Unit III

DATABASE LANGUAGE

By default, Database permit users to perform operations such as insert, select, update, delete operations. To perform those operations in Database, we need a support of Language. Language which supports to perform above said operations is called Database Language.

Types of Database language:

- Data Definition Language(DDL)
- Data Manipulation Language(DML)
- Transactional Control Language(TCL)

OPERATORS IN SQL

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

SQL ARITHMETIC OPERATORS

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left hand operand by right hand operand.	b / a will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder.	b % a will give 0

SQL COMPARISON OPERATORS

Operator	Description	Example

=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a! = b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a! < b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a!> b) is true.

SQL LOGICAL OPERATORS

Here is a list of all the logical operators available in SQL.

S.No.	Operator & Description
1	**ALL**

	The ALL operator is used to compare a value to all values in another value set.
2	**AND** The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
3	**ANY** The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	**BETWEEN** The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
5	**EXISTS** The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	**IN** The IN operator is used to compare a value to a list of literal values that have been specified.
7	**LIKE** The LIKE operator is used to compare a value to similar values using wildcard operators.
8	**NOT** The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. This is a negate operator.
9	**OR** The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	**IS NULL** The NULL operator is used to compare a value with a NULL value.

SQL SET OPERATORS

Set operators are used to join the results of two (or more) SELECT statements. The SET operators available are UNION, UNION ALL, INTERSECT and MINUS.

UNION

UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and data type must be same in both the tables, on which UNION operation is being applied.

Example of UNION

The **First** table,

ID	Name
1	abhi
2	Adam

The **Second** table,

ID	Name

2	Adam
3	Chester

Union SQL query will be,

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

The result set table will look like,

ID	**NAME**
1	Abhi
2	adam
3	Chester

UNION ALL

This operation is similar to Union. But it also shows the duplicate rows.

Example of Union All

Union All query will be like,

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

The resultset table will look like,

ID	NAME
1	abhi
2	Adam
2	Adam
3	Chester

INTERSECT

Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same.

Example of Intersect

Intersect query will be,

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

The resultset table will look like

ID	**NAME**
2	Adam

MINUS

The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.

Example of Minus

Minus query will be,

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

The result set table will look like,

ID	NAME
1	abhi

DATA DEFINITION LANGUAGE

It is also known as DDL in short form. It is used to define the physical structure of the Database. It is used to modify the physical structure of existing Database. Using this Language, user can remove physical structure of Database as well as records of Database.

There are many commands to perform those operations

- **Create command:**
  This command is used to create a new table in Database.
  Using this command, user can create only one table at a time.

Syntax:

Create table <table name> (Columnname Datatype,)

Note:

- Keywords are defined as predefined words which has the predefined meaning.
- Any table in the Database may have one or more than one columns.
- The size of VARCHAR() to datatype is 255 Characters.

Example:

Create table emp (eno integer, ename varchar(60), sal integer)

Note:

- Table name in the Database should be unique.(No Duplicate value)
- We are also permitted to create new table.

- **Drop command:**
  This command is used to remove the table from Database. It means that it removes Physical Structure of table as well as records from the table. By default, table has Physical Structure as well as Logical Structure.

  Syntax:

  Drop table <table name>

Example:

Drop table emp

- **Truncate Command:**
  This command is used to delete records from the Database table. This command does not delete the table.

  Syntax:

  Truncate table<table name>

  Example:

  Truncate table emp

- **Alter Command:**
  This Command is used to modify the structure of the Database table. Using this command, we can add new columns to the existing table. Apart from that, we are permitted to change datatype of existing columns of any table. To accomplish the last said, the table should not have any records. It is also used to drop the column of any existing table. We are permitted to add more than one columns at a time and we are also permitted to drop (or) delete only one columns at a time. Still, we can add constraints to existing table using this command. It is possible to remove constraints from table using command.

Syntax:

Alter table <table name>

Example:

- Adding a column alter

  Eg:
  Alter table emp add dno integer

- Adding more than one columns

Eg:
Alter table emp add dno integer, dname varchar (60)

- Modify the data type

  Eg:
  Alter table emp modify column sal float

- Drop a column

  Eg:
  Alter table emp drop column dno

- Adding a constraints

  Eg:

  Alter table emp add constraints p1 NOTNULL (ename)

- Dropping a constraints

  Eg:

  Alter table emp drop constraints p1

Note:

- When we modify the datatype of the existing columns, there should not be any records in Database table. It means that modifying datatype with existing columns when table contain records.
- At a time, we can modify only one columns.
- It is possible to drop any columns from existing table. Using this above query, we can drop only one columns at a time.

- **Rename Command**

  This Command is used to change the name of the existing table.

  Syntax:

  Rename 'old table name ' to 'new table name'

Eg: Rename emp to employee

DATA MANIPULATION LANGUAGE:

It is also referred as DML in short form. These commands are used to perform various operations such as insert, delete, update and select on the Database.

Insert command:

This command is used to insert a record into a Database table. Using this command, we can insert only one record at a time.

Syntax:

Insert into <table name>values (value list)

Example:

- Insert into emp values(1,'ram',1000,10)
- Insert into (eno, ename) emp values(3,'ravi')

Note:

- The number of values passed to the table should be equal to the number of columns in the table.
- Character values should be given within single quotes in parenthesis('')
- We are permitted to pass integer values directly.

Select command:

This command is used to fetch records from Database table. Using this command, we can fetch only one record, set of records, all records, values of one column, value of more than one columns from the Database table.

Syntax:

Select * from<table name> where <condition>

Note:

- Insert command does not support where class and select command have where class.
- Where class is used to define condition. It can have more than one condition.

Example:

- Select one record
  Eg:

  Select * from emp where eno=1

- Select more than one record

Eg:

  Select * from emp where dno=10

- Select All records
  Eg:
  Select * from emp

- Select one columns
  Eg:
  Select ename from emp

- Select more than one columns
  Eg:
  Select eno, ename from emp

Delete command:

This command is used to delete the records from Database table. Using this command, we can delete only one records, set of records, all records from the Database table.

Syntax:

Delete from<table name> where <condition>

Example:

- Delete one record
  Eg:
  Delete from emp where eno=1

- Delete set of records
  Eg:
  Delete from emp where dno=10

- Delete all records
  Eg:
  Delete from emp

Note:
Using this command, delete value of particular columns is not possible.

Update command:

This command is used to update (or) modify the records of the Database table. Using this command, we can update one record, set of records(or) more than one record, all records, value of particular columns, value of more than one columns.

Syntax:

Update<table name> set column_name=<new value> where <condition>

Example:

- Update one record
  Eg:
  Update emp set sal=4000 where eno=3

- Update set of records
  Eg:
  Update emp set sal=sal+1000 where dno=10

- Update all records

  Eg:

  Update emp set sal=sal+1000

TRANSACTIONAL CONTROL LANGUAGE:

Transaction Control Language (TCL) commands is used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions. Here we have three commands they are as follows

- COMMIT
- ROLLBACK
- SAVEPOINT

COMMIT COMMAND:

This command is used to save the transaction permanently in the Database.

ROLLBACK COMMAND:

This command is used to rollback the Database to its former state or original state, before the last commit is used.

Save point command:

This command is used to give name to the transactions. Using this command, we can **name** the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required

Using Save point and Rollback

Following is the table **class**,

id	name
1	Abhi
2	Adam
4	Alex

Lets use some SQL queries on the above table and see the results.

```
INSERT INTO class VALUES(5, 'Rahul');

COMMIT;

UPDATE class SET name = 'Abhijit' WHERE id = '5';

SAVEPOINT A;

INSERT INTO class VALUES(6, 'Chris');

SAVEPOINT B;

INSERT INTO class VALUES(7, 'Bravo');

SAVEPOINT C;

SELECT * FROM class;
```

NOTE: SELECT statement is used to show the data stored in the table.

The resultant table will look like,

id	**name**
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris
7	Bravo

Now let's use the ROLLBACK command to roll back the state of data to the **savepoint B**.

```
ROLLBACK TO B;
```

```
SELECT * FROM class;
```

Now our **class** table will look like,

id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris

Now let's again use the ROLLBACK command to roll back the state of data to the **savepoint A**

```
ROLLBACK TO A;

SELECT * FROM class;
```

Now the table will look like,

id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit

SQL Constraints

SQL constraints are used to eliminate the logical errors from database.

1. **NOT NULL** - Ensures that a column cannot have a NULL value.
2. **UNIQUE** - Ensures that all values in a column are different.
3. **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.
4. **FOREIGN KEY** - Prevents actions that would destroy links between tables.

5. **CHECK** - Ensures that the values in a column satisfy a specific condition.
6. **DEFAULT** - Sets a default value for a column if no value is specified.

   Example:

   CREATE TABLE Colleges (college_id INT PRIMARY KEY, college_code VARCHAR (20) NOT NULL UNIQUE, college_name VARCHAR (50) NOT NULL, college_country VARCHAR (20) DEFAULT 'US', CHECK (college_id > 0));

SQL CLAUSES

WHERE CLAUSE

The SQL **WHERE** clause is used to specify a condition while fetching the data from a single table or by joining with multiple tables. If the given condition is satisfied, then only it returns a specific value from the table. You should use the WHERE clause to filter the records and fetching only the necessary records.

The WHERE clause is not only used in the SELECT statement, but it is also used in the UPDATE, DELETE statement, etc., which we would examine in the subsequent chapters.

Syntax

The basic syntax of the SELECT statement with the WHERE clause is as shown below.

SELECT column1, column2, columnN FROM table_name WHERE [condition]

You can specify a condition using the comparison or logical operators like >, <, =, **LIKE, NOT**, etc. The following examples would make this concept clear.

Example

Consider the CUSTOMERS table having the following records −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

The following code is an example which would fetch the ID, Name and Salary fields from the CUSTOMERS table, where the salary is greater than 2000.

```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY >
2000;
```

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

The following query is an example, which would fetch the ID, Name and Salary fields from the CUSTOMERS table for a customer with the name **Hardik**.

Here, it is important to note that all the strings should be given inside single quotes (''). Whereas, numeric values should be given without any quote as in the above example.

```
SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE NAME =
'Hardik';
```

This would produce the following result –

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  5 | Hardik   |  8500.00 |
+----+----------+----------+
```

GROUP BY CLAUSE

The GROUP BY clause in SQL is used to group rows by one or more columns. It is often used with aggregate functions such as COUNT(), SUM(), AVG(), MIN(), and MAX() to perform calculations on each group of rows[1]. Here is an example of using the GROUP BY clause:

```
CREATE TABLE Orders (order_id int, item varchar(255), quantity int);

INSERT INTO Orders VALUES (1, 'Apple', 5);
INSERT INTO Orders VALUES (2, 'Banana', 3);
INSERT INTO Orders VALUES (3, 'Apple', 2);
INSERT INTO Orders VALUES (4, 'Orange', 4);

SELECT COUNT(order_id), item FROM Orders GROUP BY item;
```

Output:

COUNT(order_id)	**item**
2	Apple
1	Banana
1	Orange

In this example, the GROUP BY clause groups the rows in the Orders table by the value of the item column. The COUNT() function is then used to count the number of rows in each group, which represents the number of orders for each item. Is there anything else you would like to know about the GROUP BY clause

ORDER BY CLAUSE

The ORDER BY clause in SQL is used to sort the result set in ascending or descending order. It is often used with the SELECT statement to sort the result set by one or more columns. Here is an example of using the ORDER BY clause with sample data and output:

```
CREATE TABLE Customers (id int, name varchar(255), country varchar(255));

INSERT INTO Customers VALUES (1, 'John', 'USA');
INSERT INTO Customers VALUES (2, 'Jane', 'USA');
INSERT INTO Customers VALUES (3, 'Bob', 'Canada');
INSERT INTO Customers VALUES (4, 'Alice', 'Canada');

SELECT * FROM Customers ORDER BY country, name;
```

Output:

id	**name**	**country**
4	Alice	Canada
3	Bob	Canada
2	Jane	USA
1	John	USA

In this example, the ORDER BY clause sorts the result set by the value of the country column in ascending order. If there are multiple rows with the same value for the country column, then those rows are further sorted by the value of the name column in ascending order. Is there anything else you would like to know about the ORDER BY clause.

HAVING CLAUSE

The HAVING clause in SQL is used to filter groups of rows based on the result of an aggregate function. It is often used in combination with the GROUP BY clause to filter groups of rows after they have been grouped. Here is an example of using the HAVING clause with sample data and output.

```
CREATE TABLE Orders (order_id int, item varchar(255), quantity int);

INSERT INTO Orders VALUES (1, 'Apple', 5);
INSERT INTO Orders VALUES (2, 'Banana', 3);
INSERT INTO Orders VALUES (3, 'Apple', 2);
INSERT INTO Orders VALUES (4, 'Orange', 4);

SELECT COUNT(order_id), item FROM Orders GROUP BY item HAVING
COUNT(order_id) > 1;
```

Output

COUNT(order_id)	**item**
2	Apple

In this example, the GROUP BY clause groups the rows in the Orders table by the value of the item column. The HAVING clause then filters the groups based on the result of the COUNT() function. Only groups where the count of rows is greater than 1 are included in the result set. Is there anything else you would like to know about the HAVING clause.

LIMIT CLAUSE

The LIMIT clause in SQL is used to restrict the number of rows returned by a query. It is often used in combination with the ORDER BY clause to retrieve a specific range of rows from the result set[1]. The LIMIT clause can be used to implement pagination, where the result set is divided into pages and only one page of results is returned at a time[2].
It is important to note that the LIMIT clause is not supported by all SQL databases. For example, MySQL supports the LIMIT clause, while Oracle uses the FETCH FIRST and OFFSET clauses, and SQL Server uses the TOP clause

```
SELECT * FROM Customers ORDER BY CustomerName LIMIT 5;
```

This query retrieves the first 5 rows from the Customers table, ordered by the value of the CustomerName column. Is there anything else you would like to know about the LIMIT clause.

SQL CLONING OPERATION

There may be a situation when you need an exact copy of a table with the same columns, attributes, indexes, default values and so forth. Instead of spending time on creating the exact same version of an existing table using the CREATE TABLE statement, you can clone the table without disturbing the original table.

Cloning operation in SQL allows the user to create the exact copy of an existing table along with its definition, that is completely independent from the original table. Thus, if any changes are made to the cloned table, they will not be reflected in the original table. This operation comes in handy during testing processes, where there is a need to perform sample testing using the existing database tables.

There are three types of cloning possible using SQL in various RDBMS; they are listed below −

- Simple Cloning − Creates a new table without copying any constraints or indexes etc.
- Shallow Cloning − Creates a new empty table with the same table structure of an existing table.
- Deep Cloning − Creates a new table and copies the table structure and data of an existing table to the new table.

However, there are still no direct ways to fully clone a table in an SQL server. You have to perform a sequence of steps to clone a table including its definition and data.

Simple Cloning

Simple Cloning operation only copies the data from the existing table and copies them into the new table created. To break this down, a new table is created using the CREATE TABLE statement; and the data from the selected columns of an existing table, as a result of SELECT statement, is copied into the new table.

Syntax

Following is the basic syntax to perform simple cloning in MySQL RDBMS −

```
CREATE TABLE new_table SELECT * FROM original_table;
```

Example

In the following example, we are trying to perform simple cloning operation on the given CUSTOMERS table.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | Kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

With the following query, we are trying to create a new table "testCUSTOMERS" by cloning the "CUSTOMERS" table.

```
CREATE TABLE testCUSTOMERS SELECT * FROM CUSTOMERS;
```

Output

The output is displayed as −

```
Query OK, 7 rows affected (0.06 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

Verification

To verify whether the new table contains all the information from the existing table CUSTOMERS, we can use the following SELECT query −

```
SELECT * FROM testCUSTOMERS;
```

The testCUSTOMERS table will be retrieved as shown −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | Kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Shallow Cloning

Shallow Cloning operation only copies the structure of the existing table into the new table created. This operation excludes the data in the existing table so only an empty new table can be created.

Syntax

Following is the basic syntax to perform shallow cloning in MySQL RDBMS −

```
CREATE TABLE new_table LIKE original_table;
```

Example

In the following example, we are trying to perform shallow cloning operation on the given CUSTOMERS table.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | Kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

With the following query, we are trying to create a new table "testCUSTOMERS1" by cloning the "CUSTOMERS" table.

```
CREATE TABLE testCUSTOMERS1 LIKE CUSTOMERS;
```

Output

The output is displayed as −

```
Query OK, 0 rows affected (0.06 sec)
```

Verification

To verify whether the new table is created or not, we can use the following SELECT query −

```
SELECT * FROM testCUSTOMERS1;
```

The testCUSTOMERS table will be retrieved as shown −

```
Empty set (0.00 sec)
```

A new table is thus cloned from the existing table with no data copied into it.

Deep Cloning

Deep Cloning operation is a combination of simple cloning and shallow cloning. It not only copies the structure of the existing table but also its data into the newly created table. Hence, the new table will have all the attributes of the existing table and also its contents.

Since it is a combination of shallow and simple cloning, this type of cloning will have two different queries to be executed: one with CREATE TABLE statement and one with INSERT INTO statement. The CREATE TABLE statement will create the new table by including all the attributes of existing table; and INSERT INTO statement will insert the data from existing table into new table.

Syntax

Following is the basic syntax to perform deep cloning in MySQL RDBMS −

```
CREATE TABLE new_table LIKE original_table;
INSERT INTO new_table SELECT * FROM original_table;
```

Example

In the following example, we are trying to perform deep cloning operation on the given CUSTOMERS table.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | Kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

With the following query, we are trying to create a new table "testCUSTOMERS2" by cloning the "CUSTOMERS" table, i.e. perform shallow cloning first.

```
CREATE TABLE testCUSTOMERS2 LIKE CUSTOMERS;
```

The output is displayed as −

```
Query OK, 0 rows affected (0.06 sec)
```

Now using the following query, we are trying to insert data from "CUSTOMERS" table into new table "testCUSTOMERS2", i.e. performing simple cloning.

```
INSERT INTO testCUSTOMERS2 SELECT * FROM CUSTOMERS;
```

Output

The output is displayed as −

```
Query OK, 7 rows affected (0.01 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

Verification

To verify whether the new table is created or not with all the data present in it, we can use the following SELECT query −

```
SELECT * FROM testCUSTOMERS2;
```

The testCUSTOMERS table will be retrieved as shown −

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | Kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

SQL INJECTIONS

Concept:

SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web pge input.

Example:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

The SQL above is valid and will return ALL rows from the "Users" table, since **OR 1=1** is always TRUE.