DATABASE:

CREATE DATABASE database_name;
Creates a new database.

USE database_name;
Uses the specified database.

DDL COMMANDS:

```
CREATE TABLE table_name (
       column1 datatype,
       column2 datatype,
       column3 datatype);
```
Used to create a new table in a database.

ALTER TABLE table_name ADD column_name datatype;
Used to add a new column to an existing table.

ALTER TABLE table_name DROP COLUMN column_name ;
Used to drop a column from an existing table.

ALTER TABLE table_name RENAME TO table_newname ;
Used to change the name of an existing table.

ALTER TABLE table_name RENAME col_name TO col_newname;
Used to rename the column of an existing table.

DROP TABLE table_name;
Used to delete both records and structure of a table.

TRUNCATE TABLE table_name;
Used to delete the records but not the structure of a table.

DML COMMANDS:

```
INSERT INTO table_name
VALUES (value1, value2);

INSERT INTO table_name (column1, column2)
VALUES (value1, value2);
```

Used to add a new record (row) to a table.

```
DELETE FROM table_name
WHERE some_column = some_value;
```

Used to delete records (rows) from a table.

```
UPDATE table_name
SET
     column1 = value1,
     column2 = value2
WHERE some_column = some_value;
```

Used to modify records (rows) of a table.

DCL COMMANDS:

GRANT SELECT, UPDATE ON TABLE_1 TO USER_1, USER_2;
Used to grant access privileges of a database to a user.

REVOKE SELECT, UPDATE ON TABLE_1 FROM USER_1, USER_2;
Used to revoke the granted permissions from a user.

TCL COMMANDS:

COMMIT; - Used to save all the transactions made on a database.
ROLLBACK; - Used to undo the transactions which have not yet been saved.

SAVEPOINT savepoint_name; - Used to roll the transaction back to a certain point without having to roll back the entirety of the transaction.

DQL COMMANDS:

SELECT * FROM table_name;
Retrieve data from all the columns in the table.

SELECT col1,col2.. FROM table_name;
Retrieve data from the specified columns in a table

SELECT col1,col2.. FROM table_name WHERE condition;
Used to filter the records based on a particular condition.

SQL Constraints:

NOT NULL: Specifies that this column cannot store a NULL value.
UNIQUE: Specifies that this column can only have Unique values.
Primary Key: It is a field using which it is possible to uniquely identify each row in a table.
Foreign Key: It is a field using which it is possible to uniquely identify each row in some other table.
CHECK: It validates if all values in a column satisfy a specific condition or not
DEFAULT: It sets a default value for a column when no value is specified.

Operators:

AND - The AND operator allows multiple conditions to be combined. Records must match both conditions.

OR - The OR operator allows multiple conditions to be combined. Records must match either one of the conditions.

NOT - The NOT operator allows the negation of the condition.

IN - The IN operator is used to check whether a specified value is contained within a set of values or elements.

BETWEEN - The BETWEEN operator can be used to filter by a range of values.

LIKE - The LIKE operator can be used to match a specified pattern.

% Wildcard - The % wildcard can be used in a LIKE operator pattern to match zero or more unspecified character(s).

_ Wildcard - The _ wildcard can be used in a LIKE operator pattern to match any single unspecified character.

AS - Columns or Tables can be aliased using the AS clause.

ALL - It compares a value to all the values in another set.

ANY - It compares the values in the list based on a specified condition.

EXIST - It is used to search for the presence of a row in a table.

IS NULL - It checks if a value or column contains a NULL value.

IFNULL(): It returns the first expression if it is not NULL; otherwise, it returns the second expression.

COALESCE(): It returns the first non-NULL value from a list of expressions.

Numeric Functions:

ROUND() - It is used to round a numeric value to a specified number of decimal places.
Syntax: ROUND(numeric_expression, decimal_places)

CEIL() - It returns the smallest integer greater than or equal to the specified numeric expression.
Syntax: CEIL(numeric_expression)

FLOOR() - It returns the largest integer less than or equal to the specified numeric expression.
Syntax: FLOOR(numeric_expression)

ABS() - It returns the absolute (positive) value of a numeric expression.
Syntax: ABS(numeric_expression)

SQRT() - It returns the square root of a numeric expression.
Syntax: ABS(numeric_expression)

String Functions:

CONCAT() - It is used to concatenate two or more strings together.
Syntax: CONCAT(string1, string2, ...)

UPPER()/LOWER() - It is used to convert a string to uppercase or lowercase, respectively.
Syntax:

- UPPER(string_expression)
- LOWER(string_expression)

LENGTH() - It returns the length of a string.
Syntax: LENGTH(string_expression)

SUBSTR() - It is used to extract a substring from a string.
Syntax: SUBSTRING(string_expression, position, length)

LEFT() / RIGHT() - It extracts a specified number of characters from the left or right side of a string.

Syntax:

- LEFT(string_expression, length)
- RIGHT(string_expression, length)

TRIM() - It removes leading and trailing spaces from a string.
Syntax: TRIM(string_expression)

REPLACE() - It replaces all occurrences of a specified substring within a string with another substring.
Syntax: REPLACE(original_string, old_substring, new_substring)

CASE:

```
SELECT column_name,
  CASE
    WHEN condition THEN 'output'
    WHEN condition THEN 'output'
    .
    .
    ELSE 'output'
  END AS new_colname
FROM table_name;
```

SUBQUERY:

```
SELECT COUNT(*) from(SELECT col1, COUNT(col2) from table_name
GROUP BY col1) AS inner_query WHERE condition;
```

The inner query is executed first, and then the result is passed to the outer query, which is executed next.

AGGREGATE FUNCTIONS:

AVG() - Returns the average value of a list
SUM() - Returns the sum of values in a list.
MIN() - Returns the minimum value of a list.

MAX() - Returns the maximum value of a list.
COUNT() - Returns the number of elements in a list.

1. COUNT(*) - When * is used as an argument, it simply counts the total number of rows including the NULLs.

2. COUNT(1)- With COUNT(1), there is a misconception that it counts records from the first column. What COUNT(1) does is that it replaces all the records you get from query results with the value 1 and then counts the rows meaning it even replaces a NULL with 1 meaning it considers NULLs while counting.

3. COUNT(column_name)- When a column name is used as an argument, it simply counts the total number of rows excluding the NULLs meaning it will not consider NULLs.

4. COUNT() function with the **DISTINCT** clause eliminates the repetitive appearance of the same data. The DISTINCT can come only once in a given select statement.

```
COUNT(DISTINCT expr,[expr...])
```

Querying Data:

SELECT DISTINCT(column_name) FROM table_name;
Retrieves unique values from the specified column in the table.

SELECT * FROM table_name LIMIT n;
Limits the result set to the specified number of rows.

SELECT col1, col2 FROM table_name ORDER BY col1 ASC [DESC];
Sorts the result set in ascending or descending order based on the values in col1.

SELECT col1, col2 FROM table_name ORDER BY col1 LIMIT n OFFSET offset;
Skips the specified number of rows (offset) and returns the next n rows based on the LIMIT.

SELECT col1, aggregate(col2) FROM table_name GROUP BY col1;
Group rows by unique values in col1 and apply an aggregate function to col2 within each group.

SELECT col1, aggregate(col2) FROM table_name GROUP BY col1 HAVING condition;
Group rows by unique values col1 and filter the groups using the HAVING clause based on a specified condition.

JOINS:

SELECT col1, col2 FROM table_name t1 INNER JOIN table_name t2 ON condition;
Inner join of two tables, t1 and t2

SELECT col1, col2 FROM table_name t1 LEFT JOIN table_name t2 ON condition;
Left join of two tables, t1 and t2

SELECT col1, col2 FROM table_name t1 RIGHT JOIN table_name t2 ON condition;
Right join of two tables, t1 and t2

SELECT col1, col2 FROM table_name t1 FULL OUTER JOIN table_name t2 ON condition;
Full Outer join of two tables, t1 and t2

SELECT col1, col2 FROM table_name t1 CROSS JOIN table_name t2 ON condition;
Produces a Cartesian Product of rows in two tables, t1 and t2

SELECT col1, col2 FROM table_name
UNION DISTINCT
SELECT col1, col2 FROM table_name;
Combine rows from two queries without any duplicates.

SELECT col1, col2 FROM table_name
UNION ALL

SELECT col1, col2 FROM table_name;
Combine rows from two queries with duplicates.

SELECT col1, col2 FROM table_name
INTERSECT
SELECT col1, col2 FROM table_name;
Return rows that are common among two queries.

SELECT col1, col2 FROM table_name
MINUS
SELECT col1, col2 FROM table_name;
Returns the values from the first table after removing the values from the second table.

Window Functions:

OVER()- Used to define a window or a subset of rows within the result set for analytical functions.

PARTITION BY - divides the result set into partitions or groups based on the specified column(s) or expressions.

ROW_NUMBER() - Assigns a unique sequential row number to each row in the result set based on the specified column(s).

RANK() - Assigns a rank to each row based on the specified column, and if there are duplicate values, it assigns the same rank to those rows, potentially skipping the next rank.

DENSE_RANK() - Assigns a rank to each row based on the specified column, and if there are duplicate values, it assigns the same rank to those rows, but ranks are not skipped.

LAG()- Retrieves the value from the previous row within the specified window frame. The first row's value is NULL as there is no previous row.

LEAD() - Retrieves the value from the next row within the specified window frame. The last row's value is NULL as there is no next row.

FIRST_VALUE() - Provides the value in the first row within the specified window frame.

LAST_VALUE() - Provides the value in the last row within the specified window frame.

NTH_VALUE() - Provides the value in the nth row within the specified window frame.

NTILE() - Divides the result set into 'n' equal-sized buckets.

VIEWS:

SELECT VIEW view_name AS SELECT * FROM table_name;
It creates a simple view.

SELECT VIEW view_name AS SELECT col1, col2 FROM table_name t1 INNER JOIN table_name t2 ON condition;
It creates a complex view.

DROP VIEW view_name;
Delete a view

Order of Execution:

All query elements are processed in a very strict order:

- **FROM** - The database gets the data from tables in the FROM clause and if necessary, performs the JOINs.
- **JOIN** - Depending on the type of JOIN used in the query and the conditions specified for joining the tables in the **ON** clause, the database engine matches rows from the virtual table created in the FROM clause.
- **WHERE** - After the JOIN operation, the data is filtered based on the conditions specified in the WHERE clause. Rows that do not meet the criteria are excluded.
- **GROUP BY** - If the query includes a GROUP BY clause, the rows are grouped based on the specified columns.

- **Aggregate Functions** - The aggregate functions are applied to the groups created in the GROUP BY clause.
- **HAVING** - The HAVING clause filters the groups of rows based on the specified conditions
- **Window Functions**
- **SELECT** - After grouping and filtering are done, the SELECT statement determines which columns to include in the final result set.
- **DISTINCT** - The DISTINCT keyword is applied within the SELECT clause to ensure that unique values are returned for the specified columns.
- **UNION/INTERSECT/MINUS** - After generating the result sets from individual SELECT queries, the database applies Set Operations.
- **ORDER BY** - It allows you to sort the result set based on one or more columns in ascending or descending order.
- **OFFSET** - The specified number of rows is skipped from the beginning of the result set.
- **LIMIT** - After skipping the rows, the LIMIT clause is applied to restrict the number of rows returned.

SQL Partitioning:

It is a database optimization technique that involves dividing large tables into smaller, more manageable pieces called partitions.

RANGE Partitioning - Rows are distributed across partitions based on a specified range of values for a chosen column.

Syntax:

```
CREATE TABLE table_name (
  column1 datatype,
  ... )
PARTITION BY RANGE (column_partition)
(
  PARTITION partition_name1 VALUES LESS THAN (value1),
  ... );
```

LIST Partitioning - Rows are distributed across partitions based on a discrete list of values for a chosen column.

Syntax:

```
CREATE TABLE table_name (
  column1 datatype,
  ... )
PARTITION BY LIST (column_partition)
(
  PARTITION partition_name1 VALUES IN (value1, value2,
...),
   ... );
```

HASH Partitioning - A hash function is used to distribute rows evenly across partitions.

Syntax:

```
CREATE TABLE table_name (
  column1 datatype,
  ... )
PARTITION BY HASH (column_partition)
PARTITIONS no_of_partitions;
```

SQL Indexes:

Indexes are database structures that enhance the speed of data retrieval operations on a database table.

Create an index on one or more columns of a table.

Syntax:

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

Remove an existing index from a table.

Syntax:

```
DROP INDEX index_name
ON table_name;
```