SQL CHEAT SHEET

What is SQL?

SQL allows us to interact with the databases and bring out/manipulate data within them. Using SQL, we can create our own databases and then add data into these databases in the form of tables.

The following functionalities can be performed on a database using SQL:

- Create or Delete a Database.
- Create or Alter or Delete some tables in a Database.
- SELECT data from tables.
- INSERT data into tables.
- **UPDATE** data in tables.
- DELETE data from tables.
- Create Views in the database.
- Execute various aggregate functions.

SQL-Create Table:

We use the CREATE command to create a table. The table can be created with the following code:

```
CREATE TABLE student(
   ID INT NOT NULL,
   Name varchar(25),
   Phone varchar(12),
   Class INT
);
```

SQL-Delete Table:

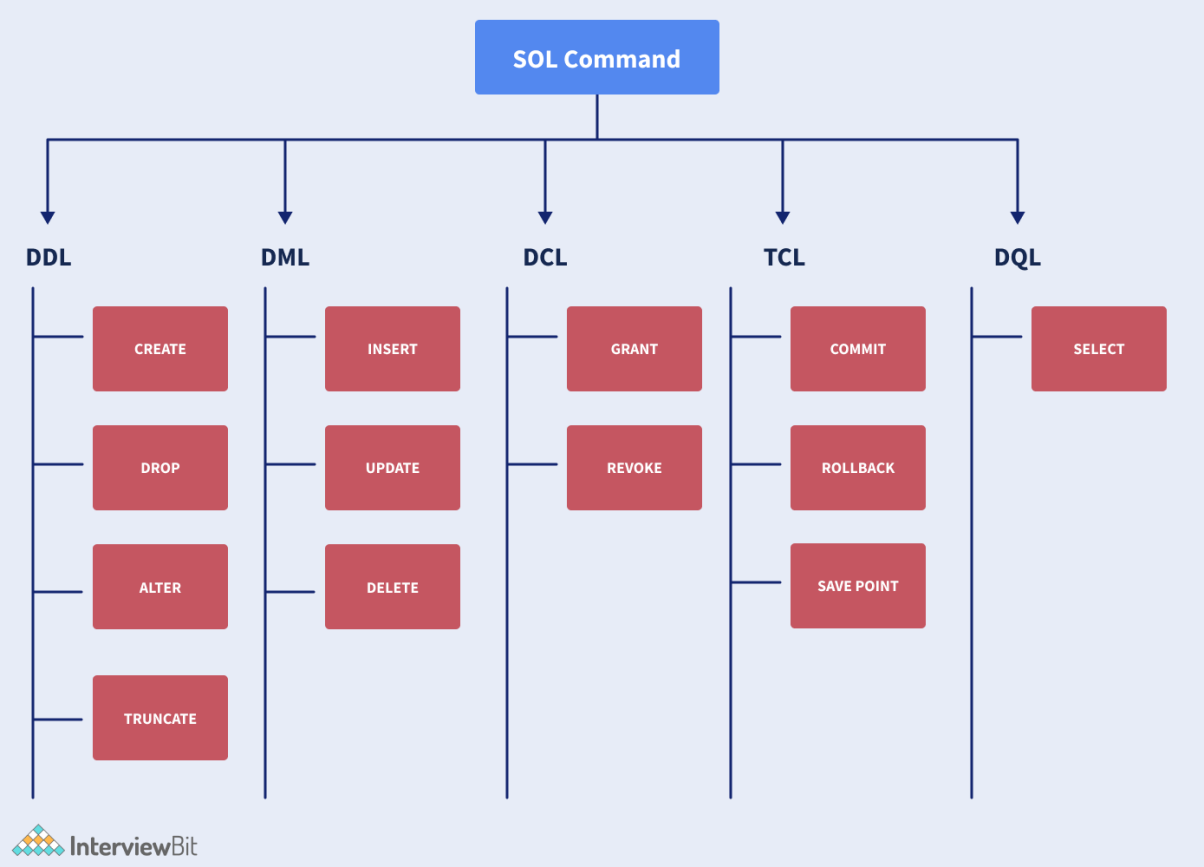
To delete a table from a database, we use the DROP command.

```
DROP TABLE student;
```

SQL Commands

SQL Commands are instructions that are used by the user to communicate with the database, to perform specific tasks, functions and queries of data.

Types of SQL Commands:



The above image broadly shows the different types of SQL commands available in SQL in the form of a chart.

1. Data Definition Language(DDL): It changes a table's structure by adding, deleting and altering its contents. Its changes are auto-committed(all changes are automatically permanently saved in the database). Some commands that are a part of DDL are:

- **CREATE:** Used to create a new table in the database.

Example:

```
CREATE TABLE STUDENT(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
```

- **ALTER:** Used to alter contents of a table by adding some new column or attribute, or changing some existing attribute.

Example:

```
ALTER TABLE STUDENT ADD(ADDRESS VARCHAR2(20));
ALTER TABLE STUDENT MODIFY (ADDRESS VARCHAR2(20));
```

- **DROP:** Used to delete the structure and record stored in the table.

Example:

```
DROP TABLE STUDENT;
```

- **TRUNCATE:** Used to delete all the rows from the table, and free up the space in the table.

Example:

```
TRUNCATE TABLE STUDENT;
```

2. Data Manipulation Language(DML): It is used for modifying a database, and is responsible for any form of change in a database. These commands are not auto-committed, i.e all changes are not automatically saved in the database. Some commands that are a part of DML are:

- **INSERT:** Used to insert data in the row of a table.

Example:

```
INSERT INTO STUDENT (Name, Subject) VALUES ("Shreya", "DSA");
```

In the above example, we insert the values “Shreya” and “DSA” in the columns Name and Subject in the STUDENT table.

- **UPDATE:** Used to update value of a table’s column.

Example:

```
UPDATE STUDENT
SET User_Name = 'Devansh'
WHERE Student_Id = '2'
```

In the above example, we update the name of the student, whose Student_ID is 2, to the User_Name = “Devansh”.

- **DELETE:** Used to delete one or more rows in a table.

Example:

```
DELETE FROM STUDENT
WHERE Name = "Sneha";
```

In the above example, the query deletes the row where the Name of the student is "Sneha" from the STUDENT table.

3. Data Control Language(DCL): These commands are used to grant and take back access/authority (revoke) from any database user. Some commands that are a part of DCL are:

- **Grant:** Used to grant a user access privileges to a database.

Example:

```
GRANT SELECT, UPDATE ON TABLE_1 TO USER_1, USER_2;
```

In the above example, we grant the rights to SELECT and UPDATE data from the table TABLE_1 to users - USER_1 and USER_2.

- **Revoke**: Used to revoke the permissions from an user.

Example:

```
REVOKE SELECT, UPDATE ON TABLE_1 FROM USER_1, USER_2;
```

In the above example we revoke the rights to SELECT and UPDATE data from the table TABLE_1 from the users- USER_1 and USER_2.

4. Transaction Control Language: These commands can be used only with DML commands in conjunction and belong to the category of auto-committed commands. Some commands that are a part of TCL are:

- **COMMIT:** Saves all the transactions made on a database.

Example:

```
DELETE FROM STUDENTS
WHERE AGE = 16;
COMMIT;
```

In the above database, we delete the row where AGE of the students is 16, and then save this change to the database using COMMIT.

- **ROLLBACK:** It is used to undo transactions which are not yet been saved.

Example:

```
DELETE FROM STUDENTS
```

```
WHERE AGE = 16;
ROLLBACK;
```

By using ROLLBACK in the above example, we can undo the deletion we performed in the previous line of code, because the changes are not committed yet.

- **SAVEPOINT:** Used to roll transaction back to a certain point without having to roll back the entirity of the transaction.

Example:

```
SAVEPOINT SAVED;
DELETE FROM STUDENTS
WHERE AGE = 16;
ROLLBACK TO SAVED;
```

In the above example, we have created a savepoint just before performing the delete operation in the table, and then we can return to that savepoint using the ROLLBACK TO command.

5. Data Query Language: It is used to fetch some data from a database. The command belonging to this category is:

- SELECT: It is used to retrieve selected data based on some conditions which are described using the WHERE clause. It is to be noted that the WHERE clause is also optional to be used here and can be used depending on the user's needs.

Example:

```
SELECT Name
FROM Student
WHERE age >= 18;
```

Keys in SQL

A database consists of multiple tables and these tables and their contents are related to each other by some relations/conditions. To identify each row of these tables uniquely, we make use of SQL keys. A SQL key can be a single column or a group of columns used to uniquely identify the rows of a table. SQL keys are a means to ensure that no row will have duplicate values. They are also a means to establish relations between multiple tables in a database.

Types of Keys:

1. Primary Key: They uniquely identify a row in a table.

Properties:

- Only a single primary key for a table. (A special case is a composite key, which can be formed by the composition of 2 or more columns, and act as a single candidate key.)
- The primary key column cannot have any NULL values.
- The primary key must be unique for each row.

Example:

```
CREATE TABLE Student (
   ID int NOT NULL,
   LastName varchar(255) NOT NULL,
   FirstName varchar(255),
   Class int,
   PRIMARY KEY (ID)
);
```

2. Foreign Key: Foreign keys are keys that reference the primary keys of some other table. They establish a relationship between 2 tables and link them up.

Example: In the below example, a table called Orders is created with some given attributes and its Primary Key is declared to be OrderID and Foreign Key is declared to be PersonId referenced from the Person's table. A person's table is assumed to be created beforehand.

```
CREATE TABLE Orders (
   OrderID int NOT NULL,
   OrderNumber int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrderID),
   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

3. **Super Key:** It is a group of single or multiple keys which identifies row of a table.
4. **Candidate Key:** It is a collection of unique attributes that can uniquely identify tuples in a table.
5. **Alternate Key:** It is a column or group of columns that can identify every row in a table uniquely.
6. **Compound Key:** It is a collection of more than one record that can be used to uniquely identify a specific record.
7. **Composite Key:** Collection of more than one column that can uniquely identify rows in a table.
8. **Surrogate Key:** It is an artificial key that aims to uniquely identify each record.

Joins in SQL

Joins are a SQL concept that allows us to fetch data after combining multiple tables of a database.The following are the types of joins in SQL:

1. **INNER JOIN:** Returns any records which have matching values in both tables.

The SQL code will be as follows,

```
SELECT orders.order_id, products.product_name,customers.customer_name,pro
ducts.price
FROM orders
INNER JOIN products ON products.product_id = order.product_id
INNER JOIN customers on customers.customer_id = order.customer_id;
```

2. **NATURAL JOIN:** It is a special type of inner join based on the fact that the column names and datatypes are the same on both tables.

Syntax:

```
Select * from table1 Natural JOIN table2;
```

Example:

```
Select * from Customers Natural JOIN Orders;
```

In the above example, we are merging the Customers and Orders table shown above using a NATURAL JOIN based on the common column customer_id.

3. **RIGHT JOIN:** Returns all of the records from the second table, along with any matching records from the first.

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees
ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

4. **LEFT JOIN:** Returns all of the records from the first table, along with any matching records from the second table.

We will apply Left Join on the above tables, as follows,

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

5. **FULL JOIN:** Returns all records from both tables when there is a match.

Applying Outer Join on the above 2 tables, using the code:

```
SELECT  ID, NAME, AMOUNT, DATE
  FROM CUSTOMERS
  FULL JOIN ORDERS
  ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

Triggers in SQL

SQL codes automatically executed in response to a certain event occurring in a table of a database are called triggers. There cannot be more than 1 trigger with a similar action time and event for one table.

Syntax:

```
Create Trigger Trigger_Name
(Before | After)  [ Insert | Update | Delete]
on [Table_Name]
[ for each row | for each column ]
[ trigger_body ]
```

Example:

```
CREATE TRIGGER trigger1
before INSERT
ON Student
FOR EACH ROW
SET new.total = (new.marks/ 10) * 100;
```

Here, we create a new Trigger called trigger1, just before we perform an INSERT operation on the Student table, we calculate the percentage of the marks for each row.
Some common operations that can be performed on triggers are:

- **DROP:** This operation will drop an already existing trigger from the table.
    - Syntax:

```
DROP TRIGGER trigger name;
```

- **SHOW:** This will display all the triggers that are currently present in the table.
    - Syntax:

```
SHOW TRIGGERS IN database_name;
```