UNIT-3
SQL &PL/SQL

INTRODUCTION TO SQL:

SQL stands for Structured Query Language and is a computer language that we use to interact with a relational database. SQL is a tool for *organizing*, *managing*,and *retrieving* archived data from a computer database. The original name was given by IBM as Structured English Query Language, abbreviated by the acronym SEQUEL. When data needs to be retrieved from a database, SQL is used to make the request.

The DBMS processes the SQL query retrieves the requested data and returns it to us. Rather, SQL statements describe how a collection of data should be organized or what data should be extracted or added to the database.

SQL LANGUAGES:

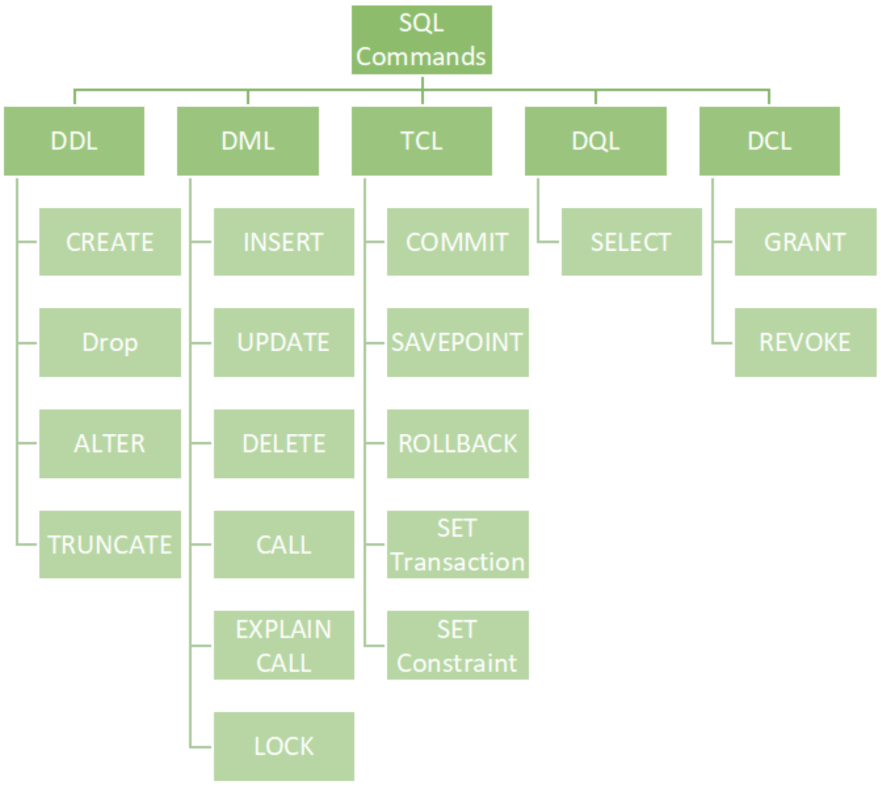
SQL uses certain commands like CREATE, DROP, INSERT, etc. to carry out the required tasks.

SQL commands are like instructions to a table. It is used to interact with the database with some operations. It is also used to perform specific tasks, functions, and queries of data. SQL can perform various tasks like creating a table, adding data to tables, dropping the table, modifying the table, set permission for users.

These SQL commands are mainly categorized into five categories:

1. DDL – Data Definition Language
2. DQL – Data Query Language
3. DML – Data Manipulation Language

4. DCL – Data Control Language
5. TCL – Transaction Control Language



Structured Query Language (SQL):

1. **Data Definition Language (DDL):** SQL provides a set of commands to define and modify the structure of a database, including creating tables, modifying table structure, and dropping tables.
2. **Data Manipulation Language (DML):** SQL provides a set of commands to manipulate data within a database, including adding, modifying, and deleting data.
3. **Query Language:** SQL provides a rich set of commands for querying a database to retrieve data, including the ability to filter, sort, group, and join data from multiple tables.

4. **Transaction Control**: SQL supports transaction processing, which allows users to group a set of database operations into a single transaction that can be rolled back in case of failure.
5. **Data Integrity:** SQL includes features to enforce data integrity, such as the ability to specify constraints on the values that can be inserted or updated in a table, and to enforce referential integrity between tables.
6. **User Access Control:** SQL provides mechanisms to control user access to a database, including the ability to grant and revoke privileges to perform certain operations on the database.

SQL is a standardized language, meaning that SQL code written for one database management system can be used on another system with minimal modification.

SQL contains of some important features and they are:

1. **Data Definition language (DDL):**
   It contains of commands which defines the data. The commands are:

- **create:** It is used to create a table.
  Syntax:

create table tablename(attribute1 datatype......attributen datatype);

- **drop:** It is used to delete the table including all the attributes.
  Syntax:

  drop table tablename;

- **alter:** alter is a reserve word which modifies the structure of the table.
  Syntax:

alter table tablename add(new column1 datatype......new columnx datatype);

rename: A table name can be changed using the reserver ‘rename’
Syntax:

rename old table name to new table name;

2. Data Manipulation Language (DML):

Data Manipulation Language contains commands used to manipulate the data.

The commands are:

- **insert:** This command is generally used after the create command to insert a set of values into the table.
  Syntax:

insert into tablename values(attribute1 datatype);

Ex-**insert into tablename values (attributen datatype);**

- **delete:** A command used to delete particular tuples or rows or cardinality from the table.
  Syntax:

delete from tablename where condition;

- **update:** It updates the tuples in a table.
  Syntax:

update tablename set tuplename='attributename';

3.Transaction Control Language:

Transactions are an important element of DBMS and to control the transactions, TCL is used which has commands like commit, rollback and savepoint.

- **commit:** It saves the database at any point whenever database is consistent.
  Syntax: commit;
- **rollback:** It rollbacks/undo to the previous point of the transaction.
  Syntax: rollback;

- **savepoint:** It goes back to the previous transaction without going back to the entire transaction.
  Syntax: savepoint;

SQL CLAUSES

- SQL clause helps us to retrieve a set or bundles of records from the table.
- SQL clause helps us to specify a condition on the columns or the records of a table.

Different clauses available in the Structured Query Language are as follows:

1. WHERE CLAUSE
2. GROUP BY CLAUSE
3. HAVING CLAUSE
4. ORDER BY CLAUSE

Let's see each clause one by one with an example. We will use MySQL database for writing the queries in examples.

1. WHERE CLAUSE

A WHERE clause in SQL is used with the SELECT query, which is one of the data manipulation language commands. WHERE clauses can be used to limit the number of rows to be displayed in the result set, it generally helps in filtering the records. It returns only those queries which fulfill the specific conditions of the WHERE clause. WHERE clause is used in SELECT, UPDATE, DELETE statement, etc.

WHERE clause with SELECT Query

Asterisk symbol is used with a WHERE clause in a SELECT query to retrieve all the column values for every record from a table.

Syntax of where clause with a select query to retrieve all the column values for every record from a table:

1. **SELECT** * **FROM** TABLENAME **WHERE** CONDITION;

If according to the requirement, we only want to retrieve selective columns, then we will use below syntax:

1. **SELECT** COLUMNNAME1, COLUMNNAME2 **FROM** TABLENAME **WHERE** CONDITION;

Consider the employee table with the following data:

E_ID	Name	Salary	City	Designation	Date_of_Joining	Age
1	Sakshi Kumari	50000	Mumbai	Project Manager	2021-06-20	24
2	Tejaswini Naik	75000	Delhi	System Engineer	2019-12-24	23
3	AnujaShar	40000	Jaipur	Manager	2021-08-15	26
4	Anushka Tripathi	90000	Mumbai	Software Tester	2021-06-13	24
5	Rucha Jagtap	45000	Bangalore	Project Manager	2020-08-09	23

Example 2:

Write a query to update the employee's record and set the updated name as 'Harshada Sharma' where the employee's city name is Jaipur.

Query:

```
mysql> UPDATE employees SET Name = "Harshada Sharma" WHER
E City = "Jaipur";
```

Example 3:

Write a query to delete an employee's record where the employee's joining date is "2013-12-12".

Query:

```
mysql> DELETE FROM employees WHERE Date_of_Joining = "2013-
12-12";
```

2. GROUP BY CLAUSE

The Group By clause is used to arrange similar kinds of records into the groups in the Structured Query Language. The Group by clause in the Structured Query Language is used with Select Statement. Group by clause is placed after the where clause in the SQL statement. The Group By clause is specially used with the aggregate function, i.e., max (), min (), avg (), sum (), count () to group the result based on one or more than one column.

The syntax of Group By clause:

1. SELECT * FROM TABLENAME GROUP BY COLUMNNAME;

The above syntax will select all the data or records from the table, but it will arrange all those data or records in the groups based on the column name given in the query.

The syntax of Group By clause with Aggregate Functions:

SELECT COLUMNNAME1, Aggregate_FUNCTION (COLUMNNAME) FROM TABLENAME GROUP BY COLUMNNAME;

Let's understand the Group By clause with the help of examples.

Consider the employees table with the following data:

E_ID	Name	Salary	City	Designation	Date_of_Joining	Age
1	Sakshi Kumari	50000	Mumbai	Project Manager	2021-06-20	24
2	Tejaswini Naik	75000	Delhi	System Engineer	2019-12-24	23
3	Anuja Sharma	40000	Jaipur	Manager	2021-08-15	26
4	Anushka Tripathi	90000	Mumbai	Software Tester	2021-06-13	24
5	Rucha Jagtap	45000	Bangalore	Project Manager	2020-08-09	23

Write a query to display all the records of the employees table but group the results based on the age column.

Query:

```
mysql> SELECT * FROM employees GROUP BY Age;
```

The above query will display all the records of the employees table but grouped by the age column.

You will get the following output:

3. HAVING CLAUSE:

When we need to place any conditions on the table's column, we use the WHERE clause in SQL. But if we want to use any condition on a column in Group By clause at that time, we will use the HAVING clause with the Group By clause for column conditions.

Syntax:

```
syn:TABLENAME GROUP BY COLUMNNAME HAVING CONDITION;
```

Consider the employees table with the following data:

E_ID	Name	Salary	City	Designation	Date_of_Joining	Age
1	Sakshi Kumari	50000	Mumbai	Project Manager	2021-06-20	24
2	Tejaswini Naik	75000	Delhi	System Engineer	2019-12-24	23
3	Anuja Sharma	40000	Jaipur	Manager	2021-08-15	26
4	Anushka Tripathi	90000	Mumbai	Software Tester	2021-06-13	24

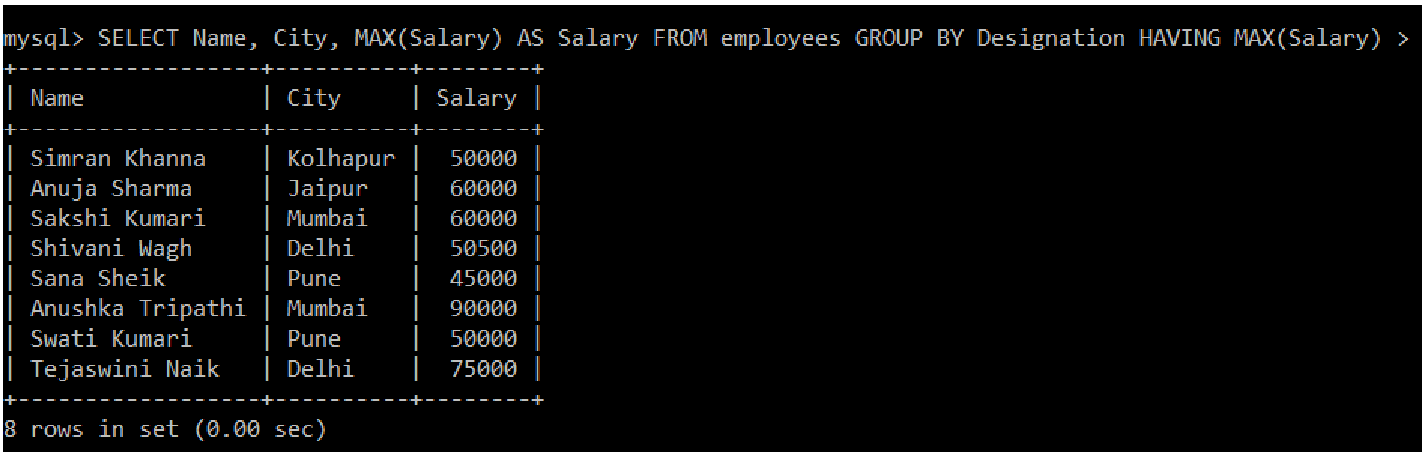
5	Rucha Jagtap	45000	Bangalore	Project Manager	2020-08-09	23

Example 1:

Write a query to display the name of employees, salary, and city where the employee's maximum salary is greater than 40000 and group the results by designation.

mysql> **SELECT Name**, City, **MAX** (Salary) **AS** Salary **FROM** employees **GROUP BY** Designation **HAVING MAX** (Salary) > 40000;

You will get the following output:



```
mysql> SELECT Name, City, MAX(Salary) AS Salary FROM employees GROUP BY Designation HAVING MAX(Salary) >
+------------------+----------+--------+
| Name             | City     | Salary |
+------------------+----------+--------+
| Simran Khanna    | Kolhapur |  50000 |
| Anuja Sharma     | Jaipur   |  60000 |
| Sakshi Kumari    | Mumbai   |  60000 |
| Shivani Wagh     | Delhi    |  50500 |
| Sana Sheik       | Pune     |  45000 |
| Anushka Tripathi | Mumbai   |  90000 |
| Swati Kumari     | Pune     |  50000 |
| Tejaswini Naik   | Delhi    |  75000 |
+------------------+----------+--------+
8 rows in set (0.00 sec)
```

The above output shows that the employee name, salary, and city of an employee where employee salary is greater than 40000 grouped by designation. (Employees with a similar designation are placed in one group, and those with other designation are placed separately).

Example 2:

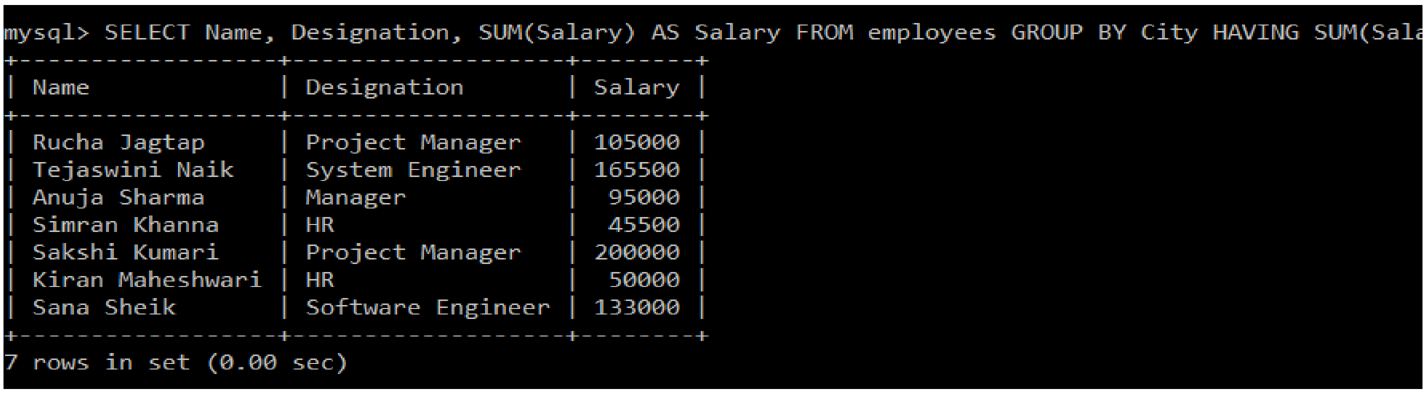
Write a query to display the name of employees and designation where the sum of an employee's salary is greater than 45000 and group the results by city.

Query:

mysql> **SELECT Name**, Designation, SUM (Salary) **AS** Salary **FROM** employees

GROUP BY City **HAVING** SUM (Salary) > 45000;

You will get the following output:



```
mysql> SELECT Name, Designation, SUM(Salary) AS Salary FROM employees GROUP BY City HAVING SUM(Sala
+------------------+-------------------+--------+
| Name             | Designation       | Salary |
+------------------+-------------------+--------+
| Rucha Jagtap     | Project Manager   | 105000 |
| Tejaswini Naik   | System Engineer   | 165500 |
| Anuja Sharma     | Manager           |  95000 |
| Simran Khanna    | HR                |  45500 |
| Sakshi Kumari    | Project Manager   | 200000 |
| Kiran Maheshwari | HR                |  50000 |
| Sana Sheik       | Software Engineer | 133000 |
+------------------+-------------------+--------+
7 rows in set (0.00 sec)
```

The above output shows the employee name, designation, and salary of an employee. The sum of salary is greater than 45000 grouped by city. (Employees with similar cities are placed in one group and those with a different city are not similar are placed separately).

4. ORDER BY CLAUSE

Whenever we want to sort anything in SQL, we use the ORDER BY clause.

The ORDER BY clause in SQL will help us to sort the data based on the specific column of a table. This means that all the data stored in the specific column on which we are executing the ORDER BY clause will be sorted.

As we all know, sorting means either in ASCENDING ORDER or DESCENDING ORDER. In the same way, ORDER BY CLAUSE sorts the data in ascending or descending order as per our requirement. The data will be sorted in ascending order whenever the **ASC keyword** is used with ORDER by clause, and the **DESC keyword** will sort the records in descending order.

By default, sorting in the SQL will be done using the ORDER BY clause in ASCENDING order if we didn't mention the sorting order.

Before moving towards the example of the ORDER BY clause to sort the records, first, we will look at syntax so it will be easy for us to go through the example.

Syntax of ORDER BY clause without asc and desc keyword:

syn:-
SELECT COLUMN_NAME1, COLUMN_NAME2 **FROM** TABLE_NAME **ORDER BY** COLUMNAME;

Syntax of ORDER BY clause to sort in ascending order:

Syn:-
SELECT COLUMN_NAME1, COLUMN_NAME2 **FROM** TABLE_NAME **ORDER BY** COLUMN_NAME **ASC**;

Syntax of ORDER BY clause to sort in descending order:

SELECT COLUMN_NAME1, COLUMN_NAME2 **FROM** TABLE_NAME **ORDER BY** COLUMN_NAME **DESC**;

Consider we have an employees table with the following data:

E_ID	Name	Salary	City	Designation	Date_of_Joining	Age
1	Sakshi Kumari	50000	Mumbai	Project Manager	2021-06-20	24
2	Tejaswini Naik	75000	Delhi	System Engineer	2019-12-24	23
3	Anuja Sharma	40000	Jaipur	Manager	2021-08-15	26
4	Anushka	900	Mum	Software	2021-06-	24

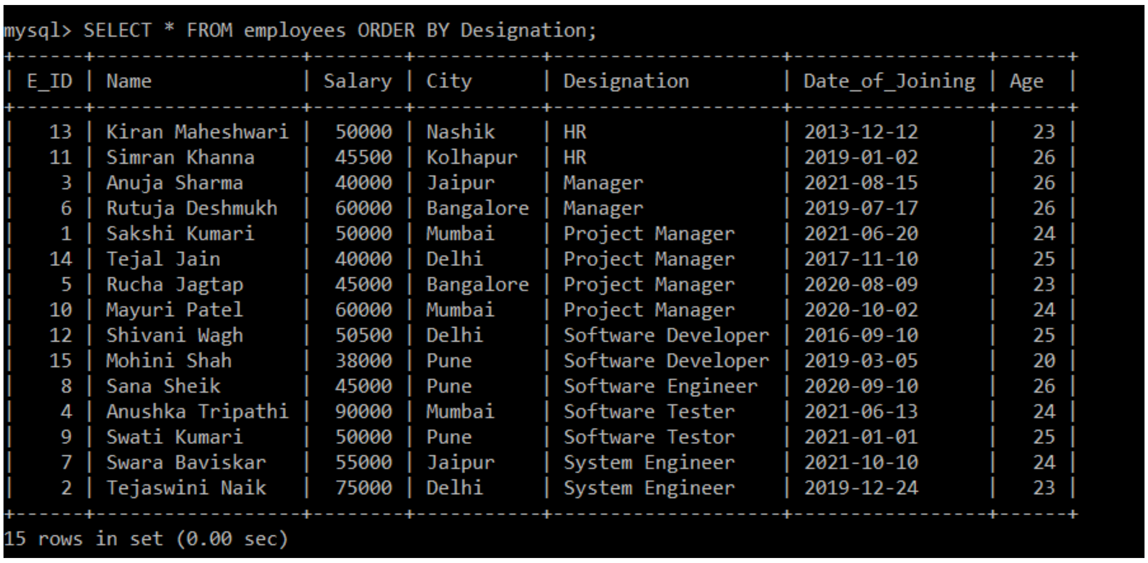
	Tripathi	00	bai	Tester	13	
5	Rucha Jagtap	45000	Bangalore	Project Manager	2020-08-09	23

Example 1:

Write a query to sort the records in the ascending order of the employee designation from the employees table.

Query:

mysql> **SELECT** * **FROM** employees **ORDER BY** Designation; Here in a SELECT query, an ORDER BY clause is applied on the column 'Designation' to sort the



```
mysql> SELECT * FROM employees ORDER BY Designation;
+------+-----------------+--------+-----------+-------------------+-----------------+------+
| E_ID | Name            | Salary | City      | Designation       | Date_of_Joining | Age  |
+------+-----------------+--------+-----------+-------------------+-----------------+------+
|   13 | Kiran Maheshwari |  50000 | Nashik    | HR                | 2013-12-12      |   23 |
|   11 | Simran Khanna   |  45500 | Kolhapur  | HR                | 2019-01-02      |   26 |
|    3 | Anuja Sharma    |  40000 | Jaipur    | Manager           | 2021-08-15      |   26 |
|    6 | Rutuja Deshmukh |  60000 | Bangalore | Manager           | 2019-07-17      |   26 |
|    1 | Sakshi Kumari   |  50000 | Mumbai    | Project Manager   | 2021-06-20      |   24 |
|   14 | Tejal Jain      |  40000 | Delhi     | Project Manager   | 2017-11-10      |   25 |
|    5 | Rucha Jagtap    |  45000 | Bangalore | Project Manager   | 2020-08-09      |   23 |
|   10 | Mayuri Patel    |  60000 | Mumbai    | Project Manager   | 2020-10-02      |   24 |
|   12 | Shivani Wagh    |  50500 | Delhi     | Software Developer | 2016-09-10      |   25 |
|   15 | Mohini Shah     |  38000 | Pune      | Software Developer | 2019-03-05      |   20 |
|    8 | Sana Sheik      |  45000 | Pune      | Software Engineer | 2020-09-10      |   26 |
|    4 | Anushka Tripathi |  90000 | Mumbai    | Software Tester   | 2021-06-13      |   24 |
|    9 | Swati Kumari    |  50000 | Pune      | Software Testor   | 2021-01-01      |   25 |
|    7 | Swara Baviskar  |  55000 | Jaipur    | System Engineer   | 2021-10-10      |   24 |
|    2 | Tejaswini Naik  |  75000 | Delhi     | System Engineer   | 2019-12-24      |   23 |
+------+-----------------+--------+-----------+-------------------+-----------------+------+
15 rows in set (0.00 sec)
```

records, but we didn't use the ASC keyword after the ORDER BY clause to sort in ascending order.

As per the expected output, the records are displayed in ascending order of the employee's designation.

Example 2:

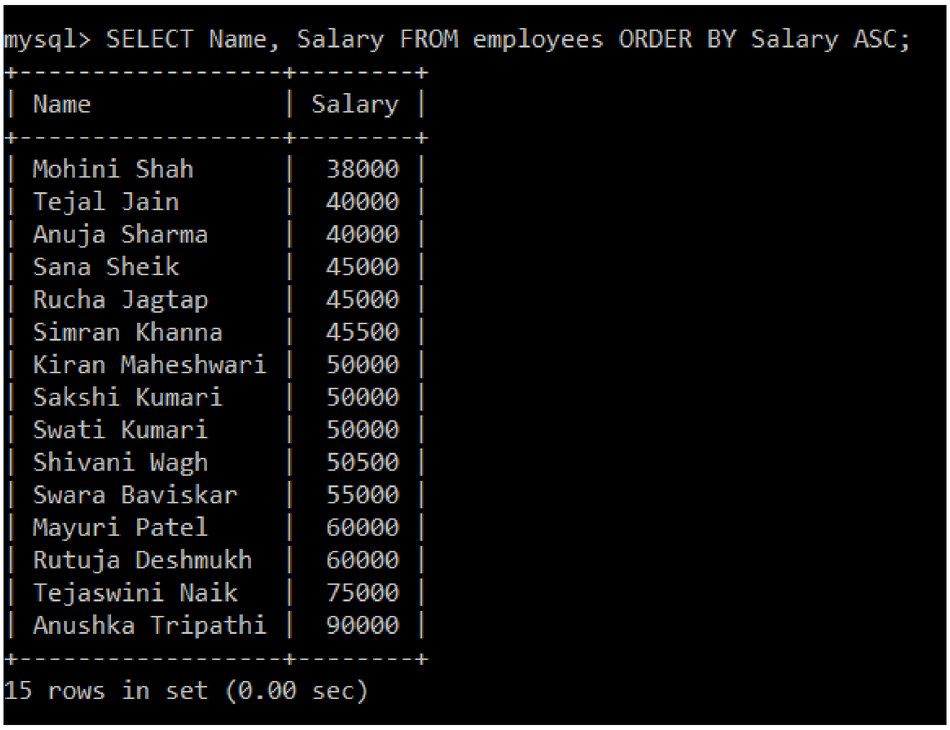
Write a query to display employee name and salary in the ascending order of the employee's salary from the employees table.

Query:

mysql> **SELECT Name**, Salary **FROM** employees **ORDER BY** Salary **ASC**;

Here in a SELECT query, an ORDER BY clause is applied to the 'Salary' column to sort the records. We have used the ASC keyword to sort the employee's salary in ascending order.

You will get the following output:



```
mysql> SELECT Name, Salary FROM employees ORDER BY Salary ASC;
+------------------+--------+
| Name             | Salary |
+------------------+--------+
| Mohini Shah      |  38000 |
| Tejal Jain       |  40000 |
| Anuja Sharma     |  40000 |
| Sana Sheik       |  45000 |
| Rucha Jagtap     |  45000 |
| Simran Khanna    |  45500 |
| Kiran Maheshwari |  50000 |
| Sakshi Kumari    |  50000 |
| Swati Kumari     |  50000 |
| Shivani Wagh     |  50500 |
| Swara Baviskar   |  55000 |
| Mayuri Patel     |  60000 |
| Rutuja Deshmukh  |  60000 |
| Tejaswini Naik   |  75000 |
| Anushka Tripathi |  90000 |
+------------------+--------+
15 rows in set (0.00 sec)
```

All the records are displayed in the ascending order of the employee's salary.

Example 3:

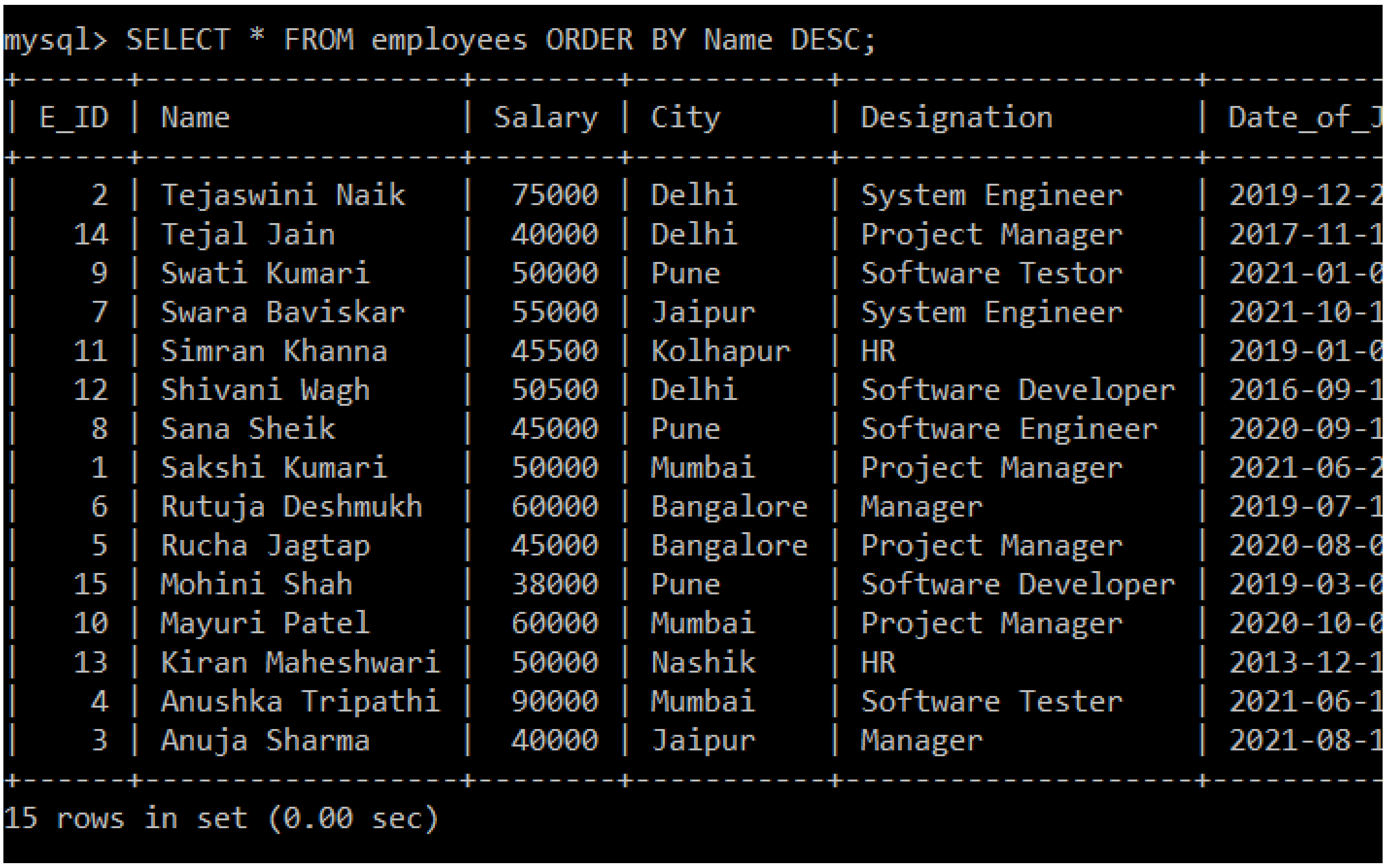
Write a query to sort the data in descending order of the employee name stored in the employees table.

Query:

mysql> **SELECT** * **FROM** employees **ORDER BY Name DESC**;

Here we have used the ORDER BY clause with the SELECT query applied on the Name column to sort the data. We have used the DESC keyword after the ORDER BY clause to sort data in descending order.

You will get the following output:



```
mysql> SELECT * FROM employees ORDER BY Name DESC;
+------+------------------+--------+-----------+--------------------+----------
| E_ID | Name             | Salary | City      | Designation        | Date_of_J
+------+------------------+--------+-----------+--------------------+----------
|    2 | Tejaswini Naik   |  75000 | Delhi     | System Engineer    | 2019-12-2
|   14 | Tejal Jain       |  40000 | Delhi     | Project Manager    | 2017-11-1
|    9 | Swati Kumari     |  50000 | Pune      | Software Testor    | 2021-01-0
|    7 | Swara Baviskar   |  55000 | Jaipur    | System Engineer    | 2021-10-1
|   11 | Simran Khanna    |  45500 | Kolhapur  | HR                 | 2019-01-0
|   12 | Shivani Wagh     |  50500 | Delhi     | Software Developer | 2016-09-1
|    8 | Sana Sheik       |  45000 | Pune      | Software Engineer  | 2020-09-1
|    1 | Sakshi Kumari    |  50000 | Mumbai    | Project Manager    | 2021-06-2
|    6 | Rutuja Deshmukh  |  60000 | Bangalore | Manager            | 2019-07-1
|    5 | Rucha Jagtap     |  45000 | Bangalore | Project Manager    | 2020-08-0
|   15 | Mohini Shah      |  38000 | Pune      | Software Developer | 2019-03-0
|   10 | Mayuri Patel     |  60000 | Mumbai    | Project Manager    | 2020-10-0
|   13 | Kiran Maheshwari |  50000 | Nashik    | HR                 | 2013-12-1
|    4 | Anushka Tripathi |  90000 | Mumbai    | Software Tester    | 2021-06-1
|    3 | Anuja Sharma     |  40000 | Jaipur    | Manager            | 2021-08-1
+------+------------------+--------+-----------+--------------------+----------
15 rows in set (0.00 sec)
```

All

SQL Operators:

Structured Query Language is a computer language that we use to interact with a relational database.Operators are the foundation of any programming language. We can define operators as symbols that help us to perform specific mathematical and logical computations on operands.
SQL operators have three different categories.

1.Arithmetic Operators

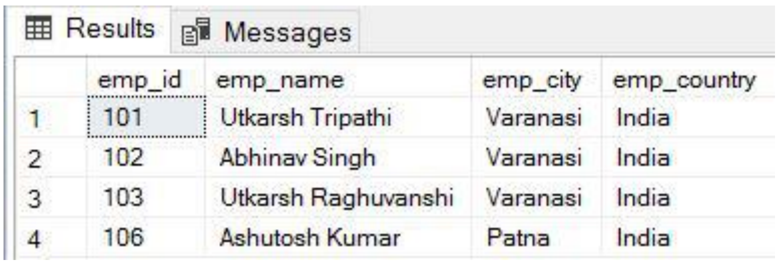
We can use various arithmetic operators on the data stored in the tables. Arithmetic Operators are:

Operator	Description
+	The addition is used to perform an addition operation on the data values.
–	This operator is used for the subtraction of the data values.
/	This operator works with the 'ALL' keyword and it calculates division operations.
*	This operator is used for multiplying data values.
%	Modulus is used to get the remainder when data is divided by another.

Example Query:

SELECT * FROM employee WHERE emp_city NOT LIKE 'A%';

Output:



```
   emp_id  emp_name             emp_city  emp_country
1  101     Utkarsh Tripathi     Varanasi  India
2  102     Abhinav Singh        Varanasi  India
3  103     Utkarsh Raghuvanshi  Varanasi  India
4  106     Ashutosh Kumar       Patna     India
```

2.Comparison Operators

Another important operator in SQL is a comparison operator, which is used to compare one expression's value to other expressions. SQL supports different types of comparison operator, which is described below:

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

Example Query:

SELECT * FROM MATHS WHERE MARKS=50;

Output:

3.Logical Operators

The Logical operators are those that are true or false. They return true or false values to combine one or more true or false values.

Operator	Description
AND	Logical AND compares two Booleans as expressions and returns true when both expressions are true.

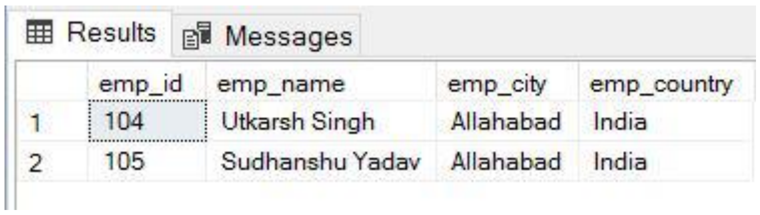
Operator	Description
OR	Logical OR compares two Booleans as expressions and returns true when one of the expressions is true.
NOT	Not takes a single Boolean as an argument and change its value from false to true or from true to false.

Example Query:

SELECT * FROM employee WHERE emp_city =

'Allahabad' AND emp_country = 'India';

Output:



```
   emp_id  emp_name         emp_city   emp_country
1  104     Utkarsh Singh    Allahabad  India
2  105     Sudhanshu Yadav  Allahabad  India
```

5. Special Operators

Operators	Description
ALL	ALL is used to select all records of a SELECT STATEMENT. It compares a value to every value in a list of results from a query. The ALL must be preceded by the comparison operators and evaluated to TRUE if the query returns no rows.
ANY	ANY compares a value to each value in a list of results from a query and evaluates to true if the result of an inner query contains at least one

Operators	Description
	row.
BETWEEN	The SQL BETWEEN operator tests an expression against a range. The range consists of a beginning, followed by an AND keyword and an end expression.
IN	The IN operator checks a value within a set of values separated by commas and retrieves the rows from the table that match.
EXISTS	The EXISTS checks the existence of a result of a subquery. The EXISTS subquery tests whether a subquery fetches at least one row. When no data is returned then this operator returns 'FALSE'.

SQL 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

Consider the two tables below as follows:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

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.

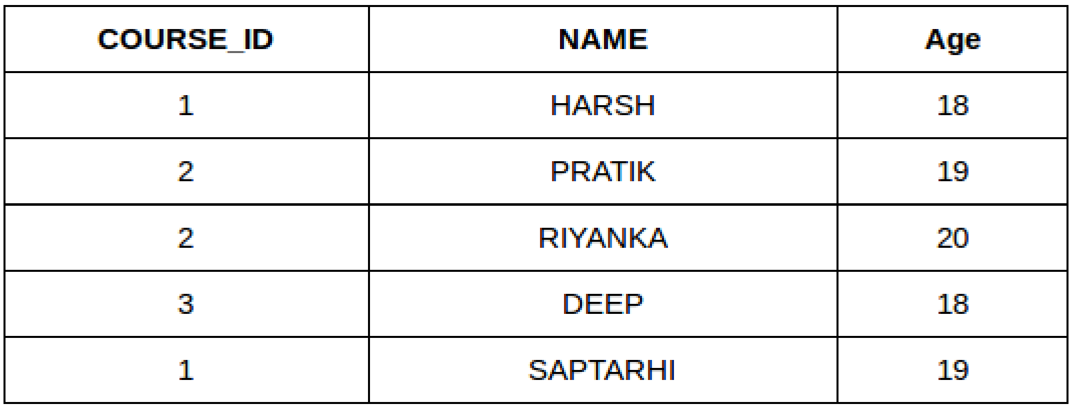
> ***Note**: We can also write JOIN instead of INNER JOIN. JOIN is same as INNER JOIN.*

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:



COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

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;
```

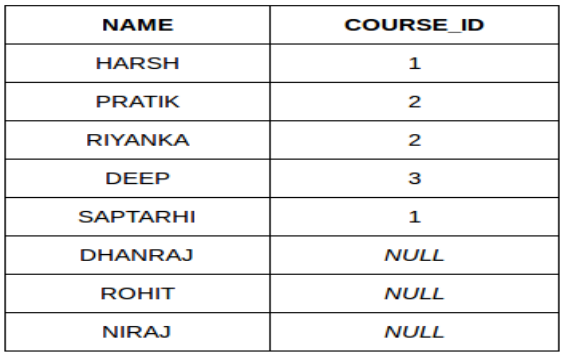
table1: First table.
table2: Second table
matching_column: Column common to both the tables.

> ***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:



NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

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;
```

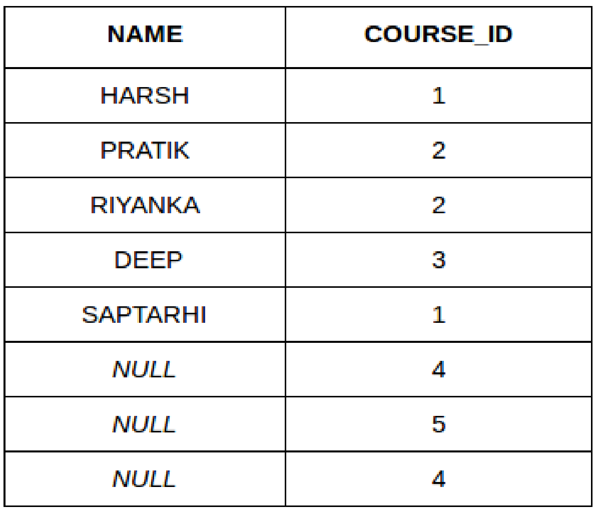
table1: First table.
table2: Second table
matching_column: Column common to both the tables.

> ***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:



NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

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

NAME	COURSE_ID
DEEP	3
SAPTARHI	1
DHANRAJ	NULL
ROHIT	NULL
NIRAJ	NULL
NULL	4
NULL	5
NULL	4

5. Cross Join:

- Join operation in SQL is used to combine multiple tables together into a single table.
- If we use the cross join to combine two different tables, then we will get the Cartesian product of the sets of rows from the joined table. When each row of the first table is combined with each row from the second table, it is known as Cartesian join or cross join.
- After performing the cross join operation, the total number of rows present in the final table will be equal to the product of the number of rows present in table 1 and the number of rows present in table 2.

- Let us take a look at the syntax of writing a query to perform the cross join operation in SQL.

SYN:-
SELECT TableName1.columnName1, TableName2.columnName2 **FROM** TableName1 CROSS JOIN TableName2 **ON** TableName1.ColumnName = TableName2.ColumnName;

EX-SELECT * **FROM** MatchScore CROSS JOIN Departments;

After executing this query, you will find the following result:

Player	Department_id	Goals	Depatment_id	Department_name
Franklin	1	2	1	IT
Alan	1	3	1	IT
Priyanka	2	2	1	IT
Rajesh	3	5	1	IT
Franklin	1	2	2	HR
Alan	1	3	2	HR
Priyanka	2	2	2	HR
Rajesh	3	5	2	HR
Frankli	1	2	3	Marketing

n				
Alan	1	3	3	Marketing
Priyanka	2	2	3	Marketing
Rajesh	3	5	3	Marketing

SET Operations in SQL:

SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

1.UNION Operation

UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its resultset. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.

Example of UNION

The **First** table,

ID	Name
1	abhi
2	adam

The **Second** table,

ID	Name
2	adam
3	Chester

Union SQL query will be,

```
SELECT * FROM First UNION SELECT * FROM Second;
```

The resultset table will look like,

ID	NAME
1	abhi
2	adam
3	Chester

2.UNION ALL

This operation is similar to Union. But it also shows the duplicate rows.

Example of Union All

Union All query will be like,

```
SELECT * FROM First UNION ALL SELECT * FROM Second;
```

Copy
The resultset table will look like,

ID	NAME
1	abhi
2	adam
2	adam
3	Chester

3.INTERSECT

Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same.

NOTE: MySQL does not support INTERSECT operator.
Intersect query will be,

```
SELECT * FROM First  INTERSECT SELECT * FROM Second;
```

Copy
The resultset table will look like

ID	NAME
2	adam

4.MINUS

The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.

Minus query will be,

```
SELECT * FROM First

MINUS

SELECT * FROM Second;
```

Copy
The resultset table will look like,

ID	NAME
1	abhi

NULL VALUES:-

In SQL there may be some records in a table that do not have values or data for every field and those fields are termed as a NULL value.

NULL values could be possible because at the time of data entry information is not available. So SQL supports a special value known as NULL which is used to represent the values of attributes that may be unknown or not apply to a tuple. SQL places a NULL value in the field in the absence of a user-defined value

Importance of NULL Value

- It is important to understand that a NULL value differs from a zero value.
- A NULL value is used to represent a missing value, but it usually has one of three different interpretations:
- The value unknown (value exists but is not known)
- Value not available (exists but is purposely withheld)
- Attribute not applicable (undefined for this tuple)

Principles of NULL values

- Setting a NULL value is appropriate when the actual value is unknown, or when a value is not meaningful.
- A NULL value is not equivalent to a value of ZERO if the data type is a number and is not equivalent to spaces if the data type is a character.
- A NULL value can be inserted into columns of any data type.
- A NULL value will evaluate NULL in any expression.
- Suppose if any column has a NULL value, then UNIQUE, FOREIGN key, and CHECK constraints will ignore by SQL.

Views in SQL

- Views in SQL are considered as a virtual table. A view also contains rows and columns.
- To create the view, we can select the fields from one or more tables present in the database.
- A view can either have specific rows based on certain condition or all the rows of a table.

Advantages of View:

1. **Complexity:** Views help to reduce the complexity. Different views can be created on the same base table for different users.
2. **Security:** It increases the security by excluding the sensitive information from the view.
3. **Query Simplicity:** It helps to simplify commands from the user. A view can draw data from several different tables and present it as a single table.
4. **Consistency:** A view can present a consistent, unchanged image of the structure of the database. Views can be used to rename the columns without affecting the base table.
5. **Data Integrity:** If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets the specified integrity constraints.
6. **Storage Capacity:** Views take very little space to store the data.
7. **Logical Data Independence:** View can make the application and database tables to a certain extent independent.

Disadvantages of View:

The DML statements which can be performed on a view created using single base table have certain restrictions are:

1. You cannot INSERT if the base table has any not null column that do not appear in view.
2. You cannot INSERT or UPDATE if any of the column referenced in the INSERT or UPDATE contains group functions or columns defined by expression.

3. You can't execute INSERT, UPDATE, DELETE statements on a view if with read only option is enabled.
4. You can't be created view on temporary tables.
5. You cannot INSERT, UPDATE, DELETE if the view contains group functions GROUP BY, DISTINCT or a reference to a psuedocolumn rownum.

Sample table:

Student_Detail

STU_ID	NAME	ADDRESS
1	Stephan	Delhi
2	Kathrin	Noida
3	David	Ghaziabad
4	Alina	Gurugram

Student_Marks

STU_ID	NAME	MARKS	AGE
1	Stephan	97	19
2	Kathrin	86	21
3	David	74	18

4	Alina	90	20
5	John	96	18

1. Creating view

A view can be created using the **CREATE VIEW** statement. We can create a view from a single table or multiple tables.

ADVERTISEMENT

Syntax:

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

2. Creating View from a single table

In this example, we create a View named DetailsView from the table Student_Detail.

Query:

```
CREATE VIEW DetailsView AS
SELECT NAME, ADDRESS
FROM Student_Details
WHERE STU_ID < 4;
```

Just like table query, we can query the view to view the data.

```
SELECT * FROM DetailsView;
```

Output:

NAME	ADDRESS
Stephan	Delhi
Kathrin	Noida
David	Ghaziabad

ADVERTISEMENT

3. Creating View from multiple tables

View from multiple tables can be created by simply include multiple tables in the SELECT statement.

In the given example, a view is created named MarksView from two tables Student_Detail and Student_Marks.

Query:

```
CREATE VIEW MarksView AS
SELECT Student_Detail.NAME, Student_Detail.ADDRESS, Student_Marks.MARKS FROM Student_Detail, Student_Mark
WHERE Student_Detail.NAME = Student_Marks.NAME;
```

To display data of View MarksView:

```
SELECT * FROM MarksView;
```

NAME	ADDRESS	MARKS
Stephan	Delhi	97
Kathrin	Noida	86
David	Ghaziabad	74
Alina	Gurugram	90

4. Deleting View

A view can be deleted using the Drop View statement.

Syntax

```
1. DROP VIEW view_name;
```

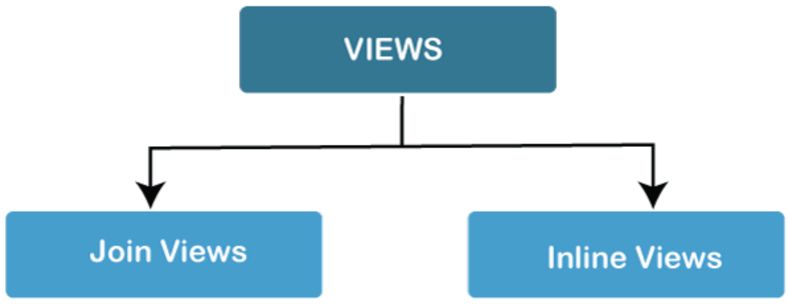
Example:

If we want to delete the View **MarksView**, we can do this as:

```
1. DROP VIEW MarksView;
```

Types of Views:

There are two types of views.



1. **Join View:** A join view is a view that has more than one table or view in its from clause and it does not use any Group by Clause, Rownum, Distinct and set operation.
2. **Inline View:** An inline view is a view which is created by replacing a subquery in the from clause which defines the data source that can be referenced in the main query. The sub query must be given an alias for efficient working.

SUBQUERIES IN SQL:

A Subquery is a query within another SQL query and embedded within the WHERE clause.

Important Rule:

- A subquery can be placed in a number of SQL clauses like WHERE clause, FROM clause, HAVING clause.
- You can use Subquery with SELECT, UPDATE, INSERT, DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.
- A subquery is a query within another query. The outer query is known as the main query, and the inner query is known as a subquery.
- Subqueries are on the right side of the comparison operator.
- A subquery is enclosed in parentheses.
- In the Subquery, ORDER BY command cannot be used. But GROUP BY command can be used to perform the same function as ORDER BY command.

1. Subqueries with the Select Statement

SQL subqueries are most frequently used with the Select statement.

Syntax

```
SELECT column_name
1. FROM table_name
2. WHERE column_name expression operator
3. ( SELECT column_name  from table_name WHERE ... );
```

Example

Consider the EMPLOYEE table have the following records:

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
6	Harry	42	China	4500.00
7	Jackson	25	Mizoram	10000.00

The subquery with a SELECT statement will be:

```
SELECT *
   FROM EMPLOYEE
   WHERE ID IN (SELECT ID
   FROM EMPLOYEE
   WHERE SALARY > 4500);
```

This would produce the following result:

ID	NAME	AGE	ADDRESS	SALARY
4	Alina	29	UK	6500.00
5	Kathrin	34	Bangalore	8500.00
7	Jackson	25	Mizoram	10000.00

2. Subqueries with the INSERT Statement

- SQL subquery can also be used with the Insert statement. In the insert statement, data returned from the subquery is used to insert into another table.
- In the subquery, the selected data can be modified with any of the character, date functions.

Syntax:

```
INSERT INTO table_name (column1, column2, column3....)
```

```
SELECT *
FROM table_name
WHERE VALUE OPERATOR
```

Example

Consider a table EMPLOYEE_BKP with similar as EMPLOYEE.

Now use the following syntax to copy the complete EMPLOYEE table into the EMPLOYEE_BKP table.

```
INSERT INTO EMPLOYEE_BKP
  SELECT * FROM EMPLOYEE
  WHERE ID IN (SELECT ID
  FROM EMPLOYEE);
```

3. Subqueries with the UPDATE Statement

The subquery of SQL can be used in conjunction with the Update statement. When a subquery is used with the Update statement, then either single or multiple columns in a table can be updated.

Syntax

```
              UPDATE table
SET column_name = new_value
WHERE VALUE OPERATOR
  (SELECT COLUMN_NAME
  FROM TABLE_NAME
  WHERE condition);
```

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example updates the SALARY by .25 times in the EMPLOYEE table for all employee whose AGE is greater than or equal to 29.

```
1. UPDATE EMPLOYEE
2.   SET SALARY = SALARY * 0.25
3.   WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
4.     WHERE AGE >= 29);
```

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
4	Alina	29	UK	1625.00
5	Kathrin	34	Bangalore	2125.00
6	Harry	42	China	1125.00
7	Jackson	25	Mizoram	10000.00

4. Subqueries with the DELETE Statement

The subquery of SQL can be used in conjunction with the Delete statement just like any other statements mentioned above.

Syntax

```
     DELETE FROM TABLE_NAME
1. WHERE VALUE OPERATOR
2.   (SELECT COLUMN_NAME
```

3. FROM TABLE_NAME
4. WHERE condition);

Ex-

DELETE FROM EMPLOYEE WHERE AGE IN (SELECT AGE FROM EMPLOY EE_BKP WHERE AGE >= 29);

This would impact three rows, and finally, the EMPLOYEE table would have the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	John	20	US	2000.00
2	Stephan	26	Dubai	1500.00
3	David	27	Bangkok	2000.00
7	Jackson	25	Mizoram	10000.00

PL/SQL Introduction:

PL/SQL is a block structured language that enables developers to combine the power of SQL with procedural statements.All the statements of a block are passed to oracle engine all at once which increases processing speed and decreases the traffic.

Basics of PL/SQL

- •PL/SQL stands for Procedural Language extensions to the Structured Query Language (SQL).
- PL/SQL is a combination of SQL along with the procedural features of programming languages.

- Oracle uses a PL/SQL engine to processes the PL/SQL statements.
- PL/SQL includes procedural language elements like conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variable of those types and triggers.

Disadvantages of SQL:

- SQL doesn’t provide the programmers with a technique of condition checking, looping and branching.
- SQL statements are passed to Oracle engine one at a time which increases traffic and decreases speed.
- SQL has no facility of error checking during manipulation of data.

Features of PL/SQL:

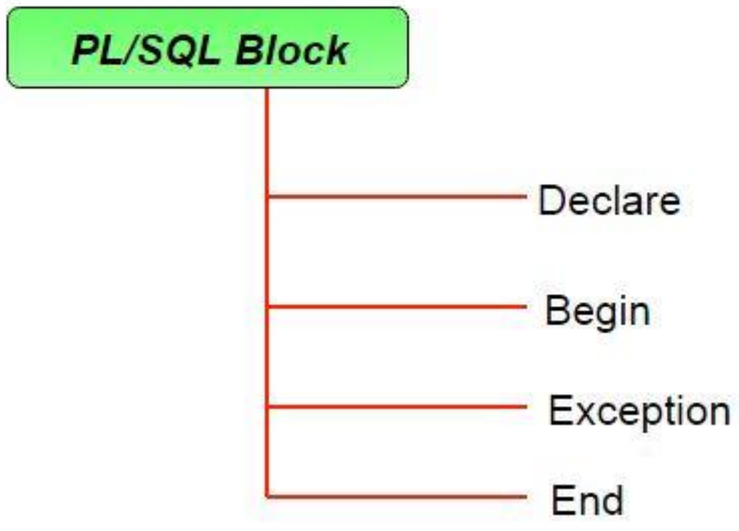
1. PL/SQL is basically a procedural language, which provides the functionality of decision making, iteration and many more features of procedural programming languages.
2. PL/SQL can execute a number of queries in one block using single command.
3. One can create a PL/SQL unit such as procedures, functions, packages, triggers, and types, which are stored in the database for reuse by applications.
4. PL/SQL provides a feature to handle the exception which occurs in PL/SQL block known as exception handling block.

Differences between SQL and PL/SQL:

SQL	PL/SQL
SQL is a single query that is used to perform DML and DDL operations.	PL/SQL is a block of codes that used to write the entire program blocks/ procedure/ function, etc.
It is declarative, that defines what needs to be done, rather than how things need to be done.	PL/SQL is procedural that defines how the things needs to be done.
Execute as a single statement.	Execute as a whole block.
Mainly used to manipulate data.	Mainly used to create an application.
Cannot contain PL/SQL code in it.	It is an extension of SQL, so it can contain SQL inside it.

Structure of PL/SQL Block:

PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other.



Typically, each block performs a logical action in the program. A block has the following structure:

```
DECLARE
   declaration statements;
BEGIN
   executable statements
EXCEPTIONS
   exception handling statements
END;
```

```
SET SERVEROUTPUT ON;

SQL> DECLARE
   var1 INTEGER;
   var2 REAL;
   var3 varchar2(20) ;

BEGIN
   null;
END;
```

Functions in PL/SQL:

The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

Syntax to create a function:

```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
1. [(parameter_name [IN | OUT | IN OUT] type [, ...])]
2. RETURN return_datatype
3. {IS | AS}
4. BEGIN
5.    < function_body >
6. END [function_name];
```

Here:

- **Function_name:** specifies the name of the function.
- **[OR REPLACE]** option allows modifying an existing function.
- The **optional parameter list** contains name, mode and types of the parameters.
- **IN** represents that value will be passed from outside and OUT represents that this parameter will be used to return a value outside of the procedure.

The function must contain a return statement.

- RETURN clause specifies that data type you are going to return from the function.
- Function_body contains the executable part.

- The AS keyword is used instead of the IS keyword for creating a standalone function.

PL/SQL Function Example

Let's see a simple example to **create a function**.

PL/SQL Function Example

Let's take an example to demonstrate Declaring, Defining and Invoking a simple PL/SQL function which will compute and return the maximum of two values.

```
DECLARE
   a number;
   b number;
   c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
    z number;
BEGIN
   IF x > y THEN
      z:= x;
   ELSE
      Z:= y;
   END IF;

   RETURN z;
END;
BEGIN
   a:= 23;
   b:= 45;

   c := findMax(a, b);
   dbms_output.put_line(' Maximum of (23,45): ' || c);
END;
/
```

PROCEDURES IN PL/SQL

A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the **calling program**.

A subprogram can be created −

- At the schema level
- Inside a package
- Inside a PL/SQL block

At the schema level, subprogram is a **standalone subprogram**. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

Parts of a PL/SQL Subprogram:

Each PL/SQL subprogram has a name, and may also have a parameter list. Like anonymous PL/SQL blocks, the named blocks will also have the following three parts −

S.No	Parts & Description
1	**Declarative Part** It is an optional part. However, the declarative part for a

subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution.

Executable Part

2 This is a mandatory part and contains statements that perform the designated action.

Exception-handling

3 This is again an optional part. It contains the code that handles run-time errors.

Creating a Procedure

A procedure is created with the **CREATE OR REPLACE PROCEDURE** statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] datatype,...])]{IS | AS}
BEGIN
< procedure_body >;
END procedure_name;
```

Where,

- *procedure-name* specifies the name of the procedure.
-
- [OR REPLACE] option allows the modification of an existing procedure.
-
-

The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.

•

•

procedure-body contains the executable part.

•

•

The AS keyword is used instead of the IS keyword for creating a standalone procedure.

•

Example

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
CREATE OR REPLACE PROCEDURE greetings
AS

 BEGIN
   dbms_output.put_line('Hello World!');
END;/
```

When the above code is executed using the SQL prompt, it will produce the following result −

Procedure created.

Executing a Standalone Procedure

A standalone procedure can be called in two ways −

•

Using the **EXECUTE** keyword

•

-
- Calling the name of the procedure from a PL/SQL block
-

The above procedure named **'greetings'** can be called with the EXECUTE keyword as −

EXEC greetings;

greetings;

The above call will display −

Hello World

PL/SQL procedure successfully completed.

The procedure can also be called from another PL/SQL block −

```
BEGIN
  greetings;END;/
```

The above call will display −

Hello World

IN & OUT Mode Example 2

This procedure computes the square of value of a passed value. This example shows how we can use the same parameter to accept a value and then return another result.

```
DECLARE
  a number;
PROCEDURE squareNum(x IN OUT number) IS BEGIN
  x := x * x;
END;
```

```
BEGIN
  a:=23;
  squareNum(a);
  dbms_output.put_line(' Square of (23): '|| a);
END;/
```

PL/SQL procedure successfully completed.

Deleting a Standalone Procedure

A standalone procedure is deleted with the **DROP PROCEDURE** statement. Syntax for deleting a procedure is −

DROP PROCEDURE procedure-name;

You can drop the greetings procedure by using the following statement −

DROP PROCEDURE greetings;

PL/SQL Cursor

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains information on a select statement and the rows of data accessed by it.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

- Implicit Cursors
- Explicit Cursors

1) PL/SQL Implicit Cursors

The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement.

These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.

Orcale provides some attributes known as Implicit cursor's attributes to check the status of DML operations. Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.

The following table soecifies the status of the cursor with each of its attribute.

Attribute	Description
%FOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE.
%NOTFOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise it returns FALSE. It is a just opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements.

%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement.

PL/SQL Implicit Cursor Example

Create customers table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Let's execute the following program to update the table and increase salary of each customer by 5000. Here, SQL%ROWCOUNT attribute is used to determine the number of rows affected:

Create procedure:

```
DECLARE
   total_rows number(2);
BEGIN
   UPDATE  customers
   SET salary = salary + 5000;
```

```
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/
```

Output:

```
6 customers updated
PL/SQL procedure successfully completed.
```

Now, if you check the records in customer table, you will find that the rows are updated.

```
select * from customers;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	25000
2	Suresh	22	Kanpur	27000
3	Mahesh	24	Ghaziabad	29000
4	Chandan	25	Noida	31000
5	Alex	21	Paris	33000
6	Sunita	20	Delhi	35000

2) PL/SQL Explicit Cursors

The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Following is the syntax to create an explicit cursor:

Syntax of explicit cursor

Following is the syntax to create an explicit cursor:

Syn:-CURSOR cursor_name **IS** select_statement;;

Steps:

You must follow these steps while working with an explicit cursor.

1. Declare the cursor to initialize in the memory.
2. Open the cursor to allocate memory.
3. Fetch the cursor to retrieve data.
4. Close the cursor to release allocated memory.

1) Declare the cursor:

It defines the cursor with a name and the associated SELECT statement.

Syntax for explicit cursor decleration

1. **CURSOR name IS**
2. **SELECT** statement;

2) Open the cursor:

It is used to allocate memory for the cursor and make it easy to fetch the rows returned by the SQL statements into it.

Syntax for cursor open:

```
1. OPEN cursor_name;
```

3) Fetch the cursor:

It is used to access one row at a time. You can fetch rows from the above-opened cursor as follows:

Syntax for cursor fetch:

```
1. FETCH cursor_name INTO variable_list;
```

4) Close the cursor:

It is used to release the allocated memory. The following syntax is used to close the above-opened cursors.

Syntax for cursor close:

```
1. Close cursor_name;
```

PL/SQL Explicit Cursor Example

Explicit cursors are defined by programmers to gain more control over the context area. It is defined in the declaration section of the

PL/SQL block. It is created on a SELECT statement which returns more than one row.

Let's take an example to demonstrate the use of explicit cursor. In this example, we are using the already created CUSTOMERS table.

Create customers table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Create procedure:

Execute the following program to retrieve the customer name and address.

```
DECLARE
   c_id customers.id%type;
   c_name customers.name%type;
   c_addr customers.address%type;
   CURSOR c_customers is
```

```
     SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
    FETCH c_customers into c_id, c_name, c_addr;
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```

Output:

```
1  Ramesh  Allahabad

2  Suresh  Kanpur

3  Mahesh  Ghaziabad

4  Chandan  Noida

5  Alex  Paris

6  Sunita  Delhi

PL/SQL procedure successfully completed.
```

PL/SQL Trigger:

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

Advantages of Triggers

These are the following advantages of Triggers:

- Trigger generates some derived column values automatically
- Enforces referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating a trigger:

Syntax for creating trigger:

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
```

```
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Here,

- CREATE [OR REPLACE] TRIGGER trigger_name: It creates or replaces an existing trigger with the trigger_name.
- {BEFORE | AFTER | INSTEAD OF} : This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.

- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

PL/SQL Trigger Example

Let's take a simple example to demonstrate the trigger. In this example, we are using the following CUSTOMERS table:

Create table and have records:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

Ex-

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
```

```
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

After the execution of the above code at SQL Prompt, it produces the following result.

```
Trigger created.
```

PL/SQL Trigger Example

Let's take a simple example to demonstrate the trigger. In this

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

example, we are using the following CUSTOMERS table:

Check the salary difference by procedure:

Use the following code to get the old salary, new salary and salary difference after the trigger created.

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
/
```

Output:

```
Old salary: 20000
New salary: 25000
Salary difference: 5000
Old salary: 22000
New salary: 27000
Salary difference: 5000
Old salary: 24000
New salary: 29000
Salary difference: 5000
Old salary: 26000
New salary: 31000
Salary difference: 5000
Old salary: 28000
New salary: 33000
```

```
Salary difference: 5000
Old salary: 30000
New salary: 35000
Salary difference: 5000
6 customers updated
```

Note: As many times you executed this code, the old and new both salary is incremented by 5000 and hence the salary difference is always 5000.

sql Data Types:

Oracle String data types

CHAR(size)	It is used to store character data within the predefined length. It can be stored up to 2000 bytes.
NCHAR(size)	It is used to store national character data within the predefined length. It can be stored up to 2000 bytes.
VARCHAR2(size)	It is used to store variable string data within the predefined length. It can be stored up to 4000 byteS.
VARCHAR(SIZE)	It is the same as VARCHAR2(size). You can also use VARCHAR(size), but it is suggested to use VARCHAR2(size)
NVARCHAR2(size)	It is used to store Unicode string data within the predefined length. We have to must specify the size of NVARCHAR2 data type. It can be stored up to 4000 bytes.

Oracle Numeric Data Types

NUMBER(p, s)	It contains precision p and scale s. The precision p can range from 1 to 38, and the scale s can range from -84 to 127.
FLOAT(p)	It is a subtype of the NUMBER data type. The precision p can range from 1 to 126.
BINARY_FLOAT	It is used for binary precision(32-bit). It requires 5 bytes, including length byte.
BINARY_DOUBLE	It is used for double binary precision (64-bit). It requires 9 bytes, including length byte.

Oracle Date and Time Data Types

DATE	It is used to store a valid date-time format with a fixed length. Its range varies from January 1, 4712 BC to December 31, 9999 AD.
TIMESTAMP	It is used to store the valid date in YYYY-MM-DD with time hh:mm:ss format.

Oracle Large Object Data Types (LOB Types)

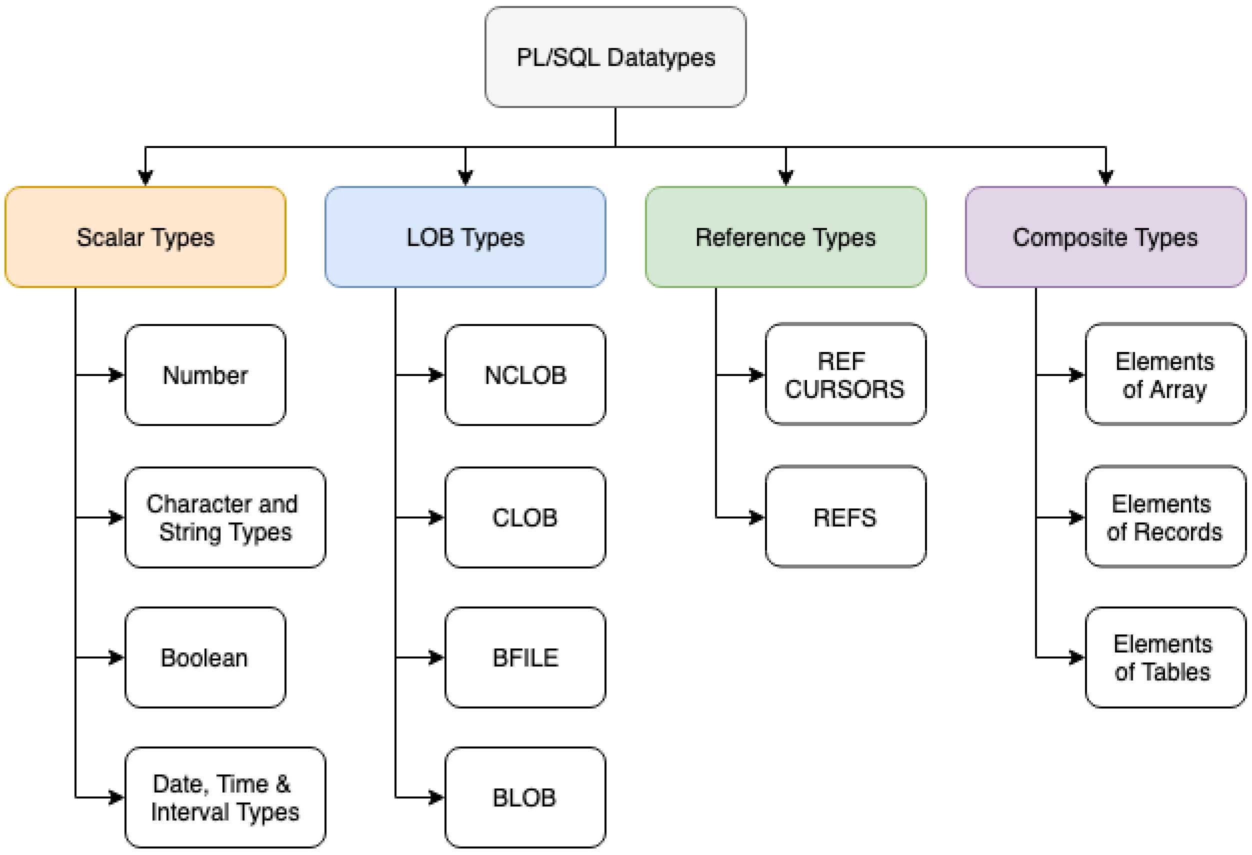
BLOB	It is used to specify unstructured binary data. Its range goes up to 2^{32}-1 bytes or 4 GB.
BFILE	It is used to store binary data in an external file. Its range goes up to 2^{32}-1 bytes or 4 GB.
CLOB	It is used for single-byte character data. Its range goes up to 2^{32}-1 bytes or 4 GB.
NCLOB	It is used to specify single byte or fixed length multibyte national character set (NCHAR) data.

	Its range is up to 2^{32}-1 bytes or 4 GB.
RAW(size)	It is used to specify variable length raw binary data. Its range is up to 2000 bytes per row. Its maximum size must be specified.
LONG RAW	It is used to specify variable length raw binary data. Its range up to 2^{31}-1 bytes or 2 GB, per row.

PL/SQL DATATYPES:

PL/SQL datatypes are not just limited to writing SQL queries but they are used in the PL/SQL block as well, just like any other programming language.

Datatype defines the type of data being used, whether it is a number or a word(string) or a single character etc. Following datatypes can be used in PL/SQL depending upon the type of data required:



So we have 4 broader categories of datatypes and they are:

1. **Scalar Types**: These are basic datatypes which generally holds a single value like a number or a string of characters. Scalar types have 4 different categories which are listed in the diagram above, namely **Number Types**, **Character and String**, **Boolean Types** and **Date and Time** etc.

2. **LOB Types**: This datatype deals with large objects and is used to specify location of these large objects like text files, images etc which are generally not stored outside the database.

3. **Reference Types**: This datatype is used to hold pointer values which generally stores address of other program items.

4. **Composite Types**: Last but not the least, as the name suggests this type of data is a composition of individual data which can be manipulated/processed separatel as well.

We won't be covering all these different datatypes below, but we will be covering the ones which are most widely used.

NUMBER(p,s)

Range: p= 1 to 38 s= -84 to 127

This datatype is used to store numeric data. Here, p is precision s is scale.

Example:

1. Age NUMBER(2); where , **Age** is a variable that can store **2** digits
2. percentage NUMBER(4,2); where, **percentage** is a variable that can store 4 (p) digits before decimal and 2 (s) digits after decimal.

CHAR(size)

Range: 1 to 2000 bytes

- This datatype is used to store alphabetical string of fixed length.
- Its value is quoted in single quotes.
- Occupies the whole declared size of memory even if the space is not utilized by the data.

Example:

1. **rank CHAR(10);** where, **rank** is a variable that can store upto 10 characters. If the length of data(charcaters) stored in rank

is 5 then it will still occupy all the 10 spaces. 5 space in the memory will get used and the rest blank memory spaces will be wasted.

VARCHAR(size)

Range: 1 to 2000 bytes

- This datatype is used to store alphanumeric string of variable length.
- Its value is quoted in single quotes.
- Occupies the whole declared size of memory even if the space is not utilized by the data.

Example:

1. address VARCHAR(10); where, **address** is a variable that can occupy maximum 10 bytes of memory space and can store alphanumeric value in it. Unused spaces are wasted.

VARCHAR2(size)

Range: 1 to 4000 bytes

- This datatype is used to store alphanumeric string of variable length.
- Its value is quoted in single quotes.
- It releases the unused space in memory, hence saving the unused space.

Example:

1. name VARCHAR2(10); where, **name** is a variable that can occupy maximum 10 bytes of memory to store an alphanumeric value. The unused memory space is released.

DATE

Range: 01-Jan-4712 BC to 31-DEC-9999

- It stores the data in date format **DD-MON-YYYY**
- The value for this datatype is written in single quotes.

Example:

1. DOB DATE; where, **DOB** is a variable that stores date of birth in defined format (i.e,'13-FEB-1991')

%TYPE

- It stores value of that variable whose datatype is unknown and when we want the variable to inherit the datatype of the table column.
- Also, its value is generally being retrieved from an existing table in the database, hence it takes the datatype of the column for which it is used.

Example:

1. Student sno %TYPE;, where **Student** is the name of the table created in database and **sno** is variable whose datatype is unknown and %TYPE is used to store its value.

BOOLEAN

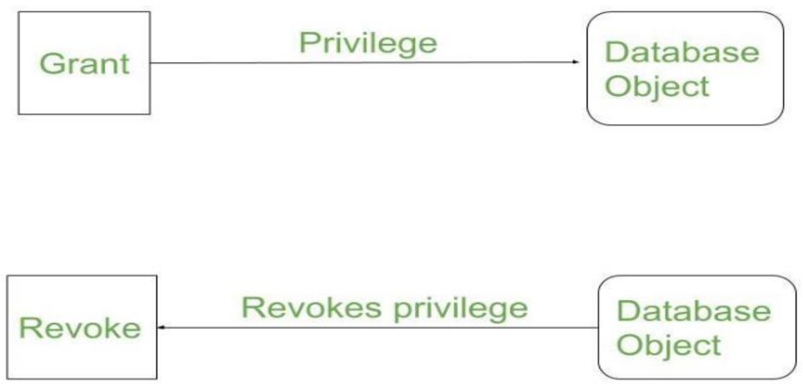
- This datatype is used in conditional statements.
- It stores logical values.
- It can be either TRUE or FALSE

Example:

1. isAdmin BOOLEAN; where, **isAdmin** is a variable whose value can be TRUE or FALSE depending upon the condition being checked.

AUTHORIZATION IN SQL:

Data Controlling Language (DCL) helps users to retrieve and modify the data stored in the database with some specified queries. Grant and Revoke belong to these types of commands of the Data controlling Language. DCL is a component of SQL commands.



Grant and Revoke Commands

1. Grant :

SQL Grant command is specifically used to provide privileges to database objects for a user. This command also allows users to

grant permissions to other users too.
Syntax:

grant privilege_name on object_name

to {user_name | public | role_name}

Here privilege_name is which permission has to be granted, object_name is the name of the database object, user_name is the user to which access should be provided, the public is used to permit access to all the users.

2. Revoke :

Revoke command withdraw user privileges on database objects if any granted. It does operations opposite to the Grant command. When a privilege is revoked from a particular user U, then the privileges granted to all other users by user U will be revoked.

Syntax:

revoke privilege_name on object_name

from {user_name | public | role_name}

Example:

grant insert, select on accounts to Ram

By the above command user ram has granted permissions on accounts database object like he can query or insert into accounts.

Ex-revoke insert, select on accounts from Ram

By the above command user ram's permissions like query or insert on accounts database object has been removed.

----xxx---

The image part with relationship ID rId21 was not found in the file.