**1. History of SQL**

SQL was first developed in the early 1970s at IBM by a team led by Donald D. Chamberlin and Raymond F. Boyce. The original name of the language was SEQUEL, which stood for "Structured English Query Language." SEQUEL was designed to be a simple and intuitive way to access and manipulate data stored in IBM's System R relational database management system. SEQUEL was later renamed to SQL, and IBM released the first commercial implementation of the language in 1981 as part of their System R relational database management system (RDBMS).

SQL quickly became the standard language for interacting with relational databases, and it was soon adopted by other RDBMS vendors such as Oracle, Sybase, and Microsoft. The American National Standards Institute (ANSI) and the International Standards Organization (ISO) published official SQL standards in 1986 and 1987, respectively.

Over the next several decades, SQL evolved and expanded to include new features and capabilities. For example, the ANSI SQL-92 standard introduced support for triggers, stored procedures, and views. The SQL:1999 standard added support for recursive queries and user-defined data types.

In addition to these standardization efforts, SQL also saw the emergence of several popular open-source implementations of the language, such as MySQL and PostgreSQL. Oracle and Microsoft's implementations and commercial offerings are widely used today.

In recent years, SQL has been complemented by new technologies such as NoSQL databases and big data platforms like Hadoop and Spark. These technologies are optimized for handling large amounts of unstructured data and distributed computing, whereas SQL is optimized for structured data and single-machine computing.

Despite the rise of these newer technologies, SQL remains a crucial tool for managing and manipulating relational databases. It is widely used in various industries, including finance, healthcare, and retail. Additionally, SQL is often used with other technologies, such as data warehousing, business intelligence, and data mining, making it an essential tool for businesses to gain insights from their data.

In conclusion, SQL has a rich history and remains one of the most important and widely used programming languages for managing relational databases. From its initial development in the 1970s to its current state, SQL has evolved and expanded to meet the changing needs of businesses and organizations. It is a powerful tool that allows companies to gain insights from their data, and it will continue to play an essential role in the future of data management.

**2. Characteristics of SQL**

* SQL is easy to learn.
* SQL is used to access data from relational database management systems.
* SQL can execute queries against the database.
* SQL is used to describe the data.
* SQL is used to define the data in the database and manipulate it when needed.
* SQL is used to create and drop the database and table.
* SQL is used to create a view, stored procedure, function in a database.
* SQL allows users to set permissions on tables, procedures, and views.

**3. Advantages of SQL**

High speed

Using the SQL queries, the user can quickly and efficiently retrieve a large amount of records from a database.

No coding needed

In the standard SQL, it is very easy to manage the database system. It doesn't require a substantial amount of code to manage the database system.

Well defined standards

Long established are used by the SQL databases that are being used by ISO and ANSI.

### Portability

SQL can be used in laptop, PCs, server and even some mobile phones.

### Interactive language

SQL is a domain language used to communicate with the database. It is also used to receive answers to the complex questions in seconds.

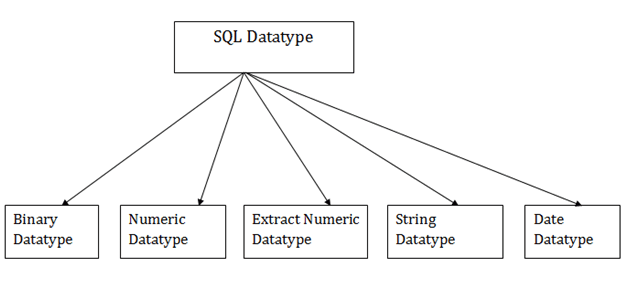
### Multiple data view

Using the SQL language, the users can make different views of the database structure.

# **4. SQL Datatype**

* SQL Datatype is used to define the values that a column can contain.
* Every column is required to have a name and data type in the database table.

## **Datatype of SQL:**



### 1. Binary Datatypes

There are Three types of binary Datatypes which are given below:

|  |  |
| --- | --- |
| **Data Type** | **Description** |
| Binary | It has a maximum length of 8000 bytes. It contains fixed-length binary data. |
| varbinary | It has a maximum length of 8000 bytes. It contains variable-length binary data. |
| Image | It has a maximum length of 2,147,483,647 bytes. It contains variable-length binary data. |

### 2. Approximate Numeric Datatype :

The subtypes are given below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Data type** | **From** | **To** | **Description** |
| Float | -1.79E + 308 | 1.79E + 308 | It is used to specify a floating-point value e.g. 6.2, 2.9 etc. |
| Real | -3.40e + 38 | 3.40E + 38 | It specifies a single precision floating point number |

### 3. Exact Numeric Datatype

The subtypes are given below:

|  |  |
| --- | --- |
| **Data type** | **Description** |
| Int | It is used to specify an integer value. |
| smallint | It is used to specify small integer value. |
| Bit | It has the number of bits to store. |
| decimal | It specifies a numeric value that can have a decimal number. |
| numeric | It is used to specify a numeric value. |

### 4. Character String Datatype

The subtypes are given below:

ADVERTISEMENT

|  |  |
| --- | --- |
| **Data type** | **Description** |
| Char | It has a maximum length of 8000 characters. It contains Fixed-length non-unicode characters. |
| Varchar | It has a maximum length of 8000 characters. It contains variable-length non-unicode characters. |
| Text | It has a maximum length of 2,147,483,647 characters. It contains variable-length non-unicode characters. |

### 5. Date and time Datatypes

The subtypes are given below:

|  |  |
| --- | --- |
| **Datatype** | **Description** |
| Date | It is used to store the year, month, and days value. |
| Time | It is used to store the hour, minute, and second values. |
| timestamp | It stores the year, month, day, hour, minute, and the second value. |

# 

# **5. SQL Commands**

* SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
* SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

## **Types of SQL Commands**

There are five types of SQL commands: DDL, DML, DCL, TCL, and DQL.



### 1. Data Definition Language (DDL)

* DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
* All the command of DDL are auto-committed that means it permanently save all the changes in the database.

Here are some commands that come under DDL:

* CREATE
* ALTER
* DROP
* TRUNCATE

**a. CREATE** It is used to create a new table in the database.

1. CREATE TABLE TABLE\_NAME (COLUMN\_NAME DATATYPES[,....]);

**Example:**

1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);

**b. DROP:** It is used to delete both the structure and record stored in the table.

**Syntax**

1. DROP TABLE table\_name;

**Example**

1. DROP TABLE EMPLOYEE;

**c. ALTER:** It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

**Syntax:**

To add a new column in the table

1. ALTER TABLE table\_name ADD column\_name COLUMN-definition;

To modify existing column in the table:

1. ALTER TABLE table\_name MODIFY(column\_definitions....);

**EXAMPLE**

1. ALTER TABLE STU\_DETAILS ADD(ADDRESS VARCHAR2(20));
2. ALTER TABLE STU\_DETAILS MODIFY (NAME VARCHAR2(20));

**d. TRUNCATE:** It is used to delete all the rows from the table and free the space containing the table.

**Syntax:**

1. TRUNCATE TABLE table\_name;

**Example:**

1. TRUNCATE TABLE EMPLOYEE;

2. Data Manipulation Language

* DML commands are used to modify the database. It is responsible for all form of changes in the database.
* The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

Here are some commands that come under DML:

* INSERT
* UPDATE
* DELETE

**a. INSERT:** The INSERT statement is a SQL query. It is used to insert data into the row of a table.

**Syntax:**

1. INSERT INTO TABLE\_NAME
2. (col1, col2, col3,.... col N)
3. VALUES (value1, value2, value3, .... valueN);

Or

1. INSERT INTO TABLE\_NAME
2. VALUES (value1, value2, value3, .... valueN);

**For example:**

1. INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");

**b. UPDATE:** This command is used to update or modify the value of a column in the table.

**Syntax:**

1. UPDATE table\_name SET [column\_name1= value1,...column\_nameN = valueN] [WHERE CONDITION]

**For example:**

1. UPDATE students
2. SET User\_Name = 'Sonoo'
3. WHERE Student\_Id = '3'

**c. DELETE:** It is used to remove one or more row from a table.

**Syntax:**

1. DELETE FROM table\_name [WHERE condition];

**For example:**

1. DELETE FROM javatpoint
2. WHERE Author="Sonoo";

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

* Grant
* Revoke

**a. Grant:** It is used to give user access privileges to a database.

**Example**

1. GRANT SELECT, UPDATE ON MY\_TABLE TO SOME\_USER, ANOTHER\_USER;

**b. Revoke:** It is used to take back permissions from the user.

**Example**

1. REVOKE SELECT, UPDATE ON MY\_TABLE FROM USER1, USER2;

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

* COMMIT
* ROLLBACK
* SAVEPOINT

**a. Commit:** Commit command is used to save all the transactions to the database.

**Syntax:**

1. COMMIT;

**Example:**

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;

**b. Rollback:** Rollback command is used to undo transactions that have not already been saved to the database.

**Syntax:**

1. ROLLBACK;

**Example:**

1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;

**c. SAVEPOINT:** It is used to roll the transaction back to a certain point without rolling back the entire transaction.

**Syntax:**

1. SAVEPOINT SAVEPOINT\_NAME;

# 6. Subquery

In SQL a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query. Important rules for Subqueries:

* You can place the Subquery in a number of SQL clauses: [WHERE](https://www.geeksforgeeks.org/sql-where-clause/) clause,[HAVING](https://www.geeksforgeeks.org/having-vs-where-clause/) clause, FROM clause. Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
* A subquery is a query within another query. The outer query is called as **main query** and inner query is called as**subquery**.
* The subquery generally executes first when the subquery doesn’t have any**co-relation** with the **main query**, when there is a co-relation the parser takes the decision **on the fly**on which query to execute on **precedence** and uses the output of the subquery accordingly.
* Subquery must be enclosed in parentheses.
* Subqueries are on the right side of the comparison operator.
* [ORDER BY](https://www.geeksforgeeks.org/sql-order-by/) command **cannot** be used in a Subquery. [GROUPBY](https://www.geeksforgeeks.org/sql-group-by/)command can be used to perform same function as ORDER BY command.
* Use single-row operators with singlerow Subqueries. Use multiple-row operators with multiple-row Subqueries.

**Syntax:** There is not any general syntax for Subqueries. However, Subqueries are seen to be used most frequently with SELECT statement as shown below:

SELECT column\_name

FROM table\_name

WHERE column\_name *expression operator*

( SELECT COLUMN\_NAME from TABLE\_NAME WHERE ... );

**Sample Table**:

DATABASE

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | ROLL\_NO | LOCATION | PHONE\_NUMBER |
| Ram | 101 | Chennai | 9988775566 |
| Raj | 102 | Coimbatore | 8877665544 |
| Sasi | 103 | Madurai | 7766553344 |
| Ravi | 104 | Salem | 8989898989 |
| Sumathi | 105 | Kanchipuram | 8989856868 |

STUDENT

|  |  |  |
| --- | --- | --- |
| NAME | ROLL\_NO | SECTION |
| Ravi | 104 | A |
| Sumathi | 105 | B |
| Raj | 102 | A |

**Sample Queries**

:

* To display NAME, LOCATION, PHONE\_NUMBER of the students from DATABASE table whose section is A

Select NAME, LOCATION, PHONE\_NUMBER from DATABASE

WHERE ROLL\_NO IN

(SELECT ROLL\_NO from STUDENT where SECTION=’A’);

* **Explanation :** First subquery executes “ SELECT ROLL\_NO from STUDENT where SECTION=’A’ ” returns ROLL\_NO from STUDENT table whose SECTION is ‘A’.Then outer-query executes it and return the NAME, LOCATION, PHONE\_NUMBER from the DATABASE table of the student whose ROLL\_NO is returned from inner subquery. Output:

|  |  |  |  |
| --- | --- | --- | --- |
| NAME | ROLL\_NO | LOCATION | PHONE\_NUMBER |
| Ravi | 104 | Salem | 8989898989 |
| Raj | 102 | Coimbatore | 8877665544 |

* Insert Query Example:

Table1: Student1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | ROLL\_NO | LOCATION | PHONE\_NUMBER |  |
| Ram | 101 | chennai | 9988773344 |  |
| Raju | 102 | coimbatore | 9090909090 |  |
| Ravi | 103 | salem | 8989898989 |  |

Table2: Student2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | ROLL\_NO | LOCATION | PHONE\_NUMBER |  |
| Raj | 111 | chennai | 8787878787 |  |
| Sai | 112 | mumbai | 6565656565 |  |
| Sri | 113 | coimbatore | 7878787878 |  |

* To insert Student2 into Student1 table:

INSERT INTO Student1 SELECT \* FROM Student2;

* Output:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | ROLL\_NO | LOCATION | PHONE\_NUMBER |  |
| Ram | 101 | chennai | 9988773344 |  |
| Raju | 102 | coimbatore | 9090909090 |  |
| Ravi | 103 | salem | 8989898989 |  |
| Raj | 111 | chennai | 8787878787 |  |
| Sai | 112 | mumbai | 6565656565 |  |
| Sri | 113 | coimbatore | 7878787878 |  |

* To delete students from Student2 table whose rollno is same as that in Student1 table and having location as chennai

DELETE FROM Student2

WHERE ROLL\_NO IN ( SELECT ROLL\_NO

FROM Student1

WHERE LOCATION = ’chennai’);

* Output:

1 row delete successfully.

* **Display Student2 table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | ROLL\_NO | LOCATION | PHONE\_NUMBER |  |
| Sai | 112 | mumbai | 6565656565 |  |
| Sri | 113 | coimbatore | 7878787878 |  |

* To update name of the students to geeks in Student2 table whose location is same as Raju,Ravi in Student1 table

UPDATE Student2

SET NAME=’geeks’

WHERE LOCATION IN ( SELECT LOCATION

FROM Student1

WHERE NAME IN (‘Raju’,’Ravi’));

* Output:

1 row updated successfully.

* **Display Student2 table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | ROLL\_NO | LOCATION | PHONE\_NUMBER |  |
| Sai | 112 | mumbai | 6565656565 |  |
| geeks | 113 | coimbatore | 7878787878 |  |

## **7. SQL Aggregate Functions**

An aggregate function is a function that performs a calculation on a set of values, and returns a single value.

Aggregate functions are often used with the GROUP BY clause of the SELECT statement. The GROUP BY clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

The most commonly used SQL aggregate functions are:

* MIN() - returns the smallest value within the selected column
* MAX() - returns the largest value within the selected column
* COUNT() - returns the number of rows in a set
* SUM() - returns the total sum of a numerical column
* AVG() - returns the average value of a numerical column

## **Aggregate Functions in SQL**

Below is the list of SQL aggregate functions, with examples

### ****Count():****

* ***Count(\*):*** Returns the total number of records .i.e 6.
* ***Count(salary):*** Return the number of Non-Null values over the column salary. i.e 5.
* ***Count(Distinct Salary):*** Return the number of distinct Non-Null values over the column salary .i.e 4

### ****Sum():****

* ***sum(salary):*** Sum all Non-Null values of Column salary i.e., 310
* ***sum(Distinct salary):***Sum of all distinct Non-Null values i.e., 250.

### ****Avg():****

* ***Avg(salary)*** = Sum(salary) / count(salary) = 310/5
* ***Avg(Distinct salary)*** = sum(Distinct salary) / Count(Distinct Salary) = 250/4

### ****Min():****

* ***Min(salary):***Minimum value in the salary column except NULL i.e., 40.

### Max():

* ***Max(salary):***Maximum value in the salary i.e., 80.

## Demo SQL Database

In this tutorial on aggregate functions, we will use the following table for examples:

| **Id** | **Name** | **Salary** |
| --- | --- | --- |
| 1 | A | 802 |
| 2 | B | 403 |
| 3 | C | 604 |
| 4 | D | 705 |
| 5 | E | 606 |
| 6 | F | NULL |

You can also create this table on your system, by writing the following queries:

MySQL

CREATE TABLE Employee (

Id INT PRIMARY KEY,

Name CHAR(1), -- Adjust data type and length if names can be longer than a single character

Salary DECIMAL(10,2) -- Adjust precision and scale if needed for salaries

);

INSERT INTO Employee (Id, Name, Salary)

VALUES (1, 'A', 802),

(2, 'B', 403),

(3, 'C', 604),

(4, 'D', 705),

(5, 'E', 606),

(6, 'F', NULL);

## Aggregate Function Example

In this example, we will use multiple aggregate functions on the data.

#### Queries

--Count the number of employees  
**SELECT COUNT**(\*) **AS** TotalEmployees **FROM** Employee;  
  
-- Calculate the total salary  
**SELECT** **SUM**(Salary) **AS** TotalSalary **FROM** Employee;  
  
-- Find the average salary  
**SELECT AVG**(Salary) **AS** AverageSalary **FROM** Employee;  
  
-- Get the highest salary  
**SELECT MAX(**Salary) **AS** HighestSalary **FROM** Employee;  
  
-- Determine the lowest salary  
**SELECT MIN**(Salary) **AS** LowestSalary **FROM** Employee;

#### Output

**TotalEmployees**  
6  
**TotalSalary**  
3120  
**AverageSalary**  
624  
**HighestSalary**  
802  
**LowestSalary**  
403

## **8. SQL JOIN**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Let's look at a selection from the "Orders" table:

|  |  |  |
| --- | --- | --- |
| **OrderID** | **CustomerID** | **OrderDate** |
| 10308 | 2 | 1996-09-18 |
| 10309 | 37 | 1996-09-19 |
| 10310 | 77 | 1996-09-20 |

Then, look at a selection from the "Customers" table:

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **CustomerName** | **ContactName** | **Country** |
| 1 | Alfreds Futterkiste | Maria Anders | Germany |
| 2 | Ana Trujillo Emparedados y helados | Ana Trujillo | Mexico |
| 3 | Antonio Moreno Taquería | Antonio Moreno | Mexico |

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

## **Different Types of SQL JOINs**

Here are the different types of the JOINs in SQL:

* (INNER) JOIN: Returns records that have matching values in both tables
* LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
* RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
* FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table

Top of Form

## **Test Yourself Wi**

# SQL | Join (Inner, Left, Right and Full Joins)

**SQL Join** statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

* INNER JOIN
* LEFT JOIN
* RIGHT JOIN
* FULL JOIN
* NATURAL JOIN

Consider the two tables below as follows:

**Student**



**StudentCourse**



The simplest Join is INNER JOIN.

### ****A. INNER JOIN****

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

**Syntax**:

SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1   
INNER JOIN table2  
ON table1.matching\_column = table2.matching\_column;  
  
  
**table1**: First table.  
**table2**: Second table  
**matching\_column**: Column common to both the tables.

***Note****: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.*



**Example Queries(INNER JOIN)**

This query will show the names and age of students enrolled in different courses.

SELECT StudentCourse.COURSE\_ID, Student.NAME, Student.AGE FROM Student  
INNER JOIN StudentCourse  
ON Student.ROLL\_NO = StudentCourse.ROLL\_NO;

**Output**:



### ****B. LEFT JOIN****

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain null. LEFT JOIN is also known as LEFT OUTER JOIN.

**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1   
LEFT JOIN table2  
ON table1.matching\_column = table2.matching\_column;  
  
  
table1: First table.  
table2: Second table  
matching\_column: Column common to both the tables.

***Note****: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are the same.*



**Example Queries(LEFT JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID   
FROM Student  
LEFT JOIN StudentCourse   
ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output**:



### ****C. RIGHT JOIN****

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result-set will contain null. RIGHT JOIN is also known as RIGHT OUTER JOIN.

**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1   
RIGHT JOIN table2  
ON table1.matching\_column = table2.matching\_column;  
  
  
table1: First table.  
table2: Second table  
matching\_column: Column common to both the tables.

***Note****: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are the same.*



**Example Queries(RIGHT JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID   
FROM Student  
RIGHT JOIN StudentCourse   
ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**



### ****D. FULL JOIN****

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.



**Syntax:**

SELECT table1.column1,table1.column2,table2.column1,....  
FROM table1   
FULL JOIN table2  
ON table1.matching\_column = table2.matching\_column;  
  
  
table1: First table.  
table2: Second table  
matching\_column: Column common to both the tables.

**Example Queries(FULL JOIN)**:

SELECT Student.NAME,StudentCourse.COURSE\_ID   
FROM Student  
FULL JOIN StudentCourse   
ON StudentCourse.ROLL\_NO = Student.ROLL\_NO;

**Output:**

| **NAME** | **COURSE\_ID** |
| --- | --- |
| HARSH | 1 |
| PRATIK | 2 |
| RIYANKA | 2 |
| DEEP | 3 |
| SAPTARHI | 1 |
| DHANRAJ | NULL |
| ROHIT | NULL |
| NIRAJ | NULL |
| NULL | 4 |
| NULL | 5 |
| NULL | 4 |

[Left JOIN (Video)](https://youtu.be/LCbO2U3jzU0)   
[Right JOIN (Video)](https://youtu.be/JOAe-yua6Jw)   
[Full JOIN (Video)](https://youtu.be/WmqAKSBupsE)   
[SQL | JOIN (Cartesian Join, Self Join)](https://www.geeksforgeeks.org/sql-join-cartesian-join-self-join/)

### E. Natural join (?)

Natural join can join tables based on the common columns in the tables being joined. A natural join returns all rows by matching values in common columns having same name and data type of columns and that column should be present in both tables.

Both table must have at list one common column with same column name and same data type.

The two table are joined using Cross join.

DBMS will look for a common column with same name and data type Tuples having exactly same values in common columns are kept in result.

Example:

| **Employee** | | |
| --- | --- | --- |
| **Emp\_id** | **Emp\_name** | **Dept\_id** |
| **1** | **Ram** | **10** |
| **2** | **Jon** | **30** |
| **3** | **Bob** | **50** |

| **Department** | |
| --- | --- |
| **Dept\_id** | **Dept\_name** |
| **10** | **IT** |
| **30** | **HR** |
| **40** | **TIS** |

Query: Find all Employees and their respective departments.

Solution: (Employee) ? (Department)

| **Emp\_id** | **Emp\_name** | **Dept\_id** | **Dept\_id** | **Dept\_name** |
| --- | --- | --- | --- | --- |
| **1** | **Ram** | **10** | **10** | **IT** |
| **2** | **Jon** | **30** | **30** | **HR** |
| **Employee data** | | | **Department data** | |

# 9. Privilege and Roles in DBMS

Confidentiality, integrity, and availability are the stamps of database security. Authorization is the allowance to the user or process to access the set of objects. The type of access granted can be any like, read-only, read, and write. Privilege means different [Data Manipulation Language(DML)](https://www.geeksforgeeks.org/difference-between-ddl-and-dml-in-dbms/) operations which can be performed by the user on data like INSERT, UPDATE, SELECT and DELETE, etc.   
There are two methods by which access control is performed is done by using the following.

1. Privileges
2. Roles

Let’s discuss one by one.

**Privileges :**  
The authority or permission to access a named object as advised manner, for example, permission to access a table. Privileges can allow permitting a particular user to connect to the database. In, other words privileges are the allowance to the database by the database object.

* **Database privileges —** A privilege is permission to execute one particular type of [SQL](https://www.geeksforgeeks.org/sql-tutorial/) statement or access a second persons’ object. Database privilege controls the use of computing resources. Database privilege does not apply to the Database administrator of the database.
* **System privileges —** A system privilege is the right to perform an activity on a specific type of object. for example, the privilege to delete rows of any table in a database is system privilege. There are a total of 60 different system privileges. System privileges allow users to CREATE, ALTER, or DROP the database objects.
* **Object privilege —** An object privilege is a privilege to perform a specific action on a particular table, function, or package. For example, the right to delete rows from a table is an object privilege. For example, let us consider a row of table GEEKSFORGEEKS that contains the name of the employee who is no longer a part of the organization, then deleting that row is considered as an object privilege. Object privilege allows the user to INSERT, DELETE, UPDATE, or SELECT the data in the database object

### Following are the differences between system privileges and object privileges.

| **Sr. No** | **System privileges** | **Object privileges** |
| --- | --- | --- |
| **1.** | This privileges is normally granted by a Database Administrative to users. | This privileges are granted by the owner of the object. |
| **2.** | This privileges are used to prevent or permit DDL statements such as create View, Table, session etc. | This privileges are used to prevent or permit DML statements such as Select, Insert, Update and Delete etc. |
| **3.** | This privileges allow the users to manage database and servers. | This privileges allows users to perform certain action upon database objects. |
| **4.** | Syntax:  Grant privileges to Username; | Syntax:  Grant privileges ON object TO username; |

**Roles :**  
A role is a mechanism that can be used to allow authorization. A person or a group of people can be allowed a role or group of roles. By many roles, the head can manage access privileges very easily. The roles are provided by the [database management system](https://www.geeksforgeeks.org/introduction-of-dbms-database-management-system-set-1/) for easy and managed or controlled privilege management.

**Properties –**   
The following are the properties of the roles which allow easy privilege management inside a database:

* **Reduced privilege administration —** The user can grant the privilege for a group of users who are related instead of granting the same set of privileges to the users explicitly.
* **Dynamic privilege management —** If the privilege of the group changes then, only the right of role needs to be changed.
* **Application-specific security —** The user can also protect the use of a role by using a password. Applications can be created to allow a role when entering the correct and best password. Users are not allowed the role if they do not know about the password

Bottom of Form

# 10. **SQL Set Operation**

The SQL Set operation is used to combine the two or more SQL SELECT statements.

## **Types of Set Operation**

1. Union
2. UnionAll
3. Intersect
4. Minus



### 1. Union

* The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
* In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
* The union operation eliminates the duplicate rows from its resultset.

**Syntax**

1. SELECT column\_name FROM table1
2. UNION
3. SELECT column\_name FROM table2;

**Example:**

**The First table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |

**The Second table**

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

Union SQL query will be:

1. SELECT \* FROM First
2. UNION
3. SELECT \* FROM Second;

The resultset table will look like:

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

2. Union All

Union All operation is equal to the Union operation. It returns the set without removing duplication and sorting the data.

**Syntax:**

1. SELECT column\_name FROM table1
2. UNION ALL
3. SELECT column\_name FROM table2;

**Example:** Using the above First and Second table.

ADVERTISEMENT

ADVERTISEMENT

Union All query will be like:

1. SELECT \* FROM First
2. UNION ALL
3. SELECT \* FROM Second;

The resultset table will look like:

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |
| 3 | Jackson |
| 3 | Jackson |
| 4 | Stephan |
| 5 | David |

3. Intersect

* It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
* In the Intersect operation, the number of datatype and columns must be the same.
* It has no duplicates and it arranges the data in ascending order by default.

**Syntax**

1. SELECT column\_name FROM table1
2. INTERSECT
3. SELECT column\_name FROM table2;

**Example:**

**Using the above First and Second table.**

Intersect query will be:

1. SELECT \* FROM First
2. INTERSECT
3. SELECT \* FROM Second;

The resultset table will look like:

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 3 | Jackson |

4. Minus

* It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
* It has no duplicates and data arranged in ascending order by default.

**Syntax:**

1. SELECT column\_name FROM table1
2. MINUS
3. SELECT column\_name FROM table2;

**Example**

**Using the above First and Second table.**

Minus query will be:

1. SELECT \* FROM First
2. MINUS
3. SELECT \* FROM Second;

The resultset table will look like:

|  |  |
| --- | --- |
| **ID** | **NAME** |
| 1 | Jack |
| 2 | Harry |