

Experiment No: 1

Objective: Installing of Oracle 11g on Windows

S/w Requirement: CD of the setup Oracle 11g

Theory : Database is a collection of information in a structured way. We can say that it is a collection of a group of facts. Your personal address book is a database of names you like to keep track of, such as personal friends and members of your family.

ROLL_NO	NAME	ADDRESS	SUBJECTS
1716510001	Abc	C-6, Kanpur.	DBMS
1716510002	Pqr	A-5, Kanpur.	.DAA
1716510003	Dyz	R-2, Kanpur.	C
1716510004	Stq	M-1, Kanpur.	WebTech
1716510005	Plm	Y-7, Kanpur.	CD

Fig. 1

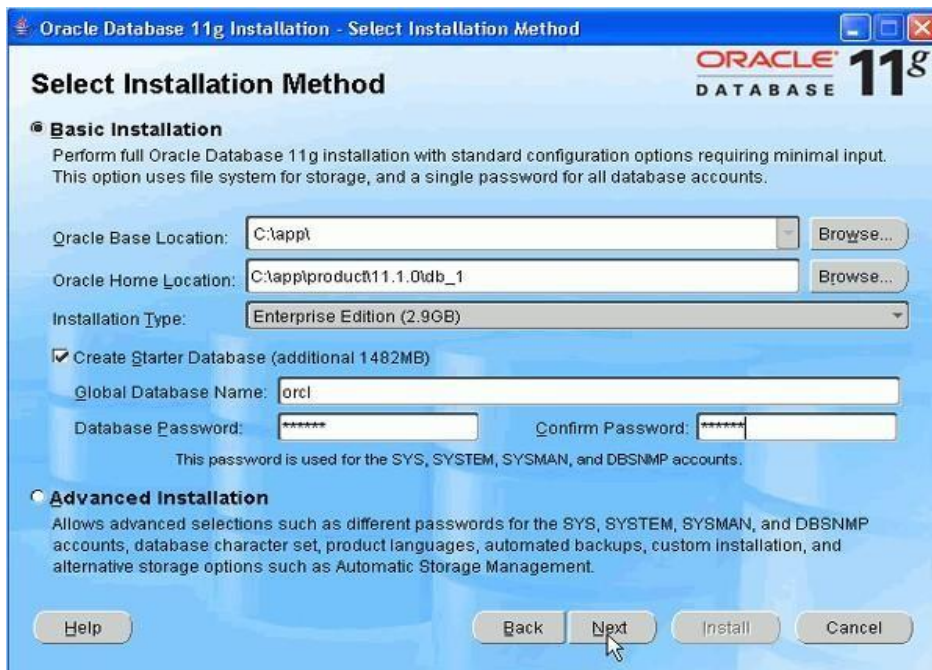
Implementation:

1. For this installation, you need either the DVDs or a downloaded version of the DVDs. In this tutorial, you install from the downloaded version. From the directory where the DVD files were unzipped, open Windows Explorer and double-click on **setup.exe** from the \db\Disk1 directory.

2.The product you want to install is **Database 11g**. Make sure the product is selected and click **Next**.



3. You will perform a basic installation with a starter database. Enter **orcl** for the Global Database Name and for Database Password and Confirm Password. Then, click **Next**



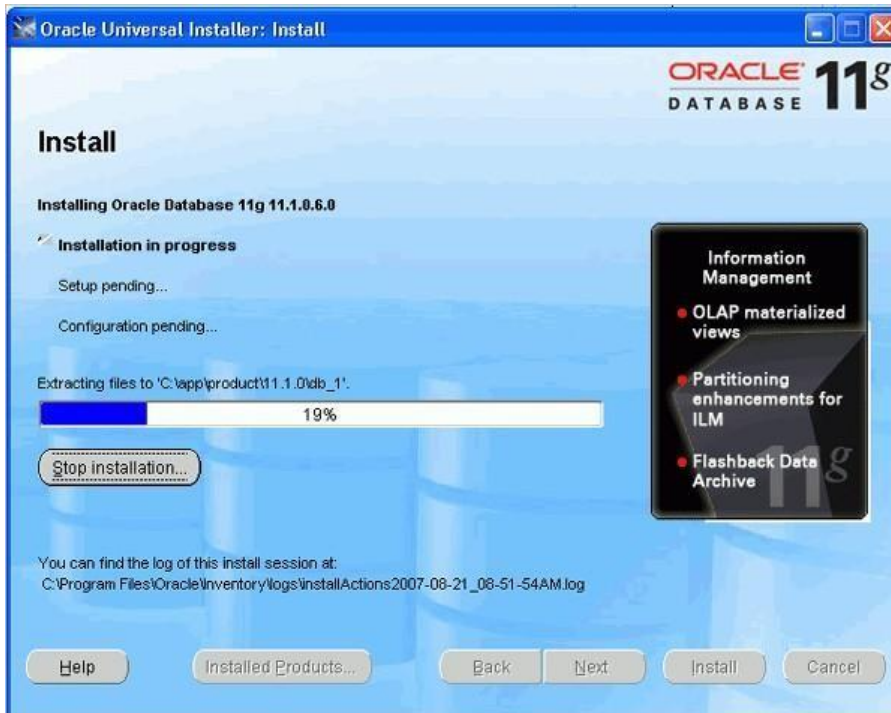
4. Configuration Manager allows you to associate your configuration information with your Metalink account. You can choose to enable it on this window. Then, click **Next**.



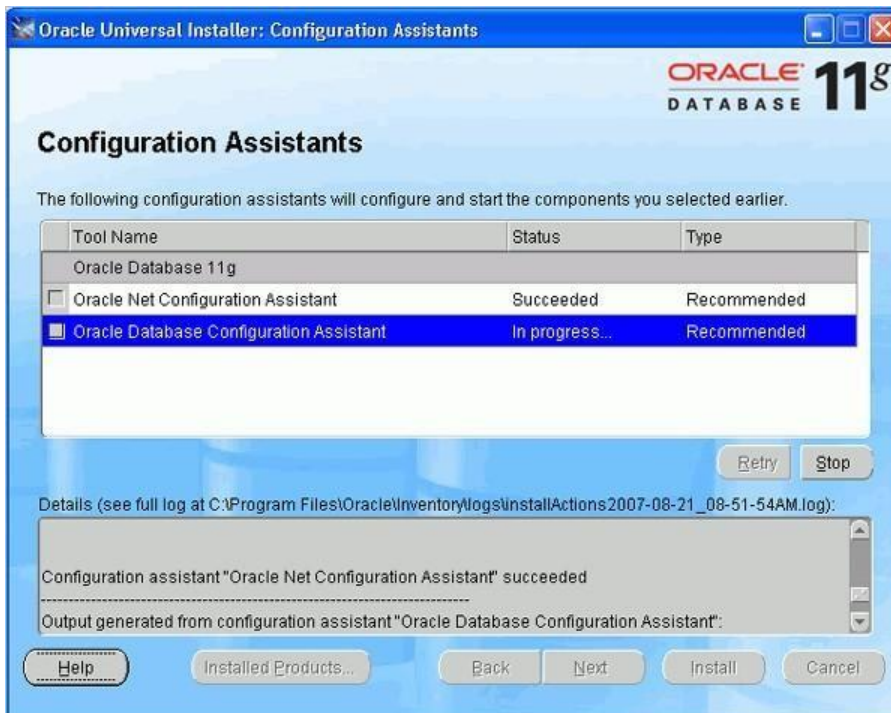
5. Review the Summary window to verify what is to be installed. Then, click **Install**.



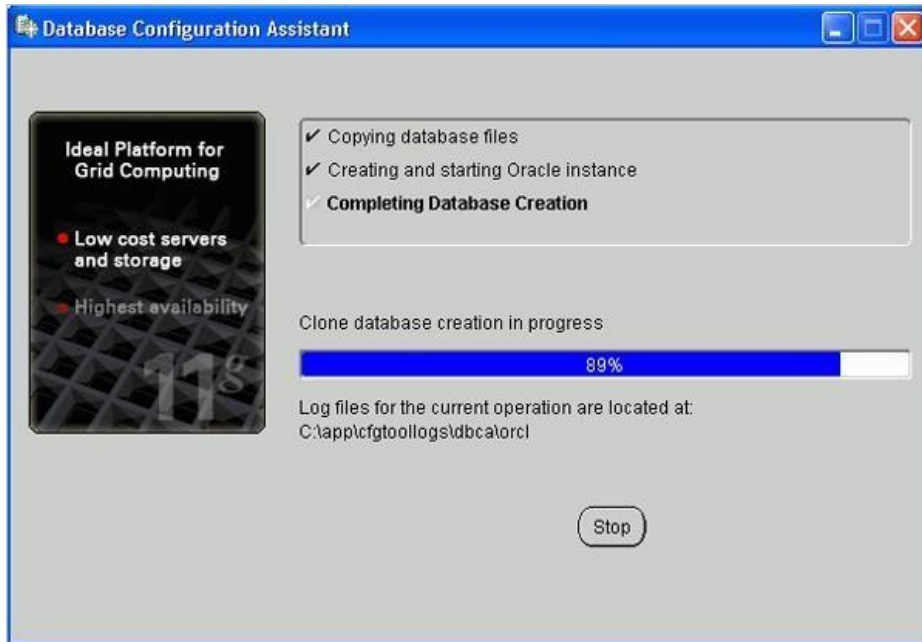
6. The progress window appears.



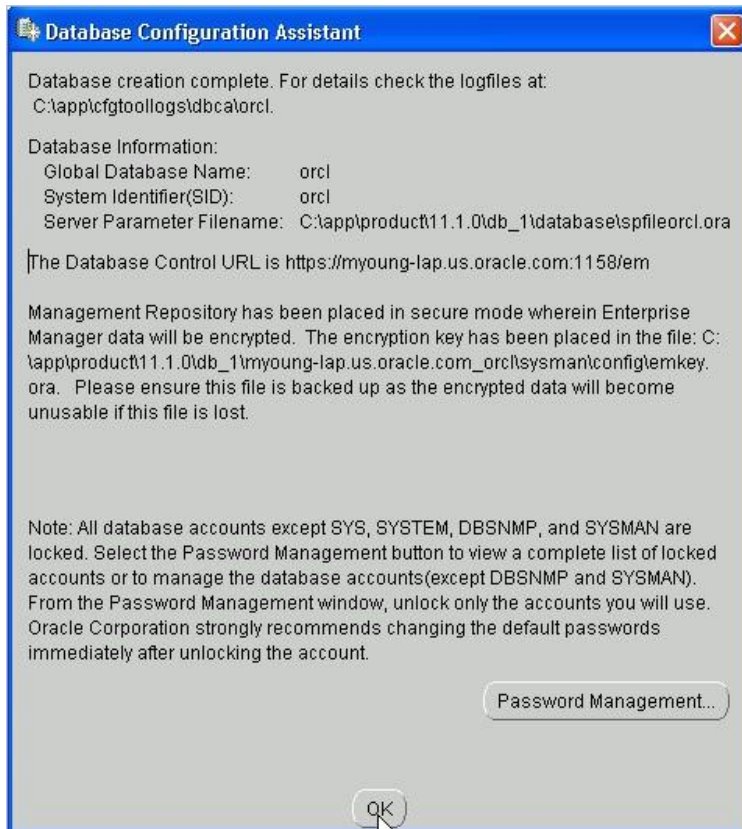
7. The Configuration Assistants window appears.



8. Your database is now being created.



9. When the database has been created, you can unlock the users you want to use. Click **OK**.



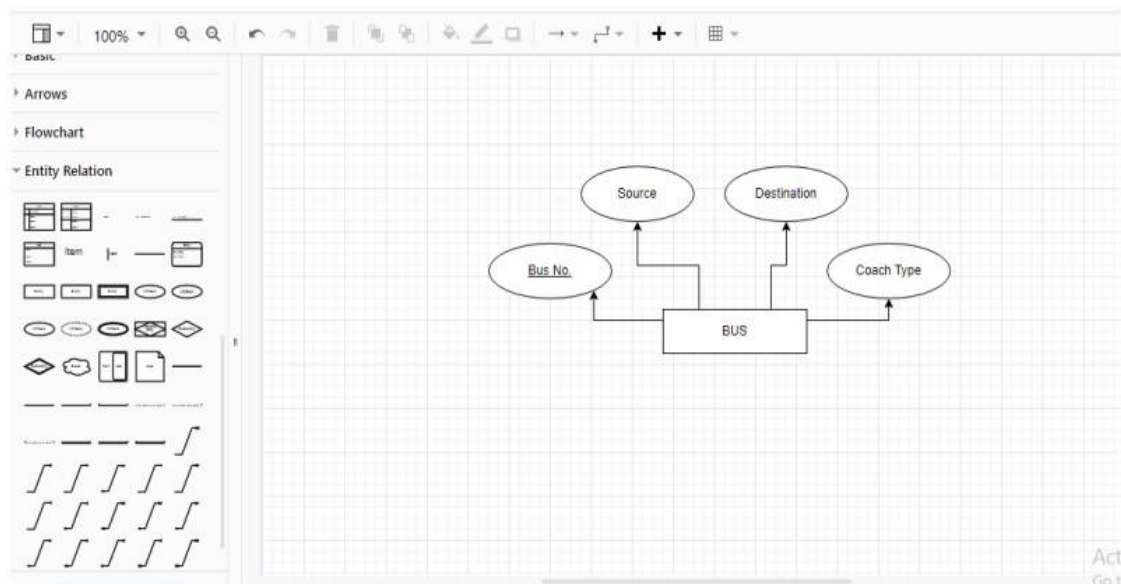
10. Click **Exit**. Click **Yes** to confirm exit.



Experiment No: 2

Objective: Creating Entity-Relationship Diagram using Case Tools (like draw.io, Creatly, Canva).

Theory: CASE stands for Computer Aided Software Engineering. It means, development and maintenance of software projects with help of various automated software tools. These tools are used to represent system components, data and control flow among various software components and system structure in a graphical form. For example, Flow Chart Maker tool for creating state-of-the-art flowcharts.

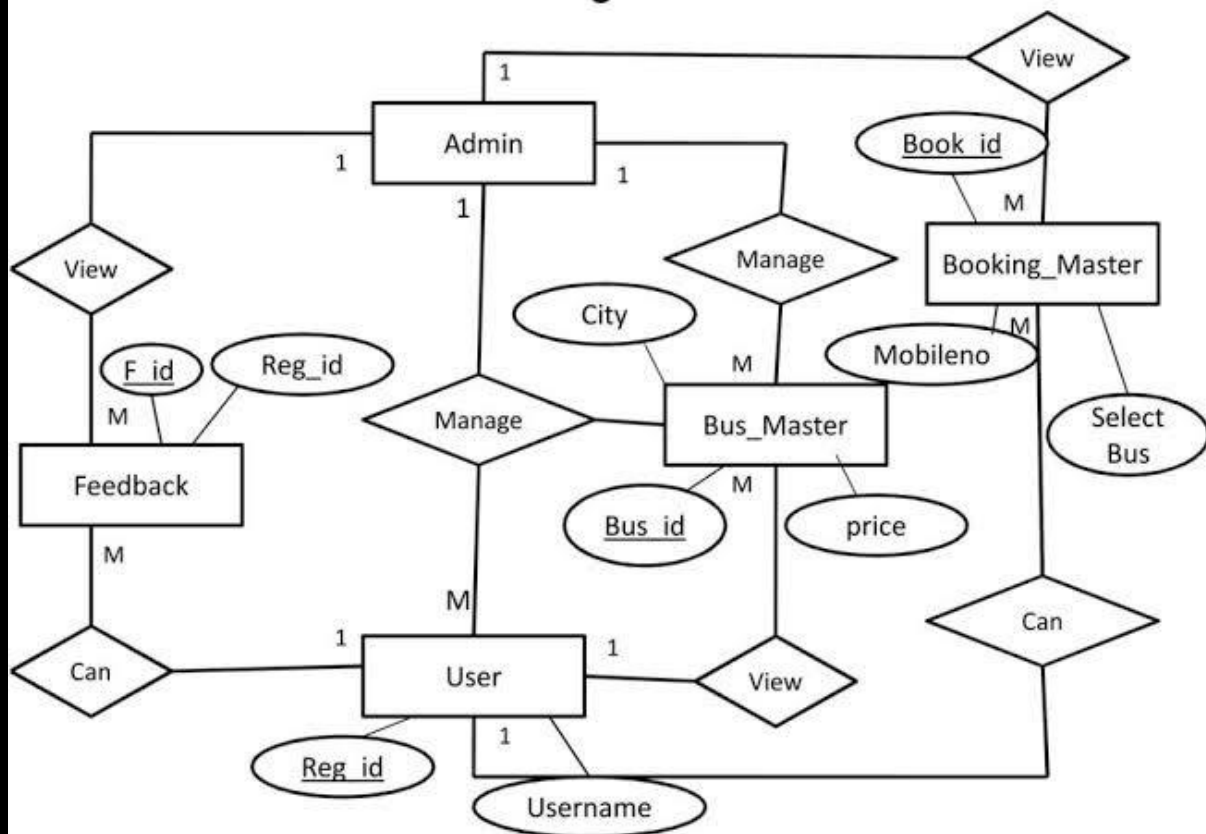


E-R Model:

The Following are the entities:

1. Admin
2. Feedback
3. User
4. Bus_master
5. Booking_master

ER Diagram



Experiment No. 3

Objective: Writing SQL statements Using Oracle/Mysql

a) Writing basic SQL Select Statements

SELECT statement is used for the purpose of projection of some or all of the attributes of a table.

SELECT clause make use of other clauses such as FROM , WHERE etc. But for SELECT statement we must need two clauses SELECT clause and FROM clause.

- **SELECT clause**

This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.

- **FROM clause**

This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.

SYNTAX:

```
SELECT column_1,column_2,... , column_k FROM table_name;
```

Or

```
SELECT * FROM
```

table_name; This is used to select all columns in a table

For example Select author_name

From

book_author;

This command will yield all the names of authors from the relation book_author.

1. Select * from customer;

Output: all columns and data of customer table

2. SELECT customerid, customername FROM customer;

Output:

CUSTOMERID	CUSTOMERNAME
-----	-----
id01	vipul manuja
id02	harshit tomar
id03	bhawna madnawat
id04	anjana gautam
id05	divyansh yadav
id06	aastha gupta
id07	bhaskar sharma

id08 kartik kumar
8 rows selected.

3. SELECT DISTINCT city FROM customer; Output:

CITY

Delhi
Greater
Noida
Gurgaon
Haryana

b) Restricting and sorting data

We can restrict the data being retrieved by the SELECT statement by using the WHERE clause.

- **WHERE clause**

This clause defines predicate or conditions, which must match in order to qualify the attributes to be projected.

SYNTAX:

```
SELECT column_1,column_2,... ,  
column_k FROM table_name  
WHERE conditional_expression;
```

For example: Select
author_name From
book_author
Where age > 50;

- **ORDER BY clause**

Sorting of data can be done by using ORDER BY clause and specifying the order in which we want to sort the data.

SYNTAX:

```
SELECT  
column_1,column_2,...,column_k FROM  
table_name ORDER  
BY column_m ASC;  
Or SELECT  
column_1,column_2,...,column_k FROM  
table_name  
ORDER BY column_m DESC;
```

For Example: SELECT
author_name

```
FROM
book_author
ORDER BY auther_name ASC;
```

1. select customerid, customername from customer266 where city='delhi';

Output:

CUSTOMERID CUSTOMERNAME

id01	vipul manuja
id04	anjana gautam
id06	aastha gupta
id07	bhaskar sharma

2. SELECT customerid, customername , city FROM Customer266 ORDER BY City;

CUSTOMERID CUSTOMERNAME CITY

id01	vipul manuja	delhi
id04	anjana gautam	delhi
id07	bhaskar sharma	delhi
id06	aastha gupta	delhi
id03	bhawna madnawat	greater noida
id05	divyansh yadav	greater noida
id08	kartik kumar	gurgaon
id09	kunal sharma	gurgaon
id02	harshit tomar	haryana
id10	deepak kaushik	shahdra

10 rows selected.

3. SELECT CUSTOMERID, CUSTOMERNAME FROM CUSTOMER266 ORDER BY CUSTOMERNAME DESC;

Output:

CUSTOMERID CUSTOMERNAME

id01	vipul manuja
id09	kunal sharma
id08	kartik kumar
id02	harshit tomar
id05	divyansh yadav
id10	deepak kaushik
id03	bhawna madnawat
id07	bhaskar sharma

id04 anjana gautam
id06 aastha
gupta 10 rows selected.

SQL> create table student266(roll_no number(2) primary key,name
char(10)); Output: Table created.

SQL> desc student266;

Output:

<u>Name</u>	<u>Null? Type</u>
ROLL_NO	NOT NULL
NUMBER(2) NAME	CHAR(10)

SQL> create table employee266(emp_no number(2)
primary key,name char(10)); Output: Table created.

SQL> desc employee; Output:

Name	Null? Type
EMP_NO	NOT NULL
NUMBER(2) NAME	CHAR(10)

SQL> insert into student266

values(1,'A'); Output: 1 row created.

SQL> insert into student266

values(2,'B'); Output: 1 row created.

SQL> insert into student266

values(3,'C'); Output: 1 row created.

SQL> select * from student266;

Output:

ROLL_NO	NAME
1	A
2	B
3	C

SQL> insert into employee266

values(7,'E'); Output: 1 row created.

SQL> insert into employee266

values(1,'A'); Output: 1 row created.

```
SQL> select * from
```

```
employee266; Output:
```

```
EMP_NO NAME
```

```
-----
```

```
7 E
```

```
1 A
```

c) **Manipulating Data using Having Function**

UPDATE/SET/WHERE clauses

This command is used for updating or modifying the values of columns in a table (relation).

SYNTAX:

```
UPDATE table_name
```

```
SET column_name = value [, column_name = value ...] [WHERE condition]
```

For example:

```
UPDATE library_record
```

```
SET Author="webmaster"
```

```
WHERE Author="anonymous";
```

HAVING clause

At times it is useful to state a condition that applies to groups rather than to tuples.

For Example:

```
SELECT
```

```
column_1,column_2 FROM
```

```
table_name
```

```
GROUP BY column_1
```

```
HAVING search_condition;
```

DELETE/FROM/WHERE clauses

This command is used for removing one or more rows from a table (relation).

SYNTAX:

```
DELETE FROM table_name [WHERE condition];
```

For example:

```
DELETE FROM
```

```
library_record266 WHERE
```

```
Author="unknown"
```

Aggregating data using group
function

GROUP BY clause

There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group or sets of tuples, we specify this wish in SQL using the GROUP BY clause. The attribute or attributes given in the group by clause are used to form group. Tuples with the same value on all attributes in the group by clause are placed in one group.

SYNTAX:

```
SELECT column_1,  
       column_2 FROM  
tablename GROUP BY  
column_1;
```

A group function returns a result based on group of rows. Some are purely mathematical functions.

- A. AVG function: It returns average of values of the column specified in the arguments in the columns.

SYNTAX: Select avg(column_name whose avg to find) from <table_name> where condition;

- B. min function: This function will give the least value of the column present in the argument.

SYNTAX: Select min(column_name whose min to find) from <table_name> where condition;

- C. max function: This function will give the maximum value of the column present in the argument.

SYNTAX: Select max(column_name whose max to find) from <table_name> where condition;

- D. sum: This function will give the sum of value of the column present in the argument.

SYNTAX: Select sum(column_name whose sum to find) from <table_name> where condition;

- E. count: This function is used to count the number of rows in function.

SYNTAX: Select count (*) from <table_name> ;

e) Displaying data from multiple tables

Theory:

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

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

ORDER-ID	CUSTOMER-ID	Order-Date
10309	2	1996-09-18
10310	37	1996-09-18
10311	77	1996-09-20

CustomerID	Customer Name	Contact-Name	Company
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo	Ana Trujillo	Mexico
3	Antonio	Antonio	Mexico

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

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

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

```
SELECT Orders.OrderID, Customers.CustomerName,
Orders.OrderDate FROM
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

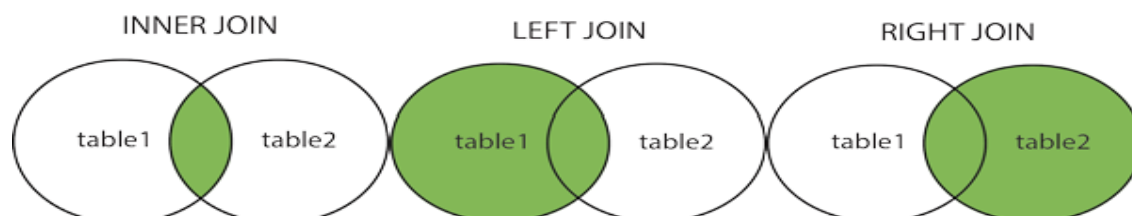
Output:

OrderID	CustomerName	OrderDate
10308	Ana Trujillo	9/18/1996
10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996

Different Types of SQL JOINS

Here are the different types of the JOINS in SQL:

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



Experiment No. 4

Objective: Creating Procedure and Functions

Theory:

PL/SQL supports the two types of programming:

Procedure

. Function.

Procedures are usually used to perform any specific task and functions are used to compute a value.

PROCEDURE

The basic syntax for the creating procedure is:

```
CREATE OR REPLACE PROCEDURE procedure _name (arguments)AS/IS  
procedure body;
```

The body of procedure is block of statements with declarative executable and exception sections.

The declarative section is located between the IS /AS keyword and BEGIN keyword.

The executable section is located between BEGIN and EXCEPTION keywords or between the BEGIN and END keywords if there is no EXCEPTION handling section.

If EXCEPTION handling is present, it is located between exception and END keywords.

Steps for Creating Procedure:

```
CREATE OR REPLACE PROCEDURE procedure _name (parameter list)AS/IS  
(Declarative section )  
BEGIN  
(Executable section  
)  
EXCEPTION  
(Error handling or exception section).  
End the procedure name
```

To execute the procedure we have to write a block of statement:

```
Begin Procedurename(data);
```

```
End;
```

Example: Lets create a table users and make a procedure for counting users.

Result Grid				
Filter Rows:				
	user_id	username	email	created_at
▶	1	JohnDoe	john@example.com	2023-12-21 14:26:30
	2	AliceSmith	alice@example.com	2023-12-21 14:26:30
	3	BobJohnson	bob@example.com	2023-12-21 14:26:30
•	NULL	NULL	NULL	NULL

Queries for creating procedure:

-- Delimiter change to create the table and procedure

DELIMITER //

-- Create the users table

```
CREATE TABLE users (
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(50) NOT NULL,
  email VARCHAR(100) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;
```

-- Insert sample data into the users table

```
INSERT INTO users (username, email) VALUES
  ('JohnDoe', 'john@example.com'),
  ('AliceSmith', 'alice@example.com'),
  ('BobJohnson', 'bob@example.com');
```

-- Create the procedure to get total users

```
CREATE PROCEDURE GetTotalUsers()
BEGIN
  DECLARE total_users INT; -- Variable to store the total number of users

  -- Select count of users into the variable
  SELECT COUNT(*) INTO total_users FROM users;

  -- Display the result
  SELECT CONCAT('Total number of users: ', total_users) AS 'Result';
END //
```

-- Reset the delimiter to default

DELIMITER ;

select* from users;

call GetTotalUsers();

Output:

Result Grid	
Filter Rows:	
	Result
▶	Total number of users: 3

FUNCTION

```
CREATE OR REPLACE FUNCTION function _name (parameter list)AS/  
IS Return datatype is/as  
(local declaration)
```

```
BEGIN
```

```
(Exception section)
```

```
EXCEPTION
```

```
(Error handling or exception  
section). End Function name;
```

A function has two parts, namely function specification and function body. The function specification begins with the keyword function and end with return clause. The function bodies begin with the keyword is/as and end with keyword end.

Now creating function for the same table users.

```
CREATE FUNCTION GetTotalUsersCount()
```

```
BEGIN
```

```
DECLARE total_users INT; -- Variable to store the total number of users
```

```
SELECT COUNT(*) INTO total_users FROM users;
```

```
RETURN total_users; -- Return the total count
```

```
END //
```

Output:

Result Grid		Filter Rows:
	Result	
▶	Total number of users: 3	

Experiment No: 5

Objective: Write a CURSOR to display list of clients in the client Master Table.

Theory:

There are two types of cursor
Explicit cursor
Implicit cursor

A. Explicit cursor

An explicit cursor is one in which cursor name is explicitly assigned to select statement. An implicit cursor is used for all other sql statements. Processing of an explicit cursor involves four steps. Processing of an implicit cursor is taken care by SQL . The declaration of the cursor is done in the declarative part of the block.

B. Implicit cursor:

These are inbuilt cursor in oracle.

In SQL, you can use a cursor to iterate through records in a table. Here's an example of how you might create and use a cursor to display a list of clients from a hypothetical "ClientMaster" table:

Assuming the "ClientMaster" table has columns like **ClientID**, **ClientName**, **Address**, and **PhoneNumber**, you can create a cursor to fetch and display the list of clients:

	ClientID	ClientName	Address	PhoneNumber
▶	1	John Doe	123 Main St, CityA	555-1234
	2	Jane Smith	456 Elm St, CityB	555-5678
	3	Alice Johnson	789 Oak St, CityC	555-9876
●	NULL	NULL	NULL	NULL

Code:

-- Create a sample ClientMaster table

```
CREATE TABLE ClientMaster (  
  ClientID INT PRIMARY KEY,  
  ClientName VARCHAR(100),  
  Address VARCHAR(255),  
  PhoneNumber VARCHAR(20)  
);
```

-- Insert sample data into ClientMaster table

```
INSERT INTO ClientMaster (ClientID, ClientName, Address, PhoneNumber)  
VALUES  
  (1, 'John Doe', '123 Main St, CityA', '555-1234'),  
  (2, 'Jane Smith', '456 Elm St, CityB', '555-5678'),  
  (3, 'Alice Johnson', '789 Oak St, CityC', '555-9876');
```

```

-- Create a stored procedure to fetch and display client information
DELIMITER //

CREATE PROCEDURE DisplayClients()
BEGIN
    -- Declare variables
    DECLARE done INT DEFAULT FALSE;
    DECLARE clientId INT;
    DECLARE clientName VARCHAR(100);
    DECLARE clientAddress VARCHAR(255);
    DECLARE clientPhoneNumber VARCHAR(20);

    -- Declare cursor
    DECLARE client_cursor CURSOR FOR
    SELECT ClientID, ClientName, Address, PhoneNumber
    FROM ClientMaster;

    -- Declare continue handler
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Open cursor
    OPEN client_cursor;

    -- Fetch data from cursor
    read_loop: LOOP
        FETCH client_cursor INTO clientId, clientName, clientAddress, clientPhoneNumber;
        IF done THEN
            LEAVE read_loop;
        END IF;

        -- Display client information
        SELECT CONCAT('Client ID: ', clientId, ', Client Name: ', clientName, ', Address: ',
clientAddress, ', Phone Number: ', clientPhoneNumber);
    END LOOP;

    -- Close cursor
    CLOSE client_cursor;
END //

DELIMITER ;

-- Call the stored procedure to display clients
CALL DisplayClients();

select * from ClientMaster

```

Output:

Result Grid		Filter Rows:	Export:
	CONCAT('Client ID: ', clientId, ', Client Name: ', clientName, ', Address: ', clientAddress, ', Phone Number: ', clientPhoneNumber)		
	NULL		

Experiment No: 6

Objective: Design and implementation of Student Information System

Specification- To create a database with an interface for implementing student information system for an educational organization.

a.) Hardware & Software Design Requirements

S. No.	Requirements	Configuration
1	System	1
2	Operating System	Windows 7 or higher / Linux
3	Front End	C++/Java/Php etc.
4	Back End	Oracle 11g/MySql

b.) Steps for Achieving Objective

Designing and implementing a Student Information System (SIS) on MySQL involves creating a database schema, tables, relationships, and potentially implementing procedures or triggers to manage student information efficiently. Here's a basic example of how you can design an SIS using MySQL:

Entity-Relationship Diagram (ERD)

Let's start by creating a simple ERD for a Student Information System with three main entities: Students, Courses, and Enrollments.

Entities:

1. ****Students:****

- student_id (Primary Key)
- name
- email
- date_of_birth
- other relevant student details

2. ****Courses:****

- course_id (Primary Key)
- course_name
- course_description
- other relevant course details

3. ****Enrollments:****

- enrollment_id (Primary Key)
- student_id (Foreign Key referencing Students table)
- course_id (Foreign Key referencing Courses table)
- enrollment_date
- other relevant enrollment details

Implementation in MySQL Workbench:

Step 1: Creating the Database

Open MySQL Workbench and create a new schema (database). Right-click on the schema section, choose "Create Schema," and name it, e.g., `student_information_system`.

Step 2: Creating Tables

Create tables for `Students`, `Courses`, and `Enrollments` based on the attributes mentioned in the ERD.

Students Table:

```
```sql
CREATE TABLE Students (
 student_id INT PRIMARY KEY AUTO_INCREMENT,
 name VARCHAR(100),
 email VARCHAR(100),
 date_of_birth DATE,
 -- Other relevant fields
);
```
```

Courses Table:

```
```sql
CREATE TABLE Courses (
 course_id INT PRIMARY KEY AUTO_INCREMENT,
 course_name VARCHAR(100),
 course_description VARCHAR(255),
 -- Other relevant fields
);
```
```

Enrollments Table:

```
```sql
CREATE TABLE Enrollments (
 enrollment_id INT PRIMARY KEY AUTO_INCREMENT,
 student_id INT,
 course_id INT,
 enrollment_date DATE,
 FOREIGN KEY (student_id) REFERENCES Students(student_id),
 FOREIGN KEY (course_id) REFERENCES Courses(course_id)
 -- Other relevant fields
);
```
```

Step 3: Establishing Relationships

Connect the `student_id` field in the `Enrollments` table to the `student_id` field in the `Students` table and the `course_id` field in the `Enrollments` table to the `course_id` field in the `Courses` table. This ensures referential integrity.

Step 4: Indexing and Optimization

Consider adding indexes to columns that are frequently used in queries, like foreign keys or columns used for searching/filtering.

Step 5: Testing

Once the tables are created and relationships established, you can insert sample data and run test queries to ensure that the system functions as intended.

Experiment No: 7

Objective: Execute the queries related to group by and having clause on the table SALES ORDER.

Theory:

GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

SYNTAX:

```
SELECT column1, column2 FROM table_name WHERE [ conditions ] GROUP BY column1, column2 ORDER BY column1, column2;
```

HAVING clause enables you to specify conditions that filter which group results appear in the final results. The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

SYNTAX:

```
SELECT column1, column2 FROM table1, table2 WHERE [ conditions ] GROUP BY column1, column2 HAVING [ conditions ] ORDER BY column1, column2;
```

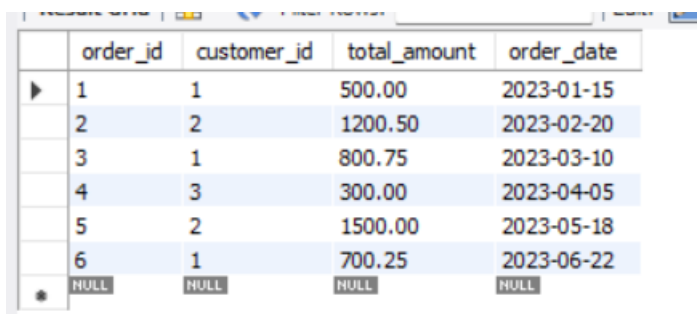
Create the table and insert values.

-- Create the Sales Order table

```
CREATE TABLE sales_order (  
    order_id INT AUTO_INCREMENT,  
    customer_id INT,  
    total_amount DECIMAL(10, 2),  
    order_date DATE,  
    PRIMARY KEY (order_id)  
);
```

-- Insert sample values into the Sales Order table

```
INSERT INTO sales_order (customer_id, total_amount, order_date) VALUES  
(1, 500.00, '2023-01-15'),  
(2, 1200.50, '2023-02-20'),  
(1, 800.75, '2023-03-10'),  
(3, 300.00, '2023-04-05'),  
(2, 1500.00, '2023-05-18'),  
(1, 700.25, '2023-06-22');
```





| | order_id | customer_id | total_amount | order_date |
|---|----------|-------------|--------------|------------|
| ▶ | 1 | 1 | 500.00 | 2023-01-15 |
| | 2 | 2 | 1200.50 | 2023-02-20 |
| | 3 | 1 | 800.75 | 2023-03-10 |
| | 4 | 3 | 300.00 | 2023-04-05 |
| | 5 | 2 | 1500.00 | 2023-05-18 |
| | 6 | 1 | 700.25 | 2023-06-22 |
| ✱ | NULL | NULL | NULL | NULL |

Now execute query using group by and having.

```
SELECT customer_id, SUM(total_amount) AS total_sales  
FROM sales_order  
GROUP BY customer_id  
HAVING SUM(total_amount) > 1000;
```

OUTPUT:

| Result Grid   Filter Rows | | |
|---|-------------|-------------|
| | customer_id | total_sales |
| ▶ | 1 | 2001.00 |
| | 2 | 2700.50 |