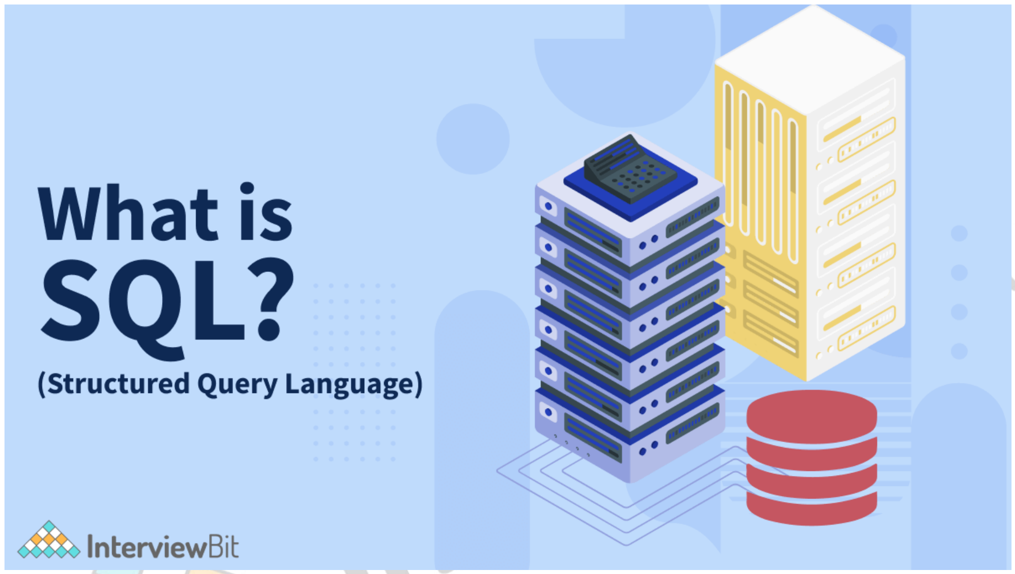


SQL Features

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.

Learn SQL: Basic to Advanced Concepts

1. Installation

To get started with using SQL, we first need to install some Database Management System server. After installing the RDBMS, the RDBMS itself will provide all the required tools to perform operations on the database and its contents through SQL. Some common RDBMS which is highly in use are:

- Oracle
- MySQL
- PostgreSQL
- Heidi SQL

To install any RDBMS, we just need to visit their official website and install the setup file from there, by following the instructions available there. With the server setup, we can set up a Query Editor, on which we can type our SQL Queries.

2. 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, where each table would be used for storing a particular kind of data and the table by themselves would be linked with each other by using some relations.

Example:

ID	Name	Phone	Class
INTEGER	VARCHAR(25)	VARCHAR(12)	INTEGER

The above example is for a table of students and stores their Name, Phone, and Class as data. The ID is assigned to each student to uniquely identify each student and using this ID, we can relate data from this table to other tables.

SQL-Create Table:

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

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

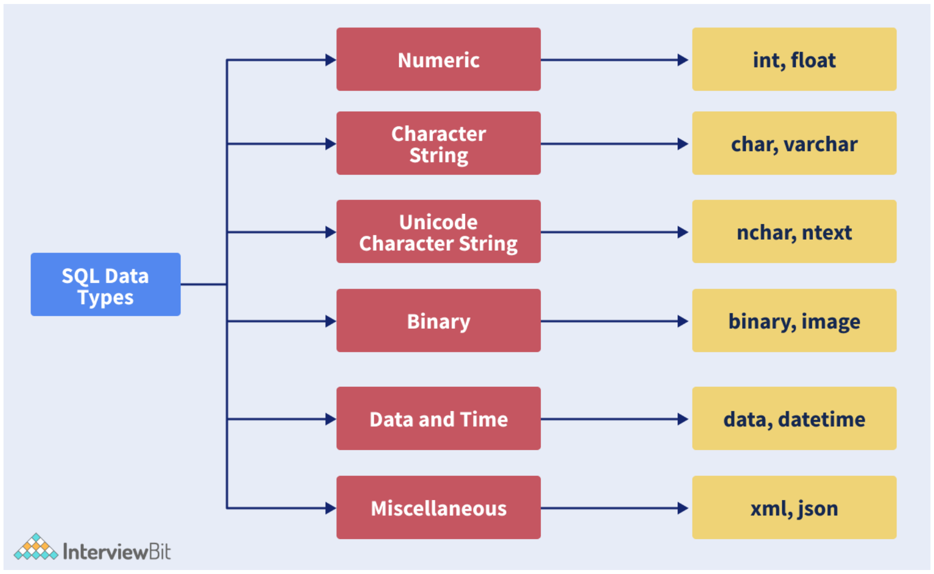
SQL-Delete Table:

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

```
DROP TABLE student;
```

3. SQL DataTypes

To allow the users to work with tables effectively, SQL provides us with various datatypes each of which can be useful based on the type of data we handle.



The above image is a chart that shows all the datatypes available in SQL along with some of their examples.

The next section describes various most popular SQL server datatypes categorised under each major division.

String Datatypes:

The table below lists all the String type datatypes available in SQL, along with their descriptions:

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.

Numeric Datatypes:

The table below lists all the Numeric Datatypes in SQL along with their descriptions:

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()

Date/Time Datatypes:

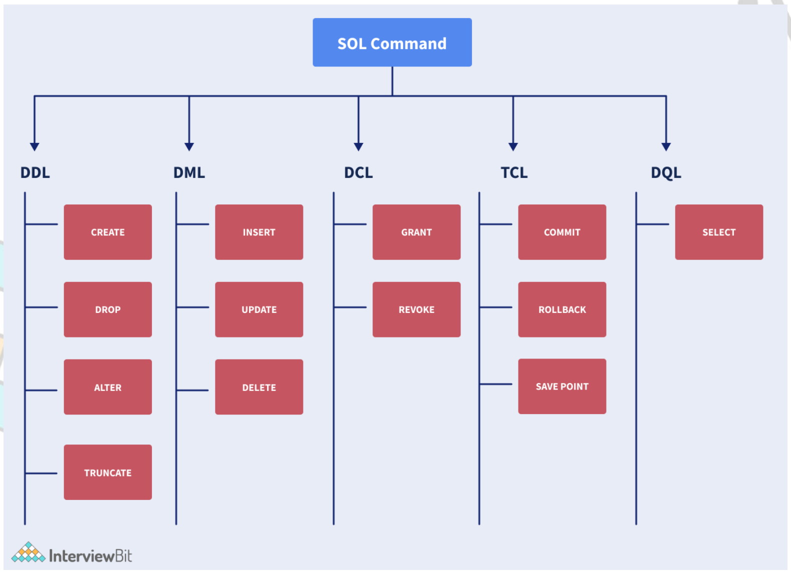
The datatypes available in SQL to handle Date/Time operations effectively are called the Date/Time datatypes. The below table lists all the Date/Time variables in SQL along with their description:

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.

4. 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 ("Scaler", "DSA");
```

In the above example, we insert the values “Scaler” 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 = 'Interviewbit'
WHERE Student_Id = '2'
```

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

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

Example:

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

In the above example, the query deletes the row where the Name of the student is “Scaler” 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: With WHERE clause,

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

Example: Without WHERE clause,

```
SELECT Name
FROM Student
```

In the first example, we will only select those names in the Student table, whose corresponding age is greater than 17. In the 2nd example, we will select all the names from the Student table.

5. SQL Constraints

Constraints are rules which are applied on a table. For example, specifying valid limits or ranges on data in the table etc.

The valid constraints in SQL are:

1. NOT NULL: Specifies that this column cannot store a NULL value.

Example:

```
CREATE TABLE Student
(
   ID int(8) NOT NULL,
   NAME varchar(30) NOT NULL,
   ADDRESS varchar(50)
);
```

In the above example, we create a table STUDENT, which has some attributes it has to store. Among these attributes we declare that the columns ID and NAME cannot have NULL values in their fields using NOT NULL constraint.

2. UNIQUE: Specifies that this column can have only Unique values, i.e the values cannot be repeated in the column.

Example:

```
CREATE TABLE Student
(
   ID int(8) UNIQUE,
   NAME varchar(10) NOT NULL,
   ADDRESS varchar(20)
);
```

In the above example, we create a table Student and declare the ID column to be unique using the UNIQUE constraint.

3. Primary Key: It is a field using which it is possible to uniquely identify each row in a table. We will get to know about this in detail in the upcoming section.

4. Foreign Key: It is a field using which it is possible to uniquely identify each row in some other table. We will get to know about this in detail in the upcoming section.

5. CHECK: It validates if all values in a column satisfy some particular condition or not.

Example:

```
CREATE TABLE Student
(
   ID int(6) NOT NULL,
   NAME varchar(10),
   AGE int CHECK (AGE < 20)
);
```

Here, in the above query, we add the CHECK constraint into the table. By adding the constraint, we can only insert entries that satisfy the condition AGE < 20 into the table.

6. DEFAULT: It specifies a default value for a column when no value is specified for that field.

Example:

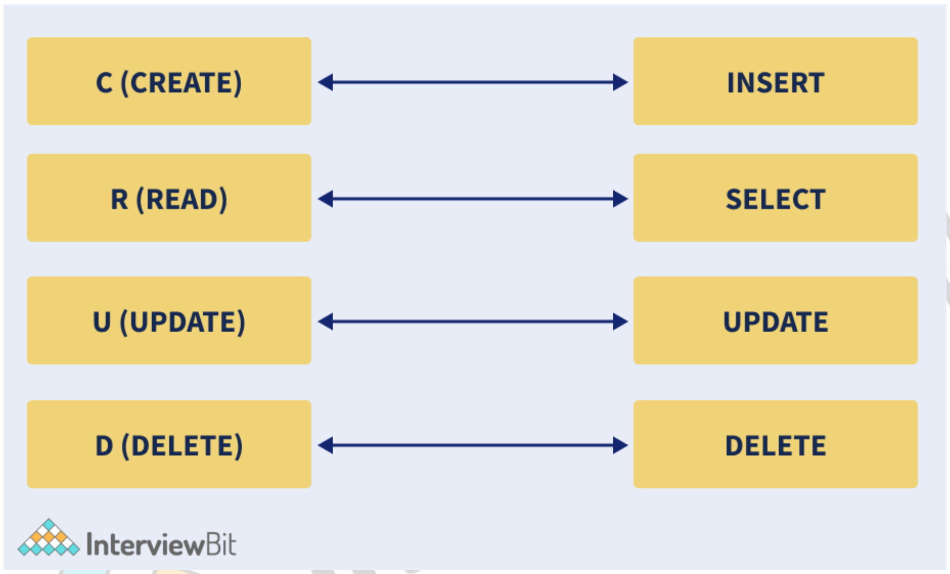
```
CREATE TABLE Student
(
   ID int(8) NOT NULL,
   NAME varchar(50) NOT NULL,
   CLASS int DEFAULT 2
);
```

In the above query, we set a default value of 2 for the CLASS attribute. While inserting records into the table, if the column has no value specified, then 2 is assigned to that column as the default value.

6. Crud Operations in SQL

CRUD is an abbreviation for **Create, Read, Update and Delete**. These 4 operations comprise the most basic database operations. The relevant commands for these 4 operations in SQL are:

- Create: INSERT
- Read: SELECT
- Update: UPDATE
- Delete: DELETE



The above image shows the pillars of SQL CRUD operations.

- **INSERT:** To insert any new data (create operation - C) into a database, we use the INSERT INTO statement.

SQL Syntax:

```
INSERT INTO name_of_table(column1, column2, ....)
   VALUES(value1, value2, ....)
```

Example:

```
INSERT INTO student(ID, name, phone, class)
   VALUES(1, 'Scaler', '+1234-4527', 12)
```

For multiple rows,

SQL Syntax:

```
INSERT INTO name_of_table(column1, column2, ....)
   VALUES(value1, value2, ....),
   (new_value1, new_value2, ...),
   (....), ... ;
```

Example:

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

The above example will insert into the student table having the values 1, Scaler, +1234-5678 and 12 to the columns ID, name, phone and class columns.

- **SELECT:** We use the select statement to perform the Read (R) operation of CRUD.

SQL Syntax:

```
SELECT column1,column2,.. FROM name_of_table;
```

Example:

```
SELECT name,class FROM student;
```

The above example allows the user to read the data in the name and class columns from the student table.

- **UPDATE:** Update is the ‘U’ component of CRUD. The Update command is used to update the contents of specific columns of specific rows.

SQL Syntax:

```
UPDATE name_of_table
SET column1=value1,column2=value2,...
WHERE conditions...;
```

Example:

```
UPDATE customers
SET phone = '+1234-9876'
WHEREID = 2;
```

The above SQL example code will update the table ‘customers’ whose ID is 2 with the new given phone number.

- **DELETE:**

The Delete command is used to delete or remove some rows from a table. It is the ‘D’ component of CRUD.

SQL Syntax:

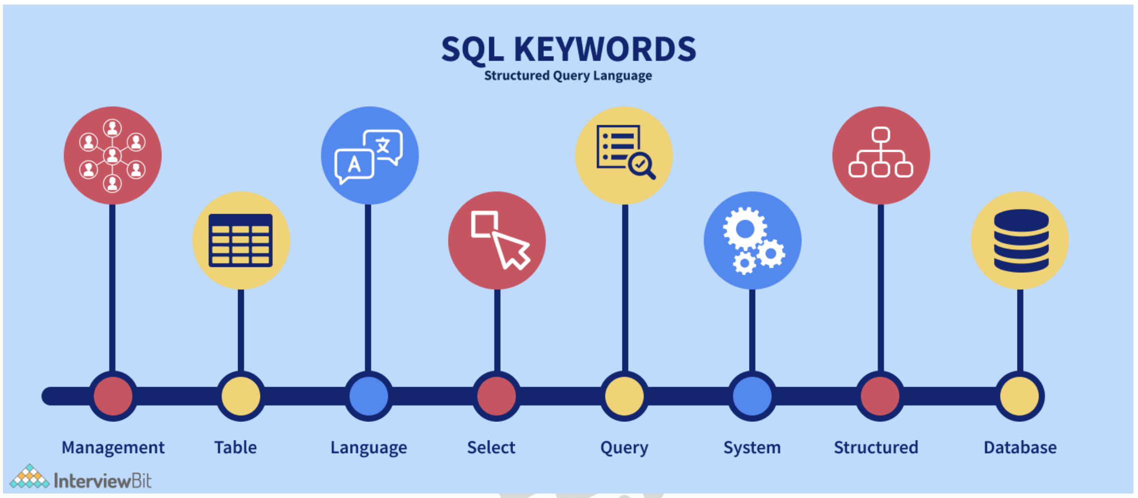
```
DELETE FROM name_of_table
WHERE condition1, condition2, ...;
```

Example:

```
DELETE FROM student
WHERE class = 11;
```

The above SQL example code will delete the row from table student, where the class = 11 conditions becomes true.

7. Important SQL Keywords



The below table lists some important keywords used in SQL, along with their description and example.

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	SELECT column1, column2, … FROM table_name ORDER BY column1, column2, … DESC;

8. 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.

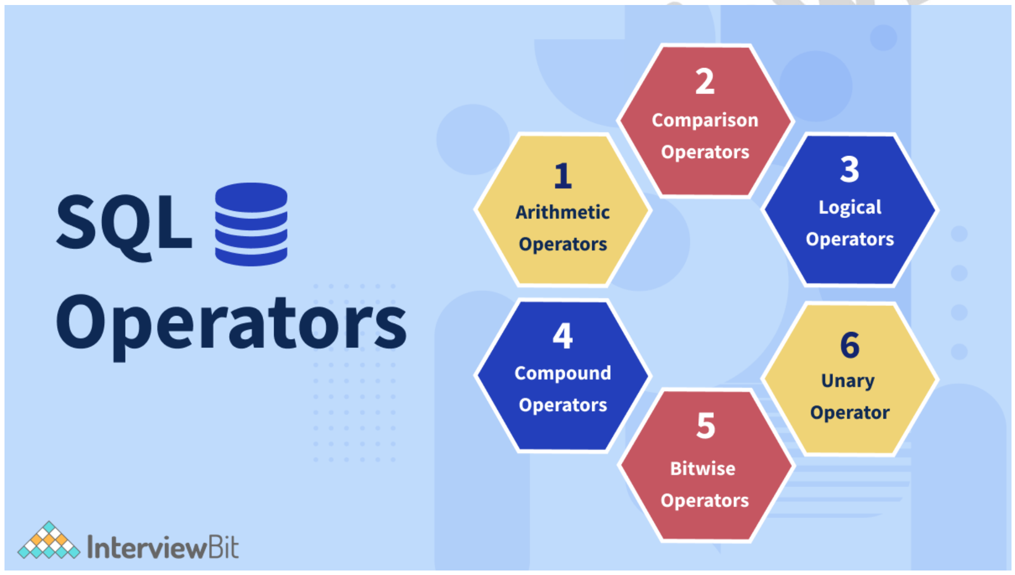
The below table lists some of the important SQL clauses and their description with examples:

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;

9. SQL Operators

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

There are 3 main types of operators: Arithmetic, Comparision and Logical operators, each of which will be described below.



- **Arithmetic Operators:**

Arithmetic Operators allows the user to perform arithmetic operations in SQL. The table below shows the list of arithmetic operators available in SQL:

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

- **Bitwise Operators:**

Bitwise operators are used to performing Bit manipulation operations in SQL. The table below shows the list of bitwise operators available in SQL:

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR

- **Relational Operators:**

Relational operators are used to performing relational expressions in SQL, i.e those expressions whose value either result in true or false. The table below shows the list of relational operators available in SQL:

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

- **Compound Operators:**

Compound operators are basically a combination of 2 or more arithmetic or relational operator, which can be used as a shorthand while writing code. The table below shows the list of compound operators available in SQL:

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

- **Logical Operators:**

Logical operators are used to combining 2 or more relational statements into 1 compound statement whose truth value is evaluated as a whole. The table below shows the SQL logical operators with their description:

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.

10. 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)
);
```

The above example creates a table called STUDENT with some given properties(columns) and assigns the ID column as the primary key of the table. Using the value of ID column, we can uniquely identify its corresponding row.

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)
);
```

- **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.

Amongst these, the Primary and Foreign keys are most commonly used.

11. Functions in SQL

The SQL Server has many builtin functions some of which are listed below:

- **SQL Server String Functions:**

The table below lists some of the String functions in SQL with their description:

Name	Description
ASCII	Returns ASCII values for a specific character.
CHAR	Returns character based on the ASCII code.
CONCAT	Concatenates 2 strings together.
SOUNDEX	Returns similarity of 2 strings in terms of a 4 character code.
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:**

The table below lists some of the Numeric functions in SQL with their description:

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:**

The table below lists some of the Date functions in SQL with their description:

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:**

The table below lists some of the Advanced functions in SQL with their description:

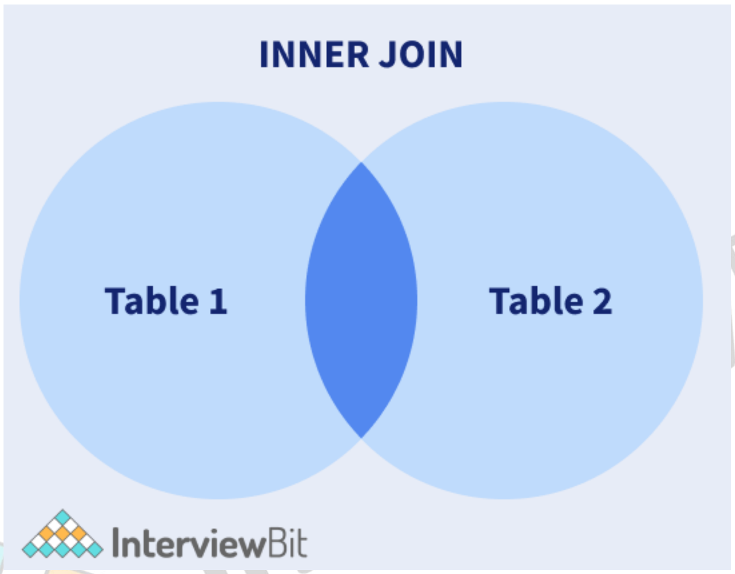
Name	Description
CAST	Typecasts a value into specified datatype.
CONVERT	Converts a value into a specified datatype.
IIF	Return a value if a condition evaluates to True, else some other value.
ISNULL	Return a specified value if the expression is NULL, else returns the expression.
ISNUMERIC	Checks if 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.

12. Joins in SQL

Joins are a SQL concept that allows us to fetch data after combining multiple tables of a database.

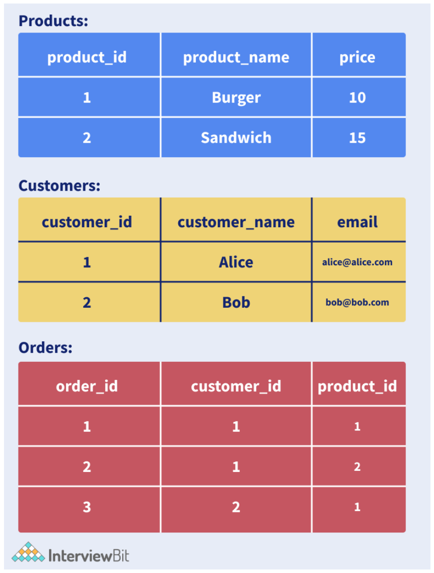
The following are the types of joins in SQL:

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



Example:

Consider the following tables,



Products:

product_id	product_name	price
1	Burger	10
2	Sandwich	15

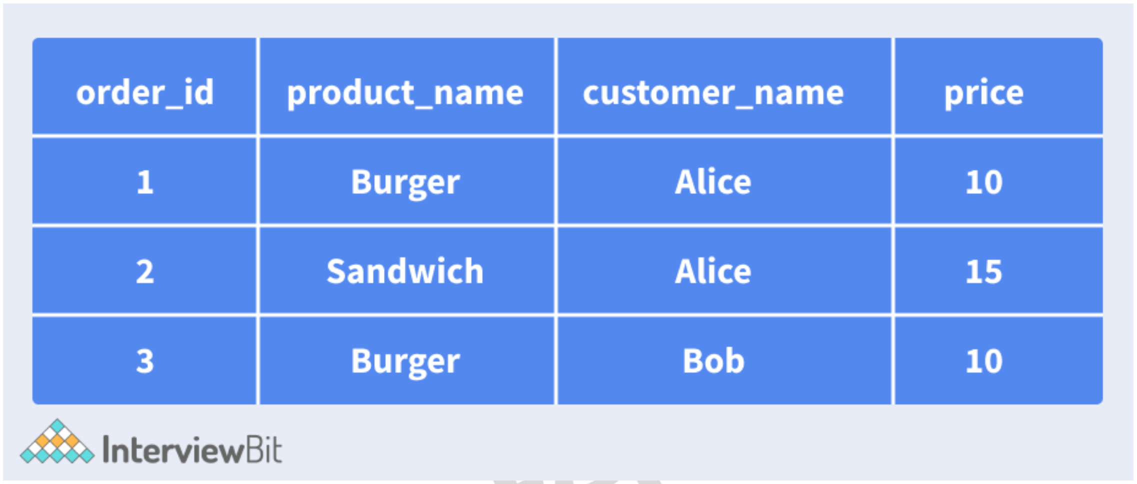
Customers:

customer_id	customer_name	email
1	Alice	alice@alice.com
2	Bob	bob@bob.com

Orders:

order_id	customer_id	product_id
1	1	1
2	1	2
3	2	1

Let us try to build the below table, using Joins,



order_id	product_name	customer_name	price
1	Burger	Alice	10
2	Sandwich	Alice	15
3	Burger	Bob	10

The SQL code will be as follows,

```
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.

Syntax:

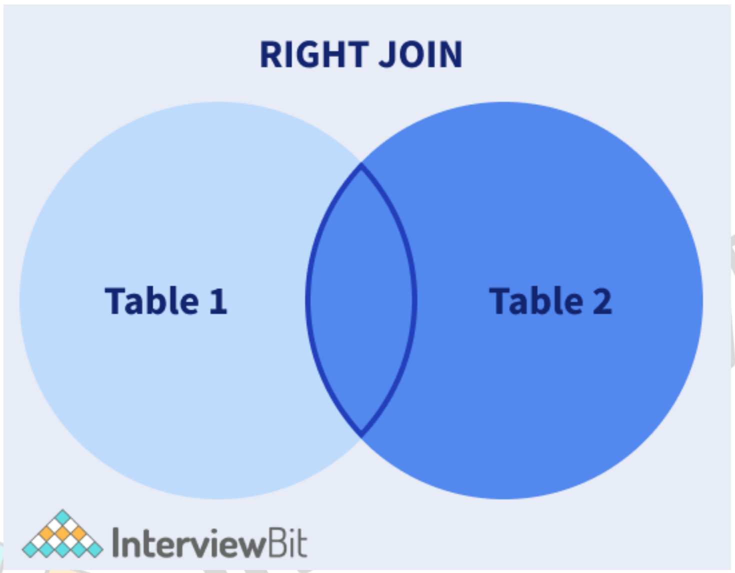
```
Select * from table1 Natural JOIN table2;
```

Example:

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

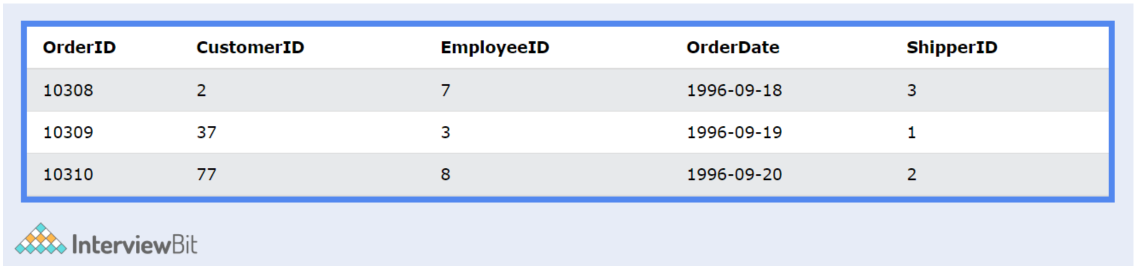
In the above example, we are merging the Customers and Orders table shown above using a NATURAL JOIN based on the common column customer_id.

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



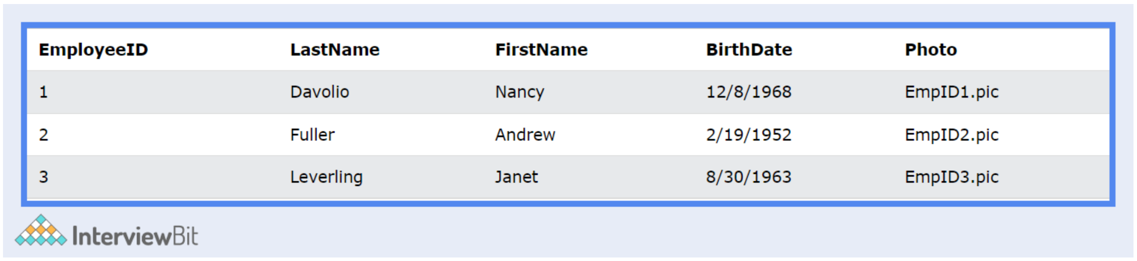
Example:

Let us define an Orders table first,



OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

Let us also define an Employee table,

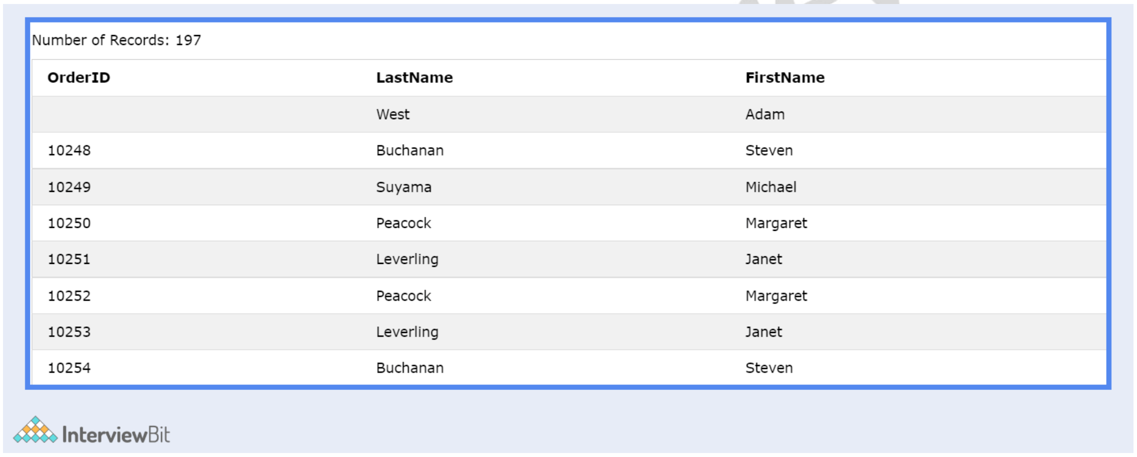


EmployeeID	LastName	FirstName	BirthDate	Photo
1	Davolio	Nancy	12/8/1968	EmpID1.pic
2	Fuller	Andrew	2/19/1952	EmpID2.pic
3	Leverling	Janet	8/30/1963	EmpID3.pic

Applying right join on these tables,

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

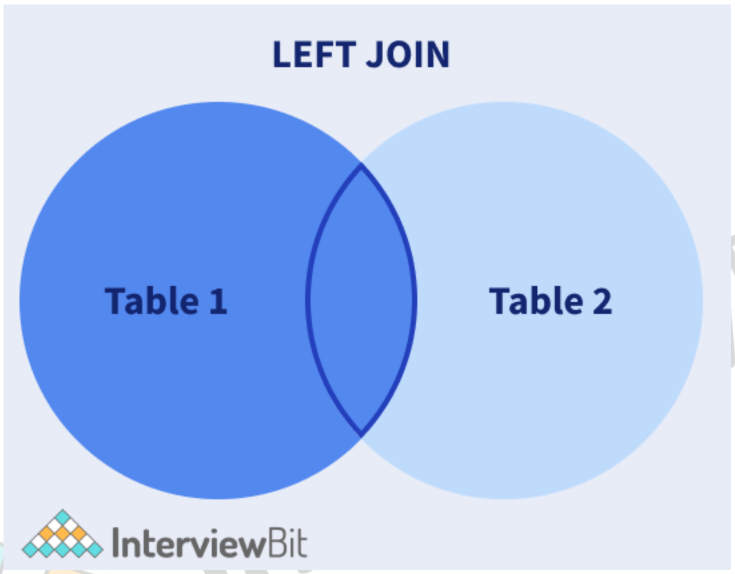
The resultant table will be,



Number of Records: 197

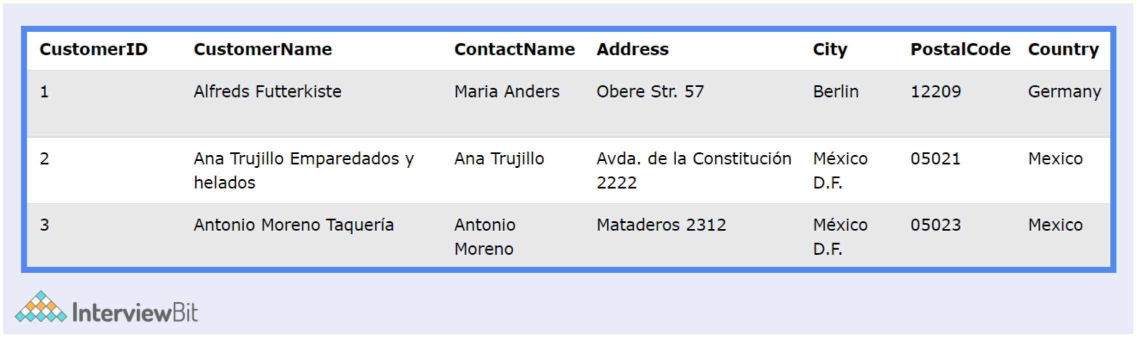
OrderID	LastName	FirstName
	West	Adam
10248	Buchanan	Steven
10249	Suyama	Michael
10250	Peacock	Margaret
10251	Leverling	Janet
10252	Peacock	Margaret
10253	Leverling	Janet
10254	Buchanan	Steven

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

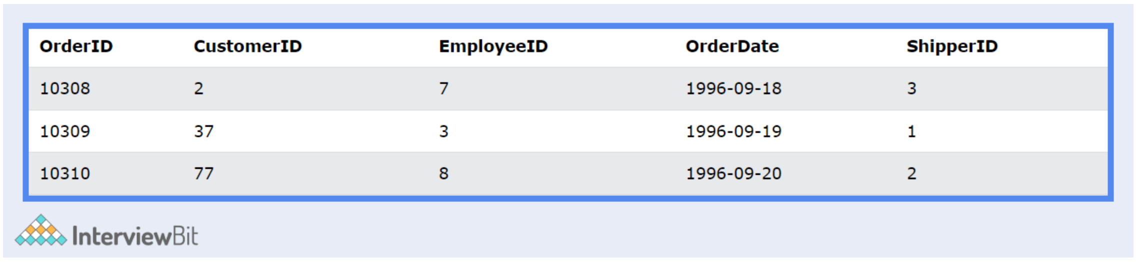


Example:

Consider the below Customer and Orders table,



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

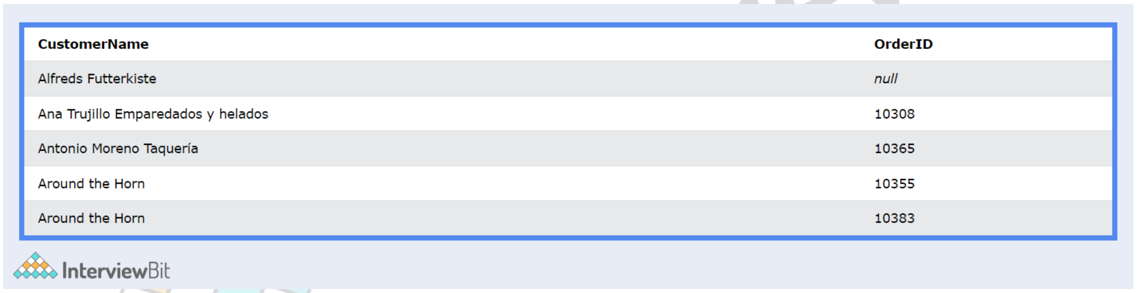


OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

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

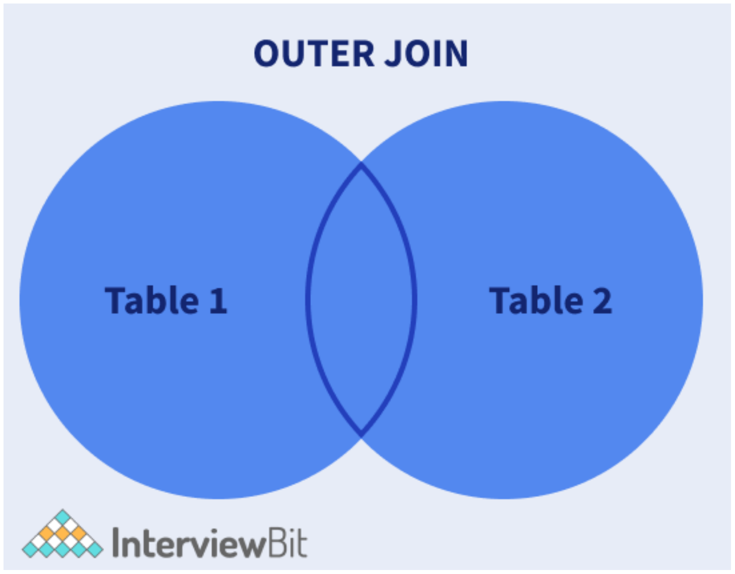
```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

The top few entries of the resultant table will appear as shown in the below image.



CustomerName	OrderID
Alfreds Futterkiste	*null*
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	10365
Around the Horn	10355
Around the Horn	10383

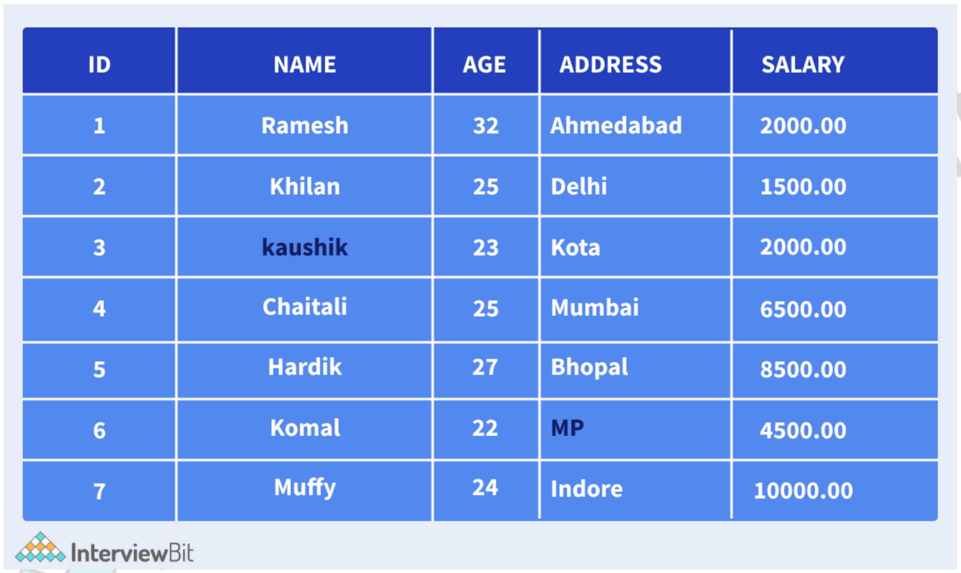
- **FULL JOIN:** Returns all records from both tables when there is a match.



Example:

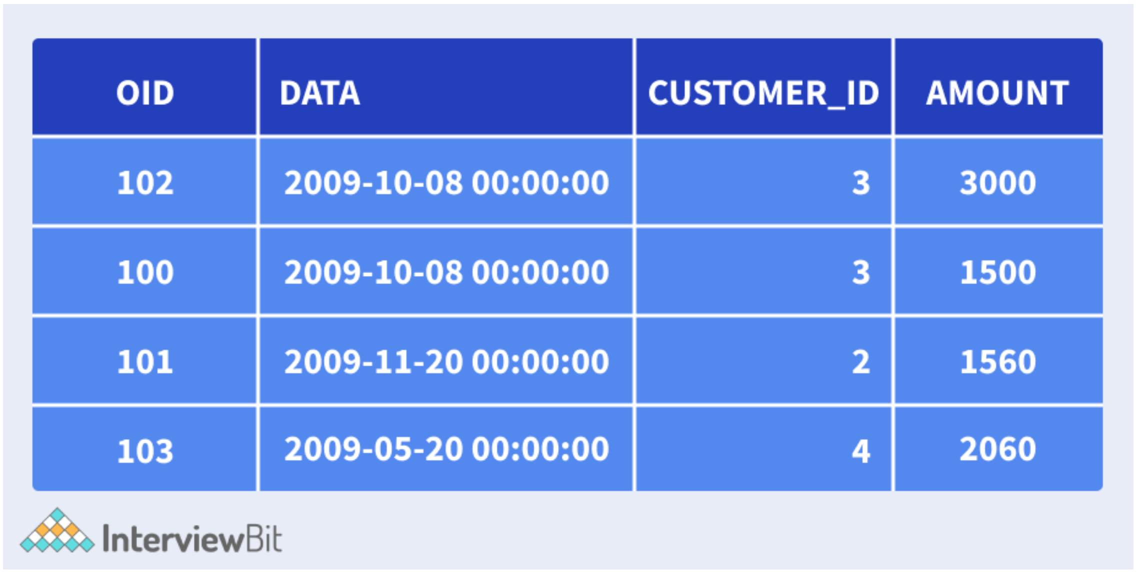
Consider the below tables, Customers and Orders,

Table Customers:



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table Orders:

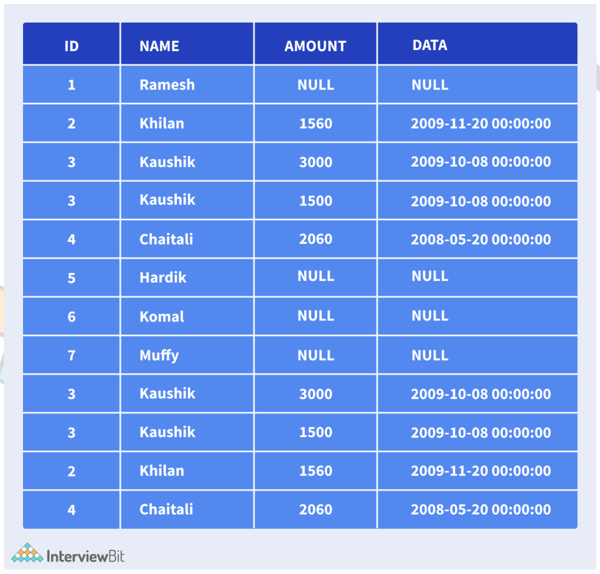


OID	DATA	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2009-05-20 00:00:00	4	2060

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;
```

We will get the following table as the result of the outer join.



ID	NAME	AMOUNT	DATA
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	Kaushik	3000	2009-10-08 00:00:00
3	Kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL
3	Kaushik	3000	2009-10-08 00:00:00
3	Kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

13. 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;
```

14. SQL Stored Procedures

SQL procedures are stored in SQL codes, which can be saved for reuse again and again.

Syntax:

```
CREATE PROCEDURE procedure_name AS sql_statement
GO;
```

To execute a stored procedure,

```
EXEC procedure_name;
```

Example:

```
CREATE PROCEDURE SelectAllCustomers AS SELECT * FROM Customers;
GO;
```

The above example creates a stored procedure called 'SelectAllCustomers', which selects all the records from the customer table.

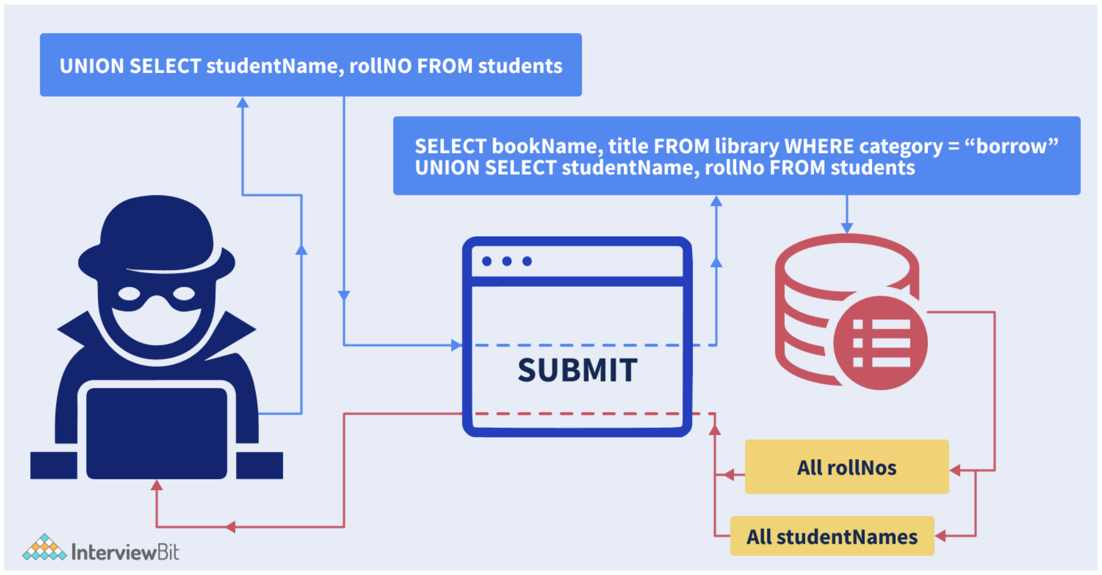
15. SQL Injection

Insertion or 'Injection' of some SQL Query from the input data of the client to the application is called SQL Injection. They can perform CRUD operations on the database and can read to vulnerabilities and loss of data.

It can occur in 2 ways:

- Data is used to dynamically construct an SQL Query.
- Unintended data from an untrusted source enters the application.

The consequences of SQL Injections can be Confidentiality issues, Authentication breaches, Authorization vulnerabilities, and breaking the Integrity of the system.


UNION SELECT studentName, rollNO FROM students
SELECT bookName, title FROM library WHERE category = "borrow" UNION SELECT studentName, rollNo FROM students
SUBMIT
All rollNos
All studentNames
InterviewBit

The above image shows an example of SQL injections, through the use of 2 tables - students and library.

Here the hacker is injecting SQL code -

```
UNION SELECT studentName, rollNo FROM students
```

into the Database server, where his query is used to JOIN the tables - students and library. Joining the 2 tables, the result of the query is returned from the database, using which the hacker gains access to the information he needs thereby taking advantage of the system vulnerability. The arrows in the diagram show the flow of how the SQL Injection causes the vulnerability in the database system, starting from the hacker's computer.

Conclusion:

Databases are growing increasingly important in our modern industry where data is considered to be a new wealth. Managing these large amounts of data, gaining insights from them and storing them in a cost-effective manner makes database management highly important in any modern software being made. To manage any form of databases/RDBMS, we need to learn SQL which allows us to easily code and manage data from these databases and create large scalable applications of the future, which caters to the needs of millions.