



**INSTITUTE OF TECHNOLOGY
& MANAGEMENT**
GWALIOR • MP • INDIA

Laboratory Manual

**Data Base Management System
(IT-405)**

For

**Second Year Students
Department: Information Technology**

Department of Information Technology Engineering

Vision of IT Department

The Department of Information Technology envisions preparing technically competent problem solvers, researchers, innovators, entrepreneurs, and skilled IT professionals for the development of rural and backward areas of the country for the modern computing challenges.

Mission of the CSE Department

- I. To offer valuable education through an effective pedagogical teaching-learning process.
- II. To shape technologically strong students for industry, research & higher studies.
- III. To stimulate the young brain entrenched with ethical values and professional behaviors for the progress of society.

Program Educational Objectives

Graduates will be able to

- I. Our graduates will show management skills and teamwork to attain employers' objectives in their careers.
- II. Our graduates will explore the opportunities to succeed in research and/or higher studies.
- III. Our graduates will apply technical knowledge of Information Technology for innovation and entrepreneurship.
- IV. Our graduates will evolve ethical and professional practices for the betterment of society.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering Fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.



Course Outcomes

DBMS (IT-405)

CO1:	Understand basic concepts and identify various data models (E-R modelling concepts) and apply these concepts for designing databases.
CO2 :	Apply relational database theory by SQL and describe relational algebra expression, tuple and domain relational expression for writing queries in relational algebra.
CO3 :	Understand and implement various Relational Database Management Systems through Oracle/SQL/PL SQL.
CO4 :	Identify and improve the database design by normalization.
CO5 :	Evaluate optimize queries and transaction processes for solving real world problems.



Course	Course Outcomes	CO Attainment	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	Understand basic concepts and identify various data models (E-R modelling concepts) and apply these concepts for designing databases.		2	1	1	1	1	0	0	0	0	0	0	0	0	0	0
CO2	Apply relational database theory by SQL and describe relational algebra expression, tuple and domain relational expression for writing queries in relational algebra.		1	1	0	1	1	0	0	0	0	0	0	0	0	2	0
CO3	Understand and implement various Relational Database Management Systems through Oracle/SQL/PL SQL.		2	2	0	1	1	0	0	0	0	0	0	0	0	1	0
CO4	Identify and improve the database design by normalization.		2	1	1	1	0	0	0	0	1	2	0	0	0	0	0
CO5	Evaluate optimize queries and transaction processes for solving real world problems.		1	2	0	2	1	0	0	0	1	1	1	0	0	0	1



List Of Experiment

S. No.	List	Course Outcome	Page No.
1	To perform various SQL Commands of DDL, DML, DCL	CO1, CO2	1-6
2	Write SQL Commands Such as insertion, deletion and updation for any schema.	CO2	7-10
3	To execute Nested Queries, Join Queries, order-by, having clause and string operation.	CO1, CO2	11-34
4	To perform set operators like Union, Intersect, Minus on a set of tables	CO1, CO2, CO5	35-47
5	To execute various commands for GROUP functions (avg, count, max, min, Sum)	CO1, CO2	48-53
6	Write a PL/SQL block for transaction application using Triggers	CO1, CO3	54-58
7	Write a DBMS program to prepare report for an application using function	CO5	59-64
8	Designing of various Input screens/Forms	CO5	65-68
9	Create reports using database connectivity of Front end with back end.	CO5	69-73
10	Create database Design with normalization and implementing in any application	CO3	74-77

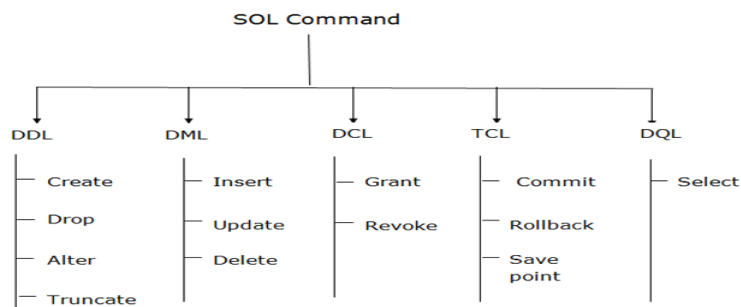
1. To perform various SQL Commands of DDL, DML, DCL.

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.

Syntax:

1. CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

Example:

1. CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DO B 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;

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

1. SELECT expressions
2. FROM TABLES
3. WHERE conditions;

For example:

1. SELECT emp_name
2. FROM employee
3. WHERE age > 20;

2. Write SQL Commands Such as insertion, deletion and updation for any schema.

SQL Sub Query

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

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

Example

Consider the EMPLOYEE table have the following records:

The subquery with a SELECT statement will be:

S.No	Name	Age	Address	Salary
------	------	-----	---------	--------

1	John	20	US	20000
2	Stephan	26	Dubai	15000
3	David	27	Bangkok	10000
4	Alina	29	UK	50000
5	Kathrin	34	Bangalore	53000
6	Harry	42	China	56000
7	Jackson	25	Mizoram	72000

1. SELECT *
2. FROM EMPLOYEE
3. WHERE ID IN (SELECT ID
4. FROM EMPLOYEE
5. WHERE SALARY > 4500);

This would produce the following result:

S.No	Name	Age	Address	Salary
1	Alina	20	US	20000
2	Kathrin	26	Dubai	15000
3	David	27	Bangkok	10000

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:

1. INSERT INTO table_name (column1, column2, column3....)
2. SELECT *
3. FROM table_name
4. 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.

1. INSERT INTO EMPLOYEE_BKP
2. SELECT * FROM EMPLOYEE
3. WHERE ID IN (SELECT ID
4. 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

1. UPDATE table
2. SET column_name = new_value
3. WHERE VALUE OPERATOR
4. (SELECT COLUMN_NAME
5. FROM TABLE_NAME
6. 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.

S.No	Name	Age	Address	Salary
------	------	-----	---------	--------

1	John	20	US	20000
2	Stephan	26	Dubai	15000
3	David	27	Bangkok	10000
4	Alina	29	UK	50000
5	Kathrin	34	Bangalore	53000
6	Harry	42	China	56000
7	Jackson	25	Mizoram	72000

2. 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

1. DELETE FROM TABLE_NAME
2. WHERE VALUE OPERATOR
3. (SELECT COLUMN_NAME
4. FROM TABLE_NAME
5. WHERE condition);

Example

Let's assume we have an EMPLOYEE_BKP table available which is backup of EMPLOYEE table. The given example deletes the records from the EMPLOYEE table for all EMPLOYEE whose AGE is greater than or equal to 29.

1. DELETE FROM EMPLOYEE
2. WHERE AGE IN (SELECT AGE FROM EMPLOYEE_BKP
3. WHERE AGE >= 29);

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

S.No	Name	Age	Address	Salary
1	Alina	20	US	20000
2	Kathrin	26	Dubai	15000
3	David	27	Bangkok	10000
4	Jackson	29	Mizoram	58700

3. To execute Nested Queries, Join Queries, order-by, having clause and string operation.

SQL Server JOINS

In real life, we store our data in multiple logical tables that are linked together by a common key value in relational databases like SQL Server, Oracle, MySQL, and others. As a result, we constantly need to get data from two or more tables into the desired output based on some conditions. We can quickly achieve this type of data in SQL Server using the SQL JOIN clause. This article gives a complete overview of JOIN and its different types with an example.

The join clause allows us to **retrieve data from two or more related tables** into a meaningful result set. We can join the table using a **SELECT** statement and a **join condition**. It indicates how SQL Server can use data from one table to select rows from another table. In general, tables are related to each other using **foreign key** constraints.

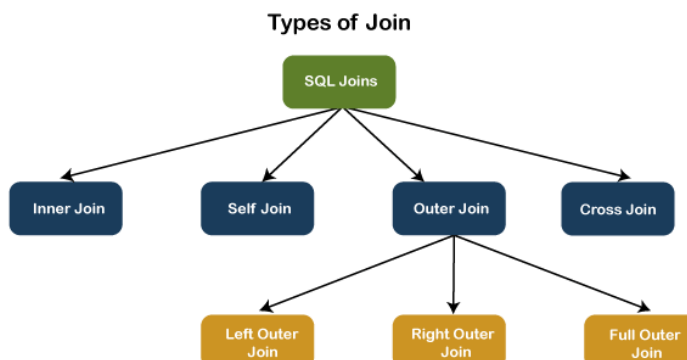
In a JOIN query, a condition indicates how two tables are related:

- Choose columns from each table that should be used in the join. A join condition indicates a foreign key from one table and its corresponding key in the other table.
- Specify the logical operator to compare values from the columns like =, <, or >.

Types of JOINS in SQL Server

SQL Server mainly supports **four types of JOINS**, and each join type defines how two tables are related in a query. The following are types of join supports in SQL Server:

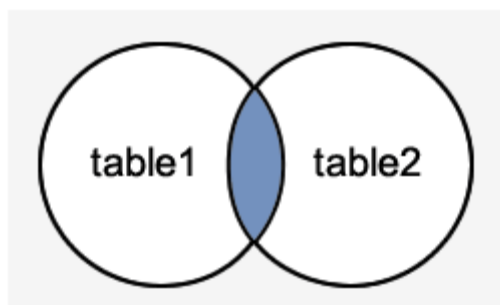
1. INNER JOIN
2. SELF JOIN
3. CROSS JOIN
4. OUTER JOIN



INNER JOIN

This JOIN returns all records from multiple tables that satisfy the specified join condition. It is the simple and most popular form of join and assumes as a **default join**. If we omit the INNER keyword with the JOIN query, we will get the same output.

The following visual representation explains how INNER JOIN returns the matching records from **table1** and **table2**:



INNER JOIN Syntax

The following syntax illustrates the use of INNER JOIN in SQL Server:

1. **SELECT** columns
2. **FROM** table1
3. **INNER JOIN** table2 **ON** condition1
4. **INNER JOIN** table3 **ON** condition2

INNER JOIN Example

Let us first create two tables "**Student**" and "**Fee**" using the following statement:

1. **CREATE TABLE** Student (

2. id **int PRIMARY KEY** IDENTITY,
3. admission_no **varchar**(45) NOT NULL,
4. first_name **varchar**(45) NOT NULL,
5. last_name **varchar**(45) NOT NULL,
6. age **int**,
7. city **varchar**(25) NOT NULL
8.);
- 9.
10. **CREATE TABLE** Fee (
11. admission_no **varchar**(45) NOT NULL,
12. course **varchar**(45) NOT NULL,
13. amount_paid **int**,
14.);

Next, we will insert some records into these tables using the below statements:

```
INSERT INTO Student (admission_no, first_name, last_name, age, city)
VALUES (3354,'Luisa', 'Evans', 13, 'Texas'),
(2135, 'Paul', 'Ward', 15, 'Alaska'),
(4321, 'Peter', 'Bennett', 14, 'California'),
(4213,'Carlos', 'Patterson', 17, 'New York'),
(5112, 'Rose', 'Huges', 16, 'Florida'),
(6113, 'Marielia', 'Simmons', 15, 'Arizona'),
(7555,'Antonio', 'Butler', 14, 'New York'),
(8345, 'Diego', 'Cox', 13, 'California');
```

```
INSERT INTO Fee (admission_no, course, amount_paid)
VALUES (3354,'Java', 20000),
(7555, 'Android', 22000),
(4321, 'Python', 18000),
(8345,'SQL', 15000),
(5112, 'Machine Learning', 30000);
```

Execute the **SELECT** statement to verify the records:

Table: Student

id	admission_no	first_name	last_name	age	city
1	3354	Luisa	Evans	13	Texas
2	2135	Paul	Ward	15	Alaska
3	4321	Peter	Bennett	14	California
4	4213	Carlos	Patterson	17	New York
5	5112	Rose	Huges	16	Florida
6	6113	Marielia	Simmons	15	Arizona
7	7555	Antonio	Butler	14	New York
8	8345	Diego	Cox	13	California

Table: Fee

admission_no	course	amount_paid
3354	Java	20000
7555	Android	22000
4321	Python	18000
8345	SQL	15000
5112	Machine Learning	30000

We can demonstrate the INNER JOIN using the following command:

1. **SELECT** Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
2. **FROM** Student
3. **INNER JOIN** Fee
4. **ON** Student.admission_no = Fee.admission_no;

This command gives the below result:

admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
4321	Peter	Bennett	Python	18000
5112	Rose	Huges	Machine Learning	30000
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

In this example, we have used the **admission_no column** as a join condition to get the data from both tables. Depending on this table, we can see the information of the students who have paid their fee.

SELF JOIN

A table is joined to itself using the SELF JOIN. It means that **each table row is combined with itself** and with every other table row. The SELF JOIN can be thought of as a JOIN of two copies of the same tables. We can do this with the help of **table name aliases** to assign a specific name to each table's instance. The table aliases enable us to use the **table's temporary** name that we are going to use in the query. It's a useful way to extract hierarchical data and comparing rows inside a single table.

SELF JOIN Syntax

The following expression illustrates the syntax of SELF JOIN in SQL Server. It works the same as the syntax of joining two different tables. Here, we use aliases names for tables because both the table name is the same.

1. **SELECT** T1.col_name, T2.col_name...
2. **FROM** table1 T1, table1 T2
3. **WHERE** join_condition;

Example

We can demonstrate the SELF JOIN using the following command:

1. **SELECT** S1.first_name, S2.last_name, S2.city
2. **FROM** Student S1, Student S2
3. **WHERE** S1.id <> S2.iD AND S1.city = S2.city
4. **ORDER BY** S2.city;

This command gives the below result:

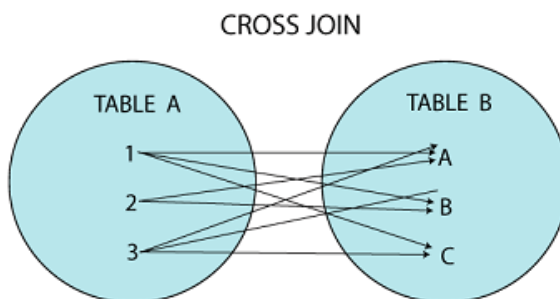
first_name	last_name	City
Peter	Cox	California
Diego	Bennett	California
Carlos	Butler	New York
Antonio	Patterson	New York

In this example, we have used the **id and city column** as a join condition to get the data from both tables.

CROSS JOIN

CROSS JOIN in SQL Server combines all of the possibilities of two or more tables and returns a result that includes every row from all contributing tables. It's also known as **CARTESIAN JOIN** because it produces the **Cartesian product** of all linked tables. The Cartesian product represents all rows present in the first table multiplied by all rows present in the second table.

The below visual representation illustrates the CROSS JOIN. It will give all the records from **table1** and **table2** where each row is the combination of rows of both tables:



CROSS JOIN Syntax

The following syntax illustrates the use of CROSS JOIN in SQL Server:

1. **SELECT** column_lists
2. **FROM** table1
3. **CROSS JOIN** table2;

Example

We can demonstrate the CROSS JOIN using the following command:

1. **SELECT** Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
2. **FROM** Student
3. **CROSS JOIN** Fee
4. **WHERE** Student.admission_no = Fee.admission_no;

This command gives the below result:

admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
4321	Peter	Bennett	Python	18000
5112	Rose	Huges	Machine Learning	30000
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

OUTER JOIN

OUTER JOIN in SQL Server **returns all records from both tables** that satisfy the join condition. In other words, this join will not return only the matching record but also return all unmatched rows from one or both tables.

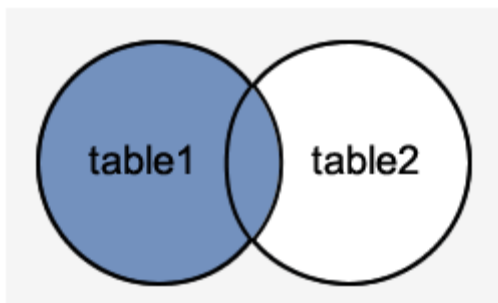
We can categories the OUTER JOIN further into three types:

- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN

LEFT OUTER JOIN

The LEFT OUTER JOIN **retrieves all the records from the left table and matching rows from the right table**. It will return NULL when no matching record is found in the right side table. Since OUTER is an optional keyword, it is also known as LEFT JOIN.

The below visual representation illustrates the LEFT OUTER JOIN:



LEFT OUTER JOIN Syntax

The following syntax illustrates the use of LEFT OUTER JOIN in SQL Server:

1. **SELECT** column_lists
2. **FROM** table1

3. LEFT [OUTER] JOIN table2
4. ON table1.column = table2.column;

Example

We can demonstrate the LEFT OUTER JOIN using the following command:

1. **SELECT** Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
2. **FROM** Student
3. LEFT OUTER JOIN Fee
4. **ON** Student.admission_no = Fee.admission_no;

This command gives the below result:

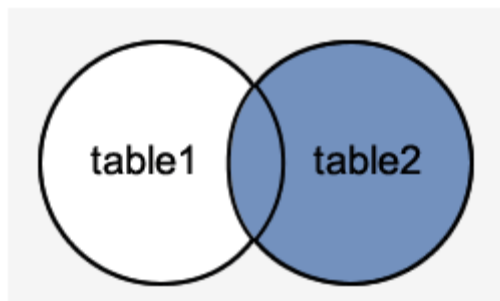
admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
2135	Paul	Ward	NULL	NULL
4321	Peter	Bennett	Python	18000
4213	Carlos	Patterson	NULL	NULL
5112	Rose	Huges	Machine Learning	30000
6113	Marielia	Simmons	NULL	NULL
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

This output shows that the unmatched row's values are replaced with NULLs in the respective columns.

RIGHT OUTER JOIN

The RIGHT OUTER JOIN retrieves all the records from the right-hand table and matched rows from the left-hand table. It will return NULL when no matching record is found in the left-hand table. Since OUTER is an optional keyword, it is also known as RIGHT JOIN.

The below visual representation illustrates the RIGHT OUTER JOIN:



RIGHT OUTER JOIN Syntax

The following syntax illustrates the use of RIGHT OUTER JOIN in SQL Server:

1. **SELECT** column_lists
2. **FROM** table1
3. **RIGHT [OUTER] JOIN** table2
4. **ON** table1.column = table2.column;

Example

The following example explains how to use the RIGHT OUTER JOIN to get records from both tables:

1. **SELECT** Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
2. **FROM** Student
3. **RIGHT OUTER JOIN** Fee
4. **ON** Student.admission_no = Fee.admission_no;

This command gives the below result:

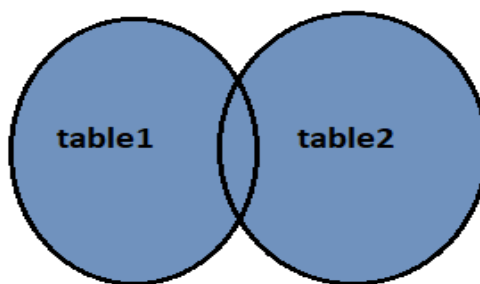
admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
7555	Antonio	Butler	Android	22000
4321	Peter	Bennett	Python	18000
8345	Diego	Cox	SQL	15000
5112	Rose	Huges	Machine Learning	30000

In this output, we can see that no column has NULL values because all rows in the Fee table are available in the student table based on the specified condition.

FULL OUTER JOIN

The FULL OUTER JOIN in SQL Server **returns a result that includes all rows from both tables**. The columns of the right-hand table return NULL when no matching records are found in the left-hand table. And if no matching records are found in the right-hand table, the left-hand table column returns NULL.

The below visual representation illustrates the FULL OUTER JOIN:



FULL OUTER JOIN Syntax

The following syntax illustrates the use of FULL OUTER JOIN in SQL Server:

1. **SELECT** column_lists
2. **FROM** table1
3. **FULL [OUTER] JOIN** table2
4. **ON** table1.column = table2.column;

Example

The following example explains how to use the FULL OUTER JOIN to get records from both tables:

1. **SELECT** Student.admission_no, Student.first_name, Student.last_name, Fee.course, Fee.amount_paid
2. **FROM** Student
3. **FULL OUTER JOIN** Fee
4. **ON** Student.admission_no = Fee.admission_no;

This command gives the below result:

admission_no	first_name	last_name	course	amount_paid
3354	Luisa	Evans	Java	20000
2135	Paul	Ward	NULL	NULL
4321	Peter	Bennett	Python	18000
4213	Carlos	Patterson	NULL	NULL
5112	Rose	Huges	Machine Learning	30000
6113	Marilia	Simmons	NULL	NULL
7555	Antonio	Butler	Android	22000
8345	Diego	Cox	SQL	15000

In this output, we can see that the column has NULL values when no matching records are found in the left-hand and right-hand table based on the specified condition.

SQL ORDER BY Clause

- Whenever we want to sort the records based on the columns stored in the tables of the SQL database, then we consider using the ORDER BY clause in SQL.
- The ORDER BY clause in SQL will help us to sort the records based on the specific column of a table. This means that all the values stored in the column on which we are applying ORDER BY clause will be sorted, and the corresponding column values will be displayed in the sequence in which we have obtained the values in the earlier step.
- Using the ORDER BY clause, we can sort the records in ascending or descending order as per our requirement. The records will be sorted in ascending order whenever the **ASC keyword** is used with ORDER by clause. **DESC keyword** will sort the records in descending order.
- *If no keyword is specified after the column based on which we have to sort the records, in that case, the sorting will be done by default in the ascending order.*

Before writing the queries for sorting the records, let us understand the syntax.

Syntax to sort the records in ascending order:

1. **SELECT** ColumnName1,...,ColumnNameN **FROM** TableName **ORDER BY** ColumnName **ASC**; Syntax to sort the records in descending order:
1. **SELECT** ColumnName1,...,ColumnNameN **FROM** TableName **ORDER BY** ColumnName **DESC**;

Syntax to sort the records in ascending order without using ASC keyword:

1. **SELECT** ColumnName1,...,ColumnNameN **FROM** TableName **ORDER BY** ColumnName;

Let us explore more on this topic with the help of examples. We will use the MySQL database for writing the queries in examples.

Consider we have customers table with the following records:

ID	Name	Age	Address	Salary
1	John	20	US	20000
2	Stephan	26	Dubai	15000
3	David	27	Bangkok	10000
4	Alina	29	UK	50000
5	Kathrin	34	Bangalore	53000
6	Harry	42	China	56000
7	Jackson	25	Mizoram	72000
8	Aakash Yadav	28	Mumbai	82000
9	Neeru Sharma	23	Pune	40000
10	Sahil Sheikh	22	Aurangabad	21000

Example 1:

Write a query to sort the records in the ascending order of the customer names stored in the customer's table.

Query:

1. mysql> **SELECT *FROM** customers **ORDER BY** Name **ASC**;

Here in a SELECT query, an ORDER BY clause is applied on the column 'Name' to sort the records. ASC keyword will sort the records in ascending order.

You will get the following output:

S.No	Name	Age	Address	Salary
1	John	20	US	20000
2	Stephan	26	Dubai	15000
3	David	27	Bangkok	10000
4	Alina	29	UK	50000
5	Kathrin	34	Bangalore	53000
6	Harry	42	China	56000
7	Jackson	25	Mizoram	72000
8	Aakash Yadav	28	Mumbai	82000
9	Neeru Sharma	23	Pune	40000
10	Sahil Sheikh	22	Aurangabad	21000

All the records present in the customers table are displayed in the ascending order of the customer's name.

Example 2:

Write a query to sort the records in the ascending order of the addresses stored in the customers table.

Query:

1. `mysql> SELECT *FROM customers ORDER BY Address;`

Here in a SELECT query, an ORDER BY clause is applied to the 'Address' column to sort the records. No keyword is used after the ORDER BY clause. Hence, the records, by default, will be sorted in ascending order.

You will get the following output:

ID	Name	Age	Address	Salary
7	Jackson	25	Mizoram	72000
10	Sahil Sheikh	22	Aurangabad	21000

All the records present in the customers table are displayed in the ascending order of the customer's address.

Example 3:

Write a query to sort the records in the descending order of the customer salary stored in the customer's table.

Query:

1. `mysql> SELECT *FROM customers ORDER BY Salary DESC;`

Here in a SELECT query, an ORDER BY clause is applied on the column ?Salary? to sort the records. DESC keyword will sort the records in descending order.

You will get the following output:

ID	NAME	AGE	ADDRESS	SALARY
10	Sahil Sheikh	35	Aurangabad	68800
3	Ajeet Bhargav	45	Meerut	65000
9	Aakash Yadav	32	Mumbai	43500
8	Neeru Sharma	29	Pune	40000
7	Rohit Shrivastav	19	Ahemdabad	38000
5	Balwant Singh	45	Varanasi	36000
4	Ritesh Yadav	36	Azamgarh	26000
6	Mahesh Sharma	26	Mathura	22000
1	Himani Gupta	21	Modinagar	22000

2	Shiva Tiwari	22	Bhopal	21000
---	--------------	----	--------	-------

All the records present in the customers table are displayed in the descending order of the customer's salary.

Example 4:

Write a query to sort the records in the descending order of the customer age stored in the customer's table.

Query:

1. `mysql> SELECT *FROM customers ORDER BY Age DESC;`

Here in a SELECT query, an ORDER BY clause is applied on the column 'Age' to sort the records. DESC keyword will sort the records in descending order.

You will get the following output:

ID	NAME	AGE	ADDRESS	SALARY
3	Ajeet Bhargav	45	Meerut	65000
5	Balwant Singh	45	Varanasi	36000
4	Ritesh Yadav	36	Azamgarh	26000
10	Sahil Sheikh	35	Aurangabad	68800
9	Aakash Yadav	32	Mumbai	43500
8	Neeru Sharma	29	Pune	40000
6	Mahesh Sharma	26	Mathura	22000
2	Shiva Tiwari	22	Bhopal	21000
1	Himani Gupta	21	Modinagar	22000
7	Rohit Shrivastav	19	Ahemdabad	38000

All the records present in the customers table are displayed in the descending order of the customer's age.

HAVING Clause in SQL

The HAVING clause places the condition in the groups defined by the GROUP BY clause in the SELECT statement.

This SQL clause is implemented after the 'GROUP BY' clause in the 'SELECT' statement.

This clause is used in SQL because we cannot use the WHERE clause with the SQL aggregate functions. Both WHERE and HAVING clauses are used for filtering the records in SQL queries.

Difference between HAVING and WHERE Clause

The difference between the WHERE and HAVING clauses in the database is the most important question asked during an IT interview.

The following table shows the comparisons between these two clauses, but the main difference is that the WHERE clause uses condition for filtering records before any groupings are made, while HAVING clause uses condition for filtering values from a group.

HAVING	WHERE
1. The HAVING clause is used in database systems to fetch the data/values from the groups according to the given condition.	1. The WHERE clause is used in database systems to fetch the data/values from the tables according to the given condition.
2. The HAVING clause is always executed with the GROUP BY clause.	2. The WHERE clause can be executed without the GROUP BY clause.
3. The HAVING clause can include SQL aggregate functions in a query or statement.	3. We cannot use the SQL aggregate function with WHERE clause in statements.
4. We can only use SELECT statement with HAVING clause for filtering the records.	4. Whereas, we can easily use WHERE clause with UPDATE, DELETE, and SELECT statements.

5. The HAVING clause is used in SQL queries after the GROUP BY clause.	5. The WHERE clause is always used before the GROUP BY clause in SQL queries.
6. We can implement this SQL clause in column operations.	6. We can implement this SQL clause in row operations.
7. It is a post-filter.	7. It is a pre-filter.
8. It is used to filter groups.	8. It is used to filter the single record of the table.

Syntax of HAVING clause in SQL

1. **SELECT** column_Name1, column_Name2,, column_NameN aggregate_function_name(column_Name) **FROM** table_name **GROUP BY** column_Name1 **HAVING** condition;

Examples of HAVING clause in SQL

In this article, we have taken the following four different examples which will help you how to use the HAVING clause with different SQL aggregate functions:

Example 1: Let's take the following **Employee** table, which helps you to analyze the HAVING clause with SUM aggregate function:

Emp_Id	Emp_Name	Emp_Salary	Emp_City
201	Abhay	2000	Goa
202	Ankit	4000	Delhi
203	Bheem	8000	Jaipur
204 Ram	2000	Goa	
205	Sumit	5000	Delhi

If you want to add the salary of employees for each city, you have to write the following query:

1. **SELECT** SUM(Emp_Salary), Emp_City **FROM** Employee **GROUP BY** Emp_City;

The output of the above query shows the following output:

SUM(Emp_Salary)	Emp_City
4000	Goa
9000	Delhi
8000	Jaipur

Now, suppose that you want to show those cities whose total salary of employees is more than 5000. For this case, you have to type the following query with the HAVING clause in SQL:

1. **SELECT** SUM(Emp_Salary), Emp_City **FROM** Employee **GROUP BY** Emp_City **HAVING** SUM(Emp_Salary)>5000;

The output of the above SQL query shows the following table in the output:

SUM(Emp_Salary)	Emp_City
000	Delhi
000	Jaipur

Example 2: Let's take the following **Student_details** table, which helps you to analyze the HAVING clause with the COUNT aggregate function:

Roll_No	Name	Marks	Age
1	Rithik	91	20
2	Kapil	50	19

3	Arun	32	17
4	Ram	22	18
5	Anuj	50	20
5	Suman	38	18
7	Sheetal	57	19
8	Anuj	54	20

Suppose, you want to count the number of students from the above table according to their age. For this, you have to write the following query:

1. **SELECT** COUNT(Roll_No), Age **FROM** Student_details **GROUP BY** Age ;

The above query will show the following output:

Count(Roll_No)	Age
3	20
2	19
1	17
2	18

Now, suppose that you want to show the age of those students whose roll number is more than and equals 2. For this case, you have to type the following query with the HAVING clause in SQL:

1. **SELECT** COUNT(Roll_No), Age **FROM** Student_details **GROUP BY** Age **HAVING** COUNT(Roll_No) >= 2 ;

The output of the above SQL query shows the following table in the output:

Count(Roll_No)	Age
3	20
2	19
2	18

Example 3: Let's take the following **Employee** table, which helps you to analyze the HAVING clause with MIN and MAX aggregate function:

Emp_ID	Name	Emp_Salary	Emp_Dept
1001	Anuj	9000	Finance
1002	Saket	4000	HR
1003	Raman	3000	Coding
1004	Renu	6000	Coding
1005	Seenu	5000	HR
1006	Mohan	10000	Marketing
1007	Anaya	4000	Coding
1008	Parul	8000	Finance

MIN Function with HAVING Clause:

If you want to show each department and the minimum salary in each department, you have to write the following query:

1. **SELECT MIN(Emp_Salary), Emp_Dept FROM Employee GROUP BY Emp_Dept;**

The output of the above query shows the following output:

MIN(Emp_Salary)	Emp_Dept
8000	Finance
4000	HR
3000	Coding
10000	Marketing

Now, suppose that you want to show only those departments whose minimum salary of employees is greater than 4000. For this case, you have to type the following query with the HAVING clause in SQL:

1. **SELECT MIN(Emp_Salary), Emp_Dept FROM Employee GROUP BY Emp_Dept HAVING MIN(Emp_Salary) > 4000 ;**

The above SQL query shows the following table in the output:

MIN(Emp_Salary)	Emp_Dept
8000	Finance
10000	Marketing

MAX Function with HAVING Clause:

In the above employee table, if you want to list each department and the maximum salary in each department. For this, you have to write the following query:

1. **SELECT MAX(Emp_Salary), Emp_Dept FROM Employee GROUP BY Emp_Dept;**

The above query will show the following output:

MAX(Emp_Salary)	Emp_Dept
9000	Finance
5000	HR
6000	Coding
10000	Marketing

Now, suppose that you want to show only those departments whose maximum salary of employees is less than 8000. For this case, you have to type the following query with the HAVING clause in SQL:

1. **SELECT MAX(Emp_Salary), Emp_Dept FROM Employee GROUP BY Emp_Dept HAVING MAX(Emp_Salary) < 8000 ;**

The output of the above SQL query shows the following table in the output:

MAX(Emp_Salary)	Emp_Dept
5000	HR
6000	Coding

Example 4: Let's take the following **Employee_Dept** table, which helps you to analyze the HAVING clause with AVG aggregate function:

Emp_ID	Name	Emp_Salary	Emp_Dept
1001	Anuj	8000	Finance
1002	Saket	4000	HR
1003	Raman	3000	Coding
1004	Renu	6000	Coding

1005	Seenu	5000	HR
1006	Mohan	10000	Marketing
1007	Anaya	4000	Coding
1008	Parul	6000	Finance

If you want to find the average salary of employees in each department, you have to write the following query:

1. **SELECT** AVG(Emp_Salary), Emp_Dept **FROM** Employee_Dept **GROUP BY** Emp_Dept;

The above query will show the following output:

AVG(Emp_Salary)	Emp_Dept
7000	Finance
4500	HR
5500	Coding
10000	Marketing

Now, suppose that you want to show those departments whose average salary is more than and equals 6500. For this case, you have to type the following query with the HAVING clause in SQL:

1. **SELECT** AVG(Emp_Salary), Emp_Dept **FROM** Employee_Dept **GROUP BY** Emp_Dept **HAVING** AVG(Emp_Salary) > 6500 ;

The above SQL query will show the following table in the output:

AVG(Emp_Salary)	Emp_Dept
7000	Finance
5500	Coding
10000	Marketing

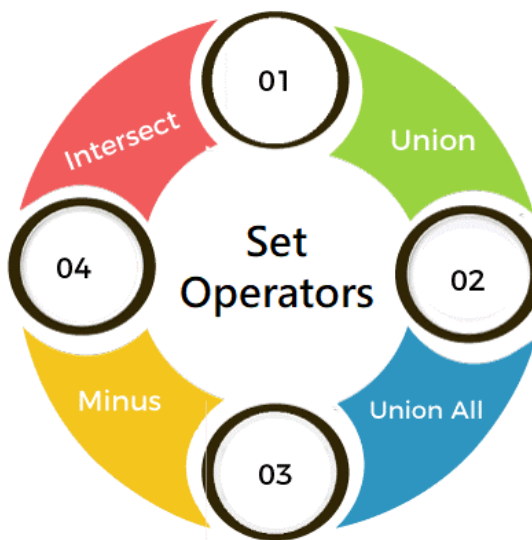
4. To perform set operators like Union, Intersect, Minus on a set of tables

SET Operators in SQL

SET operators are special type of operators which are used to *combine the result of two queries*.

Operators covered under SET operators are:

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



There are certain rules which must be followed to perform operations using SET operators in SQL. Rules are as follows:

1. **The number and order of columns must be the same.**
2. **Data types must be compatible.**

Let us see each of the SET operators in more detail with the help of examples.

All the examples will be written using the MySQL database.

Consider we have the following tables with the given data.

Table 1: t_employees

ID	Name	Age	Address	Salary
7	Jackson	25	Mizoram	72000
10	Sahil Sheikh	22	Aurangabad	21000
ID	Name	Age	Address	Salary
7	Jackson	25	Mizoram	72000
10	Sahil Sheikh	22	Aurangabad	21000
ID	Name	Age	Address	Salary
7	Jackson	25	Mizoram	72000

Table 2: t2_employees

ID	Name	Department	Salary	Year_of_Experience
1	Prashant Wagh	R&D	49000	1
2	Abhishek Pawar	Production	45000	1
3	Gautam Jain	Development	56000	4
4	Shubham Mahale	Accounts	57000	2
5	Rahul Thakur	Production	76000	4
6	Bhushan Wagh	R&D	75000	2
7	Anand Singh	Marketing	28000	1

Table 3: t_students

ID	Name	Hometown	Percentage	Favourite_Subject
1	Soniya Jain	Udaipur	89	Physics
2	Harshada Sharma	Kanpur	92	Chemistry
3	Anuja Rajput	Jaipur	78	History
4	Pranali Singh	Nashik	88	Geography
5	Renuka Deshmukh	Panipat	90	Biology
6	Swati Kumari	Faridabad	93	English
7	Prachi Jaiswal	Gurugram	96	Hindi

Table 4: t2_students

ID	Name	Hometown	Percentage	Favourite_Subject
1	Soniya Jain	Udaipur	89	Physics
2	Ishwari Dixit	Delhi	86	Hindi
3	Anuja Rajput	Jaipur	78	History
4	Pakhi Arora	Surat	70	Sanskrit
5	Renuka Deshmukh	Panipat	90	Biology
6	Jayshree Patel	Pune	91	Maths
7	Prachi Jaiswal	Gurugram	96	Hindi

1. UNION:

- UNION will be used to combine the result of two select statements.
- Duplicate rows will be eliminated from the results obtained after performing the UNION operation.

Example 1:

Write a query to perform union between the table t_employees and the table t2_employees.

Query:

1. mysql> **SELECT *FROM t_employees UNION SELECT *FROM t2_employees;**

Here, in a single query, we have written two SELECT queries. The first SELECT query will fetch the records from the t_employees table and perform a UNION operation with the records fetched by the second SELECT query from the t2_employees table.

You will get the following output:

ID	Name	Department	Salary	Year of Experience
1	Aakash Singh	Development	72000	2
2	Abhishek Pawar	Production	45000	1
3	Pranav Deshmukh	HR	59900	3
4	Shubham Mahale	Accounts	57000	2
5	Sunil Kulkarni	Development	87000	3
6	Bhushan Wagh	R&D	75000	2
7	Paras Jaiswal	Marketing	32000	1
1	Prashant Wagh	R&D	49000	1
3	Gautam Jain	Development	56000	4
5	Rahul Thakur	Production	76000	4
7	Anand Singh	Marketing	28000	1

Since we have performed union operation between both the tables, so only the records from the first and second table are displayed except for the duplicate records.

Example 2:

Write a query to perform union between the table t_students and the table t2_students.

Query:

- mysql> **SELECT *FROM t_students UNION SELECT *FROM t2_students;**

Here, in a single query, we have written two SELECT queries. The first SELECT query will fetch the records from the t_students table and perform a UNION operation with the records fetched by the second SELECT query from the t2_students table.

You will get the following output:

ID	Name	Department	Salary	Year_of_Experience
1	Soniya Jain	Udaipur	89	Physics
2	Harshada Sharma	Kanpur	92	Chemistry
3	Anuja Rajput	Jaipur	78	History
4	Pranali Singh	Nashik	88	Geography
5	Renuka Deshmukh	Panipat	90	Biology
6	Swati Kumari	Faridabad	93	English
7	Prachi Jaiswal	Gurugram	96	Hindi
2	Ishwari Dixit	Delhi	86	Hindi
4	Pakhi Arora	Surat	70	Sanskrit
6	Jayshree Patel	Pune	91	Maths

Since we have performed union operation between both the tables, so only the records from the first and second table are displayed except for the duplicate records.

2. UNION ALL

- This operator combines all the records from both the queries.

- Duplicate rows will be not be eliminated from the results obtained after performing the UNION ALL operation.

Example 1:

Write a query to perform union all operation between the table t_employees and the table t2_employees.

Query:

1. mysql> **SELECT *FROM t_employees UNION ALL SELECT *FROM t2_employees**
;

Here, in a single query, we have written two SELECT queries. The first SELECT query will fetch the records from the t_employees table and perform UNION ALL operation with the records fetched by the second SELECT query from the t2_employees table.

You will get the following output:

ID	Name	Department	Salary	Year_of_Experience
1	Aakash Singh	Development	72000	2
2	Abhishek Pawar	Production	45000	1
3	Pranav Deshmukh	HR	59900	3
4	Shubham Mahale	Accounts	57000	2
5	Sunil Kulkarni	Development	87000	3
6	Bhushan Wagh	R&D	75000	2
7	Paras Jaiswal	Marketing	32000	1
1	Prashant Wagh	R&D	49000	1
2	Abhishek Pawar	Production	45000	1
3	Gautam Jain	Development	56000	4

4	Shubham Mahale	Accounts	57000	2
5	Rahul Thakur	Production	76000	4
6	Bhushan Wagh	R&D	75000	2
7	Anand Singh	Marketing	28000	1

Since we have performed union all operation between both the tables, so all the records from the first and second table are displayed, including the duplicate records.

Example 2:

Write a query to perform union all operation between the table t_students and the table t2_students.

Query:

1. `mysql> SELECT *FROM t_students UNION ALL SELECT *FROM t2_students;`

Here, in a single query, we have written two SELECT queries. The first SELECT query will fetch the records from the t_students table and perform UNION ALL operation with the records fetched by the second SELECT query from the t2_students table.

You will get the following output:

ID	Name	Hometown	Percentage	Favourite_Subject
1	Soniya Jain	Udaipur	89	Physics
2	Harshada Sharma	Kanpur	92	Chemistry
3	Anuja Rajput	Jaipur	78	History
4	Pranali Singh	Nashik	88	Geography
5	Renuka Deshmukh	Panipat	90	Biology
6	Swati Kumari	Faridabad	93	English

7	Prachi Jaiswal	Gurugram	96	Hindi
1	Soniya Jain	Udaipur	89	Physics
2	Ishwari Dixit	Delhi	86	Hindi
3	Anuja Rajput	Jaipur	78	History
4	Pakhi Arora	Surat	70	Sanskrit
5	Renuka Deshmukh	Panipat	90	Biology
6	Jayshree Patel	Pune	91	Maths
7	Prachi Jaiswal	Gurugram	96	Hindi

Since we have performed union all operation between both the tables, so all the records from the first and second table are displayed, including the duplicate records.

3. INTERSECT:

- It is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements.

Example 1:

Write a query to perform intersect operation between the table t_employees and the table t2_employees.

Query:

1. `mysql> SELECT *FROM t_employees INTERSECT SELECT *FROM t2_employee
s;`

Here, in a single query, we have written two SELECT queries. The first SELECT query will fetch the records from the t_employees table and perform INTERSECT operation with the records fetched by the second SELECT query from the t2_employees table.

You will get the following output:

ID	Name	Hometown	Percentage	Favourite_Subject
2	Abhishek Pawar	Production	45000	1
4	Shubham Mahale	Accounts	57000	2
6	Bhushan Wagh	R&D	75000	2

Since we have performed intersect operation between both the tables, so only the common records from both the tables are displayed.

Example 2:

Write a query to perform intersect operation between the table students and the table t2_students.

Query:

1. `mysql> SELECT *FROM t_students INTERSECT SELECT *FROM t2_students;`

Here, in a single query, we have written two SELECT queries. The first SELECT query will fetch the records from the students table and perform a UNION operation with the records fetched by the second SELECT query from the t2_students table.

You will get the following output:

ID	Name	Hometown	Percentage	Favourite_Subject
1	Soniya Jain	Udaipur	89	Physics
3	Anuja Rajput	Jaipur	78	History
5	Renuka Deshmukh	Panipat	90	Biology

7	Prachi Jaiswal	Gurugram	96	Hindi
---	----------------	----------	----	-------

Since we have performed intersect operation between both the tables, so only the common records from both the tables are displayed.

4. MINUS

- It displays the rows which are present in the first query but absent in the second query with no duplicates.

Example 1:

Write a query to perform a minus operation between the table employees and the table t2_employees.

Query:

- mysql> **SELECT *FROM t_employees MINUS SELECT *FROM t2_employees;**

Here, in a single query, we have written two SELECT queries. The first SELECT query will fetch the records from the t_employees table and perform MINUS operation with the records fetched by the second SELECT query from the t2_employees table.

You will get the following output:

ID	Name	Department	Salary	Year_of_Experience
1	Aakash Singh	Development	72000	2
3	Pranav Deshmukh	HR	59900	3
5	Sunil Kulkarni	Development	87000	3

7	Paras Jaiswal	Marketing	32000	1
---	---------------	-----------	-------	---

Since we have performed Minus operation between both the tables, so only the unmatched records from both the tables are displayed.

Example 2:

Write a query to perform a minus operation between the table t_students and the table t2_students.

Query:

- mysql> **SELECT *FROM t_students MINUS SELECT *FROM t2_students;**

Here, in a single query, we have written two SELECT queries. The first SELECT query will fetch the records from the t_employees table and perform a UNION operation with the records fetched by the second SELECT query from the t2_employees table.

You will get the following output:

ID	Name	Hometown	Percentage	Favourite_Subject
2	Harshada Sharma	Kanpur	92	Chemistry
4	Pranali Singh	Nashik	88	Geography
6	Swati Kumari	Faridabad	93	English

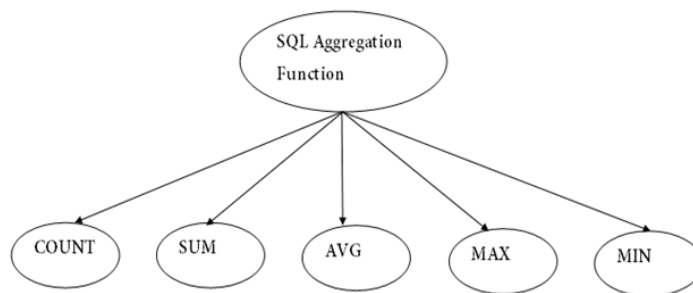
Since we have performed a minus operation between both the tables, so only the Unmatched records from both the tables are displayed.

5. To execute various commands for GROUP functions (avg, count, max, min, Sum)

SQL Aggregate Functions

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.

Types of SQL Aggregation Function



1. COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT (*) that returns the count of all the rows in a specified table. COUNT (*) considers duplicate and Null.

Syntax

- COUNT(*)
- or
- COUNT([ALL|DISTINCT] expression)

Sample table:

PRODUCT_MAST

PRODUCT	COMPANY	QTY	RATE	COST
Item1	Com1	2	10	20
Item2	Com2	3	25	75
Item3	Com1	2	30	60
Item4	Com3	5	10	50
Item5	Com2	2	20	40
Item6	Cpm1	3	25	75
Item7	Com1	5	30	150
Item8	Com1	3	10	30
Item9	Com2	2	25	50
Item10	Com3	4	30	120

Example: COUNT()

1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;

Output:

10

Example: COUNT with WHERE

1. SELECT COUNT(*)
2. FROM PRODUCT_MAST;
3. WHERE RATE >= 20;

Output:

7

Example: COUNT() with DISTINCT

1. SELECT COUNT(DISTINCT COMPANY)
2. FROM PRODUCT_MAST;

Output:

3

Example: COUNT() with GROUP BY

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY;

Output:

Com1	5
Com2	3
Com3	2

Example: COUNT() with HAVING

1. SELECT COMPANY, COUNT(*)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING COUNT(*)>2;

Output:

Com1	5
Com2	3

2. SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

1. SUM()
2. or

3. SUM([ALL|DISTINCT] expression)

Example: SUM()

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST;

Output:

670

Example: SUM() with WHERE

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>3;

Output:

320

Example: SUM() with GROUP BY

1. SELECT SUM(COST)
2. FROM PRODUCT_MAST
3. WHERE QTY>3
4. GROUP BY COMPANY;

Output:

Com1	150
Com2	170

Example: SUM() with HAVING

1. SELECT COMPANY, SUM(COST)
2. FROM PRODUCT_MAST
3. GROUP BY COMPANY
4. HAVING SUM(COST)>=170;

Output:

```
Com1  335
Com3  170
```

3. AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

1. AVG()
2. or
3. AVG([ALL|DISTINCT] expression)

Example:

1. SELECT AVG(COST)
2. FROM PRODUCT_MAST;

Output:

```
67.00
```

4. MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

1. MAX()
2. or
3. MAX([ALL|DISTINCT] expression)

Example:

1. SELECT MAX(RATE)
 2. FROM PRODUCT_MAST;
- ```
30
```

## 5. MIN Function

MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.

### Syntax

1. MIN()
2. or
3. MIN( [ALL|DISTINCT] expression )

### Example:

1. SELECT MIN(RATE)
2. FROM PRODUCT\_MAST;

### Output:

10

## 6. Write a PL/SQL block for transaction application using Triggers

### 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:

1. **CREATE** [OR REPLACE ] **TRIGGER** trigger\_name
2. { **BEFORE** | **AFTER** | **INSTEAD OF** }

3. {**INSERT** [OR] | **UPDATE** [OR] | **DELETE**}
4. [**OF** col\_name]
5. **ON** table\_name
6. [REFERENCING OLD **AS** o NEW **AS** n]
7. [**FOR EACH ROW**]
8. **WHEN** (condition)
9. **DECLARE**
10. Declaration-statements
11. **BEGIN**
12. Executable-statements
13. **EXCEPTION**
14. Exception-handling-statements
15. **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** colname]: 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  |

**Create trigger:**

Let's take a program to create a row level trigger for the CUSTOMERS table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

1. **CREATE OR REPLACE TRIGGER** display\_salary\_changes
2. **BEFORE DELETE OR INSERT OR UPDATE ON** customers
3. **FOR EACH ROW**
4. **WHEN** (NEW.ID > 0)
5. **DECLARE**
6.   sal\_diff number;
7. **BEGIN**
8.   sal\_diff := :NEW.salary - :OLD.salary;
9.   dbms\_output.put\_line('Old salary: ' || :OLD.salary);
10.   dbms\_output.put\_line('New salary: ' || :NEW.salary);
11.   dbms\_output.put\_line('Salary difference: ' || sal\_diff);
12. **END;**
13. /

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

Trigger created.

### Check the salary difference by procedure:

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

1. **DECLARE**
2.   total\_rows number(2);
3. **BEGIN**
4.   **UPDATE** customers
5.   **SET** salary = salary + 5000;
6.   **IF** sql%notfound **THEN**
7.     dbms\_output.put\_line('no customers updated');
8.   **ELSIF** sql%found **THEN**
9.     total\_rows := sql%rowcount;
10.   dbms\_output.put\_line( total\_rows || ' customers updated ');
11. **END IF;**
12. **END;**
13. /

### 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.

After the execution of above code again, you will get the following result.

Old salary: 25000  
New salary: 30000  
Salary difference: 5000  
Old salary: 27000  
New salary: 32000  
Salary difference: 5000  
Old salary: 29000  
New salary: 34000  
Salary difference: 5000  
Old salary: 31000  
New salary: 36000  
Salary difference: 5000  
Old salary: 33000  
New salary: 38000  
Salary difference: 5000  
Old salary: 35000  
New salary: 40000  
Salary difference: 5000  
6 customers updated

### Important Points

Following are the two very important point and should be noted carefully.

- OLD and NEW references are used for record level triggers these are not available for table level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent



## **7.Write a DBMS program to prepare report for an application using function**

### **PL/SQL Function**

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:**

1. **CREATE** [OR REPLACE] **FUNCTION** function\_name [parameters]
2. [(parameter\_name [IN | **OUT** | IN OUT] type [, ...])]
3. **RETURN** return\_datatype
4. {**IS** | **AS**}
5. **BEGIN**
6. < function\_body >
7. **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**.

1. **create** or replace **function** adder(n1 in number, n2 in number)
2. **return** number
3. **is**
4. n3 number(8);
5. **begin**
6. n3 :=n1+n2;
7. **return** n3;
8. **end**;
9. /

Now write another program to **call the function**.

1. **DECLARE**
2. n3 number(2);
3. **BEGIN**
4. n3 := adder(11,22);
5. dbms\_output.put\_line('Addition is: ' || n3);
6. **END**;
7. /

### Output:

```
Addition is: 33
Statement processed.
0.05 seconds
```

### Another 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.

1. **DECLARE**
2. a number;
3. b number;
4. c number;
5. **FUNCTION** findMax(x IN number, y IN number)
6. **RETURN** number

```

7. IS
8. z number;
9. BEGIN
10. IF x > y THEN
11. z:= x;
12. ELSE
13. Z:= y;
14. END IF;
15.
16. RETURN z;
17. END;
18. BEGIN
19. a:= 23;
20. b:= 45;
21.
22. c := findMax(a, b);
23. dbms_output.put_line(' Maximum of (23,45): ' || c);
24. END;
25. /

```

### Output:

```

Maximum of (23,45): 45
Statement processed.
0.02 seconds

```

### PL/SQL function example using table

Let's take a customer table. This example illustrates creating and calling a standalone function. This function will return the total number of CUSTOMERS in the customers' table.

Create customers table and have records in it.

| Customers |         |                   |        |
|-----------|---------|-------------------|--------|
| Id        | Name    | Department        | Salary |
| 1         | alex    | web developer     | 35000  |
| 2         | ricky   | program developer | 45000  |
| 3         | mohan   | web designer      | 35000  |
| 4         | dilshad | database manager  | 44000  |

### Create Function:

1. **CREATE OR REPLACE FUNCTION** totalCustomers
2. **RETURN** number **IS**
3.     total number(2) := 0;
4. **BEGIN**
5.     **SELECT** count(\*) **into** total
6.     **FROM** customers;
7.     **RETURN** total;
8. **END;**
9. /

After the execution of above code, you will get the following result.

Function created.

### Calling PL/SQL Function:

While creating a function, you have to give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task. Once the function is called, the program control is transferred to the called function.

After the successful completion of the defined task, the call function returns program control back to the main program.

To call a function you have to pass the required parameters along with function name and if function returns a value, then you can store returned value. Following program calls the function total Customers from an anonymous block:

1. **DECLARE**
2.   c number(2);
3. **BEGIN**
4.   c := totalCustomers();
5.   dbms\_output.put\_line('Total no. of Customers: ' || c);
6. **END;**
7. /

After the execution of above code in SQL prompt, you will get the following result.

Total no. of Customers: 4  
PL/SQL procedure successfully completed.

### PL/SQL Recursive Function

You already know that a program or a subprogram can call another subprogram. When a subprogram calls itself, it is called recursive call and the process is known as recursion.

Example to calculate the factorial of a number

Let's take an example to calculate the factorial of a number. This example calculates the factorial of a given number by calling itself recursively.

1. **DECLARE**
2.   num number;
3.   factorial number;
- 4.
5. **FUNCTION** fact(x number)
6. **RETURN** number
7. **IS**
8.   f number;
9. **BEGIN**
10. IF x=0 **THEN**
11.   f := 1;
12. **ELSE**
13.   f := x \* fact(x-1);
14. **END IF;**

```
15. RETURN f;
16. END;
17.
18. BEGIN
19. num:= 6;
20. factorial := fact(num);
21. dbms_output.put_line(' Factorial ' || num || ' is ' || factorial);
22. END;
23. /
```

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

```
Factorial 6 is 720
PL/SQL procedure successfully completed.
```

### PL/SQL Drop Function

#### Syntax for removing your created function:

If you want to remove your created function from the database, you should use the following syntax.

```
1. DROP FUNCTION function name;
```

## 8. Designing of various Input screens/Forms

### HTML Form

An **HTML form** is a *section of a document* which contains controls such as text fields, password fields, checkboxes, radio buttons, submit button, menus etc.

An HTML form facilitates the user to enter data that is to be sent to the server for processing such as name, email address, password, phone number, etc.

### Why use HTML Form

HTML forms are required if you want to collect some data from of the site visitor.

For example: If a user wants to purchase some items on internet, he/she must fill the form such as shipping address and credit/debit card details so that item can be sent to the given address.

### HTML Form Syntax

1. **<form** action="server URL" method="gatepost">
2. //input controls e.g. textfield, textarea, radiobutton, button
3. **</form>**

### HTML Form Tags

Let's see the list of HTML 5 form tags.

| Tag        | Description                                               |
|------------|-----------------------------------------------------------|
| <form>     | It defines an HTML form to enter inputs by the used side. |
| <input>    | It defines an input control.                              |
| <textarea> | It defines a multi-line input control.                    |
| <label>    | It defines a label for an input element.                  |
| <fieldset> | It groups the related element in a form.                  |
| <legend>   | It defines a caption for a <fieldset> element.            |
| <select>   | It defines a drop-down list.                              |

|            |                                                            |
|------------|------------------------------------------------------------|
| <optgroup> | It defines a group of related options in a drop-down list. |
| <option>   | It defines an option in a drop-down list.                  |
| <button>   | It defines a clickable button.                             |

## HTML 5 Form Tags

Let's see the list of HTML 5 form tags.

| Tag        | Description                                                   |
|------------|---------------------------------------------------------------|
| <datalist> | It specifies a list of pre-defined options for input control. |
| <keygen>   | It defines a key-pair generator field for forms.              |
| <output>   | It defines the result of a calculation.                       |

## HTML <form> element

The HTML <form> element provide a document section to take input from user. It provides various interactive controls for submitting information to web server such as text field, text area, password field, etc.

Note: The <form> element does not itself create a form but it is container to contain all required form elements, such as <input>, <label>, etc.

### Syntax:

1. <form>
2. //Form elements
3. </form>

## HTML <input> element



The HTML `<input>` element is fundamental form element. It is used to create form fields, to take input from user. We can apply different input field to gather different information from user. Following is the example to show the simple text input.

Example:

1. `<body>`
2. `<form>`
3. Enter your name `<br>`
4. `<input type="text" name="username">`
5. `</form>`
6. `</body>`

Output:

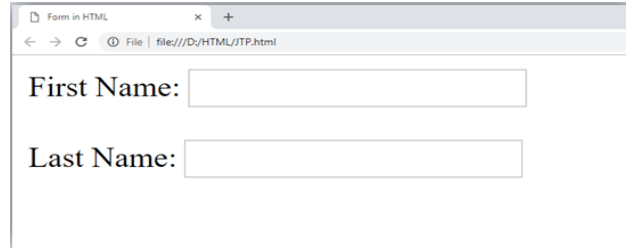
Enter your name

### HTML Text Field Control

The `type="text"` attribute of input tag creates text field control also known as single line text field control. The name attribute is optional, but it is required for the server-side component such as JSP, ASP, PHP etc.

1. `<form>`
2. First Name: `<input type="text" name="firstname"/> <br/>`
3. Last Name: `<input type="text" name="lastname"/> <br/>`
4. `</form>`

## Output:



The screenshot shows a web browser window with a single tab titled "Form in HTML". The address bar displays "File | file:///D:/HTML/ITP.html". The main content area contains two labels, "First Name:" and "Last Name:", each followed by a text input field.

## **9. Create reports using database connectivity of Front end with back end**

### **Objective/ Vision**

It provides a common platform to share the common people experiences, information's and harassment all over the world and people can discuss on any topic created by only registered user. Moreover, he/she can give the advice on any topic or report.

### **Users of the System**

1. Admin
2. Common People

### **Functional Requirements**

#### **1. Admin**

1. Can create and post the topic to be discussed and report respectively after getting logged in!
2. Can delete any report which looks like abusive matters.
3. Can view all reports and topics to be discussed and can search reports for each police station.
4. Can help in any report to proceed it further and can give it to the media.
5. Can view the previously posted comments and post a comment on each report or topic

#### **2. Common People**

1. Can view all reports posted by others after getting logged in!.
2. Can search the report of each police station.
3. Can view the previously posted comments. And post a comment on each report or topic.

### **Non-Functional Requirements**

1. Secure access of confidential data (user's details). SSL can be used.
2. 24 X 7 availability
3. Browser testing and support for IE, NN, Mozilla, and Firefox
4. Reports exportable in .XLS, .PDF

5. Create a detailed UML diagram (Component, Sequence, Class) for the system and its sub-components

### **User Interface Priorities**

1. Professional look and feel
2. Use of AJAX at least with all registration forms and with every search option and at the id of each searched result with on mouseover event.

### **Tools to be used**

1. Use any IDE to develop the project. It may be MyEclipse / Eclipse / NetBeans.
2. Oracle 10g for the database.
3. Server: Apache Tomcat/JBoss/Glassfish/WebLogic/WebSphere.

### **Front End and Back End**

1. **Front End:** JSP, JDBC, JavaScript, AJAX
2. **Back End:** Oracle

### **How project works?**

To get detail explanation about project download the document file. It includes snapshots with explanation.

### **Software Requirement to run this project**

1. You need to install an IDE Eclipse / MyEclipse / NetBeans.
2. Oracle 10g database. Here, we are using **system** for the username and **oracle** for the password.

### **How to run this project**

Import the project on the IDE and run it. All the tables will be created automatically.

Welcome Page



### Code:

```
package com.javatpoint;
import javax.servlet.*;
import java.sql.*;

public class MyListener implements ServletContextListener{

 public void contextInitialized(ServletContextEvent arg0) {

 Connection con=null;
 try{
 ResultSet rs;
 Class.forName("oracle.jdbc.driver.OracleDriver");
 con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

 PreparedStatement ps1=con.prepareStatement("Select * from FORUMREG");

 rs=ps1.executeQuery();
 if(rs.next())
 {System.out.println("your table name already exist");}
 else
 {System.out.println("else if part table does not exist new table has created");}
```

```

PreparedStatement ps2= con.prepareStatement("CREATE SEQUENCE JAVATPOINT
MINVALUE 1 MAXVALUE 999999999 INCREMENT BY 1 START WITH 1
NOCACHE NOORDER NOCYCLE");
ps2.executeUpdate();
PreparedStatement ps=con.prepareStatement("CREATE TABLE FORUMREG(ID
NUMBER,USERNAME VARCHAR2(4000),USERPASS VARCHAR2(4000),EMAIL
VARCHAR2(4000),MOBILE NUMBER,ADDRESS VARCHAR2(4000),CONSTRAINT
FORUMREG_PK PRIMARY KEY (ID) ENABLE)");
ps.executeUpdate();
PreparedStatement ps4=con.prepareStatement("CREATE TABLE FORUMREP(ID
NUMBER,COUNTRY VARCHAR2(4000),STATE VARCHAR2(4000),DISTRICT
VARCHAR2(4000),POLICE_STATION VARCHAR2(4000),REPORT
VARCHAR2(4000),STATUS VARCHAR2(4000),IMAGE BLOB,EMAIL
VARCHAR2(4000),POSTEDON DATE,CONSTRAINT FORUMREP_PK PRIMARY
KEY (ID) ENABLE)");
ps4.executeUpdate();
PreparedStatement ps5=con.prepareStatement("CREATE TABLE FORUMADVC(RID
NUMBER,ID NUMBER,CMT VARCHAR2(4000),EMAIL
VARCHAR2(4000),CONSTRAINT FORUMADVC_PK PRIMARY KEY (ID)
ENABLE)");
ps5.executeUpdate();
ps5= con.prepareStatement("CREATE TABLE FORUMTPC (ID NUMBER, TOPIC
VARCHAR2(4000), EMAIL VARCHAR2(4000),CREATEDON DATE NOT NULL
ENABLE,CONSTRAINT FORUMTPC_PK PRIMARY KEY (ID) ENABLE)");
ps5.executeUpdate();
ps5= con.prepareStatement("CREATE TABLE FORUMTADVC(ID NUMBER, TID
NUMBER, CMT VARCHAR2(4000), EMAIL VARCHAR2(4000),CONSTRAINT
FORUMTADVC_PK PRIMARY KEY (ID) ENABLE)");
ps5.executeUpdate();
Statement stmt=con.createStatement();
stmt.executeUpdate("CREATE OR REPLACE TRIGGER BI_FORUMREG before insert
on FORUMREG for each row begin select JAVATPOINT.nextval into :NEW.ID from
dual; end");
stmt.executeUpdate("CREATE OR REPLACE TRIGGER BI_FORUMREP before insert
on FORUMREP for each row begin select JAVATPOINT.nextval into :NEW.ID from
dual;end");
stmt.executeUpdate("CREATE OR REPLACE TRIGGER BI_FORUMADVC before
insert on FORUMADVC for each row begin select JAVATPOINT.nextval into :NEW.ID
from dual;end");
stmt.executeUpdate("CREATE OR REPLACE TRIGGER BI_FORUMTADVC before
insert on FORUMTADVC for each row begin select JAVATPOINT.nextval into :NEW.ID
from dual;end");
stmt.executeUpdate("CREATE OR REPLACE TRIGGER BI_FORUMTPC before insert
on FORUMTPC for each row begin select JAVATPOINT.nextval into :NEW.ID from
dual;end");}

```

```
}

 catch(Exception e){
 e.printStackTrace();

 }
}

 public void contextDestroyed(ServletContextEvent arg0) {
 System.out.println("project undeployed");
 }
}
```

## **10 Create database Design with normalization and implementing in any application**

### **Objective/ Vision**

A software product which provides solution for baby health, baby food, baby tips, baby products, baby names, parenting etc. Here, user can view baby names, baby names by religion, baby tips, baby food and baby product. Admin can add and delete baby names.

### **Live URL**

[www.babycareresolution.com](http://www.babycareresolution.com)

### **Users of the System**

1. Admin
2. Users

### **Functional Requirements**

#### **1. Admin**

1. Can login and logout.
2. Can add baby names.
3. Can view baby names.
4. Can delete baby names.
5. Can add new pages.

#### **2. Users**

1. Can view baby names.
2. Can view baby tips.
3. Can view baby food.
4. Can view baby products.

### **Non-Functional Requirements**

1. Secure access of confidential data (user's details). SSL can be used.



2. 24 X 7 availability
3. Browser testing and support for IE, NN, Mozilla, and Firefox
4. Reports exportable in .XLS, .PDF
5. Create a detailed UML diagram (Component, Sequence, Class) for the system and its sub-components

### **User Interface Priorities**

1. Professional look and feel
2. Use of AJAX at least with all registration forms and with every search option and at the id of each searched result with on mouseover event.

### **Tools to be used**

1. Use any IDE to develop the project. It may be MyEclipse / Eclipse / NetBeans.
2. Oracle 10g for the database.
3. Server: Apache Tomcat/JBoss/Glassfish/WebLogic/WebSphere.

### **Front End and Back End**

1. **Front End:** JSP, JDBC, JavaScript, AJAX
2. **Back End:** Oracle10g

### **How project works?**

To get detail explanation about project download the document file. It includes snapshots with explanation.

### **Software Requirement to run this project**

1. You need to install an IDE Eclipse / MyEclipse / NetBeans.
2. MySQL database. (Here, we are not using any username and password for MySQL database).

### **How to run this project**

- 1) Import the SQL file in MySQL (located in Web-Content directory)

2) Paste mysql-connector.jar file inside lib directory.

3) Import the project on the Eclipse IDE and run it.

### Password for Admin

Username is **admin** and password are **admin123**.

### Welcome Page

