What is SQL ?

SQL or Structured Query Language is a standard language for accessing and manipulating databases.

What is PostgreSQL ?

PostgreSQL is an open source relational database management system

What is ACID Properties of a DBMS ?

In the context of transaction processing, the acronym ACID refers to the four key properties of a transaction: atomicity, consistency, isolation, and durability.

Atomicity

All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

Consistency

Data is in a consistent state when a transaction starts and when it ends.For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

Isolation

The intermediate state of a transaction is invisible to other transactions. As a result, transactions that run concurrently appear to be serialized. For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.

Durability

After a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure. For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.

What is the importance of Commit and Rollback in a dbms transaction ?

To assure the ACID properties of a transaction, any changes made to data in the course of a transaction must be committed or rolled back.

When a transaction completes normally, a transaction processing system commits the changes made to the data; that is, it makes them permanent and visible to other transactions.

When a transaction does not complete normally, the system rolls back (or backs out) the changes; that is, it restores the data to its last consistent state.

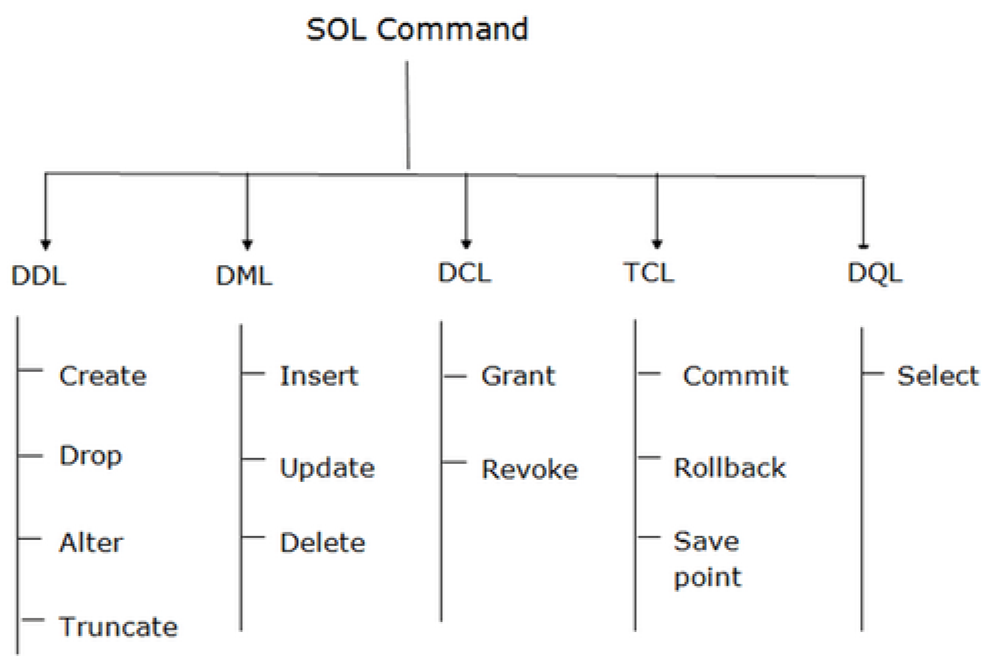
Resources that can be rolled back to their state at the start of a transaction are known as recoverable resources: resources that cannot be rolled back are nonrecoverable.

What are SQL commands and their types ?

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.

- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



1. Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL are auto-committed that means it permanently save all the changes in the database.

2. Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

PosgreSQL Commands :

List all databases

```
bitcanny=# \l
                                     List of databases
         Name         |  Owner   | Encoding | Collate | Ctype |   Access privileges
----------------------+----------+----------+---------+-------+-----------------------
 Training_development | bitcanny | UTF8     | en_IN   | en_IN |
 Training_test        | bitcanny | UTF8     | en_IN   | en_IN |
 bitcanny             | postgres | UTF8     | en_IN   | en_IN |
 postgres             | postgres | UTF8     | en_IN   | en_IN |
 template0            | postgres | UTF8     | en_IN   | en_IN | =c/postgres          +
                      |          |          |         |       | postgres=CTc/postgres
 template1            | postgres | UTF8     | en_IN   | en_IN | =c/postgres          +
                      |          |          |         |       | postgres=CTc/postgres
(6 rows)
```

Create Database

The basic syntax of CREATE DATABASE statement is as follows −
CREATE DATABASE dbname;

```
bitcanny=# CREATE DATABASE testdb;
CREATE DATABASE
bitcanny=#
```

Connect to a Database

```
bitcanny=# \c testdb;
You are now connected to database "testdb" as user "bitcanny".
testdb=#
```

Drop a Database

```
bitcanny=# DROP DATABASE testdb;
DROP DATABASE
bitcanny=#
```

Create a Table

Basic syntax of CREATE TABLE statement is as follows −

```
CREATE TABLE table_name(
  column1 datatype,
  column2 datatype,
  column3 datatype,
  .....
  columnN datatype,
  PRIMARY KEY( one or more columns )
);
```

```
bitcanny=# CREATE DATABASE testdb;
CREATE DATABASE
bitcanny=# \c testdb;
You are now connected to database "testdb" as user "bitcanny".
testdb=# CREATE TABLE students(
testdb(# ID INT PRIMARY KEY NOT NULL,
testdb(# NAME TEXT NOT NULL,
testdb(# CLASS INT NOT NULL,
testdb(# AGE INT NOT NULL );
CREATE TABLE
```

List all the tables connected to the selected database

```
testdb=# \d
          List of relations
 Schema |   Name    | Type  |  Owner
--------+----------+-------+----------
 public | students | table | bitcanny
(1 row)
```

Describe a Certain table in the Database

```
testdb=# \d students
              Table "public.students"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 name   | text    |           | not null |
 class  | integer |           | not null |
 age    | integer |           | not null |
Indexes:
    "students_pkey" PRIMARY KEY, btree (id)
```

Drop a Table

```
testdb=# \d
          List of relations
 Schema |    Name    | Type  |  Owner
--------+------------+-------+----------
 public | students   | table | bitcanny
 public | test_table | table | bitcanny
(2 rows)

testdb=# drop table test_table;
DROP TABLE
testdb=# \d
         List of relations
 Schema |   Name   | Type  |  Owner
--------+----------+-------+----------
 public | students | table | bitcanny
(1 row)
```

Create a schema

A schema is a named collection of tables. A schema can also contain views, indexes, sequences, data types, operators, and functions. Schemas are analogous to directories at the operating system level, except that schemas cannot be nested. PostgreSQL statement CREATE SCHEMA creates a schema.

```
testdb=# create schema myschema;
CREATE SCHEMA
```

Create a table in a schema

```
testdb=# create table myschema.company(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25),
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
CREATE TABLE
```

List all the Tables from a specific schema

```
testdb=# \dt myschema.*
           List of relations
  Schema  |  Name   | Type  |  Owner
----------+---------+-------+----------
 myschema | company | table | bitcanny
(1 row)
```

List all the tables from all schema

```
testdb=# \dt *.*
```

Drop a schema

To drop a schema if it is empty (all objects in it have been dropped), use the command −

DROP SCHEMA myschema;

To drop a schema including all contained objects, use the command −

DROP SCHEMA myschema CASCADE;

```
testdb=# DROP SCHEMA myschema CASCADE;
NOTICE:  drop cascades to table myschema.company
DROP SCHEMA
testdb=# \dt myschema.*
Did not find any relation named "myschema.*".
testdb=#
```

Advantages of using a Schema

- It allows many users to use one database without interfering with each other.
- It organizes database objects into logical groups to make them more manageable.
- Third-party applications can be put into separate schemas so they do not collide with the names of other objects.

Insert Records into a Table

Basic syntax of INSERT INTO statement is as follows −

INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)
VALUES (value1, value2, value3,...valueN);

Here, column1, column2,...columnN are the names of the columns in the table into which you want to insert data.

The target column names can be listed in any order. The values supplied by the VALUES clause or query are associated with the explicit or implicit column list left-to-right.

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. However, make sure the order of the values is in the same order as the columns in the table. The SQL INSERT INTO syntax would be as follows −

INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);

```
testdb=# INSERT INTO students (id,name,class,age) VALUES (1,'Rishi',12,18);
INSERT 0 1
testdb=# INSERT INTO students VALUES (2,'Rishi',12,18);
INSERT 0 1
```

Select Query

The basic syntax of SELECT statement is as follows −

SELECT column1, column2, columnN FROM table_name;

Here, column1, column2...are the fields of a table, whose values you want to fetch. If you want to fetch all the fields available in the field then you can use the following syntax −

SELECT * FROM table_name;

```
testdb=# SELECT name from STUDENTS;
 name
-------
 Rishi
 Rishi
 Rohan
(3 rows)

testdb=# SELECT * from STUDENTS;
 id | name  | class | age
----+-------+-------+-----
  1 | Rishi |    12 |  18
  2 | Rishi |    12 |  18
  3 | Rohan |    11 |  17
(3 rows)
```

Where Clause

```
testdb=# SELECT * from STUDENTS WHERE age>17;
 id | name  | class | age
----+-------+-------+-----
  1 | Rishi |    12 |  18
  2 | Rishi |    12 |  18
(2 rows)
```

```
testdb=# SELECT * from STUDENTS WHERE class=11;
 id | name  | class | age
----+-------+-------+-----
  3 | Rohan |    11 |  17
(1 row)
```

Update Query

The basic syntax of UPDATE query with WHERE clause is as follows −

UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];

```
testdb=# UPDATE students set name='nitish' where id=2;
UPDATE 1
testdb=# select * from students;
 id |  name  | class | age
----+--------+-------+-----
  1 | Rishi  |    12 |  18
  3 | Rohan  |    11 |  17
  2 | nitish |    12 |  18
(3 rows)
```

DELETE Query

The basic syntax of DELETE query with WHERE clause is as follows −

DELETE FROM table_name
WHERE [condition];

```
testdb=# DELETE FROM students WHERE id=2;
DELETE 1
```

If you want to DELETE all the records from COMPANY table, you do not need to use WHERE clause with DELETE queries, which would be as follows −

testdb=# DELETE FROM COMPANY;

Order By

```
testdb=# SELECT * FROM students ORDER BY ID ASC;
 id | name  | class | age
----+-------+-------+-----
  1 | Rishi |    12 |  18
  2 | Rishi |    12 |  18
  3 | Rohan |    11 |  17
(3 rows)

testdb=# SELECT * FROM students ORDER BY ID DESC;
 id | name  | class | age
----+-------+-------+-----
  3 | Rohan |    11 |  17
  2 | Rishi |    12 |  18
  1 | Rishi |    12 |  18
(3 rows)
```

Group By

```
testdb=# SELECT * FROM students ORDER BY ID DESC;
 id | name  | class | age
----+-------+-------+-----
  3 | Rohan |    11 |  17
  2 | Rishi |    12 |  18
  1 | Rishi |    12 |  18
(3 rows)

testdb=# SELECT name from students group by name;
 name
-------
 Rishi
 Rohan
(2 rows)
```

Having Clause

```
testdb=# select name from students group by name having count(name)>1;
 name
-------
 Rishi
(1 row)
```

Alter Table Commands

Add Column:

```
testdb=# ALTER TABLE students ADD city text;
ALTER TABLE
testdb=# \d students
              Table "public.students"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 name   | text    |           | not null |
 class  | integer |           | not null |
 age    | integer |           | not null |
 city   | text    |           |          |
Indexes:
    "students_pkey" PRIMARY KEY, btree (id)
```

Drop Column:

```
testdb=# ALTER TABLE students DROP COLUMN city;
ALTER TABLE
testdb=# \d students
              Table "public.students"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 name   | text    |           | not null |
 class  | integer |           | not null |
 age    | integer |           | not null |
Indexes:
    "students_pkey" PRIMARY KEY, btree (id)
```

Change data type:

```
testdb=# ALTER TABLE students ALTER COLUMN age TYPE text;
ALTER TABLE
testdb=# \d students
              Table "public.students"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 name   | text    |           | not null |
 class  | integer |           | not null |
 age    | text    |           | not null |
Indexes:
    "students_pkey" PRIMARY KEY, btree (id)
```

Truncate Table command

```
testdb=# Select * from students;
 id | name  | class | age | city
----+-------+-------+-----+------
  1 | Rishi |    12 |  18 |
  3 | Rohan |    11 |  17 |
  2 | Rishi |    12 |  18 |
(3 rows)

testdb=# TRUNCATE TABLE students;
TRUNCATE TABLE
testdb=# Select * from students;
 id | name | class | age | city
----+------+-------+-----+------
(0 rows)
```

Transaction Control

COMMIT Transaction:

```
testdb=# BEGIN;
BEGIN
testdb=*# SELECT *from students;
 id | name  | class | age |  city
----+-------+-------+-----+---------
  1 | rishi |    12 |  18 | kolkata
  2 | rohan |    12 |  17 | kolkata
(2 rows)

testdb=*# DELETE FROM students WHERE ID = 2;
DELETE 1
testdb=*# COMMIT;
COMMIT
testdb=#
```

ROLLBACK Transaction:

```
testdb=# SELECT *from students;
 id | name  | class | age |  city
----+-------+-------+-----+---------
  1 | rishi |    12 |  18 | kolkata
(1 row)

testdb=# BEGIN;
BEGIN
testdb=*# INSERT INTO students VALUES (2,'rohan',12,17,'kolkata');
INSERT 0 1
testdb=*# SELECT *from students;
 id | name  | class | age |  city
----+-------+-------+-----+---------
  1 | rishi |    12 |  18 | kolkata
  2 | rohan |    12 |  17 | kolkata
(2 rows)

testdb=*# ROLLBACK;
ROLLBACK
testdb=# SELECT *from students;
 id | name  | class | age |  city
----+-------+-------+-----+---------
  1 | rishi |    12 |  18 | kolkata
(1 row)
```

Constraints

Add Unique Key Constraints:

```
testdb=# \d students
              Table "public.students"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 name   | text    |           | not null |
 class  | integer |           | not null |
 age    | integer |           | not null |
 city   | text    |           |          |
Indexes:
    "students_pkey" PRIMARY KEY, btree (id)

testdb=# ALTER TABLE students ADD CONSTRAINT student_unq UNIQUE(id);
ALTER TABLE
testdb=# \d students
              Table "public.students"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 name   | text    |           | not null |
 class  | integer |           | not null |
 age    | integer |           | not null |
 city   | text    |           |          |
Indexes:
    "students_pkey" PRIMARY KEY, btree (id)
    "student_unq" UNIQUE CONSTRAINT, btree (id)
```

Not Null Constraints:

```
testdb=# \d students
              Table "public.students"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 name   | text    |           | not null |
 class  | integer |           | not null |
 age    | integer |           | not null |
 city   | text    |           |          |
Indexes:
    "students_pkey" PRIMARY KEY, btree (id)
    "student_unq" UNIQUE CONSTRAINT, btree (id)

testdb=# ALTER TABLE students ALTER COLUMN city SET NOT NULL;
ALTER TABLE
testdb=# \d students
              Table "public.students"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 name   | text    |           | not null |
 class  | integer |           | not null |
 age    | integer |           | not null |
 city   | text    |           | not null |
Indexes:
    "students_pkey" PRIMARY KEY, btree (id)
    "student_unq" UNIQUE CONSTRAINT, btree (id)
```

Custom check constraint:

```
testdb=# \d students
                  Table "public.students"
 Column |  Type   | Collation | Nullable |     Default
--------+---------+-----------+----------+-----------------
 id     | integer |           | not null |
 name   | text    |           | not null |
 class  | integer |           | not null |
 age    | integer |           | not null |
 city   | text    |           | not null | 'KOLKATA'::text
Indexes:
    "students_pkey" PRIMARY KEY, btree (id)
    "student_unq" UNIQUE CONSTRAINT, btree (id)

testdb=# ALTER TABLE students ADD CONSTRAINT age_check CHECK(age>4);
ALTER TABLE
testdb=# \d students
                  Table "public.students"
 Column |  Type   | Collation | Nullable |     Default
--------+---------+-----------+----------+-----------------
 id     | integer |           | not null |
 name   | text    |           | not null |
 class  | integer |           | not null |
 age    | integer |           | not null |
 city   | text    |           | not null | 'KOLKATA'::text
Indexes:
    "students_pkey" PRIMARY KEY, btree (id)
    "student_unq" UNIQUE CONSTRAINT, btree (id)
Check constraints:
    "age_check" CHECK (age > 4)
```