE – It creates the table.

```
Create table <table name>
(
<Column name> data type constraint,
.

);
```

Before we study the **Create** command, let us first study the some of the basic **datatypes** we use in SQL.

1) CHAR :-

It stores the fixed length character data.

It can store the alphanumeric data (i.e, numbers and characters).

2) VARCHAR

It stores the variable length character data

It can store alphanumeric data.

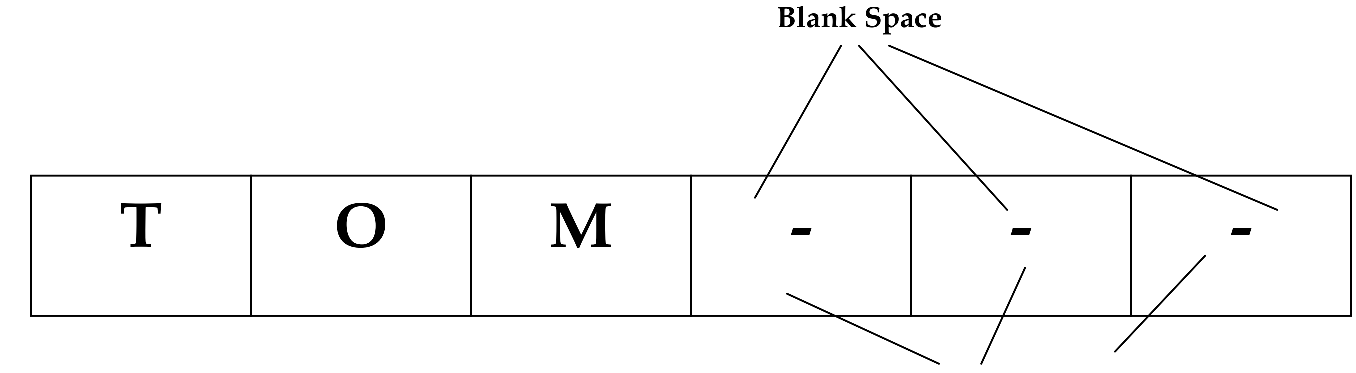
Difference between CHAR & VARCHAR

Let us consider an example as shown below to explain the difference.

Name char (6) ;

Here we are defining **name** which is of 6characters in length.

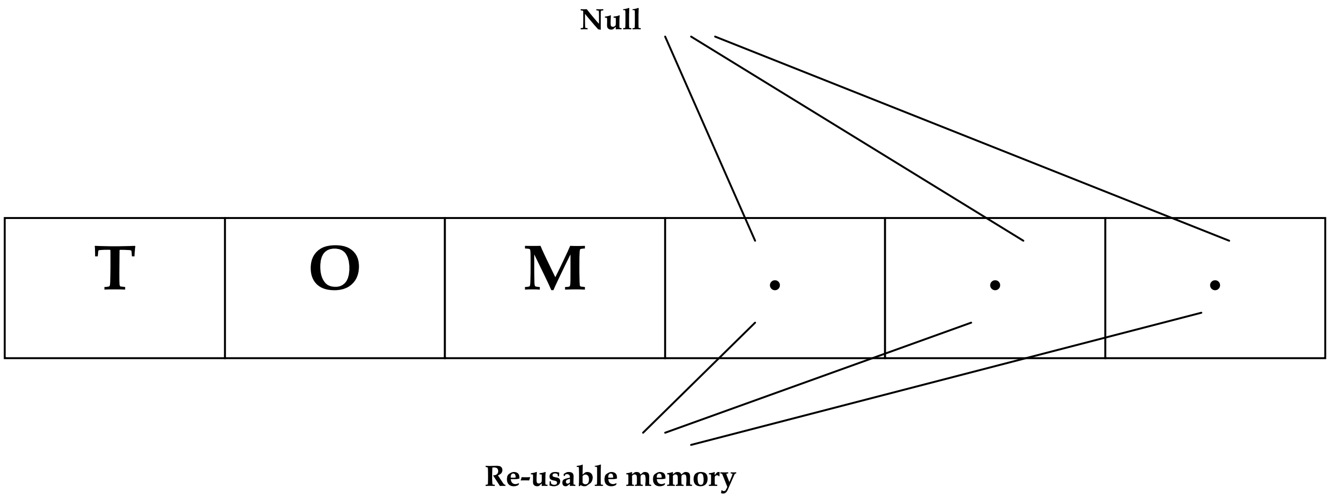
Now, let us store '***Tom***' in the name field. Let us understand how the memory is allocated for this,



Reserved / Non-reusable memory

When we declare anything of type **char**, the memory is allocated as of the size given and its fixed length – hence it cannot be altered.
Now, when we give ***tom***, it allocates 6 bytes for **name char** – only the 1st 3bytes are used to store **Tom** – the rest becomes waste as it is a blank space and it is reserved memory.
The **length(name) = 6**.

Name varchar (6) ;
Here we are defining **name** which is of 6characters in length.
Now, let us store '***Tom***' in the name field. Let us understand how the memory is allocated for this,



When we declare anything of type **varchar**, the memory is allocated as shown above and it is variable length
When we give ***tom***, it allocates 6bytes for **name varchar** – only the 1st 3bytes are used to store **tom** – the remaining 3 fields becomes **null**. As we know the property of **null** – null does not occupy any memory space **– thus the memory is not wasted here.**
The **length(name) = 3**.

Another difference is : -
In **char**, maximum value we can store is 2000 characters
In **varchar**, maximum value we can store is 4000 characters.

3) NUMBER
- it stores numeric data.

For ex – 1) sal number(4) ;
Here the maximum possible value is 9999.

2) sal number (6, 2) ;
Here, 2 – scale (total number of decimal places)
6 – precision (total number of digits including decimal places)
Maximum value is 9999.99

sal number (4, 3) ;
maximum value is 9.999
sal number (2, 2)
maximum value is .99

4) DATE

- it stores date and time
- no need to specify any length for this type.

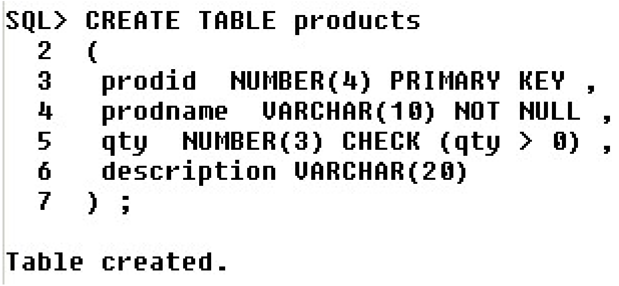
For ex, SQL > order_dt DATE ;

Date is always displayed in the default format :- **dd – month – yy**

. **Create the following tables**

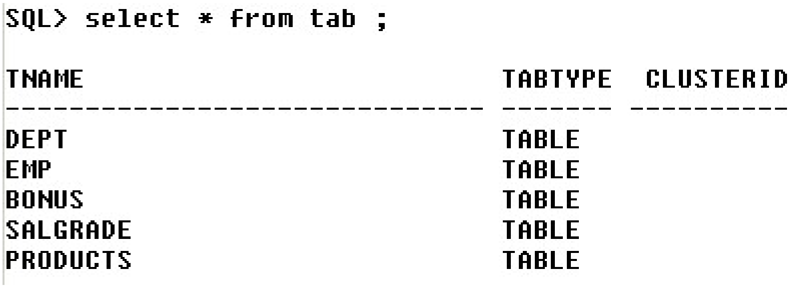
PRODUCTS
ProdID (PK)
ProdName (Not Null)
Qty (Chk > 0)
Description

ORDERS
ProdID (FK from products)
OrderID (PK)
Qty_sold (chk > 0)
Price
Order_Date



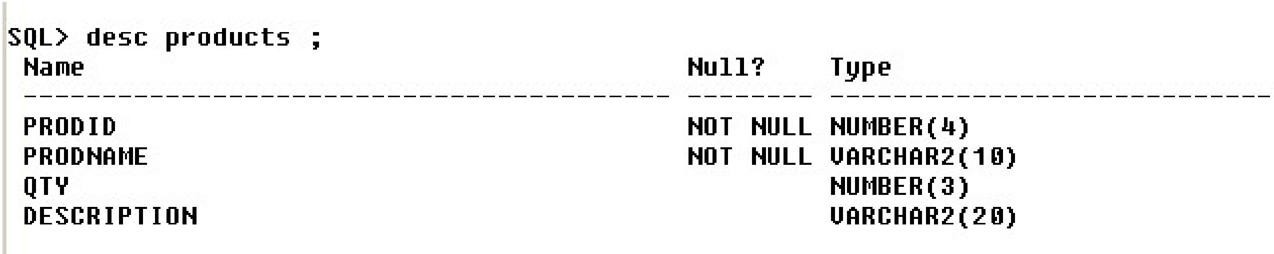
```
SQL> CREATE TABLE products
  2  (
  3   prodid  NUMBER(4) PRIMARY KEY ,
  4   prodname  VARCHAR(10) NOT NULL ,
  5   qty  NUMBER(3) CHECK (qty > 0) ,
  6   description VARCHAR(20)
  7  ) ;

Table created.
```

We can see that the table has been created.
Now, let us verify if the table has really been created and also the description of the table,



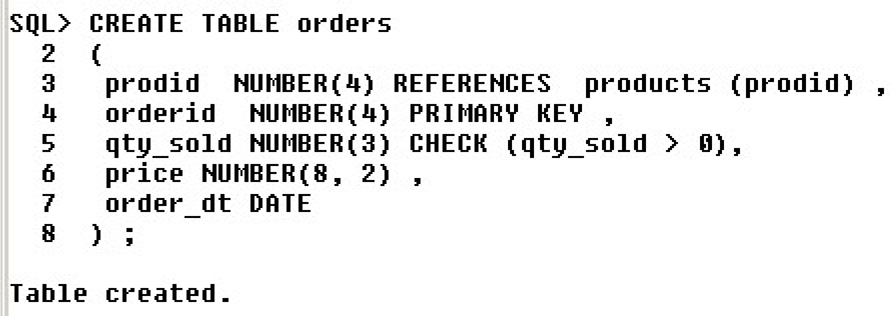
```
SQL> select * from tab ;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
DEPT                           TABLE
EMP                            TABLE
BONUS                          TABLE
SALGRADE                       TABLE
PRODUCTS                       TABLE
```

The new table **products** has been added to the database.



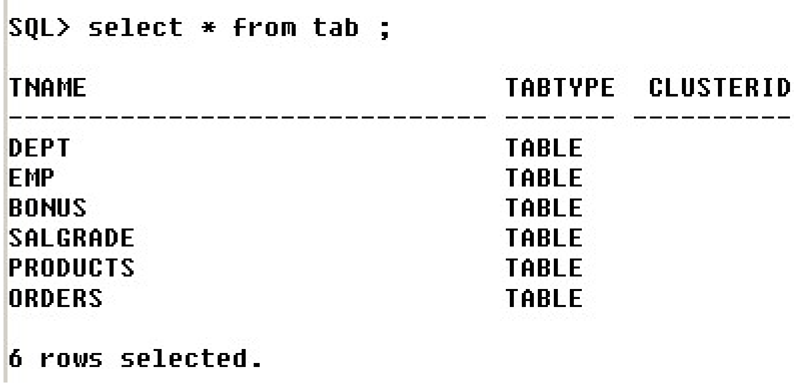
```
SQL> desc products ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODID                                    NOT NULL NUMBER(4)
 PRODNAME                                  NOT NULL VARCHAR2(10)
 QTY                                                NUMBER(3)
 DESCRIPTION                                        VARCHAR2(20)
```

Thus, we get the description of the table **products**.



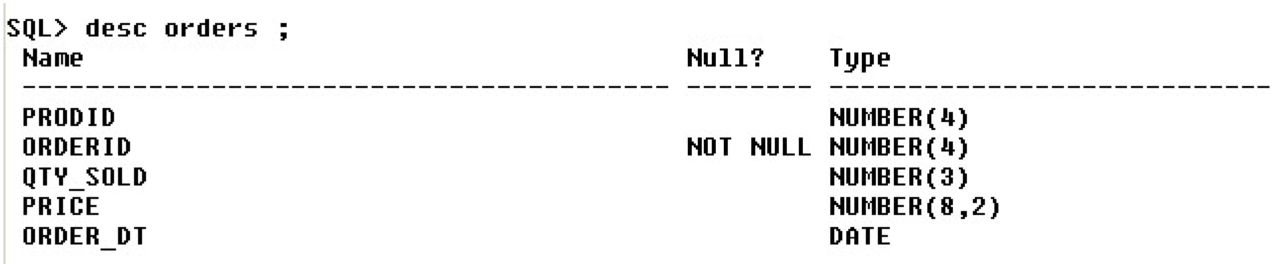
```
SQL> CREATE TABLE orders
  2  (
  3   prodid  NUMBER(4) REFERENCES  products (prodid) ,
  4   orderid  NUMBER(4) PRIMARY KEY ,
  5   qty_sold NUMBER(3) CHECK (qty_sold > 0),
  6   price NUMBER(8, 2) ,
  7   order_dt DATE
  8  ) ;

Table created.
```

The new table **orders** has been created. We can see from the above query how to reference a child table to the parent table using the **references** keyword.



```
SQL> select * from tab ;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
DEPT                           TABLE
EMP                            TABLE
BONUS                          TABLE
SALGRADE                       TABLE
PRODUCTS                       TABLE
ORDERS                         TABLE

6 rows selected.
```

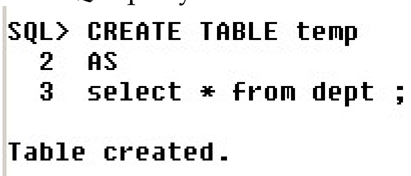
Thus we can verify that **orders** table has been created and added to the database.



```
SQL> desc orders ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODID                                             NUMBER(4)
 ORDERID                                   NOT NULL NUMBER(4)
 QTY_SOLD                                           NUMBER(3)
 PRICE                                              NUMBER(8,2)
 ORDER_DT                                           DATE
```

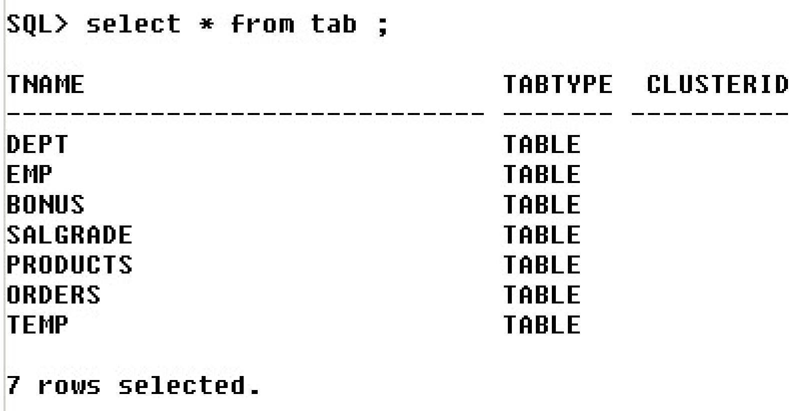
Thus, we get the description of the **orders** table.

<u>**Creating a table from another table**</u> :-
Now, we will see how to create a table from another table – i.e, it duplicates all the records and the characteristics of another table.
The SQL query for it is as follows,



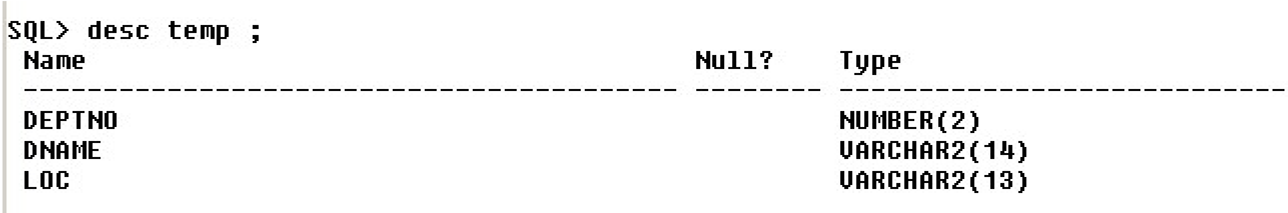
```
SQL> CREATE TABLE temp
  2  AS
  3  select * from dept ;

Table created.
```

Thus we can see that we have created another table **temp** from the table **dept**.
We can verify it as shown below,



```
SQL> select * from tab ;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
DEPT                           TABLE
EMP                            TABLE
BONUS                          TABLE
SALGRADE                       TABLE
PRODUCTS                       TABLE
ORDERS                         TABLE
TEMP                           TABLE

7 rows selected.
```

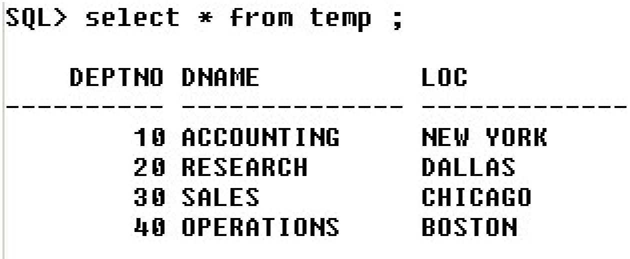
Thus, we can see that the **table temp** has been created.



```
SQL> desc temp ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                             NUMBER(2)
 DNAME                                              VARCHAR2(14)
 LOC                                                VARCHAR2(13)
```

Thus, we can see that the table **temp** has copied the structure of the table **dept**. Here, we must observe that **temp** copies all the columns, rows and NOT NULL constraints only from the table **dept**. It never copies PK, FK, Check constraints.

Thus, when in the interview somebody asks you "I have a table which has about 1million records. How do I duplicate it into another table without using Insert keyword and without inserting it individually all the records into the duplicated table ?
Answer is - Use the above query of creating a table from another table and explain it.



```
SQL> select * from temp ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Thus, from the above query – we can see that all the records of the table **dept** has been copied into the table **temp**.

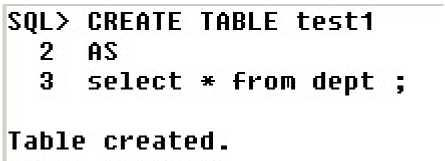
TRUNCATE

It removes all the data permanently, but the structure of the table remains as it is.
Ex – SQL > TRUNCATE TABLE temp ;

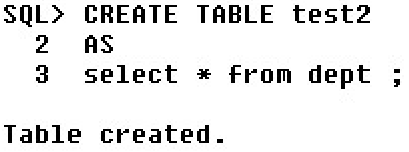
DROP

It removes both data and the structure of the table permanently from the database.
Ex – SQL > DROP TABLE test ;

Let us understand the difference between **drop & truncate** using the below shown example,

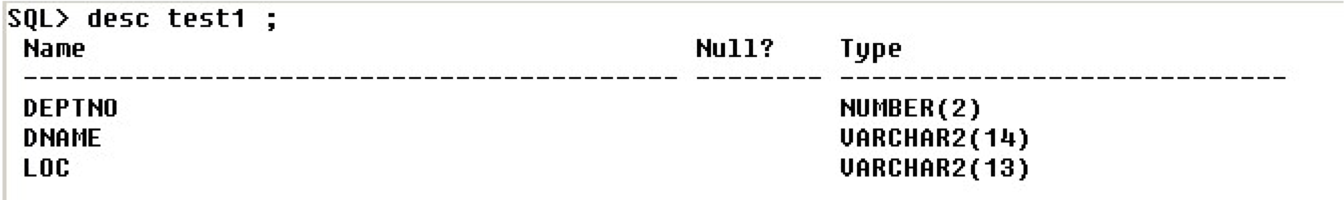


```
SQL> CREATE TABLE test1
  2  AS
  3  select * from dept ;

Table created.
```

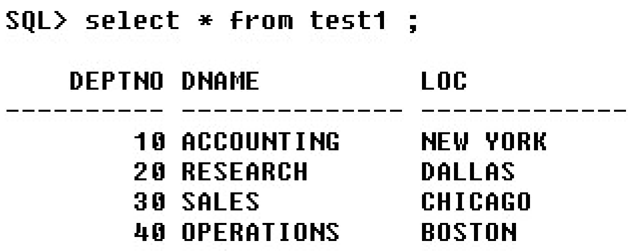


```
SQL> CREATE TABLE test2
  2  AS
  3  select * from dept ;

Table created.
```

Let us create 2 tables Test1 and Test2 as shown above.

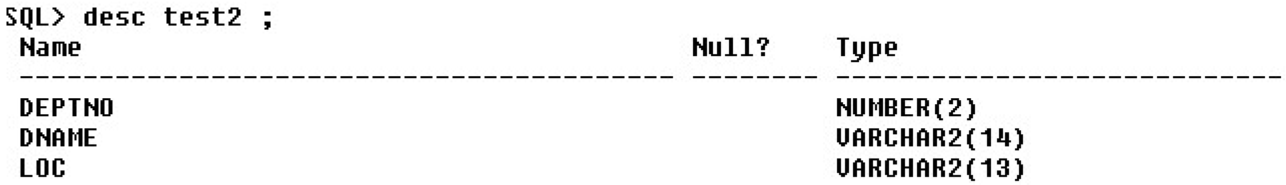


```
SQL> desc test1 ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                             NUMBER(2)
 DNAME                                              VARCHAR2(14)
 LOC                                                VARCHAR2(13)
```

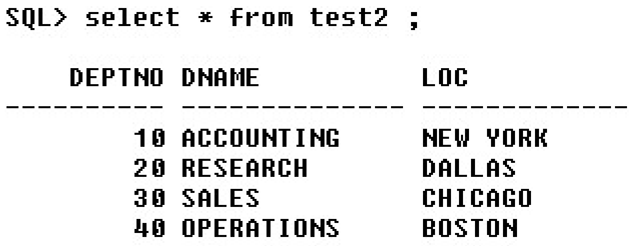


```
SQL> select * from test1 ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

The above shows the description of the table test1.



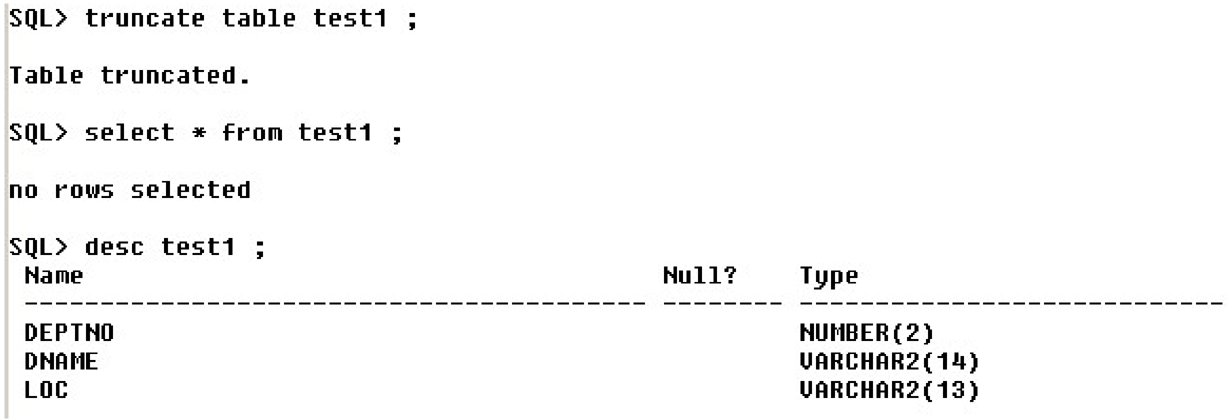
```
SQL> desc test2 ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                             NUMBER(2)
 DNAME                                              VARCHAR2(14)
 LOC                                                VARCHAR2(13)
```



```
SQL> select * from test2 ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

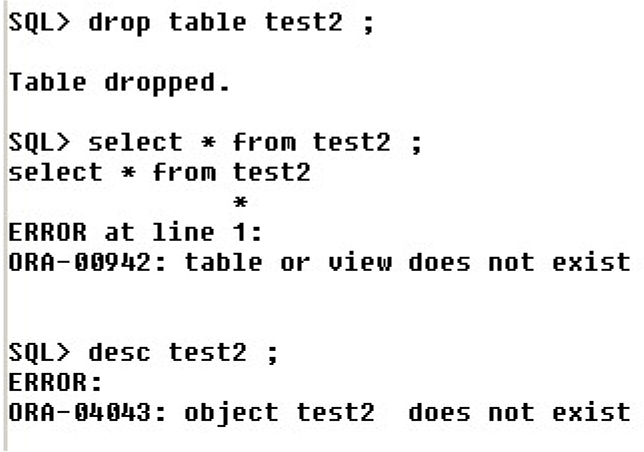
The above gives the description of the table Test2.

Now, let us use the **Truncate query on Test1** and **Drop query on Test2** and see the difference.



```
SQL> truncate table test1 ;

Table truncated.

SQL> select * from test1 ;

no rows selected

SQL> desc test1 ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPTNO                                             NUMBER(2)
 DNAME                                              VARCHAR2(14)
 LOC                                                VARCHAR2(13)
```

The above 3 queries show that – 1st query has the table test1 truncated.
2nd query – it shows **no rows selected** – thus only the records from the table has been removed.
3rd query – it shows that the structure of the table is still present. Only the records will be removed.
Thus, this **explains the truncate query.**

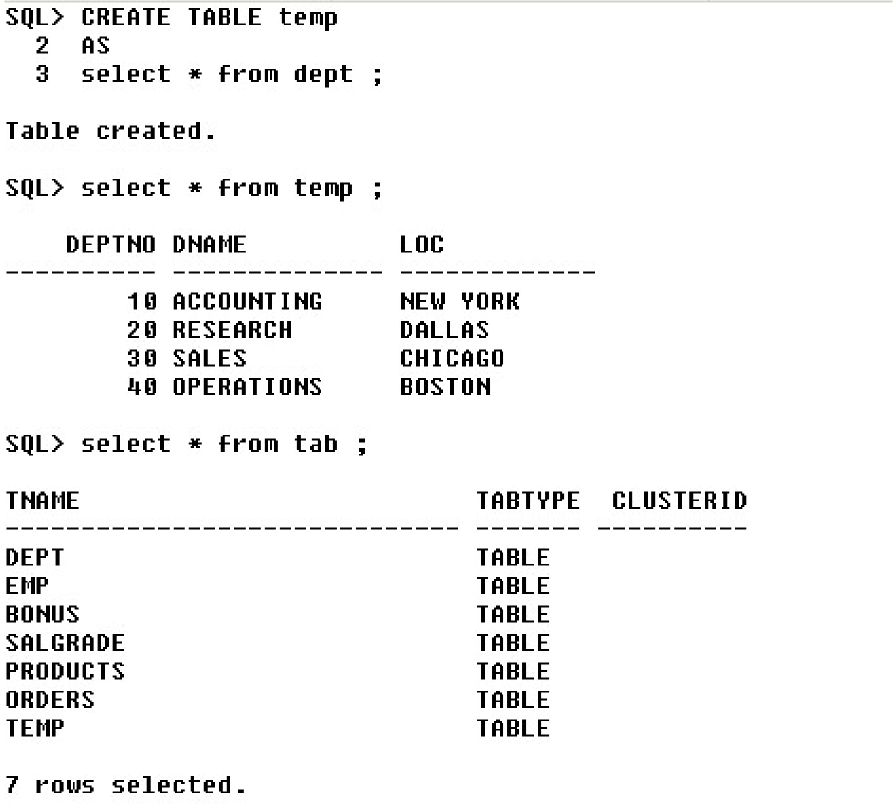


```
SQL> drop table test2 ;

Table dropped.

SQL> select * from test2 ;
select * from test2
              *
ERROR at line 1:
ORA-00942: table or view does not exist


SQL> desc test2 ;
ERROR:
ORA-04043: object test2  does not exist
```

Thus from the above queries we can explain how **drop** works. 1st query – it drops the table. Thus – the entire structure and records of the table are dropped.
2nd and 3rd query – since, there is no table – **select & desc** query for **test2** will throw an error.
Thus, this **explains the drop query.**
Hence, we have seen the difference between **drop & truncate** query.

RENAME

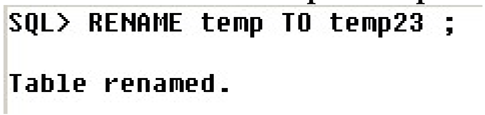
It renames a table.
For ex, let us see the query of how we do this renaming a table.



```
SQL> CREATE TABLE temp
  2  AS
  3  select * from dept ;

Table created.

SQL> select * from temp ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQL> select * from tab ;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
DEPT                           TABLE
EMP                            TABLE
BONUS                          TABLE
SALGRADE                       TABLE
PRODUCTS                       TABLE
ORDERS                         TABLE
TEMP                           TABLE

7 rows selected.
```

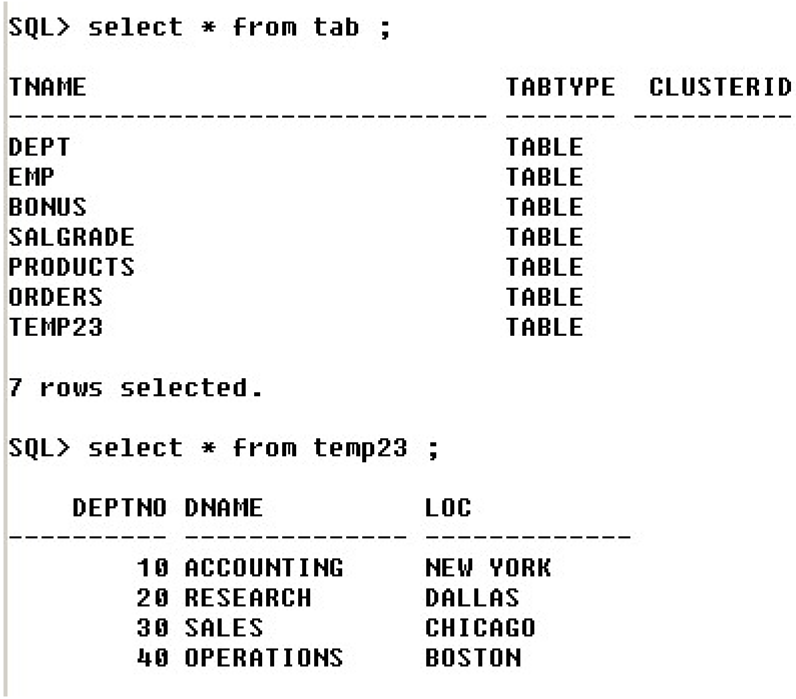
In the above 3queries – we have created a table **temp** which copies table **dept** – we see the records of the table temp – and also check if the table has really been created.

Now let us **rename temp to temp23** as shown below,



```
SQL> RENAME temp TO temp23 ;

Table renamed.
```

The above query is used to rename a table.
Now let us verify the contents of the table and check if it has really been modified,



```
SQL> select * from tab ;

TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
DEPT                           TABLE
EMP                            TABLE
BONUS                          TABLE
SALGRADE                       TABLE
PRODUCTS                       TABLE
ORDERS                         TABLE
TEMP23                         TABLE

7 rows selected.

SQL> select * from temp23 ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

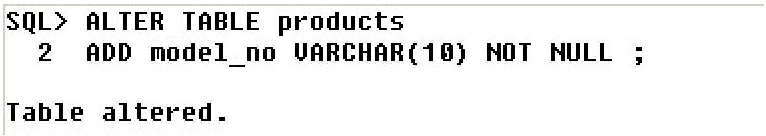
Thus the table has been renamed and its contents are verified.

ALTER

- this query alters / changes the structure of the table (i.e, - adding columns, removing columns, renaming columns etc).

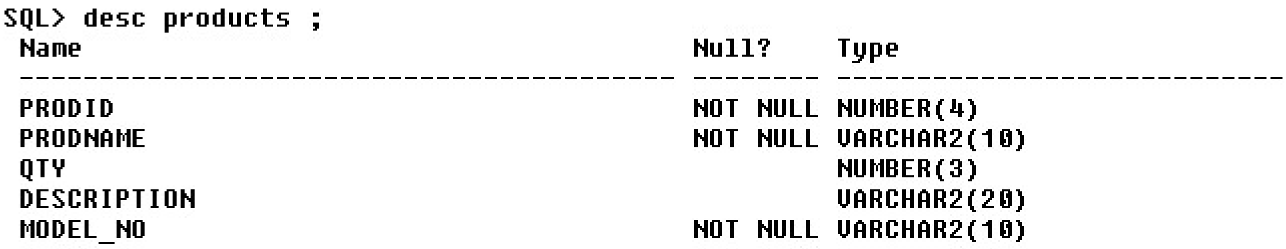
Now let us **alter** the table **products** (which we have created earlier).

1) Let us add a new column *'model_no'* to the table.



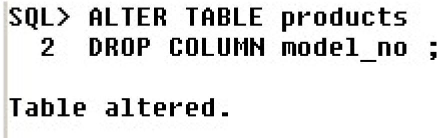
```
SQL> ALTER TABLE products
  2  ADD model_no VARCHAR(10) NOT NULL ;

Table altered.
```

Thus, a new column has been added. Let's verify it with the query shown below,



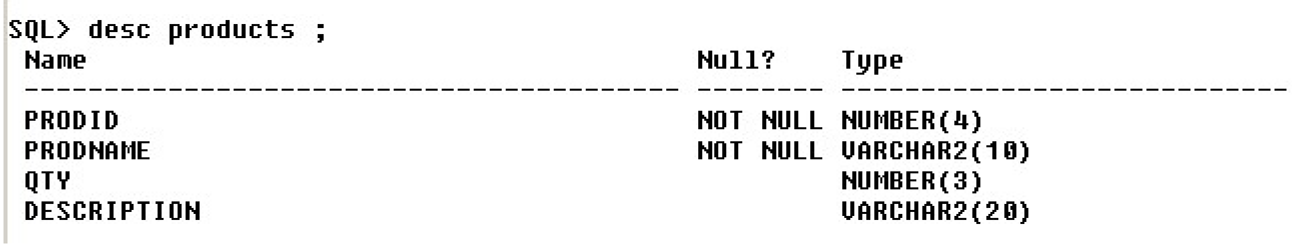
```
SQL> desc products ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODID                                    NOT NULL NUMBER(4)
 PRODNAME                                  NOT NULL VARCHAR2(10)
 QTY                                                NUMBER(3)
 DESCRIPTION                                        VARCHAR2(20)
 MODEL_NO                                  NOT NULL VARCHAR2(10)
```

2) Now let us drop the column model_no from products.



```
SQL> ALTER TABLE products
  2  DROP COLUMN model_no ;

Table altered.
```

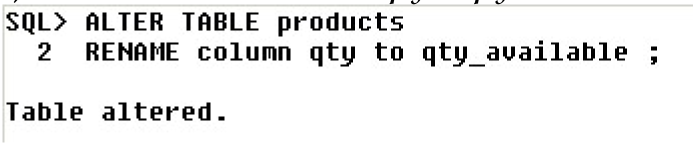
Thus, the column has been dropped.



```
SQL> desc products ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODID                                    NOT NULL NUMBER(4)
 PRODNAME                                  NOT NULL VARCHAR2(10)
 QTY                                                NUMBER(3)
 DESCRIPTION                                        VARCHAR2(20)
```

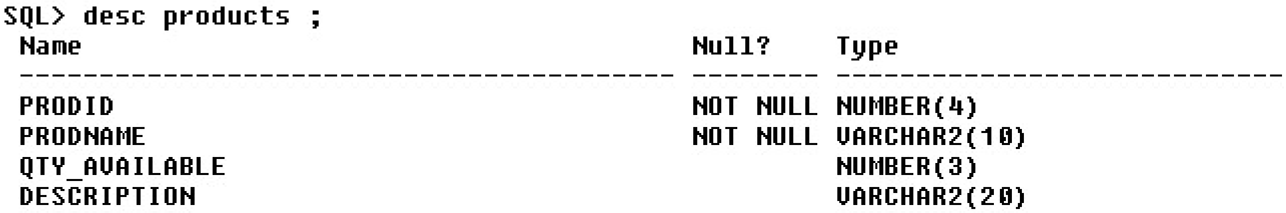
Thus, we can see from the description of the table – the column **model_no** has been dropped.

3) Let us rename the column *qty* to *qty_available*.



```
SQL> ALTER TABLE products
  2  RENAME column qty to qty_available ;

Table altered.
```

Let us verify if it has been renamed,



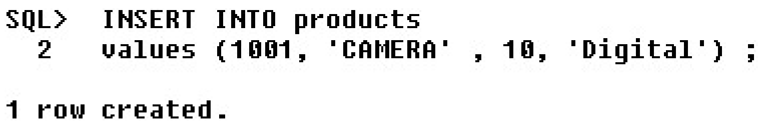
```
SQL> desc products ;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 PRODID                                    NOT NULL NUMBER(4)
 PRODNAME                                  NOT NULL VARCHAR2(10)
 QTY_AVAILABLE                                      NUMBER(3)
 DESCRIPTION                                        VARCHAR2(20)
```

NOTE : *SELECT* is neither DML nor DDL. It does not belong to any group because it does not alter anything, it just displays the data as required by the user.

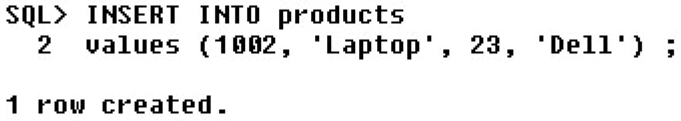
DML

INSERT

It inserts a record to a table.
Let us observe how it is done,



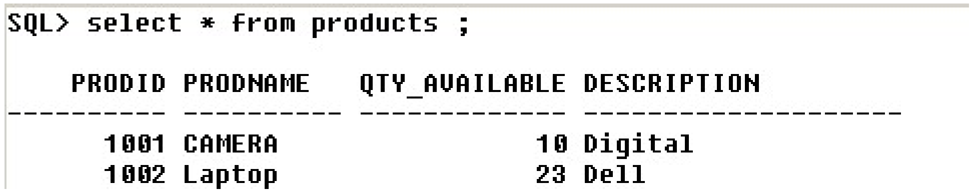
```
SQL>  INSERT INTO products
  2   values (1001, 'CAMERA' , 10, 'Digital') ;

1 row created.
```



```
SQL> INSERT INTO products
  2  values (1002, 'Laptop', 23, 'Dell') ;

1 row created.
```

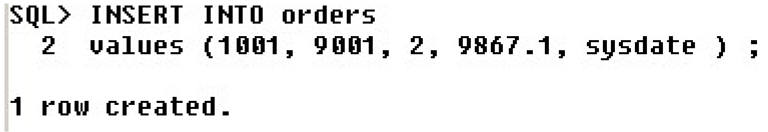
This is how we insert values into a table. All characters and alpha-numeric characters(ex – 10023sdf78) must be enclosed in single quotes (‘ ‘) and each value must be separated by comma. Also we must be careful in entering the data without violating the primary key, foreign key , unique constraints.

Now let us see the table in which the data in has been inserted,



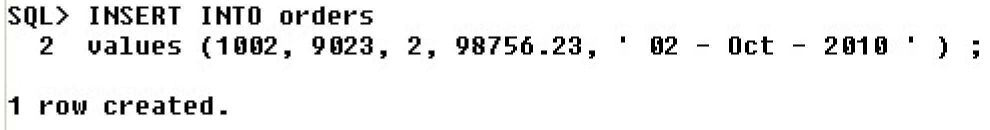
```
SQL> select * from products ;

    PRODID PRODNAME   QTY_AVAILABLE DESCRIPTION
---------- ---------- ------------- --------------------
      1001 CAMERA                10 Digital
      1002 Laptop                23 Dell
```

Now, let us insert data into the table **orders** in which a foreign key is referencing primary key,



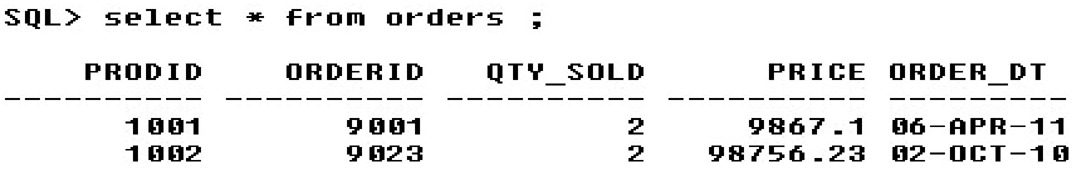
```
SQL> INSERT INTO orders
  2  values (1001, 9001, 2, 9867.1, sysdate ) ;

1 row created.
```

Here, we see that 1001 is the same prodid as of the earlier table.
Sysdate – it displays the current date set in the system .



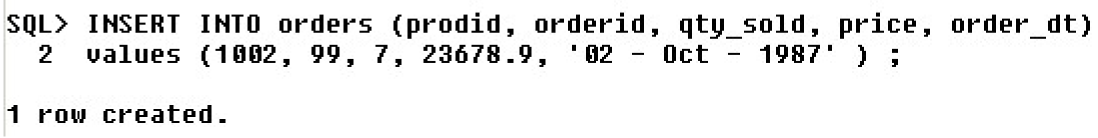
```
SQL> INSERT INTO orders
  2  values (1002, 9023, 2, 98756.23, ' 02 - Oct - 2010 ' ) ;

1 row created.
```

Now, let us see the table,



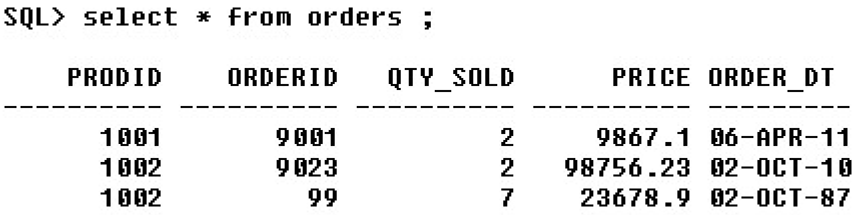
```
SQL> select * from orders ;

    PRODID    ORDERID   QTY_SOLD      PRICE ORDER_DT
---------- ---------- ---------- ---------- ---------
      1001       9001          2     9867.1 06-APR-11
      1002       9023          2   98756.23 02-OCT-10
```

Another way of inserting data into the table is shown below,



```
SQL> INSERT INTO orders (prodid, orderid, qty_sold, price, order_dt)
  2  values (1002, 99, 7, 23678.9, '02 - Oct - 1987' ) ;

1 row created.
```

Now, let us see the table,

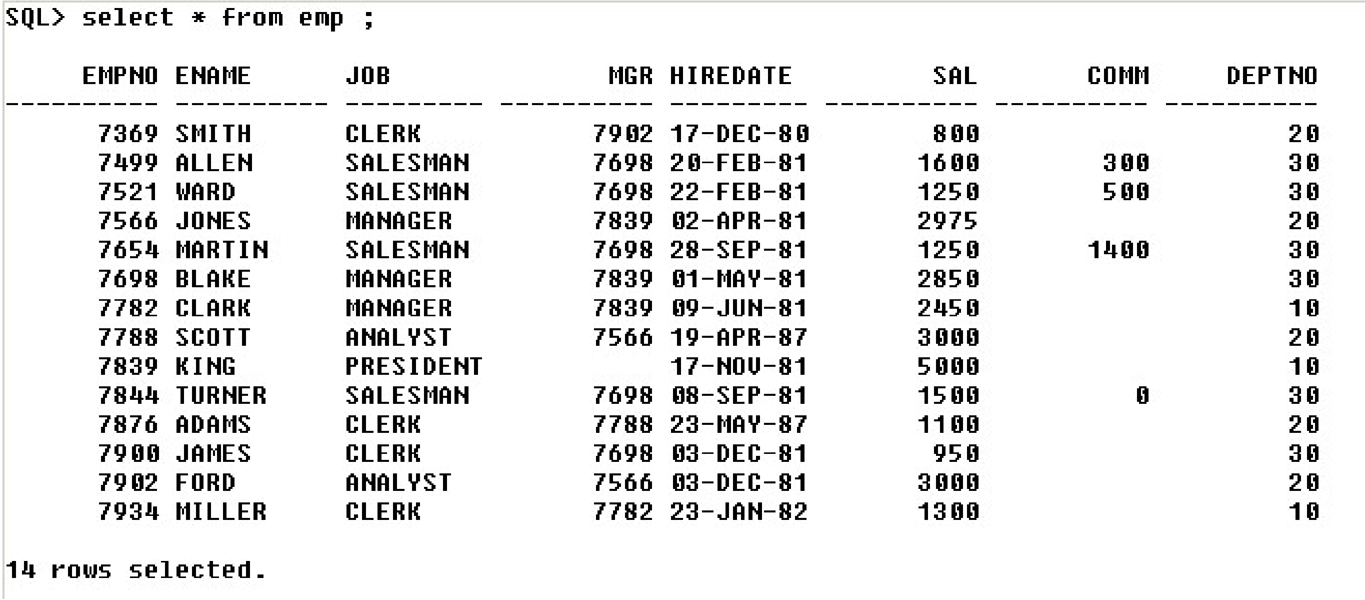


```
SQL> select * from orders ;

    PRODID    ORDERID   QTY_SOLD      PRICE ORDER_DT
---------- ---------- ---------- ---------- ---------
      1001       9001          2     9867.1 06-APR-11
      1002       9023          2   98756.23 02-OCT-10
      1002         99          7    23678.9 02-OCT-87
```

UPDATE :-

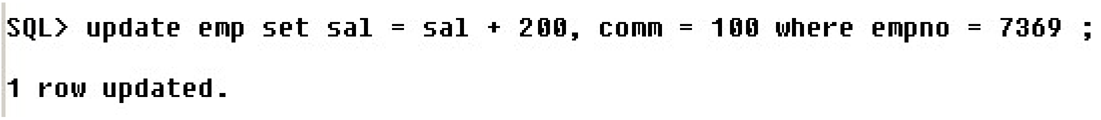
It updates one or more records.

For ex – 1) Let us update salary by increasing it by Rs200 and also give commission of Rs100 where empno = 7369.



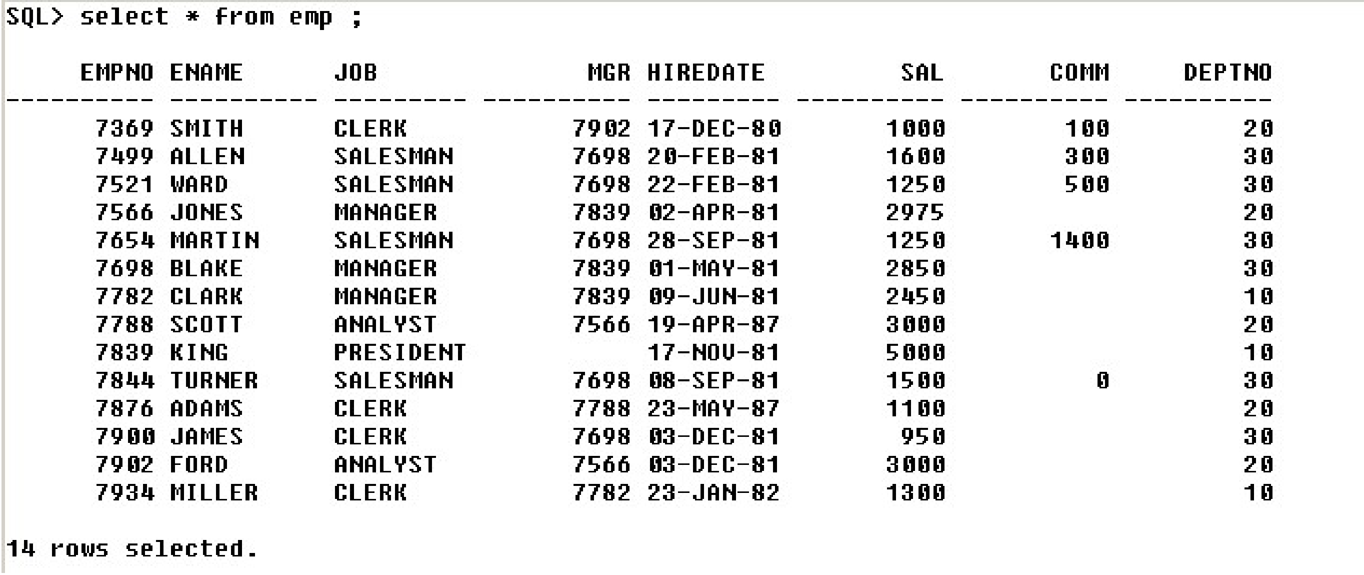
```
SQL> select * from emp ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10

14 rows selected.
```

Now, let us **update** the said record as shown below,



```
SQL> update emp set sal = sal + 200, comm = 100 where empno = 7369 ;

1 row updated.
```

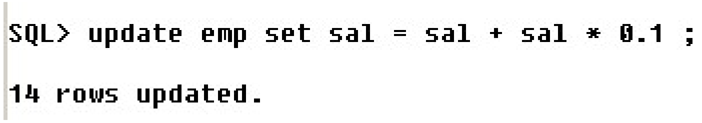
Let us verify if the record has been updated,



```
SQL> select * from emp ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80       1000        100         20
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10

14 rows selected.
```

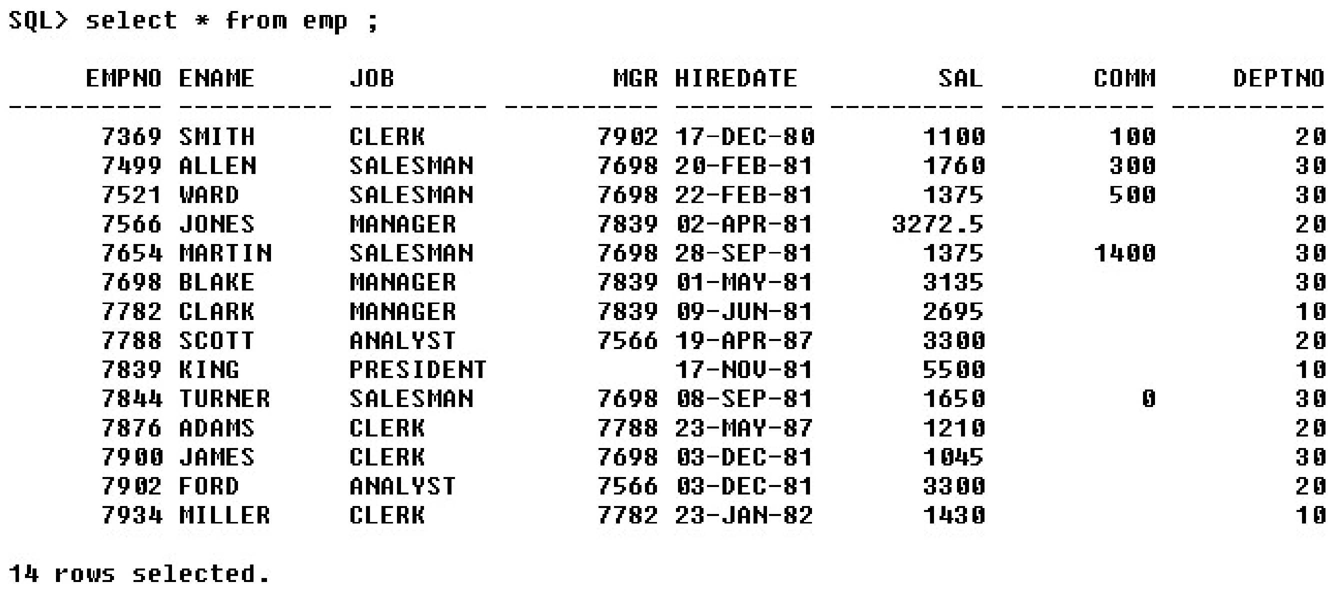
Thus, the record(empno – 7369) has been updated.

2) Increase all salary by 10%



```
SQL> update emp set sal = sal + sal * 0.1 ;

14 rows updated.
```

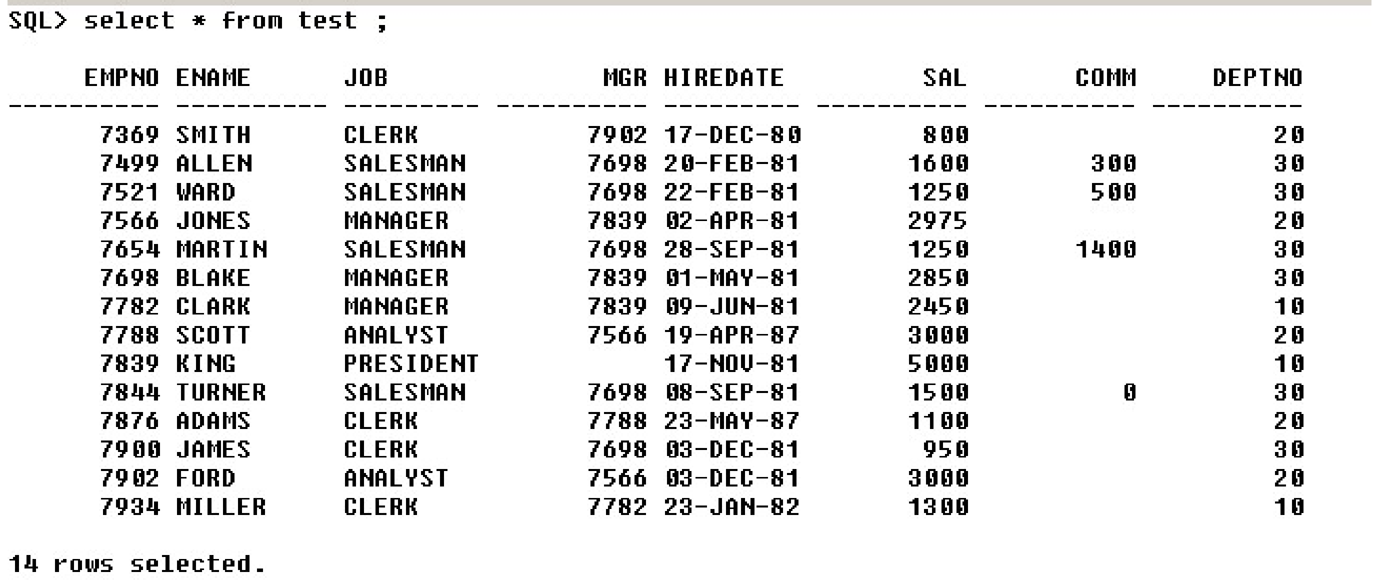
Let us verify it,



```
SQL> select * from emp ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80       1100        100         20
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1760        300         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1375        500         30
      7566 JONES      MANAGER         7839 02-APR-81     3272.5                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1375       1400         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       3135                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2695                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3300                    20
      7839 KING       PRESIDENT            17-NOV-81       5500                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1650          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       1210                    20
      7900 JAMES      CLERK           7698 03-DEC-81       1045                    30
      7902 FORD       ANALYST         7566 03-DEC-81       3300                    20
      7934 MILLER     CLERK           7782 23-JAN-82       1430                    10

14 rows selected.
```

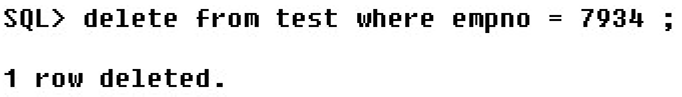
DELETE

It deletes one / some / all the records.
Let us create a table test from table emp – and see how to delete 1 record and how to delete all records from it,



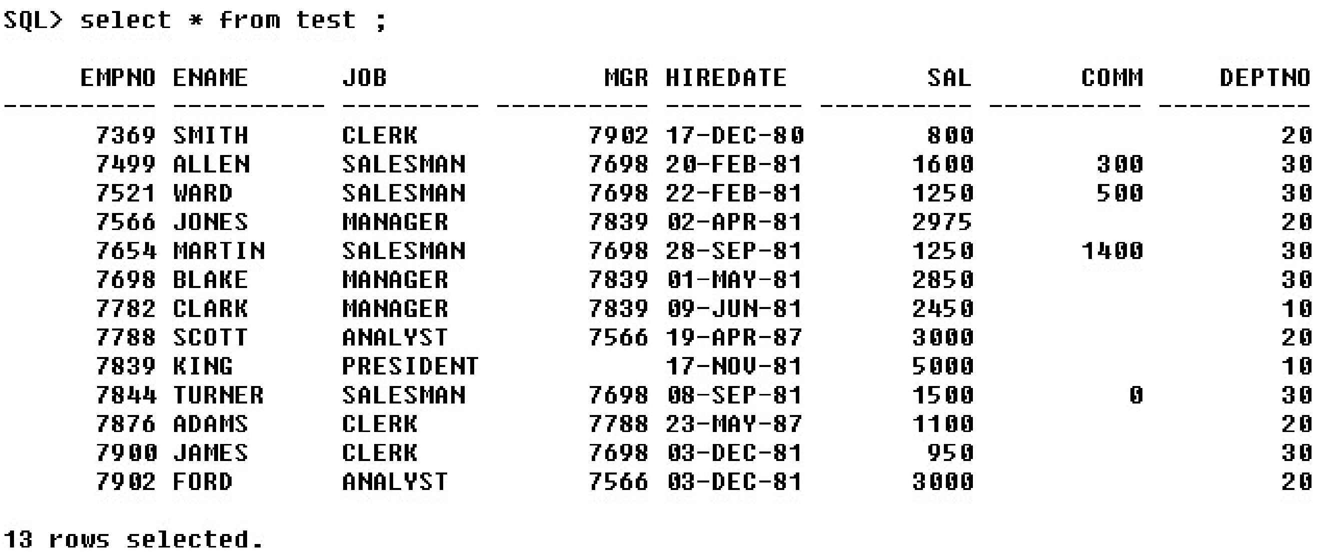
```
SQL> select * from test ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10

14 rows selected.
```

Thus, we have created the table test.



```
SQL> delete from test where empno = 7934 ;

1 row deleted.
```

Thus 1 row, ‘miller’ has been deleted.



```
SQL> select * from test ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20

13 rows selected.
```

Thus, the deletion has been confirmed.

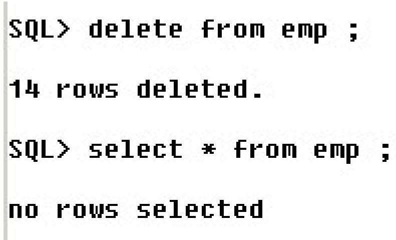
TCL

Any DML change on a table is not a permanent one.
We need to save the DML changes in order to make it permanent
We can also undo (ignore) the same DML changes on a table.

The DDL changes cannot be undone as they are implicitly saved.

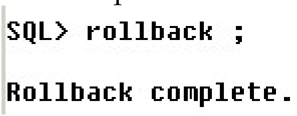
ROLLBACK

It undoes the DML changes performed on a table.
Let us see in the below example how **rollback** works,



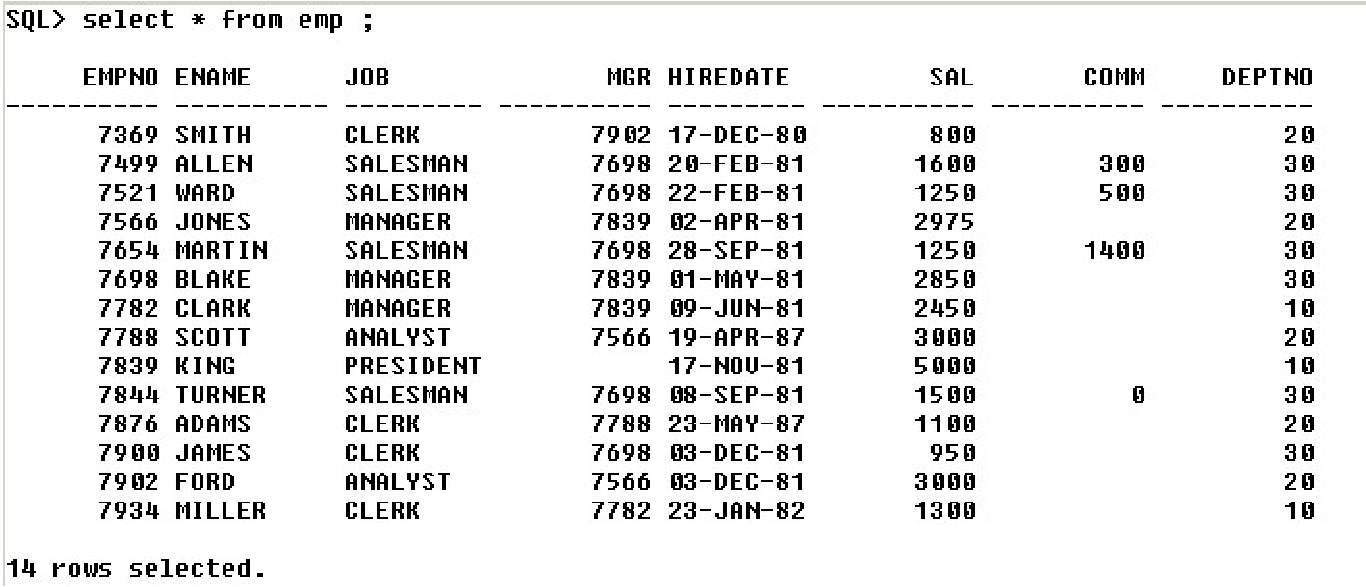
```
SQL> delete from emp ;

14 rows deleted.

SQL> select * from emp ;

no rows selected
```

Let us delete the employee table. When we perform **select** operation on emp, we can see that all the rows have been deleted.
We now perform the **rollback** operation,



```
SQL> rollback ;

Rollback complete.
```

Now let us perform the **select** operation,



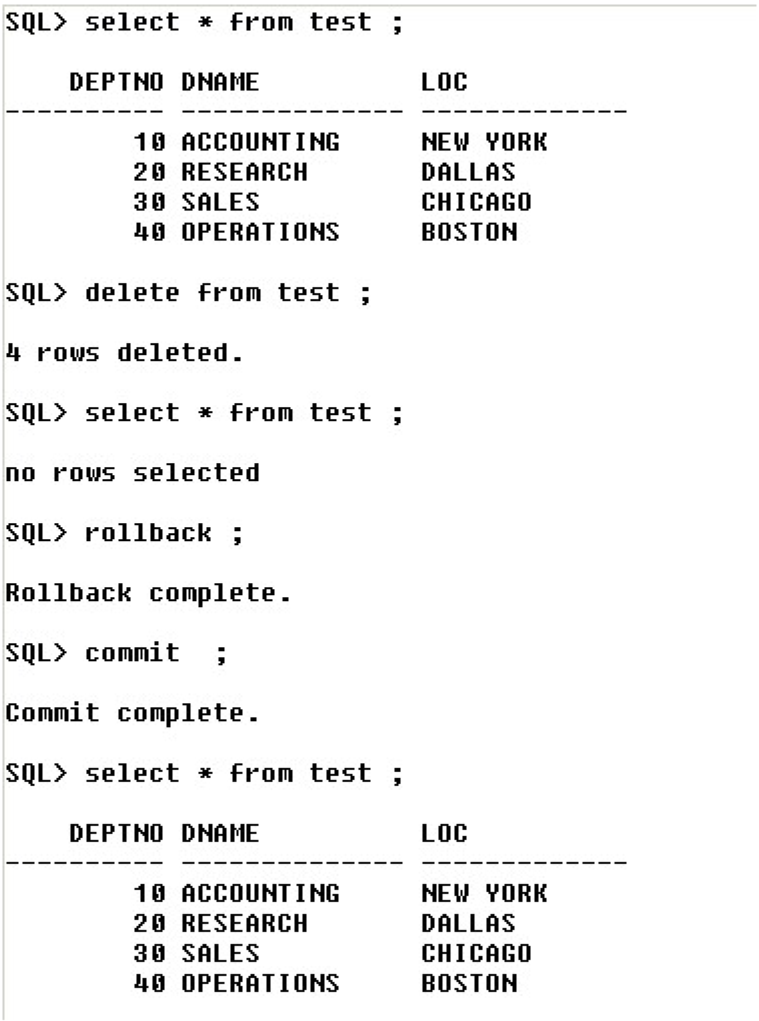
```
SQL> select * from emp ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-80        800                    20
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7839 KING       PRESIDENT            17-NOV-81       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7876 ADAMS      CLERK           7788 23-MAY-87       1100                    20
      7900 JAMES      CLERK           7698 03-DEC-81        950                    30
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10

14 rows selected.
```

Thus performing the **rollback** operation, we can retrieve all the records which had been deleted.

COMMIT

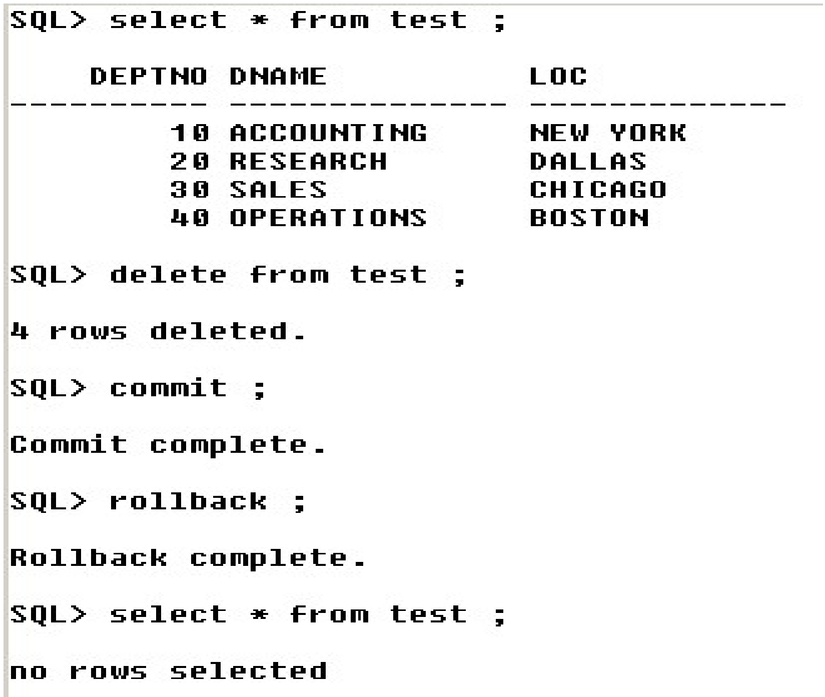
It saves the DML changes permanently to the database.

Committing after rollback & vice versa will not have any effect
Let us explain the above statement with an example,



```
SQL> select * from test ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQL> delete from test ;

4 rows deleted.

SQL> select * from test ;

no rows selected

SQL> rollback ;

Rollback complete.

SQL> commit  ;

Commit complete.

SQL> select * from test ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

We can see that **commit** has no effect after **rollback** operation.



```
SQL> select * from test ;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQL> delete from test ;

4 rows deleted.

SQL> commit ;

Commit complete.

SQL> rollback ;

Rollback complete.

SQL> select * from test ;

no rows selected
```

Thus, from above – we can see that **rollback** has no effect after **commit** operation.

During an abnormal exit – i.e, shutdown or if the SQL window is closed by mouse click – then all the DML's will be rolled back automatically.

During a normal exit – **exit ;** - all the DML's will be auto-committed – and there will be no rollback.

Ex – 1) INSERT
UPDATE
ALTER
DELETE
ROLLBACK

When we perform the following operations in the same order for a table – then INSERT, UPDATE will be committed – because ALTER is a DDL – and thus all the DML's above it will also be committed – because DDL operations cannot be undone.
Here – only DELETE will be rolled back because it's a DML.

2) INSERT
UPDATE
DELETE
ROLLBACK

Here, all are rolled back.

SAVEPOINT :

It is like a pointer (break-point) till where a DML will be rolled back.

Ex :-

Insert …

Save point x ;
Update …
Delete ..

Rollback to x ;
…
…

Here, only DELETE & UPDATE are rolled back.
INSERT is neither rolled back nor committed.

Assignments

1) Create the following tables

a) Table name :- STUDENTS
regno (PK)
name (NN)
semester
DOB
Phone

b) Table name :- BOOKS
bookno (PK)
bname
author

c) Table name :- LIBRARY
regno (FK from students)
bookno (FK from books)
DOI –date of issue
DOR – date of return

2) Insert 5 records to each of these tables

3) Differentiate between,
a) Delete and Truncate
b) Truncate and Drop
c) Char and Varchar
d) Drop and Delete

Single row functions

Functions – it is a re-usable program that returns a value.

Single row functions executes row by row that is it provide output for every record given as input.

Input argument of a single row function can be column name or expression.

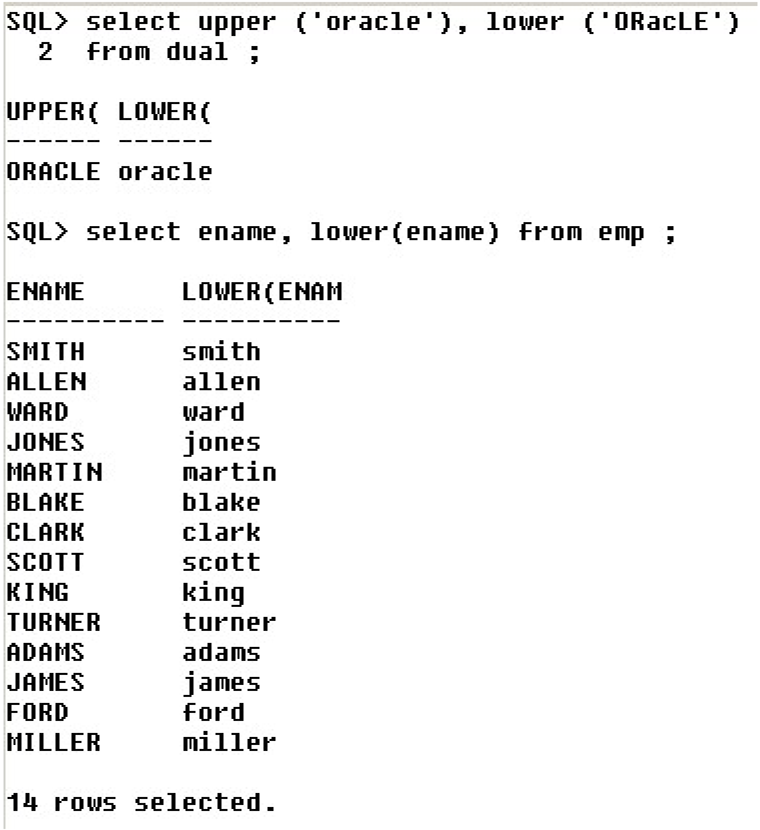
→ GROUP functions
→ CHARACTER functions
→ NUMERIC functions
→ DATE functions
→ SPECIAL functions

We have already learnt about GROUP functions.
Now, let us study the various CHARACTER functions.

<u>CHARACTER functions</u>

a) Upper: - it is used to convert the given string to upper case
b) Lower: - it is used to convert the given string to lower case
c) Length: - it is used to obtain no. of characters or digits present in the given string or no.
d)initcap: - it is used to convert the given string into init cap case.

For ex :-



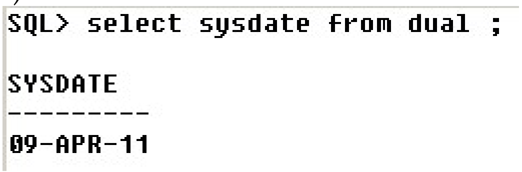
```
SQL> select upper ('oracle'), lower ('ORacLE')
  2  from dual ;

UPPER( LOWER(
------ ------
ORACLE oracle

SQL> select ename, lower(ename) from emp ;

ENAME      LOWER(ENAM
---------- ----------
SMITH      smith
ALLEN      allen
WARD       ward
JONES      jones
MARTIN     martin
BLAKE      blake
CLARK      clark
SCOTT      scott
KING       king
TURNER     turner
ADAMS      adams
JAMES      james
FORD       ford
MILLER     miller

14 rows selected.
```

In the 1st query, we see something called as **dual**.

Dual – is a dummy table which is used for performing some independent operations which will not depend on any of the existing tables.

For ex,

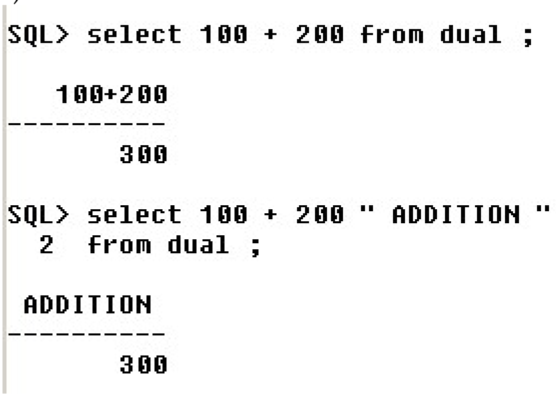
1)



```
SQL> select sysdate from dual ;

SYSDATE
---------
09-APR-11
```

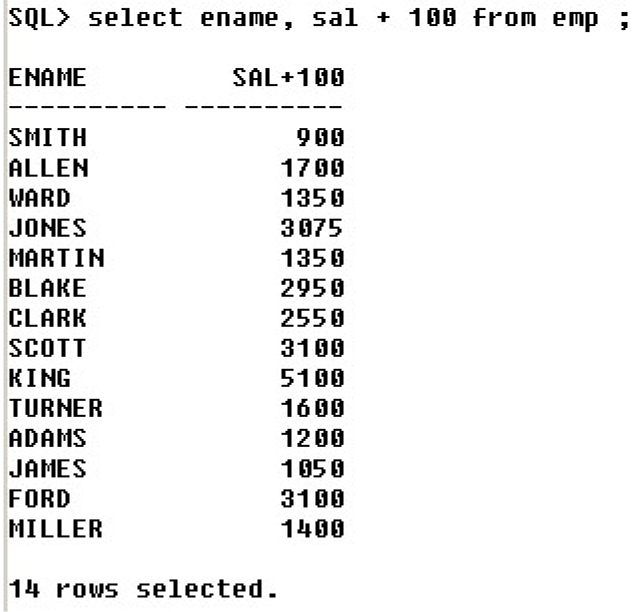
This gives the system date.

2)



```
SQL> select 100 + 200 from dual ;

   100+200
----------
       300

SQL> select 100 + 200 " ADDITION "
  2  from dual ;

 ADDITION
----------
       300
```

3)



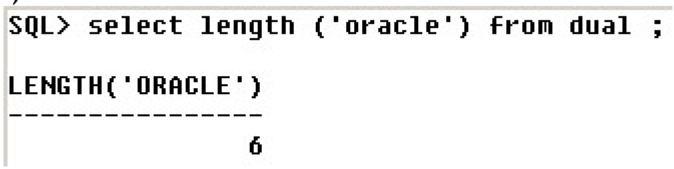
```
SQL> select ename, sal + 100 from emp ;

ENAME         SAL+100
---------- ----------
SMITH             900
ALLEN            1700
WARD             1350
JONES            3075
MARTIN           1350
BLAKE            2950
CLARK            2550
SCOTT            3100
KING             5100
TURNER           1600
ADAMS            1200
JAMES            1050
FORD             3100
MILLER           1400

14 rows selected.
```

We use dual – when the data is not present in any of the existing tables. Then we use dual.

<u>Length</u> – it returns the length of a given string.

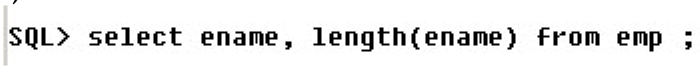
For ex,

1)

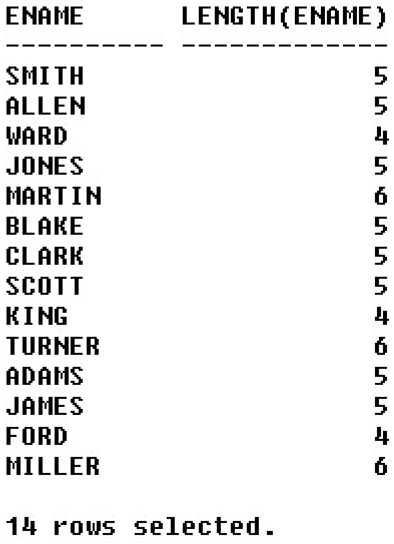


```
SQL> select length ('oracle') from dual ;

LENGTH('ORACLE')
----------------
               6
```

2)



```
SQL> select ename, length(ename) from emp ;
```



```
ENAME      LENGTH(ENAME)
---------- -------------
SMITH                  5
ALLEN                  5
WARD                   4
JONES                  5
MARTIN                 6
BLAKE                  5
CLARK                  5
SCOTT                  5
KING                   4
TURNER                 6
ADAMS                  5
JAMES                  5
FORD                   4
MILLER                 6

14 rows selected.
```

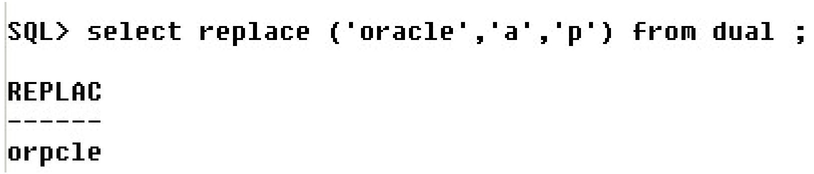
3) **<u>Display all the employees whose name & job is having exactly 5 characters</u>**

Select *
From emp
Where length(ename) and length(job) =5;

REPLACE

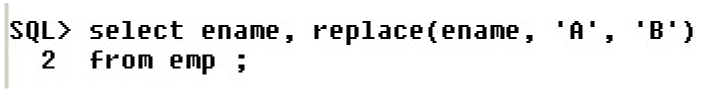
It replaces the old value with a new value in the given string.

For ex,



```
SQL> select replace ('oracle','a','p') from dual ;

REPLAC
------
orpcle
```

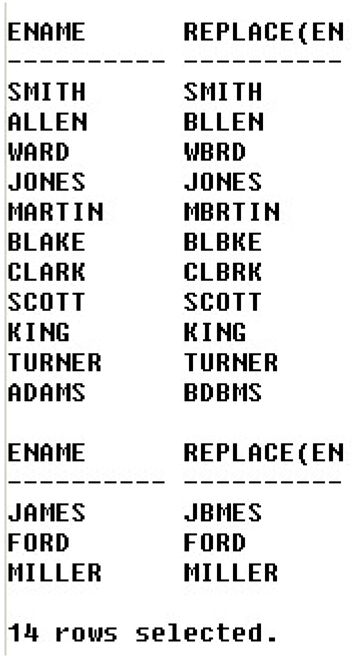
Here, **a –** is the old value to be replaced with **p** – which is the new value.



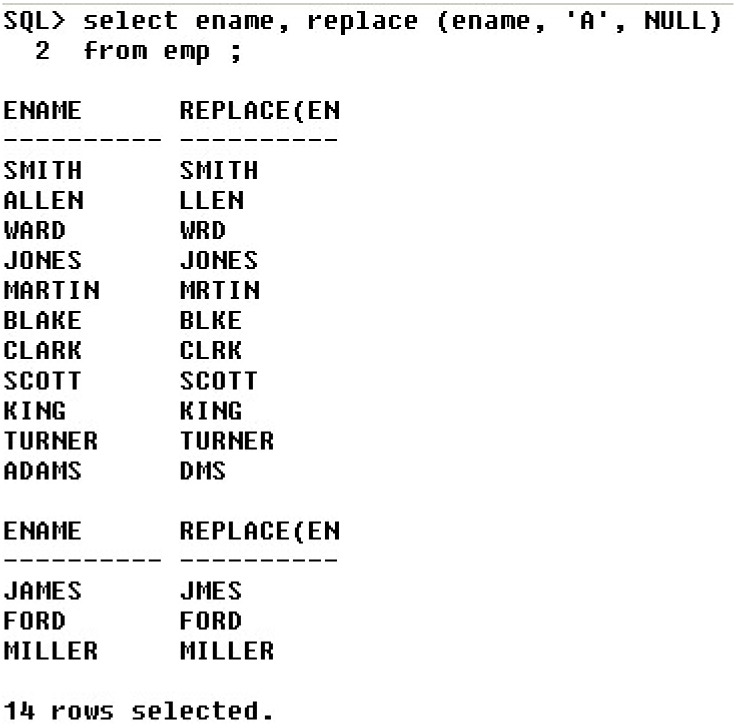
```
SQL> select ename, replace(ename, 'A', 'B')
  2  from emp ;
```

This query replaces all the names which has ‘A’ in it with ‘B’.

Let us see the output as shown below,



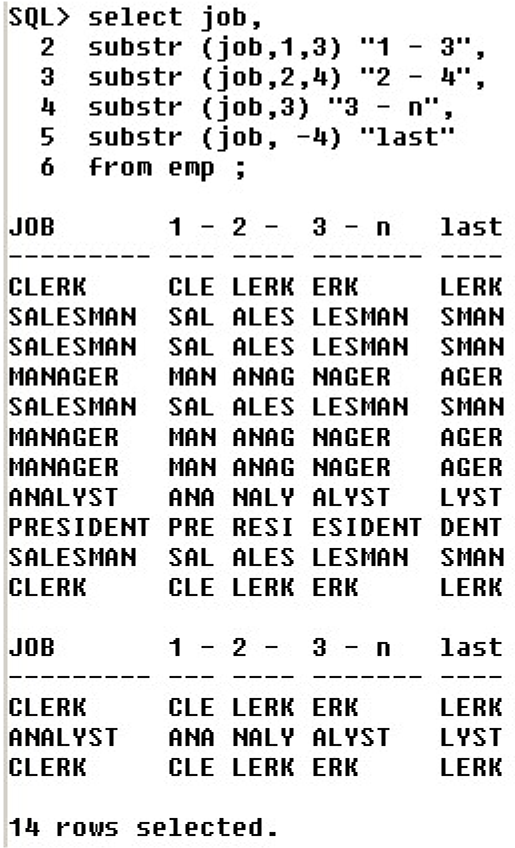
```
ENAME      REPLACE(EN
---------- ----------
SMITH      SMITH
ALLEN      BLLEN
WARD       WBRD
JONES      JONES
MARTIN     MBRTIN
BLAKE      BLBKE
CLARK      CLBRK
SCOTT      SCOTT
KING       KING
TURNER     TURNER
ADAMS      BDBMS

ENAME      REPLACE(EN
---------- ----------
JAMES      JBMES
FORD       FORD
MILLER     MILLER

14 rows selected.
```



```
SQL> select ename, replace (ename, 'A', NULL)
  2  from emp ;

ENAME      REPLACE(EN
---------- ----------
SMITH      SMITH
ALLEN      LLEN
WARD       WRD
JONES      JONES
MARTIN     MRTIN
BLAKE      BLKE
CLARK      CLRK
SCOTT      SCOTT
KING       KING
TURNER     TURNER
ADAMS      DMS

ENAME      REPLACE(EN
---------- ----------
JAMES      JMES
FORD       FORD
MILLER     MILLER

14 rows selected.
```

SUBSTR

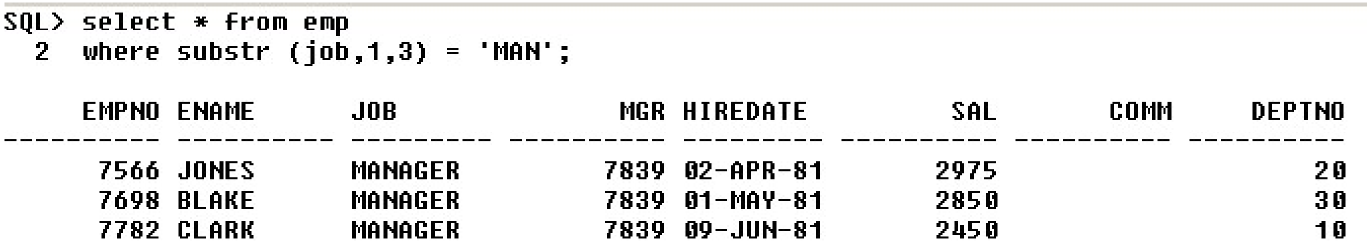
Substring function is used to obtain a new string from a given string. This is called **substring**.
It extracts 'n' characters from x(th) position of a given string.
For ex,



```
SQL> select job,
  2  substr (job,1,3) "1 - 3",
  3  substr (job,2,4) "2 - 4",
  4  substr (job,3) "3 - n",
  5  substr (job, -4) "last"
  6  from emp ;

JOB       1 - 2 -  3 - n   last
--------- --- ---- ------- ----
CLERK     CLE LERK ERK     LERK
SALESMAN  SAL ALES LESMAN  SMAN
SALESMAN  SAL ALES LESMAN  SMAN
MANAGER   MAN ANAG NAGER   AGER
SALESMAN  SAL ALES LESMAN  SMAN
MANAGER   MAN ANAG NAGER   AGER
MANAGER   MAN ANAG NAGER   AGER
ANALYST   ANA NALY ALYST   LYST
PRESIDENT PRE RESI ESIDENT DENT
SALESMAN  SAL ALES LESMAN  SMAN
CLERK     CLE LERK ERK     LERK

JOB       1 - 2 -  3 - n   last
--------- --- ---- ------- ----
CLERK     CLE LERK ERK     LERK
ANALYST   ANA NALY ALYST   LYST
CLERK     CLE LERK ERK     LERK

14 rows selected.
```

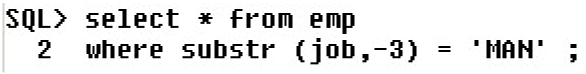
Here **, (job, '1' , '3')** – means from **job** – extract **from 1st position , 3 characters**.

1) <u>**Display the employees whose job starts with 'man'**</u>

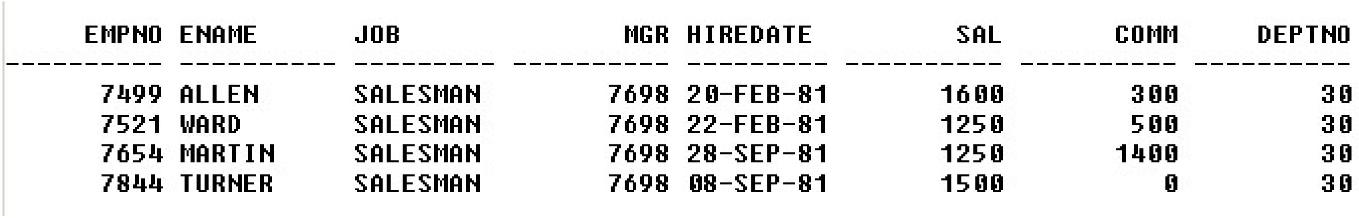


```
SQL> select * from emp
  2  where substr (job,1,3) = 'MAN';

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
```

2) <u>**Display the employees whose job ends with 'man'**</u>



```
SQL> select * from emp
  2  where substr (job,-3) = 'MAN' ;
```



```
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
```

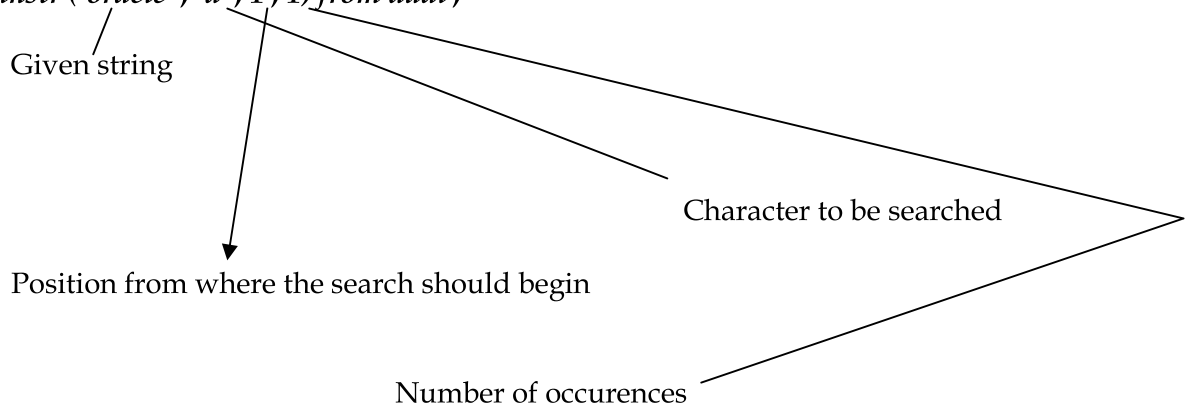
INSTR

This is also called as **instring.**

It returns position of a given character in a given string.

For ex,

Select instr ('oracle' , 'a' , 1 , 1) from dual ;

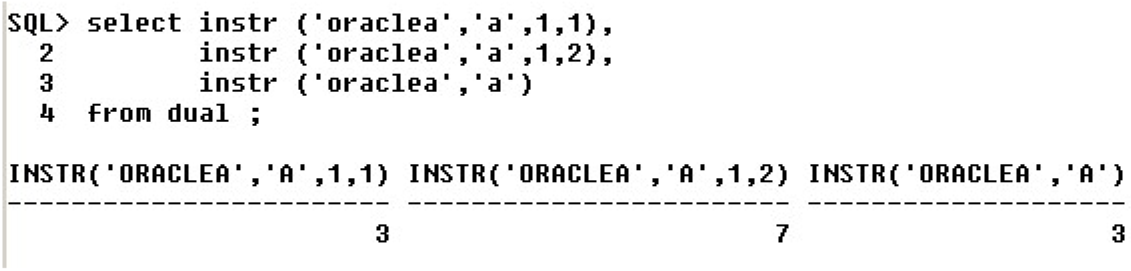


Given string

Character to be searched

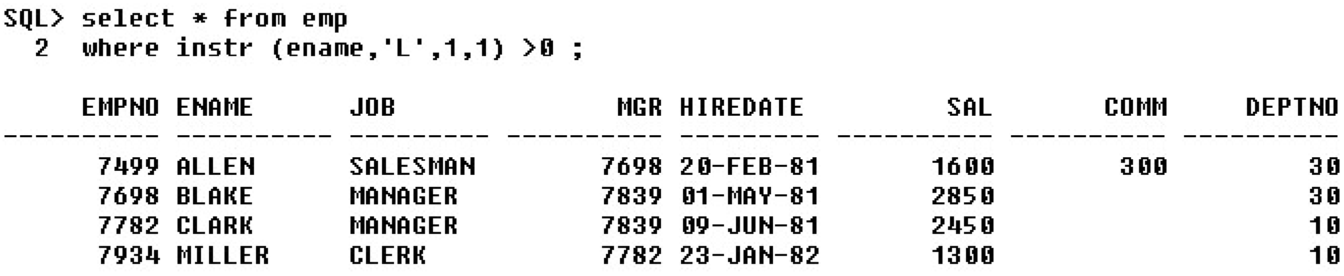
Position from where the search should begin

Number of occurences



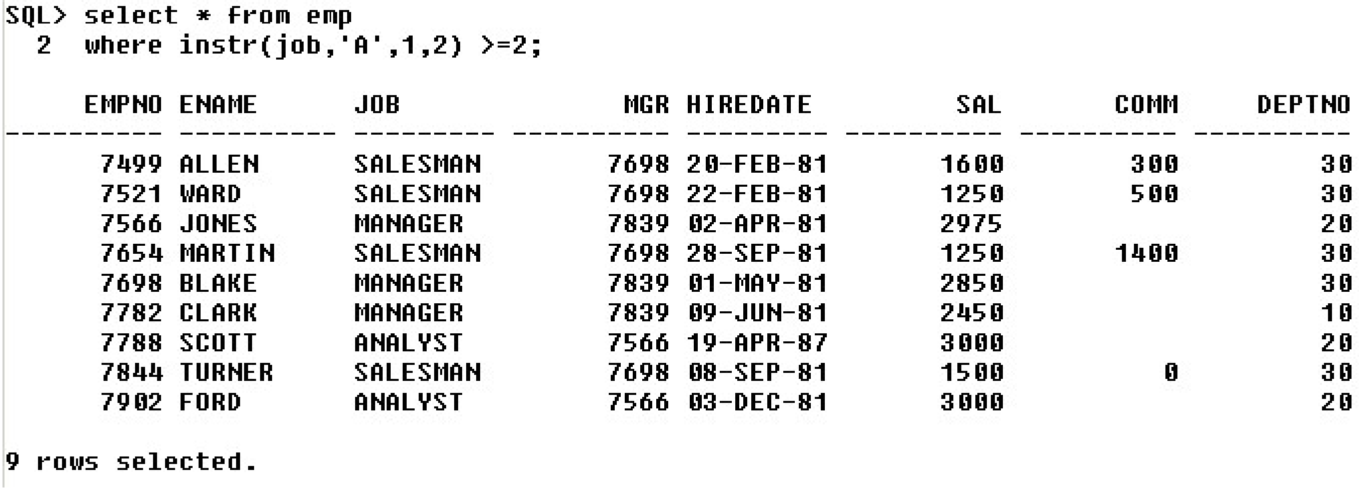
```
SQL> select instr ('oraclea','a',1,1),
  2         instr ('oraclea','a',1,2),
  3         instr ('oraclea','a')
  4  from dual ;

INSTR('ORACLEA','A',1,1) INSTR('ORACLEA','A',1,2) INSTR('ORACLEA','A')
------------------------ ------------------------ --------------------
                       3                        7                    3
```

<u>Display all the employees whose name is having 'L'</u>



```
SQL> select * from emp
  2  where instr (ename,'L',1,1) >0 ;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7934 MILLER     CLERK           7782 23-JAN-82       1300                    10
```

List the employees whose job is having atleast 2 A's in it

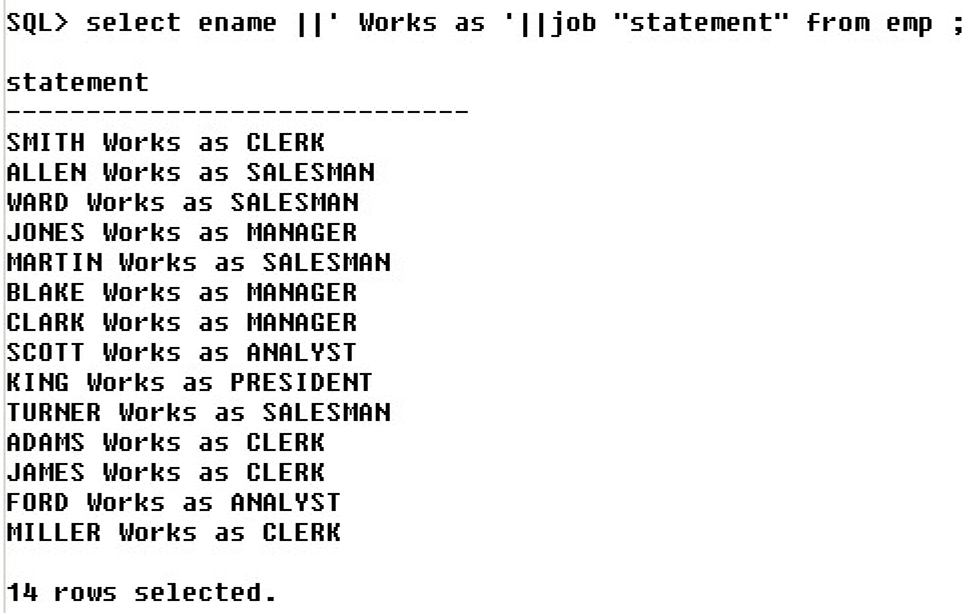


```
SQL> select * from emp
  2  where instr(job,'A',1,2) >=2;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7499 ALLEN      SALESMAN        7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN        7698 22-FEB-81       1250        500         30
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-81       1250       1400         30
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
      7788 SCOTT      ANALYST         7566 19-APR-87       3000                    20
      7844 TURNER     SALESMAN        7698 08-SEP-81       1500          0         30
      7902 FORD       ANALYST         7566 03-DEC-81       3000                    20

9 rows selected.
```

CONCAT

It concatenates any two values or columns.
It is represented by - ||

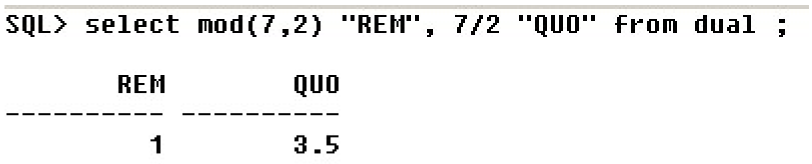
For ex,



```
SQL> select ename ||' Works as '||job "statement" from emp ;

statement
-----------------------------
SMITH Works as CLERK
ALLEN Works as SALESMAN
WARD Works as SALESMAN
JONES Works as MANAGER
MARTIN Works as SALESMAN
BLAKE Works as MANAGER
CLARK Works as MANAGER
SCOTT Works as ANALYST
KING Works as PRESIDENT
TURNER Works as SALESMAN
ADAMS Works as CLERK
JAMES Works as CLERK
FORD Works as ANALYST
MILLER Works as CLERK

14 rows selected.
```

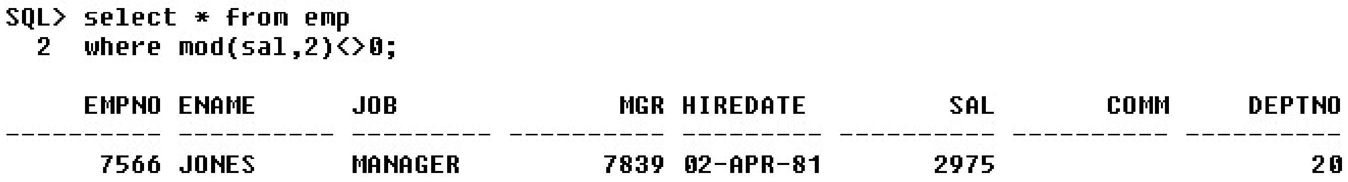
NUMERIC FUNCTIONS

1) **Mod** :- it returns the remainder when 1 number is divided by the other.



```
SQL> select mod(7,2) "REM", 7/2 "QUO" from dual ;

       REM        QUO
---------- ----------
         1        3.5
```

Display the employees earning odd numbered salaries.

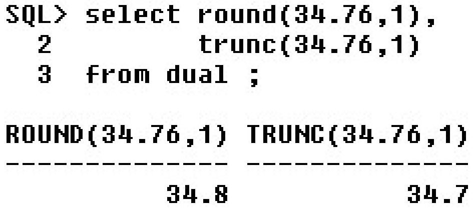


```
SQL> select * from emp
  2  where mod(sal,2)<>0;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7566 JONES      MANAGER         7839 02-APR-81       2975                    20
```

Round
It rounds off a given number to the nearest decimal place.

Trunc
It truncates the given number to the given decimal place. Truncate does not do any rounding.

For ex,



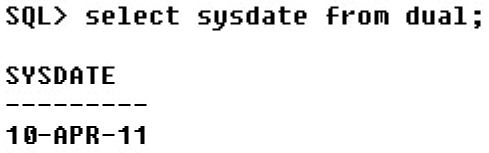
```
SQL> select round(34.76,1),
  2         trunc(34.76,1)
  3  from dual ;

ROUND(34.76,1) TRUNC(34.76,1)
-------------- --------------
          34.8           34.7
```

Here, '**1**' indicates the number of positions.

DATE FUNCTIONS

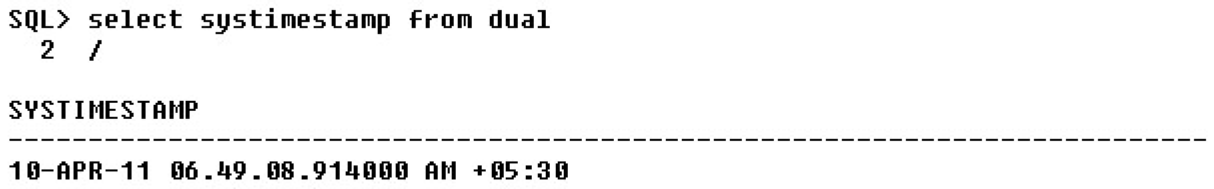
1) Sysdate
Stands for System date.
It returns both date & time, but by default – only date is displayed.
The default format is,

dd – mon – yy



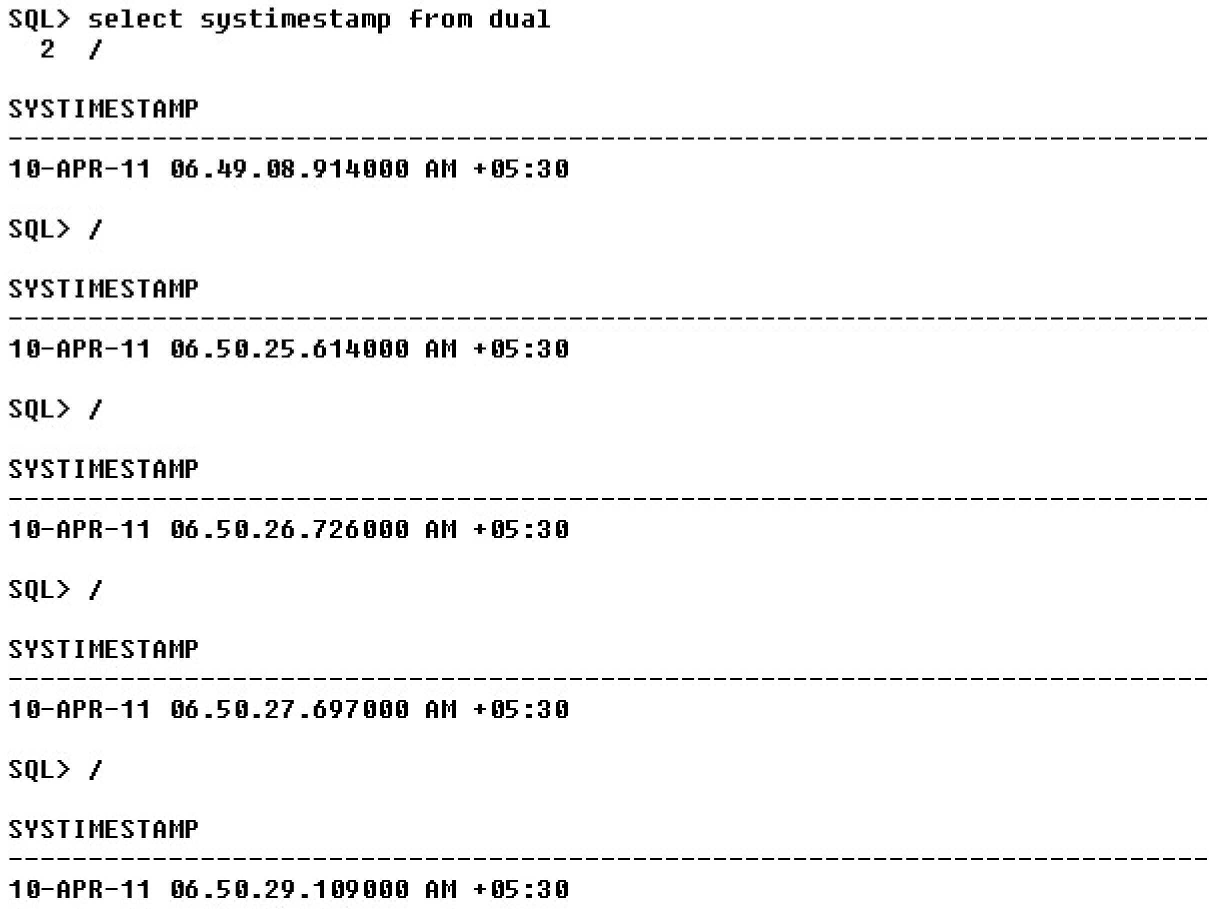
```
SQL> select sysdate from dual;

SYSDATE
---------
10-APR-11
```

2) Systimestamp
Introduced from Oracle 9i
Returns date, time and timezone.



```
SQL> select systimestamp from dual
  2  /

SYSTIMESTAMP
---------------------------------------------------------------------------
10-APR-11 06.49.08.914000 AM +05:30
```

Here, **.914000** – gives the fraction of millisecond which keeps changing as shown below,



```
SQL> select systimestamp from dual
  2  /

SYSTIMESTAMP
---------------------------------------------------------------------------
10-APR-11 06.49.08.914000 AM +05:30

SQL> /

SYSTIMESTAMP
---------------------------------------------------------------------------
10-APR-11 06.50.25.614000 AM +05:30

SQL> /

SYSTIMESTAMP
---------------------------------------------------------------------------
10-APR-11 06.50.26.726000 AM +05:30

SQL> /

SYSTIMESTAMP
---------------------------------------------------------------------------
10-APR-11 06.50.27.697000 AM +05:30

SQL> /

SYSTIMESTAMP
---------------------------------------------------------------------------
10-APR-11 06.50.29.109000 AM +05:30
```

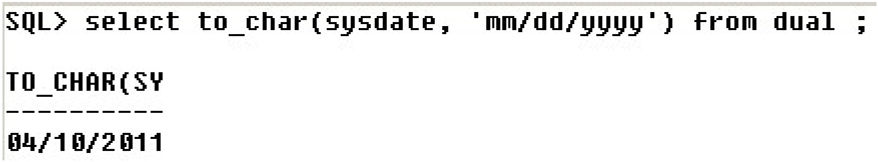
In interview – if they ask you – “ *which function contains fractions of a second* “ OR “*how to see the system time* “ – then answer is “SYSTIMESTAMP”.

SPECIAL FUNCTIONS

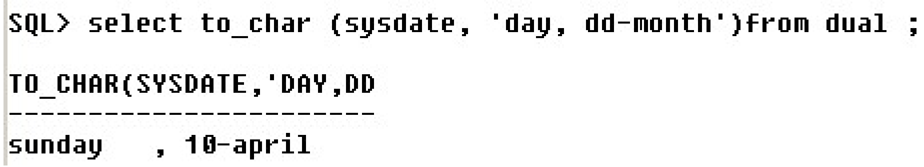
1) TO – CHAR

Used for displaying the date in different formats.

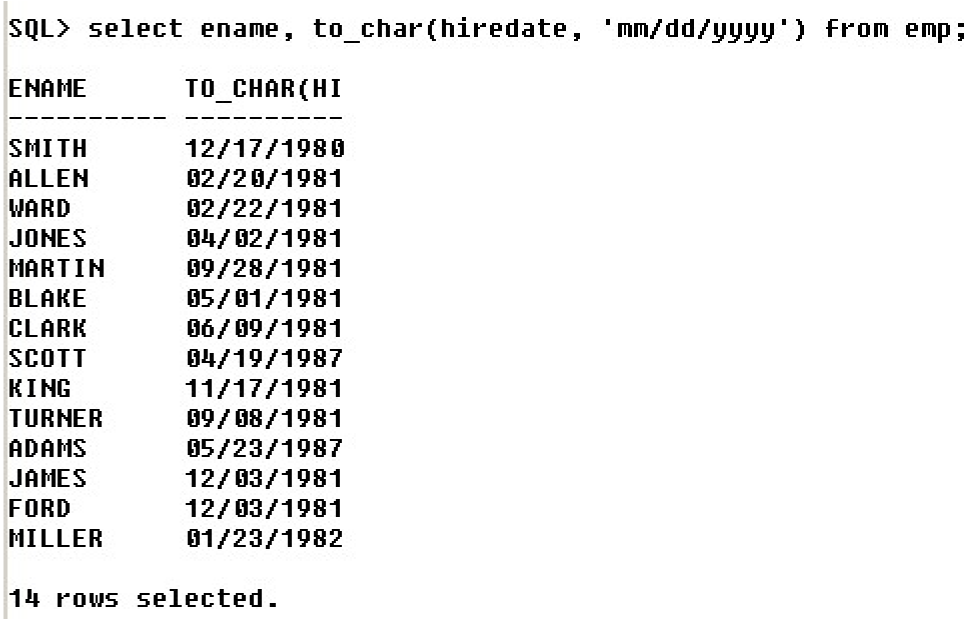
For ex,



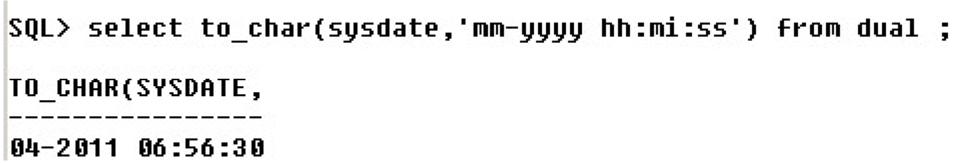
```
SQL> select to_char(sysdate, 'mm/dd/yyyy') from dual ;

TO_CHAR(SY
----------
04/10/2011
```



```
SQL> select to_char (sysdate, 'day, dd-month')from dual ;

TO_CHAR(SYSDATE,'DAY,DD
-----------------------
sunday   , 10-april
```

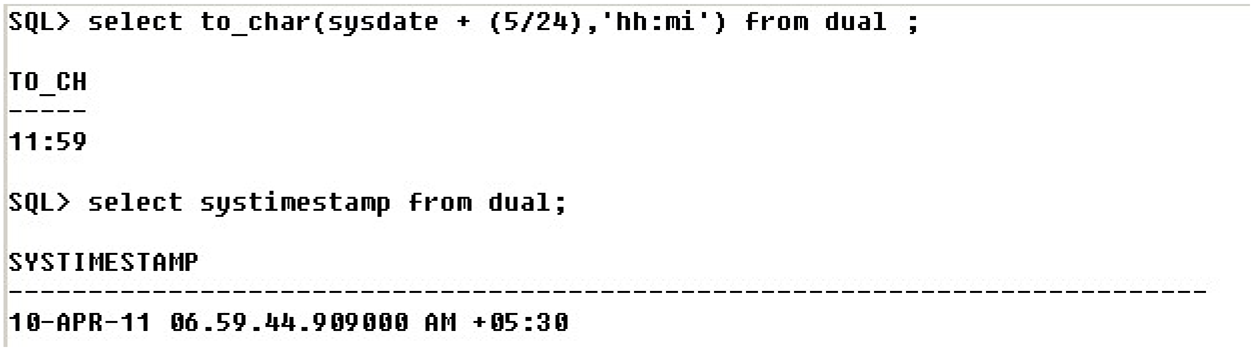


```
SQL> select ename, to_char(hiredate, 'mm/dd/yyyy') from emp;

ENAME      TO_CHAR(HI
---------- ----------
SMITH      12/17/1980
ALLEN      02/20/1981
WARD       02/22/1981
JONES      04/02/1981
MARTIN     09/28/1981
BLAKE      05/01/1981
CLARK      06/09/1981
SCOTT      04/19/1987
KING       11/17/1981
TURNER     09/08/1981
ADAMS      05/23/1987
JAMES      12/03/1981
FORD       12/03/1981
MILLER     01/23/1982

14 rows selected.
```



```
SQL> select to_char(sysdate,'mm-yyyy hh:mi:ss') from dual ;

TO_CHAR(SYSDATE,
----------------
04-2011 06:56:30
```

Now, let us see how to add 5 hrs to the existing time,



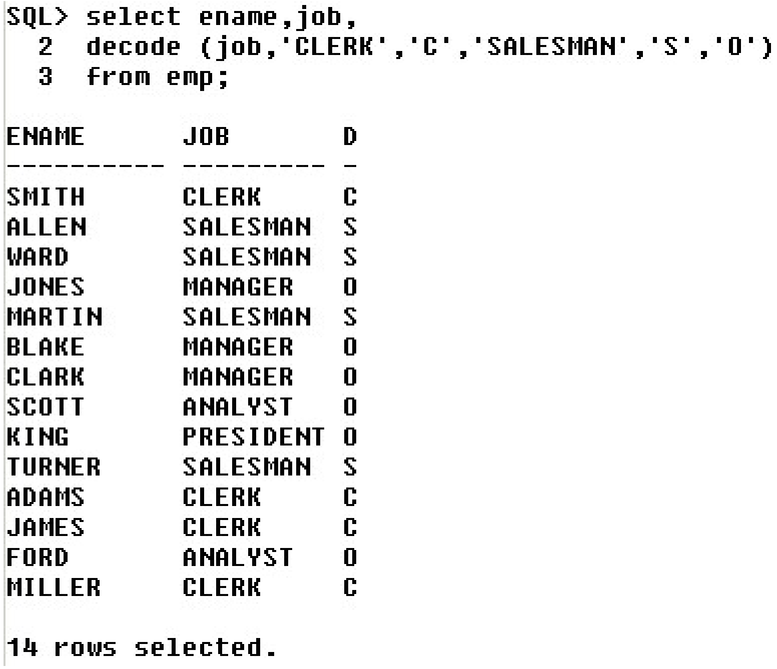
```
SQL> select to_char(sysdate + (5/24),'hh:mi') from dual ;

TO_CH
-----
11:59

SQL> select systimestamp from dual;

SYSTIMESTAMP
---------------------------------------------------------------------------
10-APR-11 06.59.44.909000 AM +05:30
```

We can see that 5 hrs has been added to the current time.

DECODE

It works like '**if – then – else**' statement.

For ex,



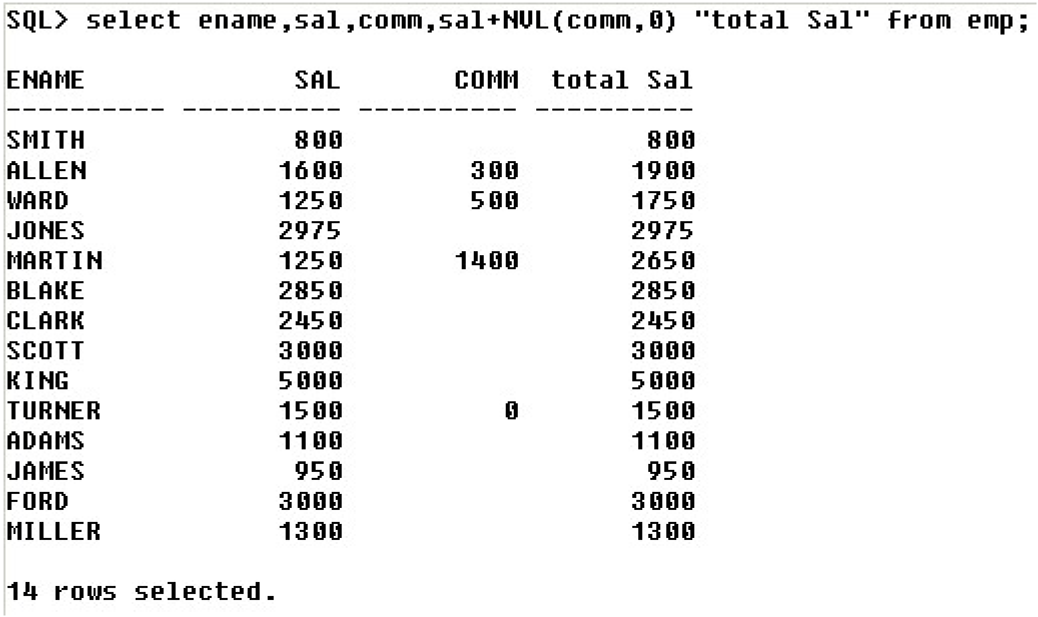
```
SQL> select ename,job,
  2  decode (job,'CLERK','C','SALESMAN','S','O')
  3  from emp;

ENAME      JOB       D
---------- --------- -
SMITH      CLERK     C
ALLEN      SALESMAN  S
WARD       SALESMAN  S
JONES      MANAGER   O
MARTIN     SALESMAN  S
BLAKE      MANAGER   O
CLARK      MANAGER   O
SCOTT      ANALYST   O
KING       PRESIDENT O
TURNER     SALESMAN  S
ADAMS      CLERK     C
JAMES      CLERK     C
FORD       ANALYST   O
MILLER     CLERK     C

14 rows selected.
```

The above query states that – in job, if clerk is there, replace with C – else if salesman is there, replace it with S – else replace with 'O'.

<u>NVL</u>

It substitutes a value for a null.

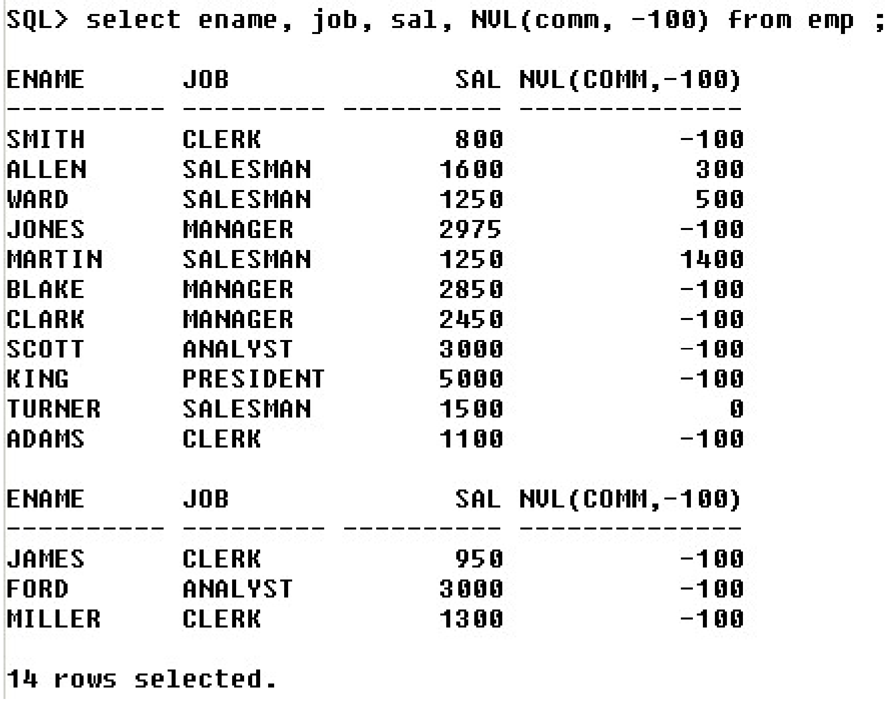
For ex,



```
SQL> select ename,sal,comm,sal+NVL(comm,0) "total Sal" from emp;

ENAME             SAL       COMM  total Sal
---------- ---------- ---------- ----------
SMITH             800                   800
ALLEN            1600        300       1900
WARD             1250        500       1750
JONES            2975                  2975
MARTIN           1250       1400       2650
BLAKE            2850                  2850
CLARK            2450                  2450
SCOTT            3000                  3000
KING             5000                  5000
TURNER           1500          0       1500
ADAMS            1100                  1100
JAMES             950                   950
FORD             3000                  3000
MILLER           1300                  1300

14 rows selected.
```

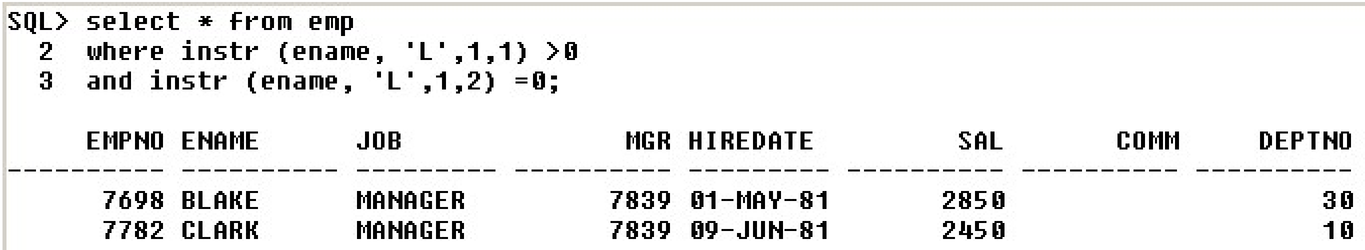
The above query means – if the employee has commission, then add sal + comm. To get total salary – else add 0 to the sal and display total salary.

<u>Display employee name, job, salary and commission. If the commission is NULL, then display -100</u>



```
SQL> select ename, job, sal, NVL(comm, -100) from emp ;

ENAME      JOB              SAL NVL(COMM,-100)
---------- --------- ---------- --------------
SMITH      CLERK            800           -100
ALLEN      SALESMAN        1600            300
WARD       SALESMAN        1250            500
JONES      MANAGER         2975           -100
MARTIN     SALESMAN        1250           1400
BLAKE      MANAGER         2850           -100
CLARK      MANAGER         2450           -100
SCOTT      ANALYST         3000           -100
KING       PRESIDENT       5000           -100
TURNER     SALESMAN        1500              0
ADAMS      CLERK           1100           -100

ENAME      JOB              SAL NVL(COMM,-100)
---------- --------- ---------- --------------
JAMES      CLERK            950           -100
FORD       ANALYST         3000           -100
MILLER     CLERK           1300           -100

14 rows selected.
```

Display all employees whose name is having exactly 1 ‘L’ in it



```
SQL> select * from emp
  2  where instr (ename, 'L',1,1) >0
  3  and instr (ename, 'L',1,2) =0;

     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM     DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7698 BLAKE      MANAGER         7839 01-MAY-81       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-81       2450                    10
```