Functions

SQL provides a wide variety of functions, and the exact functions available depend on the database system (e.g., MySQL, PostgreSQL, SQL Server, Oracle, or BigQuery). Below is a categorized list of commonly used SQL functions across most systems. For system-specific functions, consult the documentation of the database you're using.

---

1. Aggregate Functions

Used to perform calculations on a set of values and return a single value.

- `COUNT()` - Counts rows.
- `SUM()` - Returns the sum of a column.
- `AVG()` - Calculates the average value.
- `MIN()` - Returns the smallest value.
- `MAX()` - Returns the largest value.
- `GROUP_CONCAT()` (MySQL) - Concatenates values into a single string.

---

2. String Functions

Used to manipulate or analyze strings.

- `UPPER()` / `LOWER()` - Converts text to uppercase/lowercase.
- `TRIM()` - Removes leading/trailing spaces.
- `SUBSTRING()` / `SUBSTR()` - Extracts part of a string.
- `CONCAT()` - Concatenates strings.
- `LENGTH()` / `CHAR_LENGTH()` - Returns the length of a string.
- `REPLACE()` - Replaces occurrences of a substring.
- `LEFT()` / `RIGHT()` - Extracts characters from the left/right of a string.
- `POSITION()` / `LOCATE()` - Finds the position of a substring.
- `LPAD()` / `RPAD()` - Pads a string to a specific length.

- `FORMAT()` (MySQL) - Formats numbers as strings.

---

3. Date and Time Functions

Used to work with date and time values.

- `CURRENT_DATE()` / `NOW()` - Returns the current date/time.
- `DATEADD()` / `DATE_SUB()` - Adds or subtracts a time interval.
- `DATEDIFF()` - Calculates the difference between two dates.
- `EXTRACT()` - Extracts parts of a date (e.g., year, month, day).
- `FORMAT_DATE()` (BigQuery) - Formats a date.
- `TIMESTAMP()` - Converts a value into a timestamp.
- `DATE_TRUNC()` - Truncates a date to a specific part (e.g., month).
- `TO_CHAR()` / `TO_DATE()` (PostgreSQL/Oracle) - Converts between date and string formats.

---

4. Mathematical Functions

Used for mathematical operations.

- `ABS()` - Returns the absolute value.
- `ROUND()` - Rounds a number to a specific decimal place.
- `CEIL()` / `FLOOR()` - Rounds a number up/down.
- `POWER()` - Raises a number to a power.
- `SQRT()` - Returns the square root.
- `MOD()` - Calculates the remainder of a division.
- `LOG()` / `LN()` - Returns the logarithm.
- `EXP()` - Returns `e` raised to the power of a number.
- `PI()` - Returns the value of π.

---

5. Conditional Functions

Used for conditional logic in queries.

- `CASE` - Conditional logic.

```
CASE WHEN condition THEN result ELSE alternative END
```

- `IF()` (MySQL) - Performs conditional logic.
- `NULLIF()` - Returns `NULL` if two values are equal.
- `COALESCE()` - Returns the first non-null value in a list.
- `ISNULL()` (SQL Server) / `IFNULL()` (MySQL) - Returns a default value if the expression is `NULL`.

---

6. JSON Functions (Database-Specific)

Used for working with JSON data.

- `JSON_EXTRACT()` (BigQuery/MySQL) - Extracts a value from JSON.
- `JSON_VALUE()` (SQL Server) - Extracts a scalar value from JSON.
- `JSON_OBJECT()` (MySQL) - Creates a JSON object.
- `JSON_ARRAY()` (MySQL) - Creates a JSON array.
- `JSON_QUERY()` (SQL Server) - Extracts an object or array from JSON.

---

7. Window (Analytic) Functions

Used for advanced calculations over a window of rows.

- `ROW_NUMBER()` - Assigns a unique number to each row.
- `RANK()` / `DENSE_RANK()` - Assigns a rank to rows.
- `NTILE()` - Divides rows into a specified number of groups.
- `LEAD()` / `LAG()` - Accesses data from subsequent/previous rows.
- `FIRST_VALUE()` / `LAST_VALUE()` - Returns the first/last value in a partition.
- `SUM()` / `AVG()` / `COUNT()` - Can also be used as window functions.

---

8. Numeric Conversion Functions

- `CAST()` - Converts a value to a specified data type.
- `CONVERT()` (SQL Server) - Converts a value to a specified data type.
- `FORMAT()` - Formats numbers (MySQL, SQL Server).

---

9. Geospatial Functions (Database-Specific)

- `ST_DISTANCE()` - Calculates the distance between two points.
- `ST_CONTAINS()` - Checks if one geometry contains another.
- `ST_INTERSECTS()` - Checks if two geometries intersect.

---

10. Miscellaneous Functions

- `RAND()` - Returns a random number.
- `UUID()` - Generates a unique identifier.
- `VERSION()` - Returns the database version.
- `SESSION_USER()` - Returns the current user.
- `DATABASE()` - Returns the current database name.

This list is general and may vary slightly depending on the SQL dialect. Let me know which SQL system you are using (e.g., BigQuery, MySQL, PostgreSQL, SQL Server), and I can provide specific details for that platform!

Here's a comprehensive guide to **`CREATE`**, **`ALTER`**, **`DROP`**, and related SQL statements used to define, modify, and delete database objects such as tables, views, databases, indexes, and more.

---

1. `CREATE` Statements

The `CREATE` command is used to create new database objects.

a. Create Database

```
CREATE DATABASE database_name;
```

- Creates a new database.
- **Optional (MySQL):** Add `IF NOT EXISTS` to avoid errors if the database exists.

---

b. Create Table

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

Example:

```
CREATE TABLE employees (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    hire_date DATE,
    salary DECIMAL(10, 2) DEFAULT 0.0
);
```

- **Constraints:** `PRIMARY KEY`, `NOT NULL`, `UNIQUE`, `DEFAULT`, `CHECK`, `FOREIGN KEY`.

---

c. Create View

```
CREATE VIEW view_name AS
SELECT column1, column2
FROM table_name
WHERE condition;
```

Example:

```
CREATE VIEW high_salary_employees AS
SELECT name, salary
FROM employees
WHERE salary > 100000;
```

---

d. Create Index

```
CREATE INDEX index_name ON table_name (column_name);
```

Example:

```
CREATE INDEX idx_name ON employees(name);
```

- Speeds up queries involving `WHERE` or `ORDER BY` on the indexed column.
- **Unique Index:** Prevents duplicate values:

  ```
  CREATE UNIQUE INDEX unique_idx_name ON employees(email);
  ```

---

2. `ALTER` Statements

The `ALTER` command modifies the structure of existing database objects.

a. Alter Table

- **Add a Column:**

  ```
  ALTER TABLE table_name ADD column_name datatype constraints;
  ```

  Example:

  ```
  ALTER TABLE employees ADD department_id INT;
  ```

- **Modify a Column:**

  ```
  ALTER TABLE table_name MODIFY column_name new_datatype;
  ```

  Example:

  ```
  ALTER TABLE employees MODIFY salary DECIMAL(12, 2);
  ```

- **Drop a Column:**

  ```
  ALTER TABLE table_name DROP COLUMN column_name;
  ```

  Example:

  ```
  ALTER TABLE employees DROP COLUMN department_id;
  ```

- **Rename a Column:**

  ```
  ALTER TABLE table_name RENAME COLUMN old_name TO new_name;
  ```

  Example (PostgreSQL/BigQuery):

  ```
  ALTER TABLE employees RENAME COLUMN hire_date TO joining_date;
  ```

- **Add/Drop Constraints:**

  ```
  ALTER TABLE table_name ADD CONSTRAINT constraint_name constraint_type (column_name);
  ALTER TABLE table_name DROP CONSTRAINT constraint_name;
  ```

---

b. Alter View

- Modify a view definition:

  ```
  CREATE OR REPLACE VIEW view_name AS
  SELECT column1, column2
  FROM table_name
  WHERE condition;
  ```

---

3. `DROP` Statements

The `DROP` command deletes objects from the database.

a. Drop Database

```
DROP DATABASE database_name;
```

- Deletes the database and all its contents.

b. Drop Table

```
DROP TABLE table_name;
```

- Deletes a table and its data.

c. Drop View

```
DROP VIEW view_name;
```

- Deletes a view.

d. Drop Index

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

- Removes an index.

---

4. Other Related DDL Commands

a. Rename Table

```
ALTER TABLE table_name RENAME TO new_table_name;
```

Example:

```
ALTER TABLE employees RENAME TO staff;
```

---

b. Truncate Table

```
TRUNCATE TABLE table_name;
```

- Deletes all rows from a table but retains the structure.

---

c. Create Schema

```
CREATE SCHEMA schema_name;
```

- Creates a new schema (namespace for organizing objects).

---

d. Alter Schema

- Rename a schema (database-specific):

```
ALTER SCHEMA schema_name RENAME TO new_schema_name;
```

e. Drop Schema

```
DROP SCHEMA schema_name;
```

- Deletes the schema and all its objects.

5. Notes on DDL Commands

- **Transactional Behavior:**
    - Some DDL operations (e.g., `CREATE`, `DROP`) are not transactional. Once executed, they cannot be rolled back.
- **Database-Specific Features:**
    - Certain DDL commands may have different syntaxes or capabilities in databases like MySQL, PostgreSQL, SQL Server, or BigQuery.
    - For example, BigQuery uses `CREATE OR REPLACE` for views and tables.

Let me know if you need further clarification or examples tailored to a specific SQL dialect!

```
-- CREATE DATABASE FARMERS_MARKET;

USE FARMERS_MARKET;

CREATE TABLE CUSTOMER(
    CUST_ID INT PRIMARY KEY,
    CUST_NAME VARCHAR(50) NOT NULL,
    ZIP_CODE VARCHAR(10)
);

SELECT * FROM CUSTOMER;

INSERT INTO CUSTOMER (CUST_ID,CUST_NAME,ZIP_CODE)
VALUES (1,"JASON",411111);
```

```
INSERT INTO CUSTOMER VALUES (2,"SCHOOL",453261);

INSERT INTO CUSTOMER VALUES
(3,"GOKUL",111111),
(4,"OM",111111);

INSERT INTO CUSTOMER (CUST_NAME,CUST_ID,ZIP_CODE)
VALUES ("rITIK",5,411111);

INSERT INTO CUSTOMER (CUST_NAME,CUST_ID,ZIP_CODE)
VALUES ("SAYALI",6,411111);

ALTER TABLE CUSTOMER ADD PHONE_NUMBER INT;

ALTER TABLE CUSTOMER MODIFY PHONE_NUMBER VARCHAR(10);

ALTER TABLE CUSTOMER DROP COLUMN ZIP_CODE;

ALTER TABLE CUSTOMER RENAME COLUMN CUST_NAME TO FIRST_NAME;

ALTER TABLE CUSTOMER RENAME TO CUST_TABLE;

SELECT * FROM CUST_TABLE;

SELECT * FROM CUSTOMER;

DROP TABLE CUST_TABLE;

CREATE TABLE Payments (
    payment_id INT PRIMARY KEY,
    customer_name VARCHAR(50),
    amount VARCHAR(10)  -- Storing amount as VARCHAR (incorrect data type)
);

INSERT INTO Payments VALUES (1, 'Alice', '100');
INSERT INTO Payments VALUES (2, 'Bob', '200');
INSERT INTO Payments VALUES (3, 'Charlie', '50.75');
```

```
SELECT *
FROM Payments;

SELECT payment_id, customer_name,
       CAST(amount AS DECIMAL(10,2)) AS amount_numeric
FROM Payments;

SELECT SUM(CAST(amount AS DECIMAL(10,2))) AS total_payments
FROM Payments;
```