SQL Features

* SQL allows us to interact with the database 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 of 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.

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
    - Heldi SQL

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

1. Tables

All data in the database are organized efficiently in the form of tables. A database can be formed a collection of multiple tables, where each table 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 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;

1. SQL Data Types

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

String Datatypes:

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

|  |  |
| --- | --- |
| Datatype | Description |
| CHAR(size) | A fixed-length string containing number, 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 12777215 characters. |
| LONG TEXT | Can contain a string up to 4294967295 characters. |
| BINARY(size) | Similar to CHAR() but stores binary byte strings. |
| VARBINARY(size) | Similar to VARCHAR() but stored 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 255 bytes. |
| MEDIUMBLOB | Holds blobs up to 16777215 bytes. |
| LONGBLOB | Holds blobs up to 4294967295 bytes. |

Numeric Data types:

The table below lists all the numeric data types in SQL along with their descriptions:

|  |  |
| --- | --- |
| Data type | 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 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 data type is DOUBLE(). |

Date/Time Datatypes:

The data types available in SQL to handle Date/Time operations effectively are called the Date/Time data types. 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 range of ‘1000-01-01’ to ‘9999-12-31’. |
| TIME(fsp) | Stores time in hh:mm:ss format with times in range of ‘-839:59:59’ to ‘839:59:59’. |
| 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 4 digit number format, with a range lying between -1901 to 2155. |

1. 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:

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.

CREATE TABLE Student (Name VARCHAR(20), Email VARCHAR(100), DOB DATE);

* + - ALTER :- Used to alter contents of a table by adding some new column or attribute, 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 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 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.

INSERT INTO Student (Name, Subject) VALUES (“Scalar”, “DSA”);

In the above example, we insert the values “Scalar” and “DSA” in the columns Name and Subject in the Student Table.

* + - UPDATE :- Used to update value of a table’s column.

UPDATE Student

SET User\_Name = “DeepPatel”

WHERE Student\_Id = 2;

In the above example, we update the name of the student, whose Student\_Id is 2, to the User\_Name = “DeepPatel”;

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

DELETE FROM Student

WHERE Name = “Scalar”;

In the above example, the query deletes the row where the Name of the student is “Scalar” from the Student Table.

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.

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.

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 to users – USER\_1 and USER\_2.

Transaction Control Language (TCL) :- 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.

DELETE FROM Students

WHERE AGE = 16;

COMMIT;

In the above database, we delete the row where AGE of the student is 16, and then save this changes to the database using COMMIT.

* + - ROLLBACK :- It is used to Undo transactions which are not yet been saved.

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 :- User to roll transaction back to a certain point without having to roll back the entirely of the transaction.

SAVEPOINT SAVED;

DELETE FROM STUDENTS

WHERE AGE = 16;

ROLLBACK TO SAVED;

In the above example, we have created a save point just before performing the delete operation in the table, and the we can return to that save point using the ROLLBACK TO command.

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 is also optional to be used here and can be used depending on the user’s needs.

SELECT Name FROM Student WHERE age >= 18;

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.

1. 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:

* + - 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)

);

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.

* + - 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.

* + - 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.
    - 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.
    - 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.

* + - 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.

1. 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

* + - 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'

WHERE ID = 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.

1. 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 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; |

1. Clauses in SQL

Clauses are in-built functions available in SQL and are used for filtering and analyzing 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; |

1. 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:

|  |  |
| --- | --- |
| Operators | 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. |

1. 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.

1. 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 of a number. |
| AVG | Returns average value of an expression. |
| COUNT | Count the number of record returned by select query. |
| EXP | Returns a raised to the power of a number. |
| FLOOR | Returns the greatest integer<= the number. |
| RAND | Returns the 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 date (as a string). |
| DATEPART | Returns a specified part of date (as a 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:

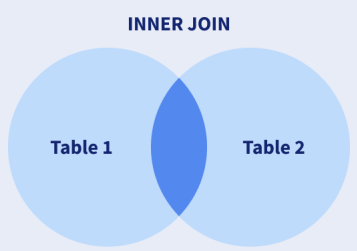
|  |  |
| --- | --- |
| Name | Description |
| CAST | Typecast a value into specified datatype. |
| CONVERT | Converts a value into specified datatype. |
| IIF | Returns a true if a condition evaluates to true, else some other value. |
| ISNULL | Returns the specified value if expression is NULL, else returns the expression. |
| ISNUMERIC | Checks if 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 aer combining multiple tables of a database.

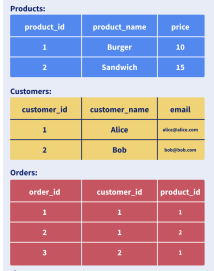
The following are the types of joins in SQL:

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

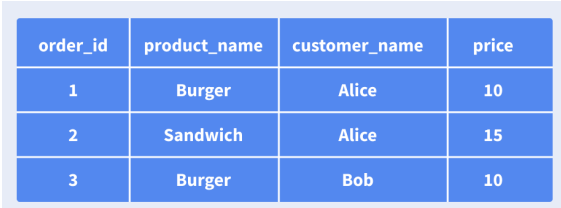


Example:

Consider the following tables,



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



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:

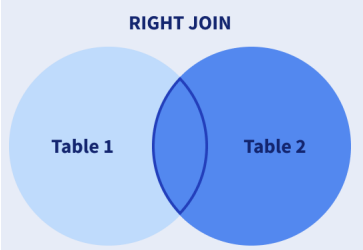
Select \* from table1 Natural JOIN table2;

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 :-

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName

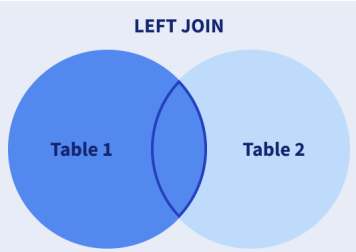
FROM Orders

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.



Example:-

SELECT Customers.CustomerName, Orders.OrderID

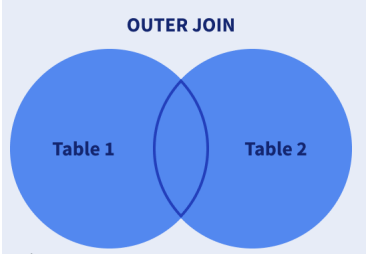
FROM Customers

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:-

SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

FULL JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

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.

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.