What is SQL?

- DBMS language, Structured Query Language, storing, manipulating and retrieving data out of a database

Database Management Systems(DBMS)

- A Database is a collection of small units of data arranged systematically
- A Relational Database Management System is a collection of tools that allows the users to manipulate, organize and visualize the contents of a database

SQL DBMS tools

- Oracle, MySQL, PostgreSQL, SQLite

Tables

- All data in the database are organized efficiently in the form of tables
- A database can be formed from a collection of multiple tables, linked with each other by using some relations.

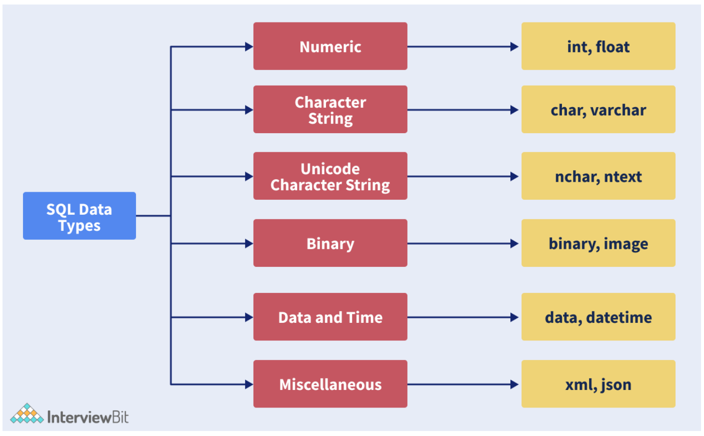
Create Table

```
CREATE TABLE student(
 ID INT NOT NULL PRIMARY KEY,--postgreSQL
 Name varchar(25),
 Phone varchar(12),
 Class INT,
 PRIMARY KEY (ID) -- MySQL
);
```

Delete Table

```
DROP TABLE student;
```

Data Types



String Datatypes:

Datatype	**Description**
CHAR(size)	A fixed-length string containing numbers, letters or special characters. Length may vary from 0-255.
VARCHAR(size)	Variable-length string where the length may vary from 0-65535. Similar to CHAR.
TEXT(size)	Can contain a string of size up to 65536 bytes.
TINY TEXT	Can contain a string of up to 255 characters.
MEDIUM TEXT	Can contain a string of up to 16777215 characters.
LONG TEXT	Can contain a string of up to 4294967295 characters.
BINARY(size)	Similar to CHAR() but stores binary byte strings.
VARBINARY(size)	Similar to VARCHAR() but stores binary byte strings.
BLOB(size)	Holds blobs up to 65536 bytes.
TINYBLOB	It is used for Binary Large Objects and has a maximum size of 255bytes.
MEDIUMBLOB	Holds blobs up to 16777215 bytes.
LONGBLOB	Holds blobs upto 4294967295 bytes.
ENUM(val1,val2,…)	String object that can have only 1 possible value from a list of size at most 65536 values in an ENUM list. If no value is inserted, a blank value is inserted.
SET(val1,val2,…)	String object with 0 or more values, chosen from a list of possible values with a maximum limit of 64 values.

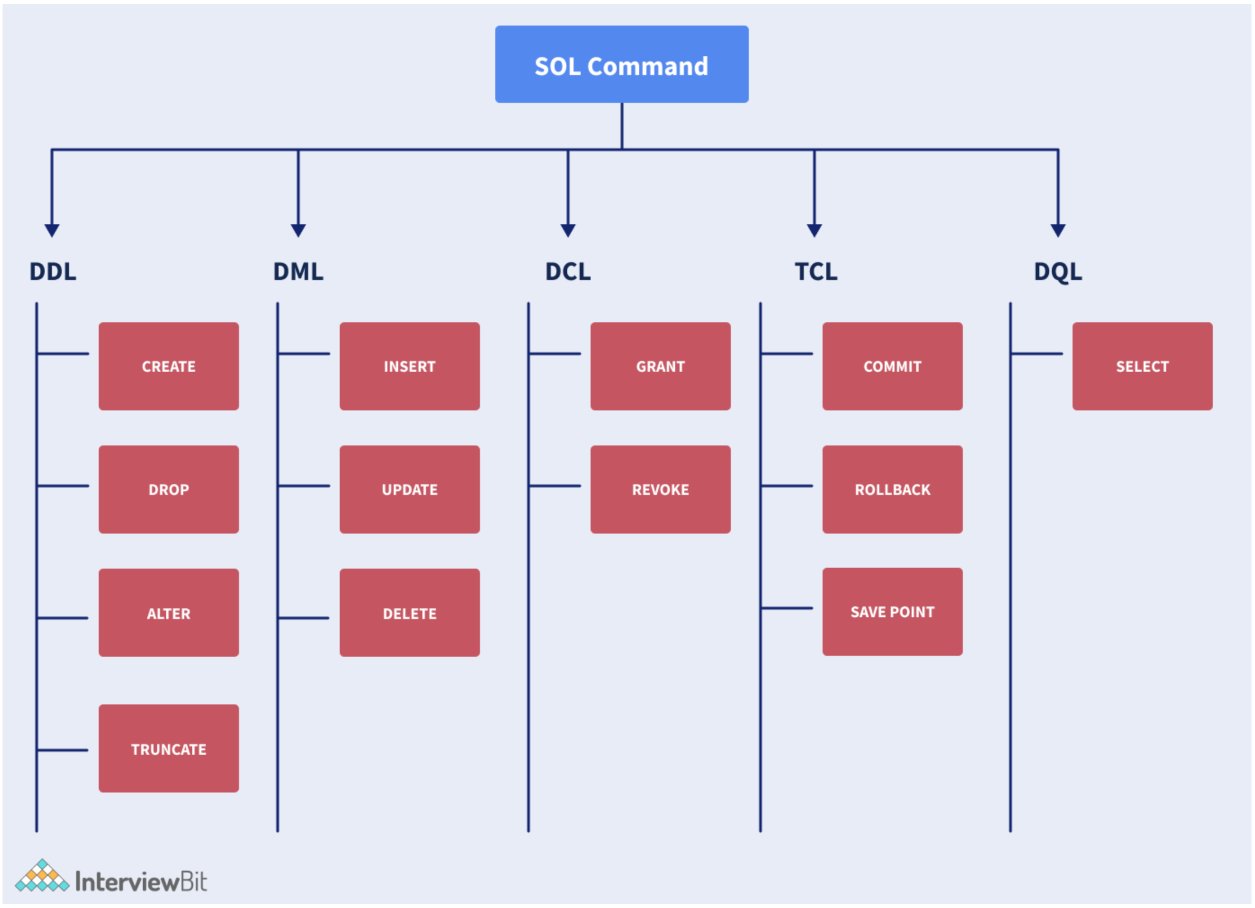
Numeric Datatypes:

Datatype	Description
BIT(size)	Bit-value type, where size varies from 1 to 64. Default value: 1
INT(size)	Integer with values in the signed range of -2147483648 to 2147483647 and values in the unsigned range of 0 to 4294967295.
TINYINT(size)	Integer with values in the signed range of -128 to 127 and values in the unsigned range of 0 to 255.
SMALLINT(size)	Integer with values in the signed range of -32768 to 32767 and values in the unsigned range of 0 to 65535.
MEDIUMINT(size)	Integer with values in the signed range of -8388608 to 8388607 and values in the unsigned range of 0 to 16777215.
BIGINT(size)	Integer with values in the signed range of 9223372036854775808 to 9223372036854775807 and values in the unsigned range of 0 to 18446744073709551615.
BOOLEAN	Boolean values where 0 is considered as FALSE and non-zero values are considered TRUE.
FLOAT (p)	The floating-point number is stored. If the precision parameter is set between 0 to 24, the type is FLOAT() else if it lies between 25 to 53, the datatype is DOUBLE().
DECIMAL(size,d)	Decimal number with a number of digits before decimal place set by size parameter, and a number of digits after the decimal point set by d parameter. Default values: size = 10, d = 10. Maximum Values: size = 65, d = 30.

Date/Time Datatypes:

Datatype	Description
DATE	Stores date in YYYY-MM-DD format with dates in the range of ‘1000-01-01’ to ‘9999-12-31’.
TIME(fsp)	Stores time in hh:mm:ss format with times in the range of ‘-838:59:59’ to ‘838:59:59’.
DATETIME(fsp)	Stores a combination of date and time in YYYY-MM-DD and hh:mm:ss format, with values in the range of ‘1000-01-01 00:00:00’ to ‘9999-12-31 23:59:59’.
TIMESTAMP(fsp)	It stores values relative to the Unix Epoch, basically a Unix Timestamp. Values lie in the range of ‘1970-01-01 00:00:01’ UTC to ‘2038-01-09 03:14:07’ UTC.
YEAR	Stores values of years as a 4digit number format, with a range lying between -1901 to 2155.

Types of SQL Commands:



Data Definition Language(DDL)

- It changes a table’s structure by adding, deleting, and altering its contents
- Its changes are auto-committee
- **CREATE:** Used to create a new table in the database.
- **CREATE TABLE** STUDENT(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
- **ALTER:** Used to alter the table contents by adding some new column or attriute, or changing some existing attribute
- **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.
- **DROP TABLE** STUDENT;
- **TRUNCATE:** Used to delete all the rows from the table, and free up the space in the table.
- **TRUNCATE TABLE** STUDENT;

Data Manipulation Language(DML)

- It is used for modifying a databae
- These commands are not auto-committed, i.e all changes are not automatically saved in the database.
- **INSERT:** Used to insert data in the row of a table.
- **INSERT INTO** STUDENT (Name, Subject) **VALUES** ("Scaler", "DSA");
- **UPDATE:** Used to update value of a table’s column.
- UPDATE STUDENT
- **SET** User_Name = ‘ABC’
- **WHERE** Student_Id = '2'
- **DELETE:** Used to delete one or more rows in a table.
- **DELETE FROM** STUDENT
- **WHERE** Name = "Scaler";

Data Control Language(DCL):

- These commands are used to grant and take back access/authority (revoke) from any database user
- **Grant:** Used to grant a user access privileges to a database
- **GRANT SELECT**, UPDATE **ON** TABLE_1 **TO** USER_1, USER_2;
- **Revoke**: Used to revoke the permissions from an user.
- **REVOKE SELECT**, UPDATE **ON** TABLE_1 **FROM** USER_1, USER_2;

Transaction Control Language:

- These commands can be used only with DML commands in conjunction and belong to the category of auto-committed commands.
- **COMMIT:** Saves all the transactions made on a database.
- **DELETE FROM** STUDENTS
- **WHERE** AGE = 16;
- **COMMIT**;
- **ROLLBACK:** It is used to undo transactions which are not yet been saved.
- **DELETE FROM** STUDENTS
- **WHERE** AGE = 16;
- **ROLLBACK**;
- **SAVEPOINT:** Used to roll transaction back to a certain point without having to roll back the entirity of the transaction.
- **SAVEPOINT** SAVED;
- **DELETE FROM** STUDENTS
- **WHERE** AGE = 16;
- **ROLLBACK TO** SAVED;

Data Query Language:

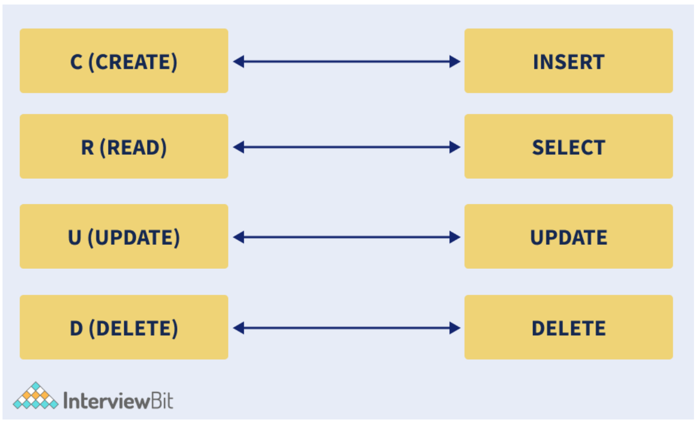
- ‘It is used to fetch some data from a database
- SELECT
- It is used to retrieve selected data based on some conditions which are described using the WHERE clause. It is to be noted that the WHERE clause is also optional to be used here and can be used depending on the user’s needs.
- **SELECT** Name
- **FROM** Student
- **WHERE** age >= 18;

SQL Constraints

- Constraints are rules which are applied on a table
- **NOT NULL:** Specifies that this column cannot store a NULL value.
- **CREATE TABLE** Student
- (
- ID int(8) **NOT NULL**,
- NAME varchar(30) **NOT NULL**,
- ADDRESS varchar(50)
-);
- **UNIQUE:** Specifies that this column can have only Unique values
- **CREATE TABLE** Student
- (
- ID int(8) **UNIQUE**,
- NAME varchar(10) **NOT NULL**,
- ADDRESS varchar(20)
-);
- **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 some particular condition or not.
- **CREATE TABLE** Student
- (
- ID int(6) **NOT NULL**,
- NAME varchar(10),
- AGE int **CHECK** (AGE < 20)
-);
- **DEFAULT:** It specifies a default value for a column when no value is specified for that field.
- **CREATE TABLE** Student
- (
- ID int(8) **NOT NULL**,
- NAME varchar(50) **NOT NULL**,
- CLASS int **DEFAULT** 2
-);

Crud Operations in SQL

- CRUD is an abbreviation for **Create, Read, Update and Delete**
- Create: INSERT
- Read: SELECT
- Update: UPDATE
- Delete: DELETE

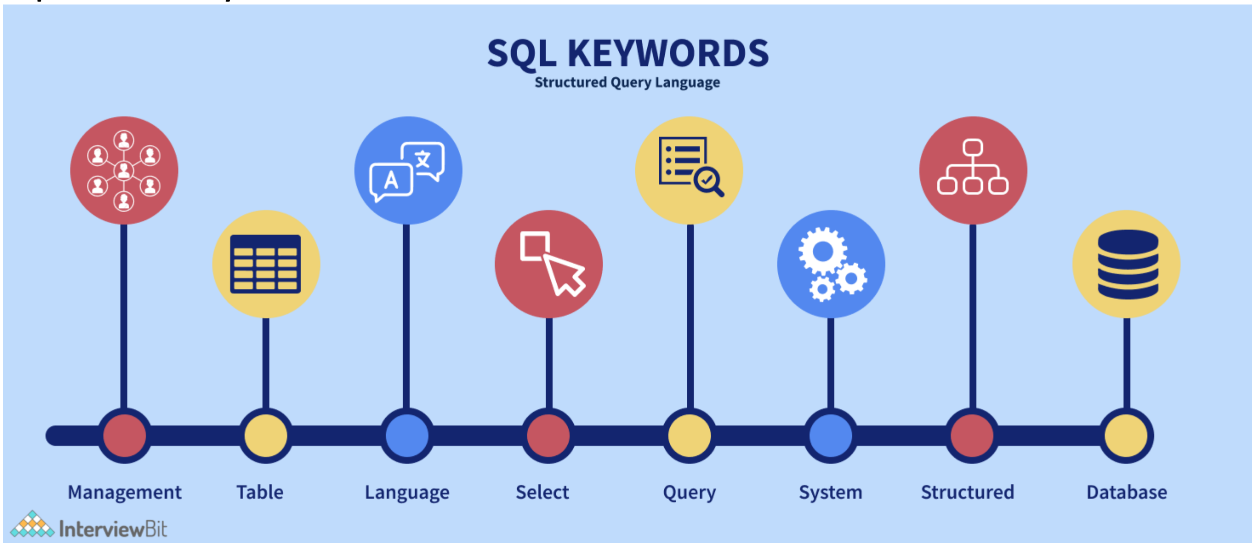


- **INSERT:** To insert any new data (create operation - C) into a database, we use the INSERT INTO statement.
- **INSERT INTO** student(ID, name, phone, class)
- **VALUES**(1, 'Scaler', '+1234-4527', 12)

- **INSERT INTO** student(ID, name, phone, class)
- **VALUES**(1, 'Scaler', '+1234-4527', 12),
- (2, 'Interviewbit', '+4321-7654', 11);

- **SELECT:** We use the select statement to perform the Read (R) operation of CRUD.
- **SELECT** name,class **FROM** student;
- **UPDATE:**The Update command is used to update the contents of specific columns of specific rows.
- UPDATE customers
- SET phone = '+1234-9876'
- WHERE ID = 2;
- **DELETE:**
- The Delete command is used to delete or remove some rows from a table.
- **DELETE FROM** student
- **WHERE** class = 11;

Important SQL Keywords



Keyword	Description	Example
ADD	Will add a new column to an existing table.	ALTER TABLE student ADD email_address VARCHAR(255)
ALTER TABLE	Adds edits or deletes columns in a table	ALTER TABLE student DROP COLUMN email_address;
ALTER COLUMN	Can change the datatype of a table's column	ALTER TABLE student ALTER COLUMN phone VARCHAR(15)
AS	Renames a table/column with an alias existing only for the query duration.	SELECT name AS student_name, phone FROM student;
ASC	Used in conjunction with ORDER BY to sort data in ascending order.	SELECT column1, column2, … FROM table_name ORDER BY column1, column2, … ASC;
DESC	Used in conjunction with ORDER BY to sort data in descending order.	SELECT column1, column2, … FROM table_name ORDER BY column1, column2, … DESC;
CHECK	Constrains the value which can be added to a column.	CREATE TABLE student(fullName varchar(255), age INT, CHECK(age >= 18));
CREATE DATABASE	Creates a new database.	CREATE DATABASE student;
DEFAULT	Sets the default value for a given column.	CREATE TABLE products(ID int, name varchar(255) DEFAULT 'Username', from date DEFAULT GETDATE());
DELETE	Delete values from a table.	DELETE FROM users WHERE user_id= 674;
DROP COLUMN	Deletes/Drops a column from a table.	ALTER TABLE student DROP COLUMN name;
DROP DATABASE	Completely deletes a database with all its content within.	DROP DATABASE student;
DROP DEFAULT	Removes a default value for a column.	ALTER TABLE student ALTER COLUMN age DROP DEFAULT;
DROP TABLE	Deletes a table from a database.	DROP TABLE students;

FROM	Determines which table to read or delete data from.	SELECT * FROM students;
IN	Used with WHERE clause for multiple OR conditionals.	SELECT * FROM students WHERE name IN('Scaler', 'Interviewbit','Academy');
ORDER BY	Used to sort given data in Ascending or Descending order.	SELECT * FROM student ORDER BY age ASC
SELECT DISTINCT	Works in the same war as SELECT, except that only unique values are included in the results.	SELECT DISTINCT age from student;
TOP	Used in conjunction with SELECT to select a fixed number of records from a table.	SELECT TOP 5 * FROM students;
VALUES	Used along with the INSERT INTO keyword to add new values to a table.	INSERT INTO Customers (CustomerName, City, Country) VALUES ('Cardinal', 'Stavanger', 'Norway');
WHERE	Filters given data based on some given condition.	SELECT * FROM students WHERE age >= 18;
UNIQUE	Ensures that all values in a column are different.	UNIQUE (ID)
UNION	Used to combine the result-set of two or more SELECT statements.	SELECT column_name(s) FROM Table1 UNION SELECT column_name(s) FROM Table2;
UNION ALL	Combines the result set of two or more SELECT statements(it allows duplicate values)	SELECT City FROM table1 UNION ALL SELECT City FROM table2 ORDER BY City;
SELECT TOP	Used to specify the number of records to return.	SELECT TOP 3 * FROM Students;
LIMIT	Puts a restriction on how many rows are returned from a query.	SELECT * FROM table1 LIMIT 3;
UPDATE	Modifies the existing records in a table.	UPDATE Customers SET ContactName = 'Scaler', City = 'India' WHERE CustomerID = 1;
SET	Used with UPDATE to specify which columns and values should be updated in a table.	UPDATE Customers SET ContactName = 'Scaler', City= 'India' WHERE CustomerID = 1;
IS NULL	Column values are tested for NULL values using this operator.	SELECT CustomerName, ContactName, Address FROM Customers WHERE Address IS NULL;
LIKE	Used to search for a specified pattern in a column.	SELECT * FROM Students WHERE Name LIKE 'a%';
ROWNUM	Returns a number indicating the order in which Oracle selects the row from a table or set of joined rows.	SELECT * FROM Employees WHERE ROWNUM < 10;
GROUP BY	Groups rows that have the same values into summary rows.	SELECT COUNT(StudentID), State FROM Students GROUP BY State;
HAVING	Enables the user to specify conditions that filter which group results appear in the results.	HAVING COUNT(CustomerID) > 5;

Clauses in SQL

- Clauses are in-built functions available in SQL and are used for filtering and analysing data quickly allowing the user to efficiently extract the required information from the database.

Name	Description	Example
WHERE	Used to select data from the database based on some conditions.	SELECT * from Employee WHERE age >= 18;
AND	Used to combine 2 or more conditions and returns true if all the conditions are True.	SELECT * from Employee WHERE age >= 18 AND salary >= 45000 ;
OR	Similar to AND but returns true if any of the conditions are True.	Select * from Employee where salary >= 45000 OR age >= 18
LIKE	Used to search for a specified pattern in a column.	SELECT * FROM Students WHERE Name LIKE 'a%';

LIMIT	Puts a restriction on how many rows are returned from a query.	SELECT * FROM table1 LIMIT 3;
ORDER BY	Used to sort given data in Ascending or Descending order.	SELECT * FROM student ORDER BY age ASC
GROUP BY	Groups rows that have the same values into summary rows.	SELECT COUNT(StudentID), State FROM Students GROUP BY State;
HAVING	It performs the same as the WHERE clause but can also be used with aggregate functions.	SELECT COUNT(ID), AGE FROM Students GROUP BY AGE HAVING COUNT(ID) > 5;

SQL Operators

- Operators are used in SQL to form complex expressions which can be evaluated to more intricate queries and extract more precise data from a database.
- **Arithmetic Operators:**

Operator	Description
+	Addition
-	Subtraction
*****	Multiplication
/	Division
%	Modulo

- **Bitwise Operators:**

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR

- **Relational Operators:**

Operator	Description
=	**Equal to**
>	**Greater than**
<	**Less than**
>=	**Greater than or equal to**
<=	**Less than or equal to**
<>	**Not equal to**

- **Compound Operators:**

Operator	Description
+=	Add equals
-=	Subtract equals
***=**	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	AND equals
|=	OR equals
^=	XOR equals

- **Logical Operators:**

Operator	Description
ALL	Returns True if all subqueries meet the given condition.
AND	Returns True if all the conditions turn out to be true
ANY	True if any of the subqueries meet the given condition
BETWEEN	True if the operand lies within the range of the conditions
EXISTS	True if the subquery returns one or more records
IN	Returns True if the operands to at least one of the operands in a given list of expressions
LIKE	Return True if the operand and some given pattern match.
NOT	Displays some record if the set of given conditions is False
OR	Returns True if any of the conditions turn out to be True
SOME	Returns True if any of the Subqueries meet the given condition.

Keys in SQL

- **Primary Key:** They uniquely identify a row in a table.

- Only a single primary key for a table.
- The primary key column cannot have any NULL values.
- The primary key must be unique for each row.
- **CREATE TABLE** Student (
- ID int **NOT NULL**,
- LastName varchar(255) **NOT NULL**,
- FirstName varchar(255),
- Class int,
- **PRIMARY** KEY (ID)
-);
- **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.
- **CREATE TABLE** Orders (
- OrderID int **NOT NULL**,
- OrderNumber int **NOT NULL**,
- PersonID int,
- **PRIMARY** KEY (OrderID),
- **FOREIGN** KEY (PersonID) **REFERENCES** Persons(PersonID)
-);
- **Super Key:** It is a group of single or multiple keys which identifies row of a table.
- **Candidate Key:** It is a collection of unique attributes that can uniquely identify tuples in a table.
- **Alternate Key:** It is a column or group of columns that can identify every row in a table uniquely.
- **Compound Key:** It is a collection of more than one record that can be used to uniquely identify a specific record.
- **Composite Key:** Collection of more than one column that can uniquely identify rows in a table.
- **Surrogate Key:** It is an artificial key that aims to uniquely identify each record.

Functions in SQL

- **SQL Server String Functions:**

Name	Description
ASCII	Returns ASCII values for a specific character.
CHAR	Returns character based on the ASCII .
CONCAT	Concatenates 2 strings together.
SOUNDEX	Returns similarity of 2 strings in terms of a 4 character .
DIFFERENCE	Compares 2 SOUNDEX values and returns the result as an integer.
SUBSTRING	Extracts a substring from a given string.
TRIM	Removes leading and trailing whitespaces from a string.
UPPER	Converts a string to upper-case.

- **SQL Server Numeric Functions:**

Name	Description
ABS	Returns the absolute value of a number.
ASIN	Returns arc sine value of a number.
AVG	Returns average value of an expression.
COUNT	Counts the number of records returned by a SELECT query.
EXP	Returns e raised to the power of a number.
FLOOR	Returns the greatest integer <= the number.
RAND	Returns a random number.
SIGN	Returns the sign of a number.
SQRT	Returns the square root of a number.
SUM	Returns the sum of a set of values.

- **SQL Server Date Functions:**

Name	Description
CURRENT_TIMESTAMP	Returns current date and time.
DATEADD	Adds a date/time interval to date and returns the new date.
DATENAME	Returns a specified part of a date(as a string).
DATEPART	Returns a specified part of a date(as an integer).
DAY	Returns the day of the month for a specified date.

GETDATE	Returns the current date and time from the database.

- **SQL Server Advanced Functions:**

Name	Description
CAST	Typecasts a value into specified datatype.
CONVERT	Converts a value into a specified datatype.
IIF	Return a value if a condition evaluates to True, else some other value.
ISNULL	Return a specified value if the expression is NULL, else returns the expression.
ISNUMERIC	Checks if an expression is numeric or not.
SYSTEM_USER	Returns the login name for the current user
USER_NAME	Returns the database user name based on the specified id.

Joins in SQL

- Joins are a SQL concept that allows us to fetch data after combining multiple tables of a database.
- **INNER JOIN:** Returns any records which have matching values in both tables.
    - **SELECT** orders.order_id, products.product_name, customers.customer_name, products.price
    - **FROM** orders
    - **INNER JOIN** products **ON** products.product_id = order.product_id
    - **INNER JOIN** customers **on** customers.customer_id = order.customer_id;
- **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.
    - **Select** * **from** Customers **Natural JOIN** Orders;
- **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;
- **LEFT JOIN:** Returns all of the records from the first table, along with any matching records from the second table.
    - **SELECT** Customers.CustomerName, Orders.OrderID
    - **FROM** Customers
    - **LEFT JOIN** Orders
    - **ON** Customers.CustomerID=Orders.CustomerID
    - **ORDER BY** Customers.CustomerName;
- **FULL JOIN:** Returns all records from both tables when there is a match.
-
- Example:
- Consider the below tables, Customers and Orders,
- **Table Customers:**
    - **SELECT** ID, NAME, AMOUNT, DATE
    - **FROM** CUSTOMERS
    - **FULL JOIN** ORDERS
    - **ON** CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

Triggers in SQL

- SQL s automatically executed in response to a certain event occurring in a table of a database are called triggers.
- **Create Trigger** Trigger_Name
- (Before | After) [**Insert** | **Update** | **Delete**]
- **on** [Table_Name]
- [**for each row** | **for each column**]
- [trigger_body]
    - **CREATE TRIGGER** trigger1
    - before **INSERT**
    - **ON** Student
    - **FOR EACH** ROW
    - **SET** new.total = (new.marks/ 10) * 100;

- **DROP:** This operation will drop an already existing trigger from the table.
- **DROP TRIGGER trigger** name;
- **SHOW:** This will display all the triggers that are currently present in the table.
- **SHOW** TRIGGERS **IN** database_name;

SQL Stored Procedures

- SQL procedures are stored in SQL s, which can be saved for reuse again and again.
- CREATE PROCEDURE procedure_name AS sql_statement
- GO;
    - EXEC procedure_name;
    - CREATE PROCEDURE SelectAllCustomers AS SELECT * FROM Customers;
    - GO;

Codes:

- Table name: airbnb_listings
- Columns: id, city, country, number_of_rooms, year _listed
- **Get all the columns from a table**
- SELECT *
- FROM airbnb_listings;

- **Return the city column from the table**
- SELECT city
- FROM airbnb_listings;

- **Get the city and year_listed columns from the table**
- SELECT city, year_listed
- FROM airbnb_listings;

- **Get the listing id, city, ordered by the number_of_rooms in ascending order**
- SELECT city, year_listed
- FROM airbnb_listings
- ORDER BY number_of_rooms ASC;

- **Get the listing id, city, ordered by the number_of_rooms in descending order**
- SELECT city, year_listed
- FROM airbnb_listings
- ORDER BY number_of_rooms DESC;

- **Get the first 5 rows from airbnb_listings**
- SELECT *
- FROM airbnb_lisitings
- LIMIT 5;

- **Get a unique list of cities where there are listings**
- SELECT DISTINCT city
- FROM airbnb_lisitings;

- **Filtering on numeric columns**
- **Get all the listings where number_of_rooms is more or equal to 3**
- SELECT *
- FROM airbnb_listings
- WHERE number_of_rooms >= 3;

- **Get all the listings where number_of_rooms is more than 3**
- SELECT *
- FROM airbnb_listings
- WHERE number_of_rooms > 3;

- **Get all the listings where number_of_rooms is exactly 3**
- SELECT *
- FROM airbnb_listings
- WHERE number_of_rooms = 3;

- **Get all the listings where number_of_rooms is lower or equal to 3**
- SELECT *
- FROM airbnb_listings
- WHERE number_of_rooms <= 3;

- **Get all the listings where number_of_rooms is lower than 3**
- SELECT *
- FROM airbnb_listings
- WHERE number_of_rooms < 3;

- **Filtering columns within a range—Get all the listings with 3 to 6 rooms**
- SELECT *
- FROM airbnb_listings
- WHERE number_of_rooms BETWEEN 3 AND 6;

- **Filtering on text columns**
- **Get all the listings that are based in 'Paris'**
- SELECT *
- FROM airbnb_listings
- WHERE city = ’Paris’;

- **Filter one column on many conditions—Get the listings based in the 'USA' and in ‘France’**
- SELECT *
- FROM airbnb_listings
- WHERE country IN (‘USA’, ‘France’);

- **Get all listings where city starts with "j" and where it does not end with "t"**
- SELECT *
- FROM airbnb_listings
- WHERE city LIKE ‘j%’ AND city NOT LIKE ‘%t’;

- **Filtering on multiple columns**
- **Get all the listings in "Paris" where number_of_rooms is bigger than 3**
- SELECT *
- FROM airbnb_listings
- WHERE city = ’Paris’ AND number_of_rooms > 3;

- **Get all the listings in "Paris" OR the ones that were listed after 2012**
- SELECT *
- FROM airbnb_listings
- WHERE city = 'Paris' OR year_listed > 2012;

- **Filtering on missing data**
- **Get all the listings where number_of_rooms is missing**
- SELECT *
- FROM airbnb_listings
- WHERE number_of_rooms IS NULL;

- **Get all the listings where number_of_rooms is not missing**

- SELECT *
- FROM airbnb_listings
- WHERE number_of_rooms IS NOT NULL;

- **Simple aggregations**
- **Get the total number of rooms available across all listings**
- SELECT SUM(number_of_rooms)
- FROM airbnb_listings;

- **Get the average number of rooms per listing across all listings**
- SELECT AVG(number_of_rooms)
- FROM airbnb_listings;

- **Get the listing with the highest number of rooms across all listings**
- SELECT MAX(number_of_rooms)
- FROM airbnb_listings;

- **Get the listing with the lowest number of rooms across all listings**
- SELECT MIN(number_of_rooms)
- FROM airbnb_listings;

- **Grouping, filtering, and sorting**
- **Get the total number of rooms for each country**
- SELECT country, SUM(number_of_rooms)
- FROM airbnb_listings
- GROUP BY country;

- **Get the average number of rooms for each country**
- SELECT country, AVG(number_of_rooms)
- FROM airbnb_listings
- GROUP BY country;

- **Get the listing with the maximum number of rooms for each country**
- SELECT country, MAX(number_of_rooms)
- FROM airbnb_listings
- GROUP BY country;

- **Get the listing with the lowest amount of rooms per country**
- SELECT country, MIN(number_of_rooms)
- FROM airbnb_listings
- GROUP BY country;

- **For each country, get the average number of rooms per listing, sorted by ascending order**
- SELECT country, AVG(number_of_rooms) AS avg_rooms
- FROM airbnb_listings
- GROUP BY country
- ORDER BY avg_rooms ASC;

- **For Japan and the USA, get the average number of rooms per listing in each country**
- SELECT country, AVG(number_of_rooms)
- FROM airbnb_listings
- WHERE country IN (‘USA’, ‘Japan’);
- GROUP BY country;

- **Get the number of cities per country, where there are listings**

- SELECT country, COUNT(city) AS number_of_cities
- FROM airbnb_listings
- GROUP BY country;

- **Get all the years where there were more than 100 listings per year**
- SELECT year_listed
- FROM airbnb_listings
- GROUP BY year_listed
- HAVING COUNT(id) > 100;