SQL:

SQL (Structured Query Language) is used to perform operations on the records stored in the database, such as updating records, inserting records, deleting records, creating and modifying database tables, views, etc.

SQL is not a database system, but it is a query language.

Suppose you want to perform the queries of SQL language on the stored data in the database. You are required to install any database management system in your systems, for example, Oracle

, MySQL

, MongoDB

, PostgreSQL

, SQL Server

, DB2

, etc.

Why SQL?

Nowadays, SQL is widely used in data science and analytics. Following are the reasons which explain why it is widely used:

* The basic use of SQL for data professionals and SQL users is to insert, update, and delete the data from the relational database.
* SQL allows the data professionals and users to retrieve the data from the relational database management systems.
* It also helps them to describe the structured data.
* It allows SQL users to create, drop, and manipulate the database and its tables.
* It also helps in creating the view, stored procedure, and functions in the relational database.
* It allows you to define the data and modify that stored data in the relational database.
* It also allows SQL users to set the permissions or constraints on table columns, views, and stored procedures.

-----------------------------------------------------

Structured Query Language is a standard Database language which is used to create, maintain and retrieve the relational database. Following are some interesting facts about SQL.

* SQL is case insensitive. But it is a recommended practice to use keywords (like SELECT, UPDATE, CREATE, etc) in capital letters and use user defined things (liked table name, column name, etc) in small letters.
* We can write comments in SQL using “–” (double hyphen) at the beginning of any line.
* SQL is the programming language for relational databases (explained below) like MySQL, Oracle, Sybase, SQL Server, Postgre, etc. Other non-relational databases (also called NoSQL) databases like MongoDB, DynamoDB, etc do not use SQL

.

**What is Relational Database?**

Relational database means the data is stored as well as retrieved in the form of relations (tables). Table 1 shows the relational database with only one relation called **STUDENT** which stores **ROLL\_NO**, **NAME**, **ADDRESS**, **PHONE** and **AGE** of students.

**STUDENT**

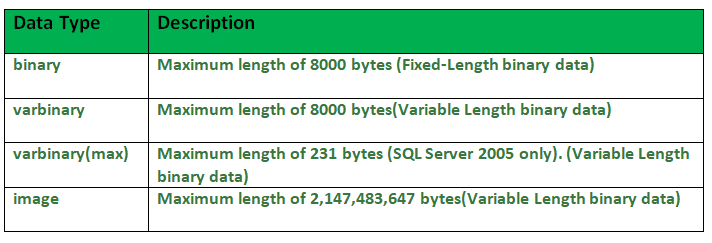
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** | **AGE** |
| 1 | RAM | DELHI | 9455123451 | 18 |
| 2 | RAMESH | GURGAON | 9652431543 | 18 |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |
| 4 | SURESH | DELHI | 9156768971 | 18 |

**Attribute:** Attributes are the properties that define a relation. e.g.; **ROLL\_NO**, **NAME** etc.

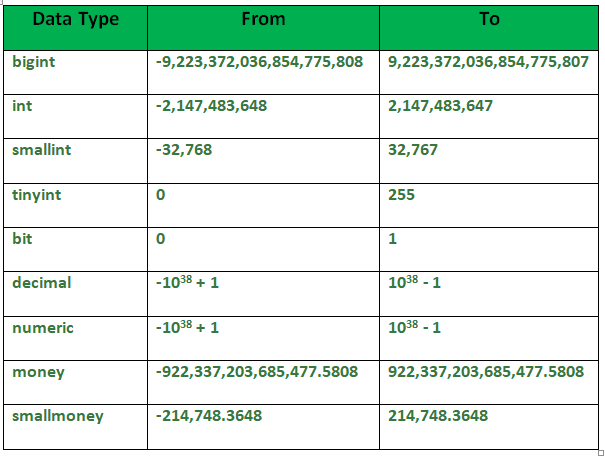
**Tuple:** Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | RAM | DELHI | 9455123451 | 18 |

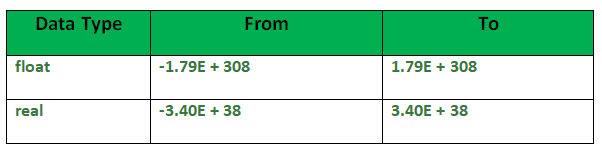
. **Binary Datatypes :**  
There are four subtypes of this datatype which are given below :



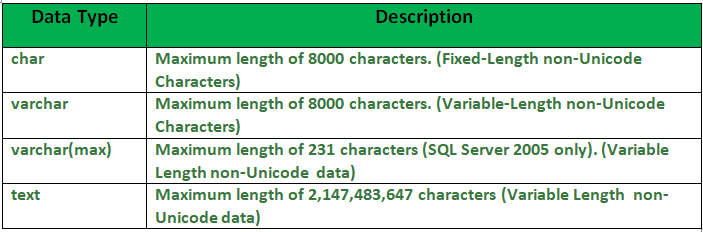
2. **Exact Numeric Datatype :**  
There are nine subtypes which are given below in the table. The table contains the range of data in a particular type.

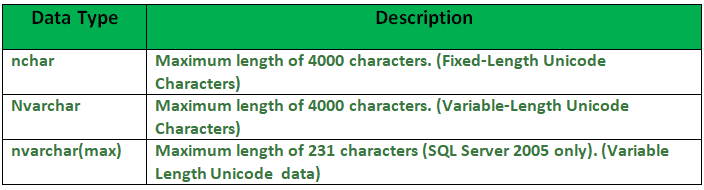


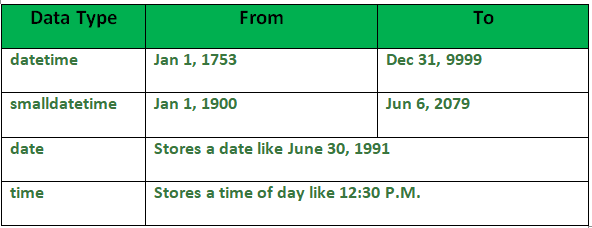
**3. Approximate Numeric Datatype :**  
The subtypes of this datatype are given in the table with the range.



**4. Character String Datatype :**  
The subtypes are given in below table –



**5. Unicode Character String Datatype :**  
The details are given in below table –  


**6. Date and Time Datatype :**  
The details are given in below table.  


1. What is the maximum value that can be stored in NUMERIC(4,2)?

A. 9999.99  
B. 99.9999  
C. 99.99  
D. 9.99

View Answer

Ans : C

Explanation: 99.99 is the maximum value that can be stored in NUMERIC(4,2)

2. Determine data type for the given column? Column Name: Price ; Description: Cost of an item in rupees and paise ; Example: 200.21

A. VARCHAR2(50)  
B. NUMBER  
C. NUMBER(5,2)  
D. NUMBER(6)

View Answer

Ans : C

Explanation: NUMBER(5,2) is the data type for the given column.

3. Determine data type for the given column? Column Name: IFSC\_Code ; Description: A 11 Character alphanumeric code that identifies a bank branch ; Example: SBIN0009044

A. VARCHAR2(50)  
B. NUMBER  
C. Char(11)  
D. NUMBER(11)

View Answer

Ans : C

Explanation: Char(11) is the data type for the given column.

4. Determine the most suitable data type for the given column? Column Name: Profile\_Image ; Description: Image of the employee

A. Clob  
B. Blob  
C. Varchar(100)  
D. None of the above

View Answer

Ans : B

Explanation: Blob is the data type for the given column.

5. The user defined data type can be created using

A. Create datatype  
B. Create data  
C. Create definetype  
D. Create type

View Answer

Ans : D

Explanation: The create type clause can be used to define new types.Syntax : create type Dollars as numeric(12,2) final; .

6. In contemporary databases, the top level of the hierarchy consists of \_\_\_\_\_\_ each of which can contain \_\_\_\_\_

A. Catalogs, schemas  
B. Schemas, catalogs  
C. Alter typeEnvironment, schemas  
D. Schemas, Environment

View Answer

Ans : A

Explanation: In contemporary databases, the top level of the hierarchy consists of Catalogs each of which can contain schemas.

7. Choose the most suitable data type in case multiple data types are possible for the column. Column Name: PIN\_Code ; Description: Six digit numeric PIN code for any address in India ; Example: 560100

A. Integer  
B. VARCHAR2(11)  
C. CHAR(11)  
D. Number(6)

View Answer

Ans : D

Explanation: Number(6) is the most suitable data type in case multiple data types are possible for the column

8. Choose the most suitable data type in case multiple data types are possible for the column. Column Name: Student\_id ; Description: Unique number assigned to every Student ; Example: 100000

A. Integer  
B. VARCHAR2(11)  
C. CHAR(11)  
D. Number(6)

View Answer

Ans : A

Explanation: Integer is the most suitable data type in case multiple data types are possible for the column

9. Choose the most suitable data type in case multiple data types are possible for the column.Column Name: Date\_Of\_Birth ; Description: Date of Birth of the employee ; Example: 1990/01/01

A. Timestamp  
B. VARCHAR2(11)  
C. Blob  
D. Date

View Answer

Ans : D

Explanation: Date is the most suitable data type in case multiple data types are possible for the column

10.  SQL allows comparison operations on the data types i.e.

A. Date  
B. Timestamp  
C. Time  
D. All of the above

View Answer

Ans : D

Explanation: SQL allows comparison operations on the all of the above data types .

DDL:

  [CREATE](https://www.w3schools.in/mysql/php-mysql-create/) - to create a database and its objects like (table, index, views, store procedure, function, and triggers)

 ALTER - alters the structure of the existing database

 DROP - delete objects from the database

 TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed

 COMMENT - add comments to the data dictionary

 RENAME - rename an object

|  |  |
| --- | --- |
| CREATE | Used for creating database objects like a database and a database table. |
| ALTER | Used for modifying and renaming elements of an existing database table. |
| DROP | Used for removing an entire database or a database table. |
| TRUNCATE | Used to remove all the records from a database table. |
| COMMENT | Used to write comments within SQL queries. |

CREATE DATABASE DatabaseName;  
CREATE TABLE TableName (Column1 Datatype1, Column2 Datatype2,…,ColumnNDatatypeN);

For example, we will create a table named ‘Emp’ with some fields and similar data types that are valid in MySQL and respectively a database named ‘EmpDB’ in MySQL server using the queries below:

Example:

CREATE DATABASE EmpDB;

CREATE TABLE empdb.Emp (Emp\_ID INT PRIMARY KEY AUTO\_INCREMENT, Emp\_Name VARCHAR(255), Emp\_City VARCHAR(255), Emp\_AdmDate DATE NOT NULL);

Assignment:

Crate table with student and columns id,name,age,address

Alter:

Syntax:

ALTER TABLE TableName ADD ColumnNameData\_Type;  
ALTER TABLE TableName DROPColumnName;  
ALTER TABLE TableName MODIFY COLUMNColumnNameData\_Type;

Example:

ALTER TABLE empdb.Emp ADD Emp\_Contact INT NOT NULL;

ALTER TABLE empdb.Emp DROP Emp\_Contact;

select \* from empdb.emp;

ALTER TABLE empdb.Emp MODIFY COLUMN Emp\_AdmDate Year;

ALTER TABLE table\_name RENAME COLUMN old\_column\_name TO new\_column\_name;

Assignment:alter student table

Drop:

Syntax:

**Code:**

DROP TABLE TableName;

Example:

DROP Table Emp;

Assignment :drop student table:

Assignment:create another table and insert records and truncate records using below command

Truncate:

Syntax:

TRUNCATE TABLE TableName;

Example:

TRUNCATE TABLE microdb.users;

#### 5. COMMENT Command

/\*selectingall columns from table Employee existing in the database: \*/

SELECT \* FROM microdb.users;

DML Commands:

DML is short name of **Data Manipulation Language** which deals with data manipulation and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE, etc., and it is used to store, modify, retrieve, delete and update data in a database.

* [SELECT](https://www.w3schools.in/mysql/php-mysql-select/) - retrieve data from a database
* [INSERT](https://www.w3schools.in/mysql/php-mysql-insert/) - insert data into a table
* [UPDATE](https://www.w3schools.in/mysql/php-mysql-update/) - updates existing data within a table
* [DELETE](https://www.w3schools.in/mysql/php-mysql-delete/) - Delete all records from a database table

#### 1. INSERT

It is used to insert or add new rows or records in the existing table.

**Syntax:**

Insert into <table\_name> values(<value1>,<value2>,<value3>…….,<valuen>);

Where,

* **table name:** The name of the table In which the data needs to be inserted.
* **values:** values for each column of the table.

create table students (roll\_no int,student\_name varchar(150),course varchar(150));

insert into students values(1,'ashish','java');

Insert into students values(2,’rahul’,’C++’);

select \* from students;

select student\_name from students;

select roll\_no, student\_name, course from students where roll\_no=3;

update:

Update<table\_name> set <column\_name>=value where <condition>;

Below is the query of the update statement:

update students set roll\_no=roll\_no+10 where student\_name='ashish';

update students12 set student\_name='aman' where roll\_no=2;

select \* from students;

4. Delete

Delete statement is used to delete rows of the table based on the specified conditions.

Syntax:

delete from <table\_name> where <condition>;

table\_name: Name of the table from which the data needs to be deleted.

condition: Condition based on which the data is to be deleted.

**.**

select \* from students;

Output:

Output-1.7

delete from students where roll\_no=11;

select \* from students;

Output:

Output-1.8

delete from students where student\_name= 'divya';

Output:

Output-1.9

delete from students12 where course='Arch';

**. DELETE :**   
DELETE is a [DML(Data Manipulation Language)](https://www.geeksforgeeks.org/difference-between-ddl-and-dml-in-dbms/) command and is used when we specify the row(tuple) that we want to remove or delete from the table or relation. The DELETE command can contain a WHERE clause. If the **WHERE** clause is used with the DELETE command then it removes or deletes only those rows(tuple) that satisfy the condition otherwise by default it removes all the tuples(rows) from the table.  Remember that DELETE logs the row deletions.

**Syntax of DELETE command :**

DELETE FROM TableName

WHERE condition;

**2. TRUNCATE :**   
TRUNCATE is a [DDL(Data Definition Language)](https://www.geeksforgeeks.org/difference-between-ddl-and-dml-in-dbms/) command and is used to delete all the rows or tuples from a table. Unlike the DELETE command, the TRUNCATE command does not contain a WHERE clause. In the TRUNCATE command, the transaction log for each deleted data page is not recorded. Unlike the DELETE command, the TRUNCATE command is fast. We cannot roll back the data after using the TRUNCATE command.

**Syntax of TRUNCATE command:-**

TRUNCATE TABLE TableName;

DQL COMMAND:

Select:

**Basic Syntax:**

**SELECT column1,column2 FROM table\_name**

**column1 , column2**: names of the fields of the table

**table\_name:** from where we want to fetch

This query will return all the rows in the table with fields column1 , column2.

* To fetch the entire table or all the fields in the table:

SELECT \* FROM table\_name;

* Query to fetch the fields ROLL\_NO, NAME, AGE from the table Student:

SELECT ROLL\_NO, NAME, AGE FROM Student;

Output:

| **ROLL\_NO** | **NAME** | **Age** |
| --- | --- | --- |
| 1 | Ram | 18 |
| 2 | RAMESH | 18 |
| 3 | SUJIT | 20 |
| 4 | SURESH | 18 |

### **DCL (Data Control Language):**

DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

List of  DCL commands:

* [**GRANT:**](https://www.geeksforgeeks.org/mysql-grant-revoke-privileges/)This commandgives users access privileges to the database.
* [**REVOKE:**](https://www.geeksforgeeks.org/difference-between-grant-and-revoke/)This command withdraws the user’s access privileges given by using the GRANT command.

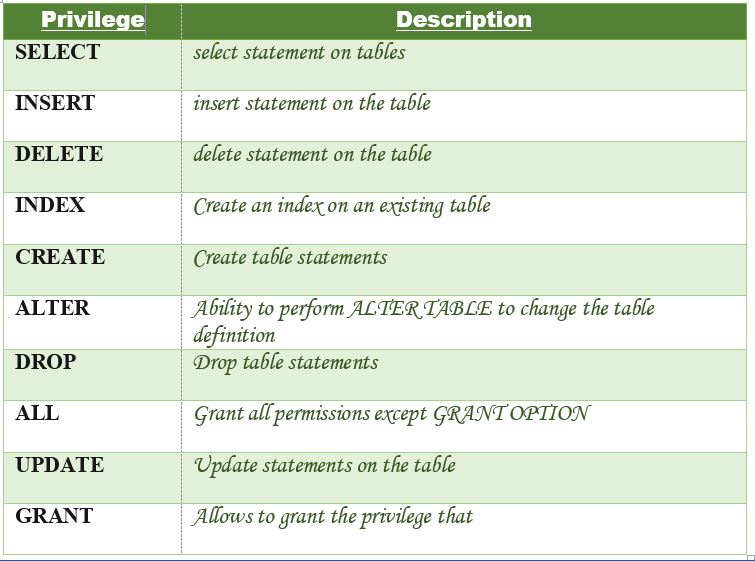
Grant:

**Syntax:**

GRANT privileges\_names ON object TO user;

**Parameters Used**:

* **privileges\_name**: These are the access rights or privileges granted to the user.
* **object:**It is the name of the database object to which permissions are being granted. In the case of granting privileges on a table, this would be the table name.
* **user:**It is the name of the user to whom the privileges would be granted.

**Privileges**:  
The privileges that can be granted to the users are listed below along with description:  


Let us now learn about different ways of granting privileges to the users:

1. **Granting SELECT Privilege to a User in a Table**: To grant Select Privilege to a table named “users” where User Name is Amit, the following GRANT statement should be executed.

GRANT SELECT ON Users TO'Amit'@'localhost;

1. **Granting more than one Privilege to a User in a Table**: To grant multiple Privileges to a user named “Amit” in a table “users”, the following GRANT statement should be executed.

GRANT SELECT, INSERT, DELETE, UPDATE ON Users TO 'Amit'@'localhost;

1. **Granting All the Privilege to a User in a Table**: To Grant all the privileges to a user named “Amit” in a table “users”, the following Grant statement should be executed.

GRANT ALL ON Users TO 'Amit'@'localhost;

1. **Granting a Privilege to all Users in a Table**: To Grant a specific privilege to all the users in a table “users”, the following Grant statement should be executed.

GRANT SELECT ON Users TO '\*'@'localhost;

In the above example the “\*” symbol is used to grant select permission to all the users of the table “users”.

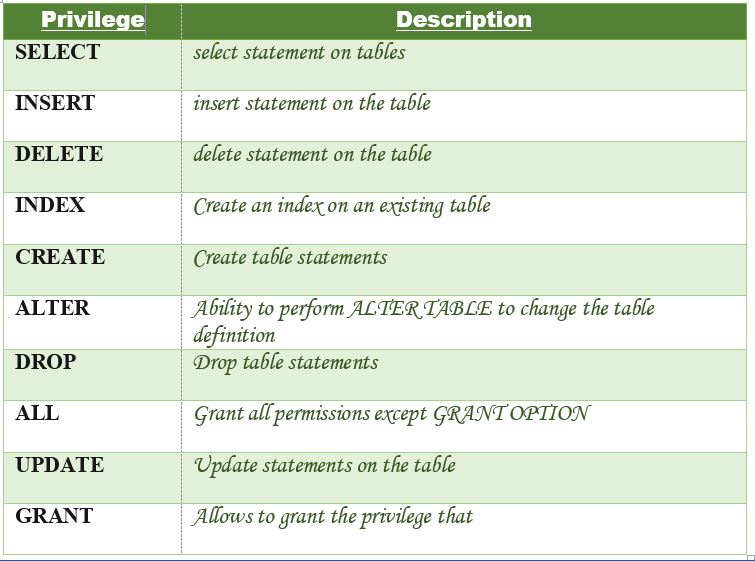
The Revoke statement is used to revoke some or all of the privileges which have been granted to a user in the past.

**Syntax:**

REVOKE privileges ON object FROM user;

**Parameters Used**:

* **object:**It is the name of the database object from which permissions are being revoked. In the case of revoking privileges from a table, this would be the table name.
* **user:**It is the name of the user from whom the privileges are being revoked.

**Privileges**  
Privileges can be of the following values:  


Different ways of revoking privileges from a user:

1. **Revoking SELECT Privilege to a User in a Table**: To revoke Select Privilege to a table named “users” where User Name is Amit, the following revoke statement should be executed.

REVOKE SELECT ON users TO 'Amit'@localhost';

1. **Revoking more than Privilege to a User in a Table**: To revoke multiple Privileges to a user named “Amit” in a table “users”, the following revoke statement should be executed.

REVOKE SELECT, INSERT, DELETE, UPDATE ON Users TO 'Amit'@'localhost;

1. **Revoking All the Privilege to a User in a Table**: To revoke all the privileges to a user named “Amit” in a table “users”, the following revoke statement should be executed.

REVOKE ALL ON Users TO 'Amit'@'localhost;

1. **Revoking a Privilege to all Users in a Table**: To Revoke a specific privilege to all the users in a table “users”, the following revoke statement should be executed.

REVOKE SELECT ON Users TO '\*'@'localhost;

# **TCL Commands in SQL**

* In SQL, TCL stands for **Transaction control language**.
* A single unit of work in a database is formed after the consecutive execution of commands is known as a transaction.
* There are certain commands present in SQL known as TCL commands that help the user manage the transactions that take place in a database.
* **COMMIT. ROLLBACK** and **SAVEPOINT** are the most commonly used TCL commands in SQL.

Now let us take a deeper dive into the TCL commands of SQL with the help of examples. All the queries in the examples will be written using the MySQL database.

### **1. COMMIT**

COMMIT command in SQL is used to save all the transaction-related changes permanently to the disk. Whenever DDL commands such as INSERT, UPDATE and DELETE are used, the changes made by these commands are permanent only after closing the current session. So before closing the session, one can easily roll back the changes made by the DDL commands. Hence, if we want the changes to be saved permanently to the disk without closing the session, we will use the commit command.

**Syntax:**

1. **COMMIT**;

**Example:**

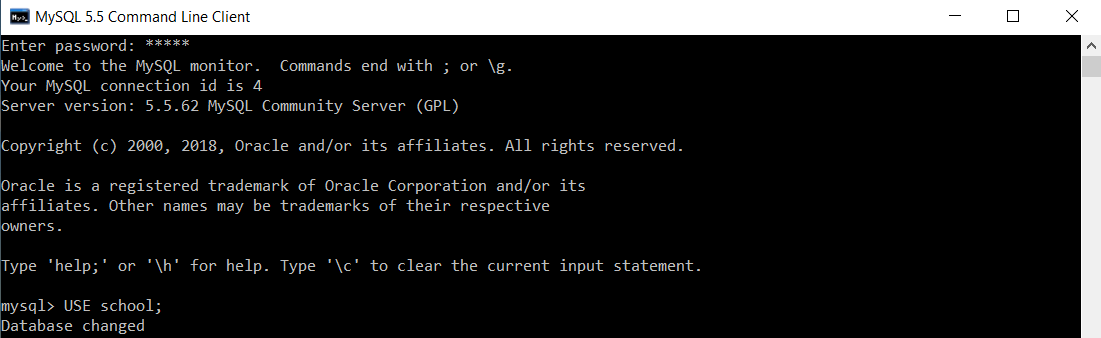
33.9M

735

Exception Handling in Java - Jav

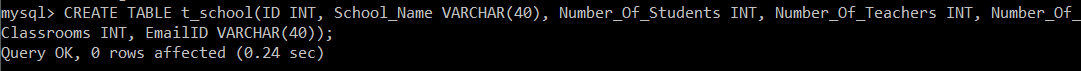
We will select an existing database, i.e., school.

1. mysql> USE school;



To create a table named t\_school, we will execute the following query:

1. mysql> **CREATE** **TABLE** t\_school(ID **INT**, School\_Name **VARCHAR**(40), Number\_Of\_Students **INT**, Number\_Of\_Teachers **INT**, Number\_Of\_Classrooms **INT**, EmailID **VARCHAR**(40));



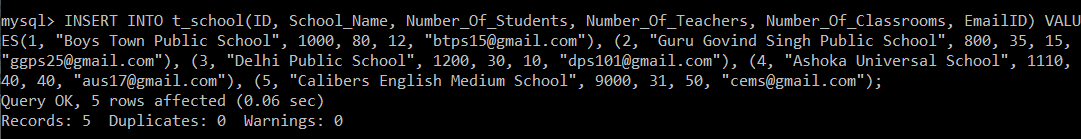
BEGIN / START TRANSACTION command is used to start the transaction.

1. mysql> START **TRANSACTION**;

TCL Commands in SQL

Now, we will execute the following query to insert multiple records at the same time in the t\_school table.

1. mysql> **INSERT** **INTO** t\_school(ID, School\_Name, Number\_Of\_Students, Number\_Of\_Teachers, Number\_Of\_Classrooms, EmailID) **VALUES**(1, "Boys Town Public School", 1000, 80, 12, "btps15@gmail.com"), (2, "Guru Govind Singh Public School", 800, 35, 15, "ggps25@gmail.com"), (3, "Delhi Public School", 1200, 30, 10, "dps101@gmail.com"), (4, "Ashoka Universal School", 1110, 40, 40, "aus17@gmail.com"), (5, "Calibers English Medium School", 9000, 31, 50, "cems@gmail.com");



We will now execute the SELECT query to verify the execution of the INSERT INTO query executed above.

1. mysql> **SELECT** \***FROM** t\_school;

After executing the SELECT query on the t\_school table, you will get the following output:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **School\_Name** | **Number\_Of\_Students** | **Number\_Of\_Teachers** | **Number\_Of\_Classrooms** | **EmailID** |
| 1 | Boys Town Public School | 1000 | 80 | 12 | btps15@gmail.com |
| 2 | Guru Govind Singh Public School | 800 | 35 | 15 | ggps25@gmail.com |
| 3 | Delhi Public School | 1200 | 30 | 10 | dps101@gmail.com |
| 4 | Ashoka Universal School | 1110 | 40 | 40 | aus17@gmail.com |
| 5 | Calibers English Medium School | 9000 | 31 | 50 | cems@gmail.com |

The output of the SELECT query shows that all the records are inserted successfully.

We will execute the COMMIT command to save the results of the operations carried on the t\_school table.

1. mysql> **COMMIT**;

TCL Commands in SQL

Autocommit is by default enabled in MySQL. To turn it off, we will set the value of autocommit as 0.

1. mysql> **SET** autocommit = 0;

TCL Commands in SQL

MySQL, by default, commits every query the user executes. But if the user wishes to commit only the specific queries instead of committing every query, then turning off the autocommit is useful.

### **2. SAVEPOINT**

We can divide the database operations into parts. For example, we can consider all the insert related queries that we will execute consecutively as one part of the transaction and the delete command as the other part of the transaction. Using the SAVEPOINT command in SQL, we can save these different parts of the same transaction using different names. **For example**, we can save all the insert related queries with the savepoint named INS. To save all the insert related queries in one savepoint, we have to execute the SAVEPOINT query followed by the savepoint name after finishing the insert command execution.

**Syntax:**

1. SAVEPOINT savepoint\_name;

### **3. ROLLBACK**

While carrying a transaction, we must create savepoints to save different parts of the transaction. According to the user's changing requirements, he/she can roll back the transaction to different savepoints. Consider a scenario: We have initiated a transaction followed by the table creation and record insertion into the table. After inserting records, we have created a savepoint INS. Then we executed a delete query, but later we thought that mistakenly we had removed the useful record. Therefore in such situations, we have an option of rolling back our transaction. In this case, we have to roll back our transaction using the ROLLBACK command to the savepoint INS, which we have created before executing the DELETE query.

**Syntax:**

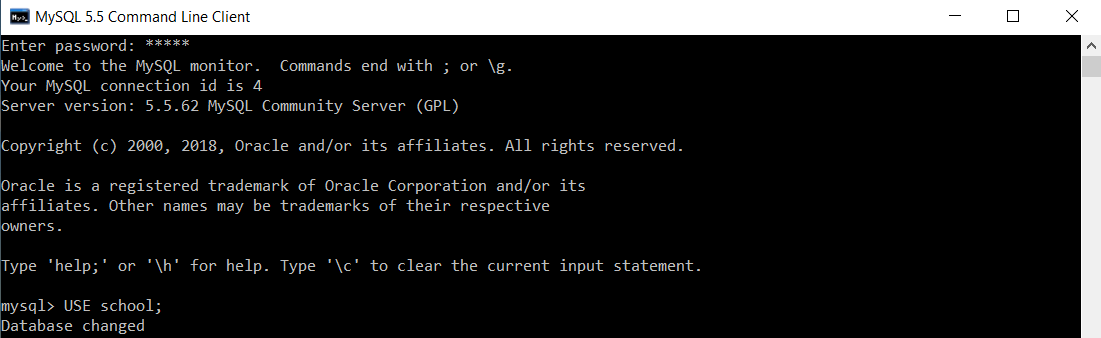
1. **ROLLBACK** **TO** savepoint\_name;

**Examples to understand the SAVEPOINT and ROLLBACK commands:**

**Example 1:**

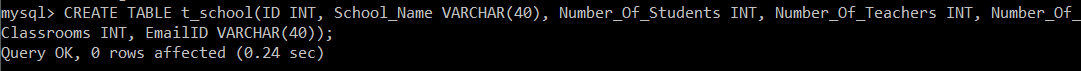
We will select an existing database, i.e., school.

1. mysql> USE school;



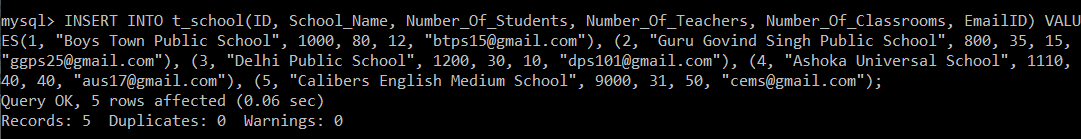
To create a table named t\_school, we will execute the following query:

1. mysql> **CREATE** **TABLE** t\_school(ID **INT**, School\_Name **VARCHAR**(40), Number\_Of\_Students **INT**, Number\_Of\_Teachers **INT**, Number\_Of\_Classrooms **INT**, EmailID **VARCHAR**(40));



Now, we will execute the following query to insert multiple records at the same time in the t\_school table.

1. mysql> **INSERT** **INTO** t\_school(ID, School\_Name, Number\_Of\_Students, Number\_Of\_Teachers, Number\_Of\_Classrooms, EmailID) **VALUES**(1, "Boys Town Public School", 1000, 80, 12, "btps15@gmail.com"), (2, "Guru Govind Singh Public School", 800, 35, 15, "ggps25@gmail.com"), (3, "Delhi Public School", 1200, 30, 10, "dps101@gmail.com"), (4, "Ashoka Universal School", 1110, 40, 40, "aus17@gmail.com"), (5, "Calibers English Medium School", 9000, 31, 50, "cems@gmail.com");



We will now execute the SELECT query to verify the execution of the INSERT INTO query executed above.

1. mysql> **SELECT** \***FROM** t\_school;

After executing the SELECT query on the t\_school table, you will get the following output:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **School\_Name** | **Number\_Of\_Students** | **Number\_Of\_Teachers** | **Number\_Of\_Classrooms** | **EmailID** |
| 1 | Boys Town Public School 1000 | 80 | 12 | btps15@gmail.com |  |
| 2 | Guru Govind Singh Public School | 800 | 35 | 15 | ggps25@gmail.com |
| 3 | Delhi Public School | 1200 | 30 | 10 | dps101@gmail.com |
| 4 | Ashoka Universal School | 1110 | 40 | 40 | aus17@gmail.com |
| 5 | Calibers English Medium School | 9000 | 31 | 50 | cems@gmail.com |

The output of the SELECT query shows that all the records are inserted successfully.

BEGIN / START TRANSACTION command is used to start the transaction.

1. mysql> START **TRANSACTION**;

TCL Commands in SQL

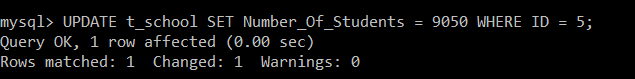
As we know, the SAVEPOINT command in SQL is used to save the different parts of the same transaction using different names. Consider till this point as one part of our transaction. We will save this part using a savepoint named Insertion.

1. mysql> SAVEPOINT Insertion;

TCL Commands in SQL

Now, we will execute the update command on the t\_school table to set the Number\_Of\_Students as 9050 for the record with ID 5.

1. mysql> **UPDATE** t\_school **SET** Number\_Of\_Students = 9050 **WHERE** ID = 5;



To verify that the record with ID 5 now has the Number\_Of\_Students as 9050, we will execute the SELECT query.

1. mysql> **SELECT** \***FROM** t\_school;

After executing the SELECT query on the t\_school table, you will get the following output:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **School\_Name** | **Number\_Of\_Students** | **Number\_Of\_Teachers** | **Number\_Of\_Classrooms** | **EmailID** |
| 1 | Boys Town Public School | 1000 | 80 | 12 | btps15@gmail.com |
| 2 | Guru Govind Singh Public School | 800 | 35 | 15 | ggps25@gmail.com |
| 3 | Delhi Public School | 1200 | 30 | 10 | dps101@gmail.com |
| 4 | Ashoka Universal School | 1110 | 40 | 40 | aus17@gmail.com |
| 5 | Calibers English Medium School | 9050 | 31 | 50 | cems@gmail.com |

The output of the SELECT query shows that the record with ID 5 is updated successfully.

Consider the update operation as one part of our transaction. We will save this part using a savepoint named Updation.

1. mysql> SAVEPOINT Updation;

TCL Commands in SQL

Suddenly, our requirement changed, and we realized that we had updated a record that was not supposed to be. In such a scenario, we need to roll back our transaction to the savepoint, which was created prior to the execution of the UPDATE command.

1. mysql> **ROLLBACK** **TO** Insertion;

TCL Commands in SQL

We didn't need the updation carried on the record. Hence, we have rolled back to the savepoint named Insertion.

For confirming that we have got the same t\_school table that we had before carrying out the updation operation, we will again execute the SELECT query.

1. mysql> **SELECT** \***FROM** t\_school;

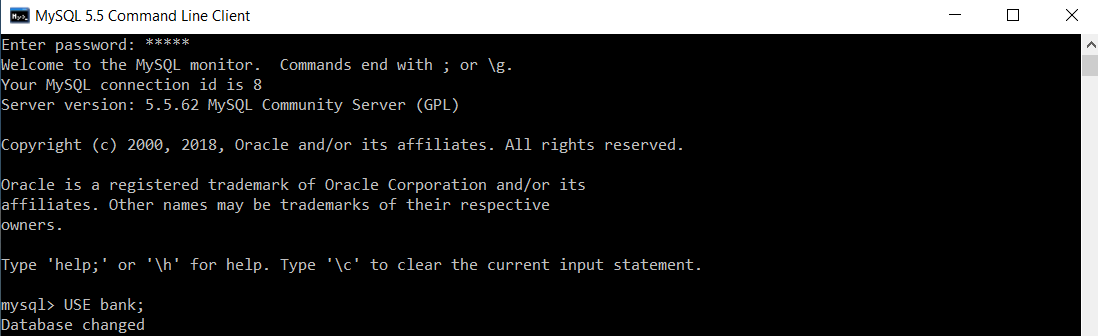
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **School\_Name** | **Number\_Of\_Students** | **Number\_Of\_Teachers** | **Number\_Of\_Classrooms** | **EmailID** |
| 1 | Boys Town Public School | 1000 | 80 | 12 | btps15@gmail.comm |
| 2 | Guru Govind Singh Public School | 800 | 35 | 15 | ggps25@gmail.comm |
| 3 | Delhi Public School | 1200 | 30 | 10 | dps101@gmail.comm |
| 4 | Ashoka Universal School | 1110 | 40 | 40 | aus17@gmail.comm |
| 5 | Calibers English Medium School | 9000 | 31 | 50 | cems@gmail.com |

The SELECT query output confirms that the transaction is now successfully rolled back to the savepoint 'Insertion'.

**Example 2:**

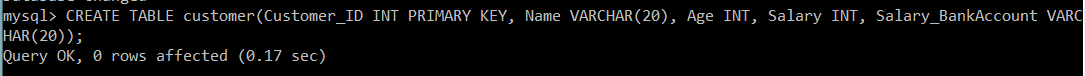
We will select an existing database, i.e., bank.

1. mysql> USE bank;



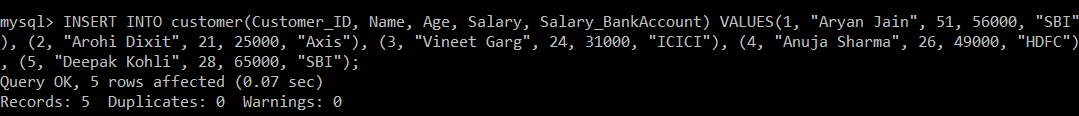
To create a table named customer, we will execute the following query:

1. mysql> **CREATE** **TABLE** customer(Customer\_ID **INT** **PRIMARY** **KEY**, **Name** **VARCHAR**(20), Age **INT**, Salary **INT**, Salary\_BankAccount **VARCHAR**(20));



Now, we will execute the following query to insert multiple records at the same time in the customer table.

1. mysql> **INSERT** **INTO** customer(Customer\_ID, **Name**, Age, Salary, Salary\_BankAccount) **VALUES**(1, "Aryan Jain", 51, 56000, "SBI"), (2, "Arohi Dixit", 21, 25000, "Axis"), (3, "Vineet Garg", 24, 31000, "ICICI"), (4, "Anuja Sharma", 26, 49000, "HDFC"), (5, "Deepak Kohli", 28, 65000, "SBI");



We will now execute the SELECT query to verify the execution of the INSERT INTO query executed above.

1. mysql> **SELECT** \***FROM** customer;

After executing the SELECT query on the t\_school table, you will get the following output:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Customer\_ID** | **Name** | **Age** | **Salary** | **Salary\_BankAccount** |
| 1 | Aryan Jain | 51 | 56000 | SBI |
| 2 | Arohi Dixit | 21 | 25000 | Axis |
| 3 | Vineet Garg | 24 | 31000 | ICICI |
| 4 | Anuja Sharma | 26 | 49000 | HDFC |
| 5 | Deepak Kohli | 28 | 65000 | SBI |

The output of the SELECT query shows that all the records are inserted successfully.

BEGIN / START TRANSACTION command is used to start the transaction.

1. mysql> START **TRANSACTION**;

TCL Commands in SQL

As we know, the SAVEPOINT command in SQL is used to save the different parts of the same transaction using different names. Consider till this point as one part of our transaction. We will save this part using a savepoint named Insertion.

1. mysql> SAVEPOINT Insertion;

TCL Commands in SQL

We will execute the delete command on the customer table to remove the record with ID 5.

1. mysql> **DELETE** **FROM** customer **WHERE** Customer\_ID = 5;

TCL Commands in SQL

We will execute the SELECT query to verify that the record with ID 5 has been removed.

1. mysql> **SELECT** \***FROM** customer;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Customer\_ID** | **Name** | **Age** | **Salary** | **Salary\_BankAccount** |
| 1 | Aryan Jain | 51 | 56000 | SBI |
| 2 | Arohi Dixit | 21 | 25000 | Axis |
| 3 | Vineet Garg | 24 | 31000 | ICICI |
| 4 | Anuja Sharma | 26 | 49000 | HDFC |

The output of the SELECT query shows that the record with ID 5 is removed successfully.

Consider the delete operation as one part of our transaction. We will save this part using a savepoint named Deletion.

1. mysql> SAVEPOINT Deletion;

TCL Commands in SQL

Suddenly, our requirement changed, and we realized that we had deleted a record that was not supposed to be. In such a scenario, we need to roll back our transaction to the savepoint, which was created prior to the execution of the DELETE command.

1. mysql> **ROLLBACK** **TO** Insertion;

We didn't need the deletion carried on the record. Hence, we have rolled back to the savepoint named Insertion.

For confirming that we have got the same customer table that we had before carrying out the deletion operation, we will again execute the SELECT query.

1. mysql> **SELECT** \***FROM** customer;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Customer\_ID** | **Name** | **Age** | **Salary** | **Salary\_BankAccount** |
| 1 | Aryan Jain | 51 | 56000 | SBI |
| 2 | Arohi Dixit | 21 | 25000 | Axis |
| 3 | Vineet Garg | 24 | 31000 | ICICI |
| 4 | Anuja Sharma | 26 | 49000 | HDFC |
| 5 | Deepak Kohli | 28 | 65000 | SBI |

The SELECT query output confirms that the transaction is now successfully rolled back to the savepoint 'Insertion'.

# SQL | Constraints

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

The available constraints in SQL are: 

* **NOT NULL**: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.
* **UNIQUE**: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
* **PRIMARY KEY**: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.
* **FOREIGN KEY**: A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.
* **CHECK**: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.
* **DEFAULT**: This constraint specifies a default value for the column when no value is specified by the user.

**How to specify constraints?**   
We can specify constraints at the time of creating the table using CREATE TABLE statement. We can also specify the constraints after creating a table using ALTER TABLE statement.

**Syntax**:   
Below is the syntax to create constraints using CREATE TABLE statement at the time of creating the table. 

CREATE TABLE sample\_table

(

column1 data\_type(size) constraint\_name,

column2 data\_type(size) constraint\_name,

column3 data\_type(size) constraint\_name,

....

);

**sample\_table**: Name of the table to be created.

**data\_type**: Type of data that can be stored in the field.

**constraint\_name**: Name of the constraint. for example- NOT NULL, UNIQUE, PRIMARY KEY etc.

Let us see each of the constraint in detail. 

**1. NOT NULL –**   
If we specify a field in a table to be NOT NULL. Then the field will never accept null value. That is, you will be not allowed to insert a new row in the table without specifying any value to this field.   
For example, the below query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row. 

CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

ADDRESS varchar(20)

);

**2. UNIQUE** **–**  
This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values. We can have more than one UNIQUE columns in a table.   
For example, the below query creates a table Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID. [Unique constraint in detail.](https://www.geeksforgeeks.org/sql-unique-constraint/) 

CREATE TABLE Student

(

ID int(6) NOT NULL UNIQUE,

NAME varchar(10),

ADDRESS varchar(20)

);

**3. PRIMARY KEY –**   
Primary Key is a field which uniquely identifies each row in the table. If a field in a table as primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field. So, in other words we can say that this is combination of NOT NULL and UNIQUE constraints.   
A table can have only one field as primary key. Below query will create a table named Student and specifies the field ID as primary key. 

CREATE TABLE Student

(

ID int(6) NOT NULL UNIQUE,

NAME varchar(10),

ADDRESS varchar(20),

PRIMARY KEY(ID)

);

1. **FOREIGN KEY –**   
   Foreign Key is a field in a table which uniquely identifies each row of a another table. That is, this field points to primary key of another table. This usually creates a kind of link between the tables.   
   Consider the two tables as shown below:

**CREATE** **TABLE** Persons (

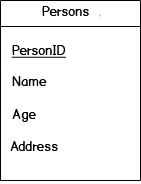
PersonID int AUTO\_INCREMENT **PRIMARY** **KEY**,

Name **VARCHAR**(20) NOT **NULL**,

Age int,

Address **VARCHAR**(100)

);



The column **PersonID** is a primary key of “Persons” table. This means that the values in **PersonID** column uniquely identify the rows in the table. The following statement display all the data in “Persons” table.

> SELECT \* FROM persons;

+----------+-----------+--------+-------------------------------+

| PersonID | Name | age | Address |

+----------+-----------+--------+-------------------------------+

| 101 | Alex | 25 | 819 Saint Francis Way |

| 102 | Emily | 15 | 171 Jarvisville Road Michigan |

| 103 | Jean | 35 | 188 Clay Street Indiana |

| 104 | Bob | 40 | 285 Java Lane Missouri |

+----------+-----------+--------+-------------------------------+

###### **Foreign Key**

A foreign key is a simple mechanism to ensure referential integrity between data in different tables. In other words, the foreign key forces a table to be linked to the data of another table. In the following example, “Orders” table is linked to “Persons” table by **PersonID**.  
   
**Example of Foreign Key:**  
Let’s assume that each person has made orders. To store the orders, you can create a new table named “Orders”:

**CREATE** **TABLE** Orders (

OrderID int AUTO\_INCREMENT **PRIMARY** **KEY**,

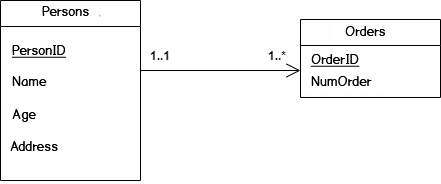
NumOrder int NOT **NULL**,

PersonID int,

**FOREIGN** **KEY** (PersonID) **REFERENCES** Persons(PersonID)

);

The column “PersonID” is a foreign key that refers to the column “PersonID” in the table “Persons”. We used the “Foreign Key” constraint to establish this relationship.



The following statement display all the data in “Orders” table.

> SELECT \* FROM orders;

+----------+-----------+----------+

| OrderID | NumOrder | PersonID |

+----------+-----------+----------+

| 55 | 00001 | 101 |

| 56 | 00002 | 101 |

| 57 | 00003 | 102 |

| 58 | 00004 | 104 |

+----------+-----------+----------+

**(i) CHECK –**   
Using the CHECK constraint we can specify a condition for a field, which should be satisfied at the time of entering values for this field.   
For example, the below query creates a table Student and specifies the condition for the field AGE as (AGE >= 18 ). That is, the user will not be allowed to enter any record in the table with AGE < 18. [Check constraint in detail](https://www.geeksforgeeks.org/sql-check-constraint/) 

CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

AGE int NOT NULL CHECK (AGE >= 18)

);

**(ii) DEFAULT –**   
This constraint is used to provide a default value for the fields. That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.   
For example, the below query will create a table named Student and specify the default value for the field AGE as 18. 

CREATE TABLE Student

(

ID int(6) NOT NULL,

NAME varchar(10) NOT NULL,

AGE int DEFAULT 18

);