

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.

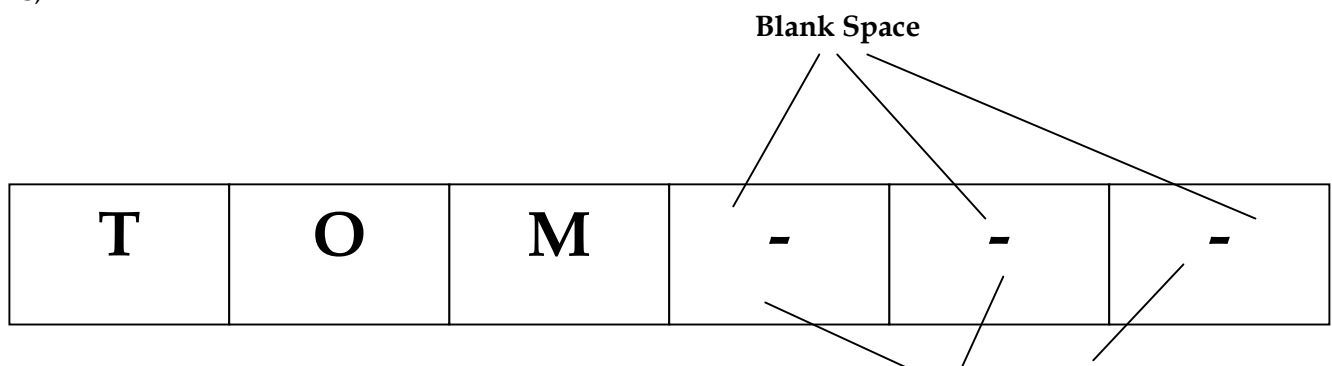
Difference between CHAR & VARCHAR

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.

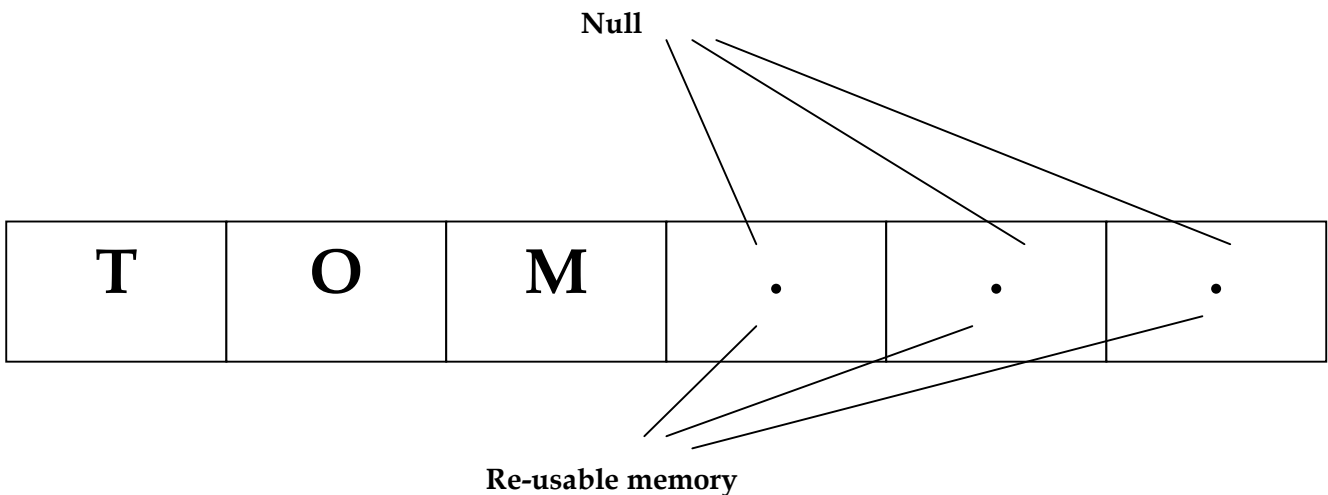
Now, when we give *tom*, it allocates 6 bytes for **name char** – only the 1st 3bytes are used to store **Tom** – the rest becomes waste as it is a blank space and it is reserved memory.

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
PROID		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.

Creating a table from another table :-

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

PROID	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 ;
```

PROID	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