DDL Commands

The CREATE command is used to create database objects, whether that's tables, indexes, views, or databases. The CREATE command syntax varies depending on the type of object being created, but the general structure is similar.

Creating database

```
CREATE DATABASE database_name;
USE database_name;                    /*to use database*/
```

Creating table

```
CREATE TABLE table_name(
  column1 datatype,
  column2 datatype,
  .....
  columnN datatype
);
```

The DROP command removes a database, schema, table, or view from a database. The IF EXISTS option avoids errors if the object to be dropped does not exist.

Drop table

```
DROP TABLE table_name;
```

Drop database

```
DROP DATABASE database_name;
```

The ALTER command is used to modify an existing table's structure, and it can be used to add, drop, or modify columns and also to rename a table.

Alter table

```
ALTER TABLE table_name
ADD column_name  column_definition;

ALTER TABLE table_name
DROP COLUMN column_name;

ALTER TABLE table_name
ALTER COLUMN column_name column_type;

E.g     alter my_table modify column
        first_name varchar(25);
```

The TRUNCATE command is used to remove all rows from a table, and it also resets any auto-incrementing identity values to their starting point.

TRUNCATE

```
truncate table table_name;
```

RENAME

```
RENAME table_name TO table_name_new;
```

COMMENT

```
--  for single line comment,
for multi-line comments
/*
Thi is
Multiline
Comments
*/
```

DML Commands

The INSERT command is used to add new rows to a table in a database. The columns you want to insert data into are specified in the column list, while the values you want to insert are specified in the VALUES clause.

INSERT

```
INSERT INTO table_name
(attribute1, attribute2, ...)
VALUES(val1, val2, ...);
```

The UPDATE command is used to modify existing data in a table. It updates the values of one or more columns in one or more table rows that satisfy the specified condition.

UPDATE

```
UPDATE table_name
SET column1 = val1,
column2 = val2,
...
WHERE CLAUSE;
```

```
EXAMPLE OF MASS UPDATE      update my_table
                            set age = age + 1;
```

The DELETE command removes one or more rows from a table that satisfy the specified condition.
DELETE

```
DELETE FROM table_name
WHERE CLAUSE;
```

Data Query Language – SELECT

The SELECT command retrieves data from one or more tables in a database. It allows you to specify which columns you want to retrieve data from and optionally filter the results using a WHERE clause.

```
SELECT column1 as c1
column2 as c2
```

```
...
from table_name;

EXAMPLE     select age as myAge
            from my_table;

select * from my_table;

select * from
my_table where age>20;

select * from my_table
where first_name like "myName_";

select * from my_table
where roll_no like "%0";
```

DCL

The GRANT command grants specific permissions to a user or role on a particular database object, such as a table, view, or stored procedure.

GRANT

```
GRANT PRIVILEGES
ON OBJECT
TO USER;

Example     GRANT SELECT
            ON tableName
            TO 'userName'@'localhost';

            GRANT SELECT, INSERT, DELETE, UPDATE
            ON tableName
            TO 'userName'@'localhost';

            GRANT ALL
            ON tableName
            TO 'userName'@'localhost';
```

The REVOKE command removes permissions from a user or role in a database. It is often used in conjunction with the GRANT command, which grants permissions to users or roles.

REVOKE

```
REVOKE privileges
ON object
FROM user;

Example
        REVOKE DELETE
```

```
ON tableName
FROM userName;

REVOKE ALL
ON tableName
FROM userName;
```

The COMMIT command permanently saves changes to a database that were made in a transaction.

COMMIT

```
commit;
```

The ROLLBACK command can be used to undo changes made within a transaction, thus restoring the database to its previous state.

ROLLBACK

```
ROLLBACK;
```

The SAVEPOINT command creates a named point within a transaction that allows for a partial rollback in case of an error.

SAVEPOINT

```
SAVEPOINT some_name;
```

DISTINCT

The DISTINCT keyword selects unique values from a column in a table. The example below returns all city values from the customers table, but the DISTINCT keywords ensures there are no duplicate values.

Example

```
SELECT DISTINCT city FROM customers;
```

ORDER BY

The ORDER BY keyword sorts query results in descending or ascending order based on one or more columns. The example below returns all values from the orders table, sorted by order_date in descending order.

Example:

```
SELECT * FROM orders ORDER BY order_date DESC;
```

GROUP BY

The GROUP BY keyword is used to group rows with the same values in one or more columns. The example below returns the number of customers in each region via the COUNT keyword (another bonus keyword for you there!).

Example:

```
SELECT region, COUNT(*) FROM customers GROUP BY region;
```

JOIN

The JOIN keyword combines the rows from two or more tables based on a related column that they share. The example below returns a list of customer_name and order_date values for all orders in the orders table that have a matching customer_id in the customers table.
Example:

```
SELECT customers.customer_name, orders.order_date
FROM customers
JOIN orders ON customers.customer_id = orders.customer_id;
```

WHERE
The WHERE keyword filters results from a query based on one or more conditions. The example below returns everything from the products table when the unit_price exceeds 10.
Example:

```
SELECT * FROM products WHERE unit_price > 10;
```