DRONACHARYA
Group of Institutions

B-27, Knowledge Park – III, Greater Noida Uttar Pradesh - 201308
Approved by: All India Council for Technical Education (AICTE), New Delhi
Affiliated to: Dr. A. P. J. Abdul Kalam Technical University (AKTU), Lucknow

DEPARTMENT OF INFORMATION TECHNOLOGY

DATABASE MANAGEMENT SYSTEM LAB

SUBJECT CODE: BCS-551

B.Tech., Semester -V

Session: 2024-25, ODD Semester

Table of Contents

DRONACHARYA GROUP OF INSTITUTIONS GREATER NOIDA

VISION

- Instilling core human values and facilitating competence to address global challenges by providing Quality Technical Education.

MISSION

- M1 - Enhancing technical expertise through innovative research and education, fostering creativity and excellence in problem-solving.
- M2 - Cultivating a culture of ethical innovation and user-focused design, ensuring technological progress enhances the well-being of society.
- M3 - Equipping individuals with the technical skills and ethical values to lead and innovate responsibly in an ever-evolving digital land.

DEPARTMENT OF INFORMATION TECHNOLOGY

VISION

To provide students with theoretical understanding and technical proficiency in Information Technology, instill with moral and ethical values, to excel in academic, industry, and research settings.

MISSION

M1: To instill in students a strong foundation of both the theory and practical application of IT skills, combined with the innovation and research approaches to keep pace with emerging technologies.

M2: To empower graduates to become global leaders specializing in field of Information Technology.

M3: To impart to students the social, ethical, and moral values necessary for them to make substantial contributions to society.

Program Outcomes (POs)

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: 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.

PO3: 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.

PO4: 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.

PO5: 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.

PO6: 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.

PO7: 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.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of engineering practice.

PO 9: Individual and teamwork: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: 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.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply theseto one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: 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.

Programme Educational Objectives (PEOs)

PEO1: Apply the knowledge of mathematics, science and engineering fundamentals to identify and solve IT and engineering problems.

PEO2: Use various software tools and technologies to solve problems related to academia, industry and society.

PEO3: Work with ethical and moral values in the multi-disciplinary teams and can communicate effectively among team members with continuous learning.

Program Specific Outcomes (PSOs)

PSO1: Ability to think logically and apply programming knowledge and practices in analyzing real world problems and provide solutions to meet the needs of society.

PSO2: Enhance the competence of technocrats to provide professional engineering solutions as per the industrial and societal needs.

PSO3: To cultivate and produce highly motivated engineers committed to lifelong learning, pursuing research, higher education, and embracing competitive challenges to excel in the future.

Database Management System Lab (BCS-551)

Cos	COURSE OUTCOMES
BCS-551.1	Understand and apply oracle 11 g for creating tables, views, indexes, sequences and other database objects
BCS-551.2	Design and implement a database schema for company data base, banking data base, library information system, payroll processing system, student information System.
BCS-551.3	Write and execute simple and complex queries using DDL, DML, DCL and TCL.
BCS-551.4	Write and execute PL/SQL blocks, procedure functions, packages and triggers, Cursors.
BCS-551.5	Enforce entity integrity, referential integrity, key constraints, domain constraints on database.

Mapping of Program Outcomes with Course Outcomes (COs)

CO-PO Matrix												
Course Outcomes	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
BCS-551.1	2	2	3	3	3	-	-	-	-	-	-	2
BCS-551.2	3	3	3	2	2	-	-	-	-	-	-	3
BCS-551.3	2	3	3	3	3	-	-	-	-	-	-	2
BCS-551.4	2	3	2	2	2	-	-	-	-	-	-	2
BCS-551.5	2	3	2	2	2	-	-	-	-	-	-	3

CO-PSO Matrix			
COs	**PSO1**	**PSO2**	**PSO3**
BCS-551.1	**1**	**2**	
BCS-551.2	**1**	**3**	
BCS-551.3	**1**	**3**	
BCS-551.4	**1**	**2**	
BCS-551.5	**1**	**3**	

List of Experiments

SR. No.	Experiments
1	Installing oracle/ MYSQL.
2	Creating Entity-Relationship Diagram using case tools.
3	Writing SQL statements Using ORACLE /MYSQL: a) Writing basic SQL SELECT statements. b) Restricting and sorting data. c) Displaying data from multiple tables. d) Aggregating data using group function. e) Manipulating data. f) Creating and managing tables.
4	Creating procedure and functions.
5	Design and implementation of Student Information System.
6	Write a CURSOR to display list of clients in the client Master Table.
7	Execute the queries related to Group By and having Clause on tables SALES_ORDER.
8	Execute the following queries: a) The NOT NULL b) The UNIQUE Constraint c) The PRIMARY KEY Constraint d) The CHECK Constraint e) Define Integrity Constraints in ALTER table Command
9	Execute Nested Queries on tables CLIENT_MASTER, PRODUCT_MASTER, SALESMAN_MASTER, SALES_ORDER, SALES_ORDER_DETAILS.
10	Execute Queries related to Exists, Not Exists, Union, Intersection, Difference, Join on tables CLIENT_MASTER, PRODUCT_MASTER, SALESMAN_MASTER, SALES_ORDER_DETAILS>

Experiment No: 1

Program Name: Installing Oracle

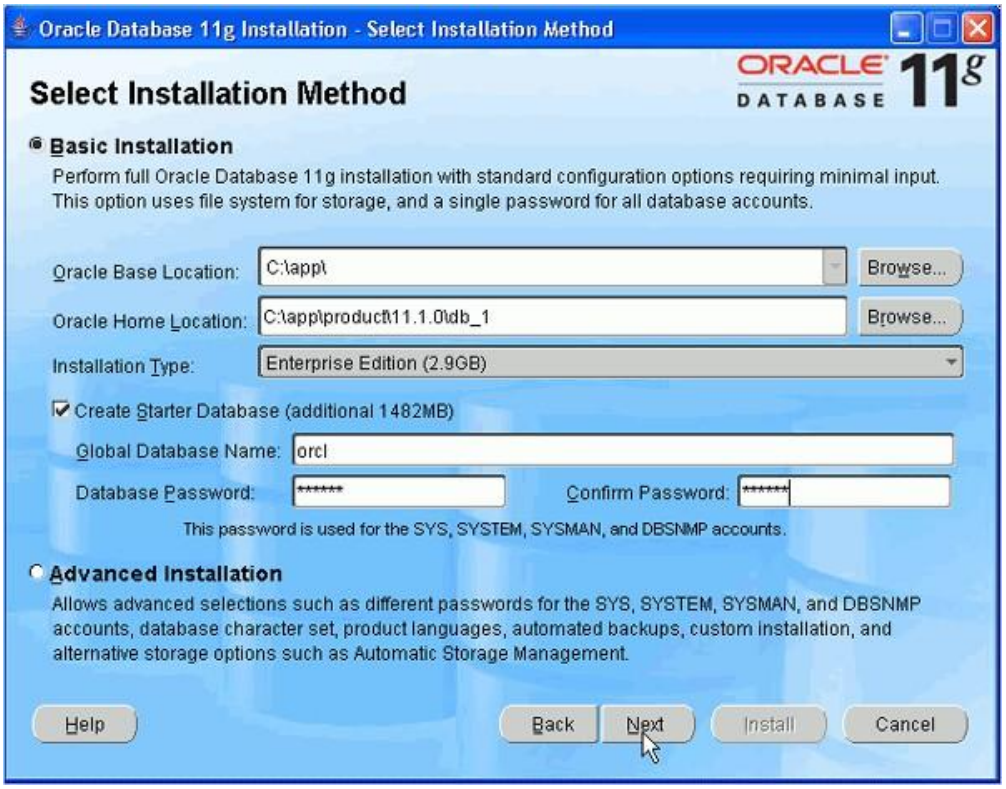
Theory Concept: To install the software, you must use the Universal installer.

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 wereunzipped, open Windows Explorer and double-click on **setup.exe** from the \db\Disk1 directory.

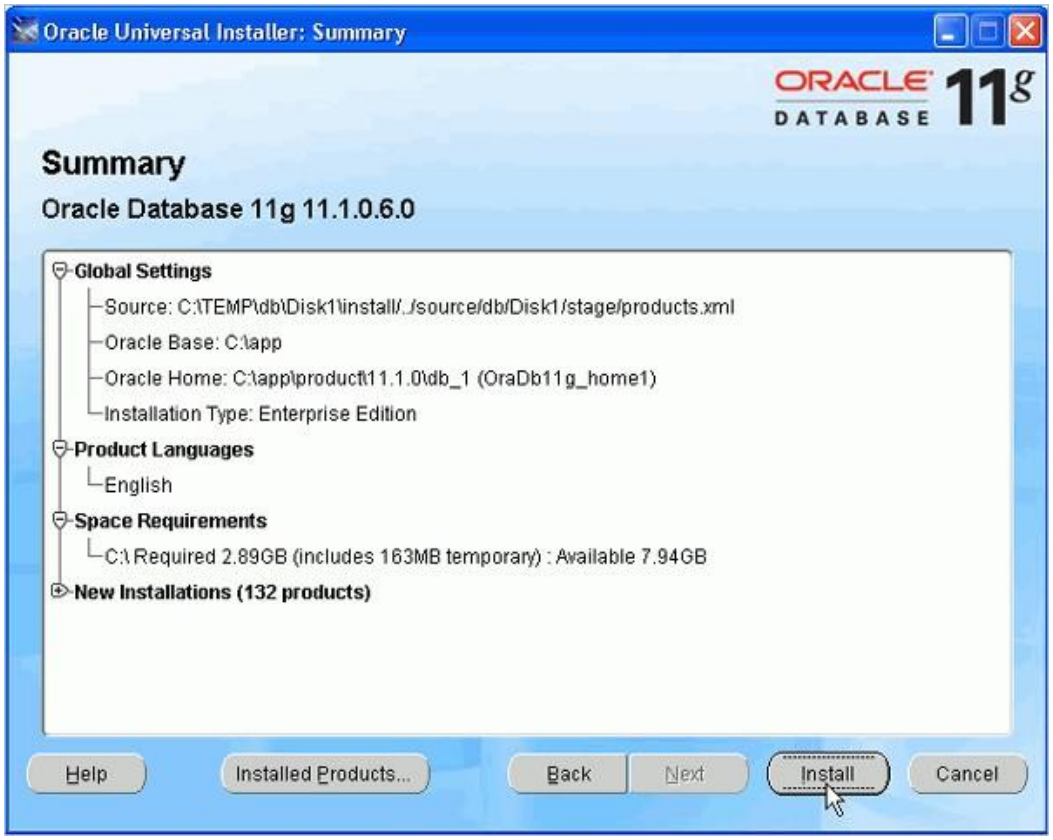
2. The product you want to install is **Database 11g**. Make sure the product is selected and click **Next**.

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**

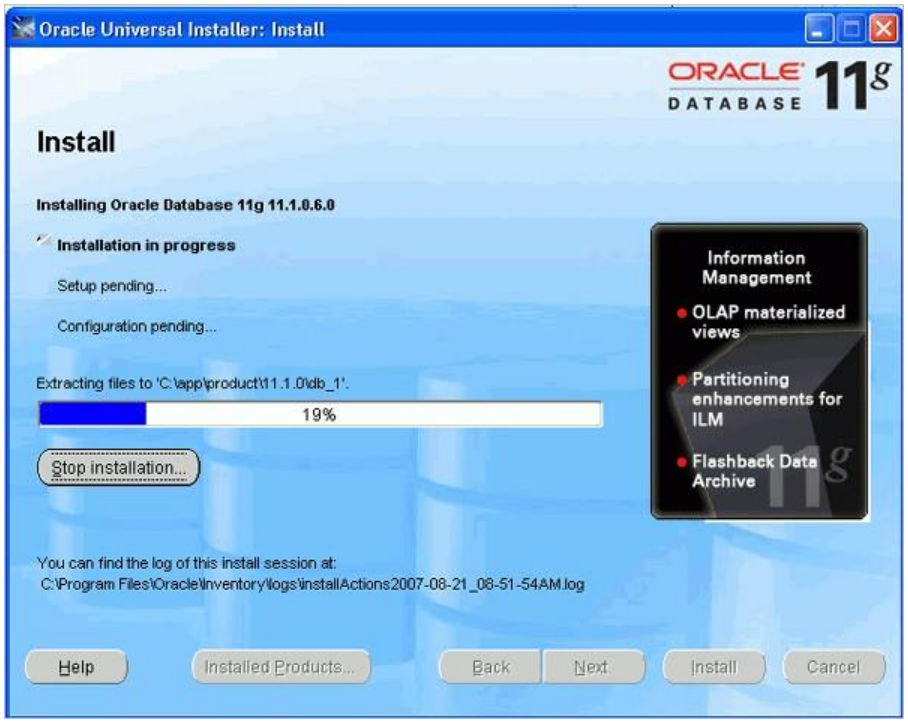


4. Configuration Manager allows you to associate your configuration information with your Metalink account. You can choose to enable it on this window. Then, click **Next.**

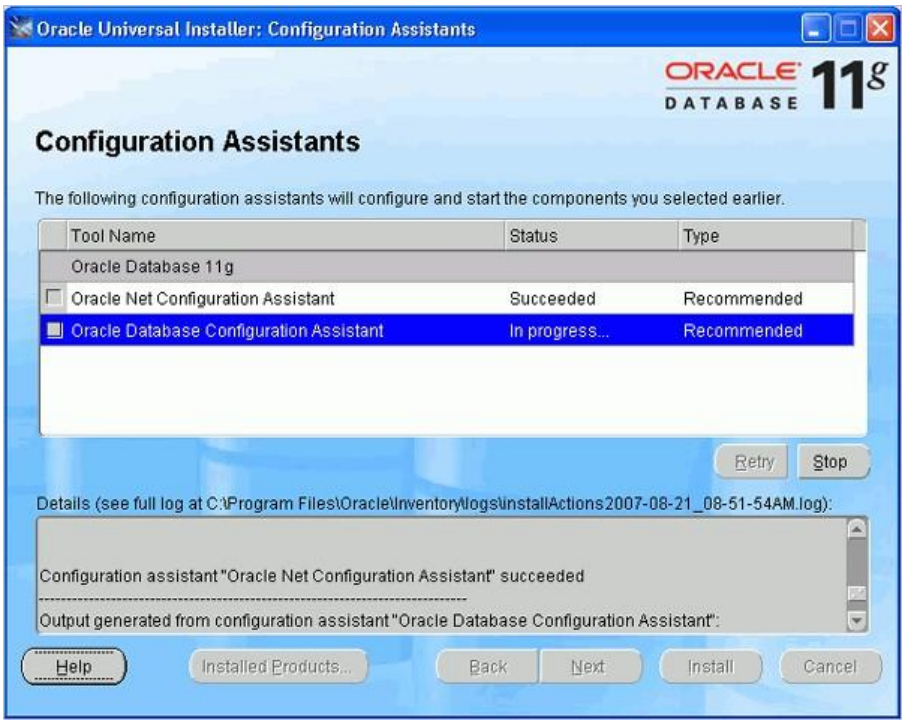
5. Review the Summary window to verify what is to be installed. Then, click **Install**.



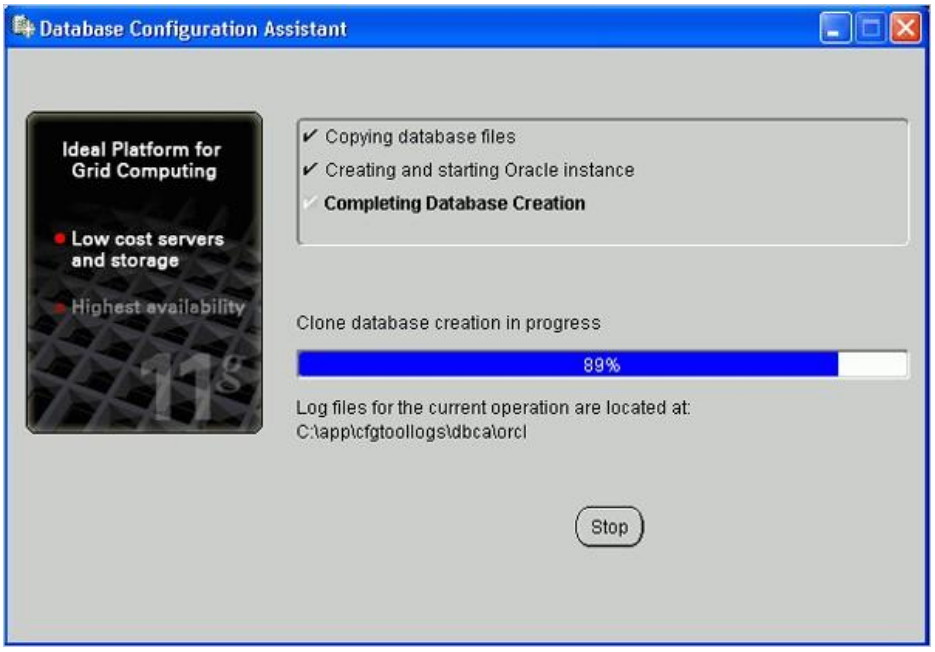
6. The progress window appears.



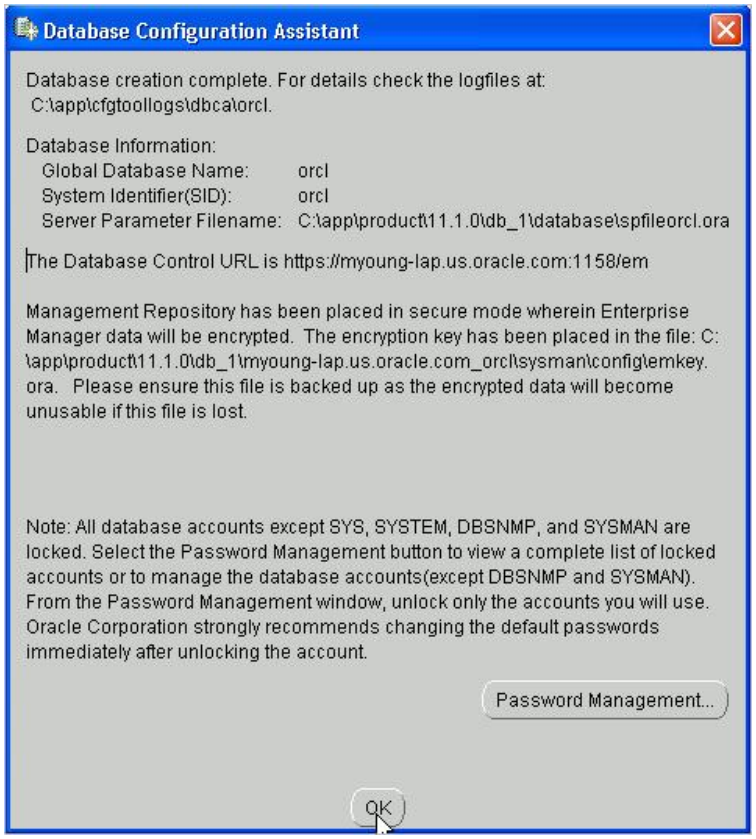
7. The Configuration Assistants window appears.



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

Program Name: Creating Entity-Relationship Diagram using case tools.

Steps:

Step 1: Install MySQL Workbench
If you don't already have MySQL Workbench installed, you can download it from the official MySQL website: https://www.mysql.com/products/workbench/

Step 2: Launch MySQL Workbench
After installation, launch MySQL Workbench on your computer.

Step 3: Create a New EER Diagram

Click on "File" in the menu bar.
Select "New Model" to create a new Entity-Relationship Diagram (ERD).
Step 4: Add Entities and Attributes

In the diagram canvas, you can add entities by clicking on the "Entity" button in the toolbar and then clicking on the canvas to place the entity.
Double-click on the entity to give it a name.
To add attributes to an entity, right-click on the entity and select "Add Attribute."

Step 5: Define Relationships

To define relationships between entities, select the "Relationship" tool from the toolbar.
Click on one entity and then click on the related entity to establish a relationship.
Specify the cardinality and other properties of the relationship.
Step 6: Save Your ERD

It's important to save your work. Click on "File" and then "Save" to save the model.

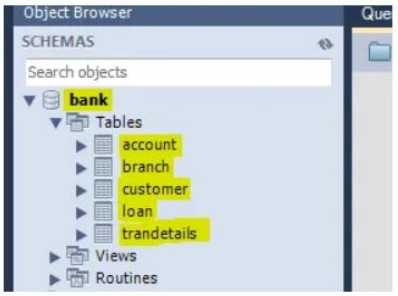
Step 7: Generate SQL Script (Optional)

MySQL Workbench allows you to generate SQL scripts from your ERD. You can do this by clicking on "Database" and then "Forward Engineer..." to create a database schema based on your ERD.
Step 8: Review and Export (Optional)

You can review your ERD, make any necessary changes, and then export it in different formats, such as PNG or PDF.

Output Examples:

1. First make sure you have a **Database** and **Tables** created on the MySQL server.

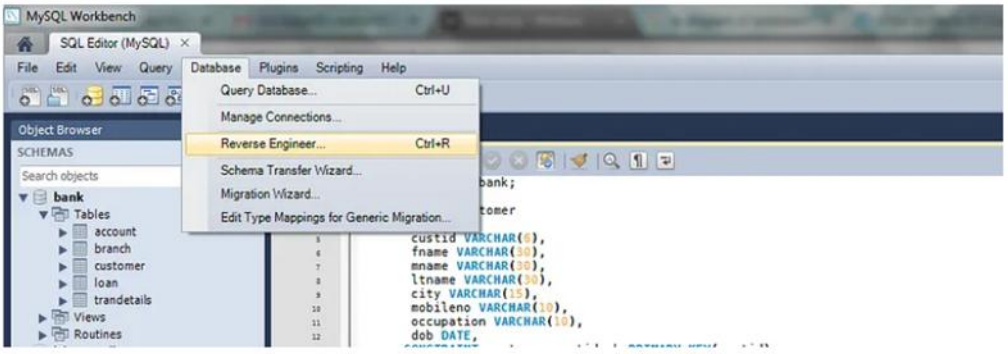


Example :-

Database - *bank.*

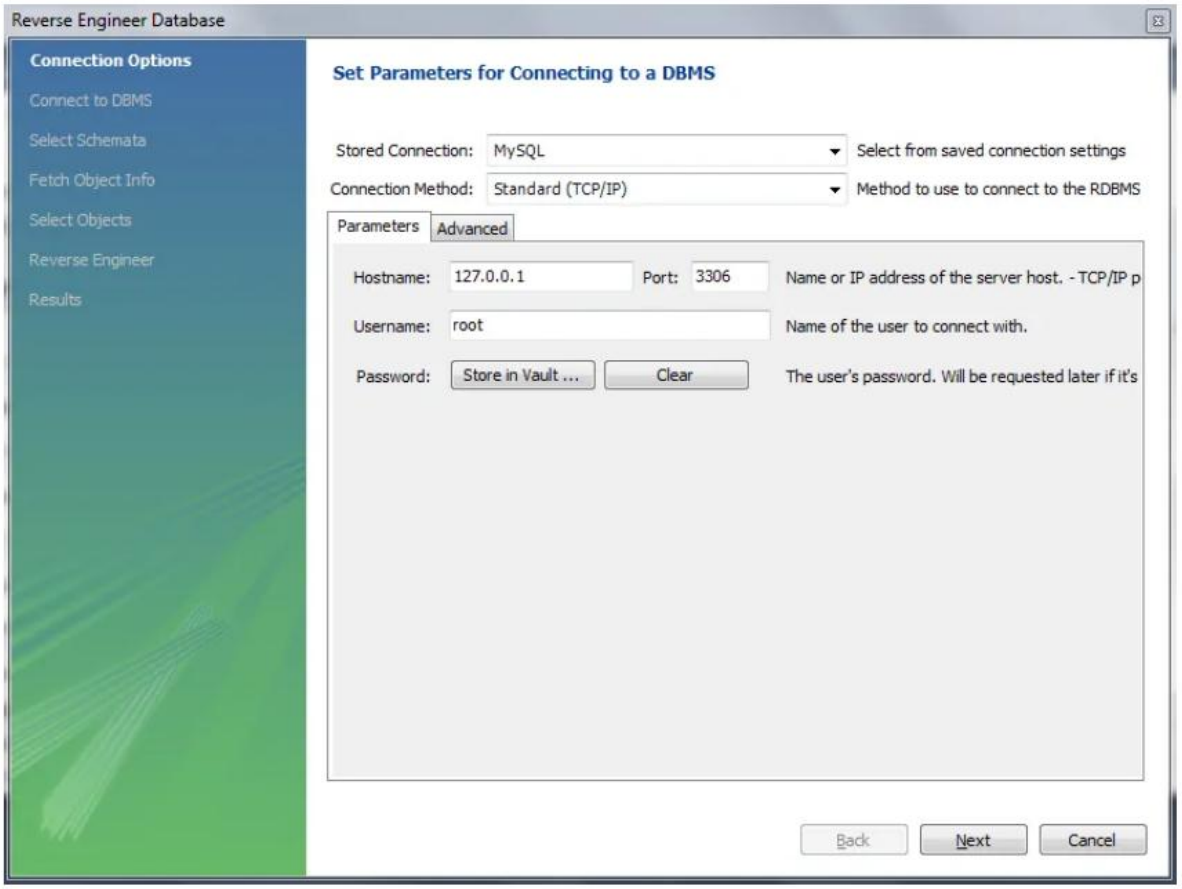
Tables - *account, branch, customer, loan, trandetails.*

2. Click on **Database** -> **Reverse Engineer.**

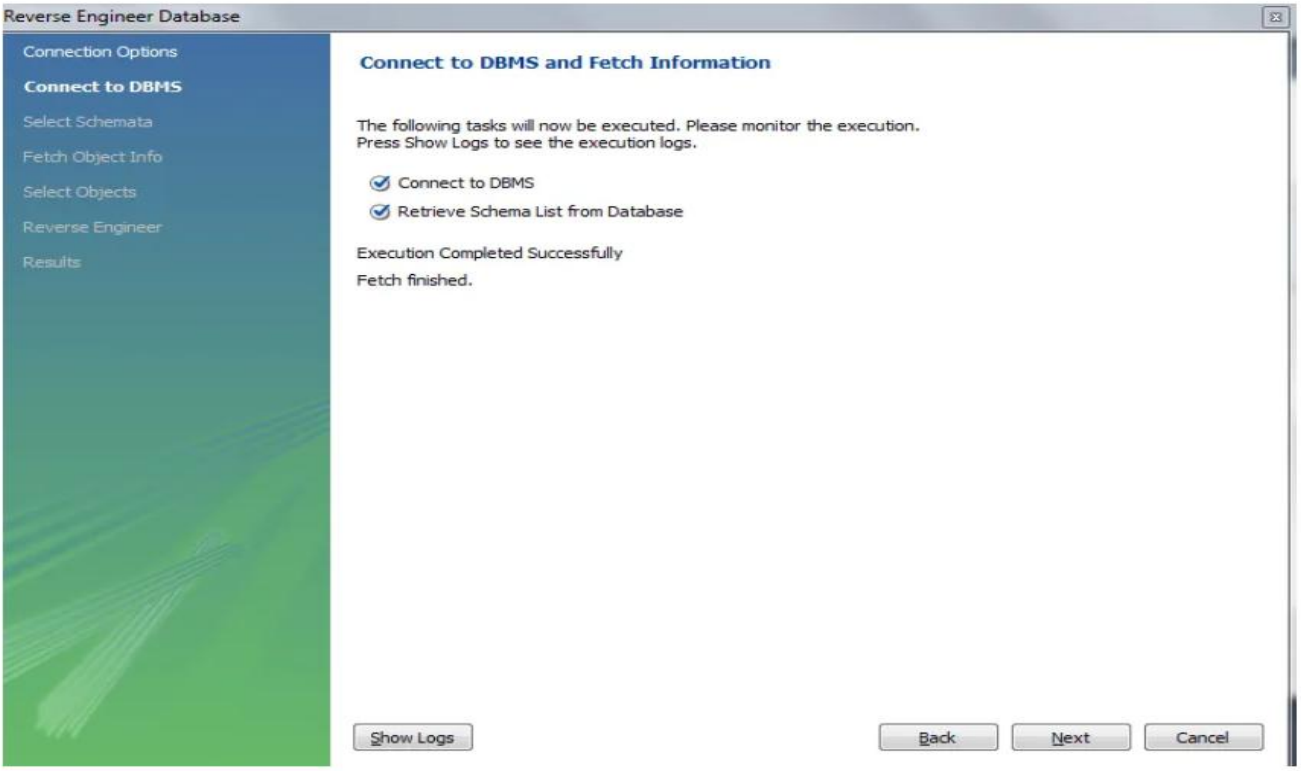


3. Select your **stored connection** *(for connecting to your MySQL Server in which database is present)* from the dropdown. Then click **Next.**

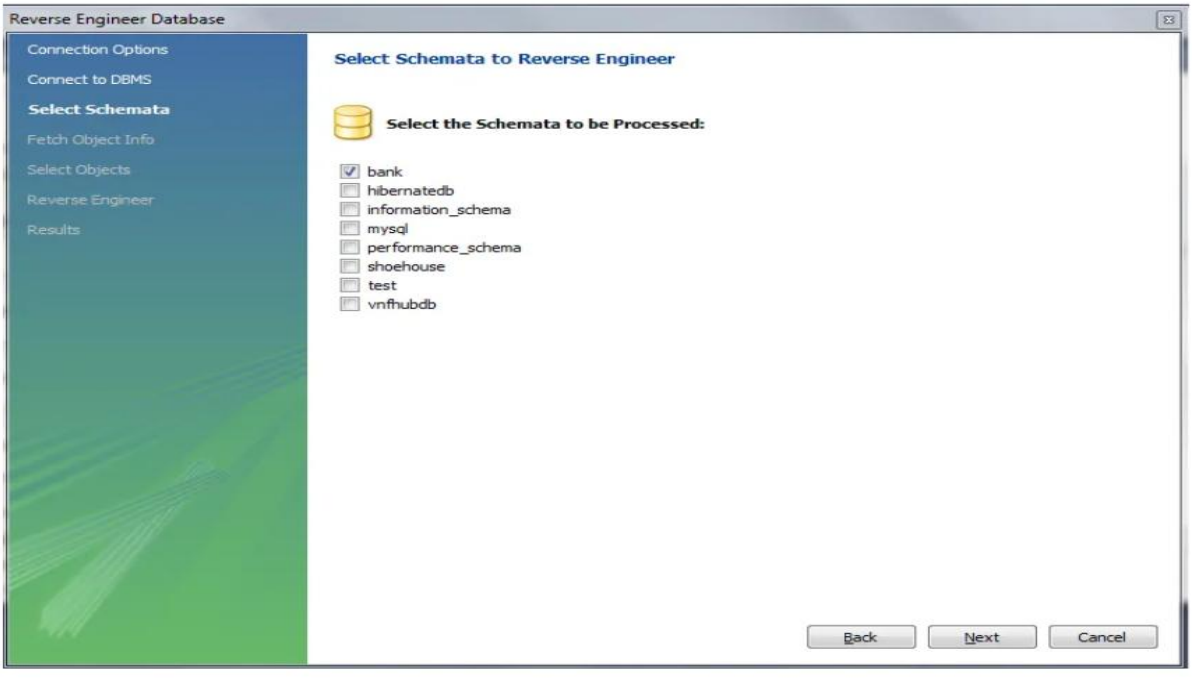
3. Select your **stored connection** *(for connecting to your MySQL Server in which database is present)* from the dropdown. Then click **Next.**



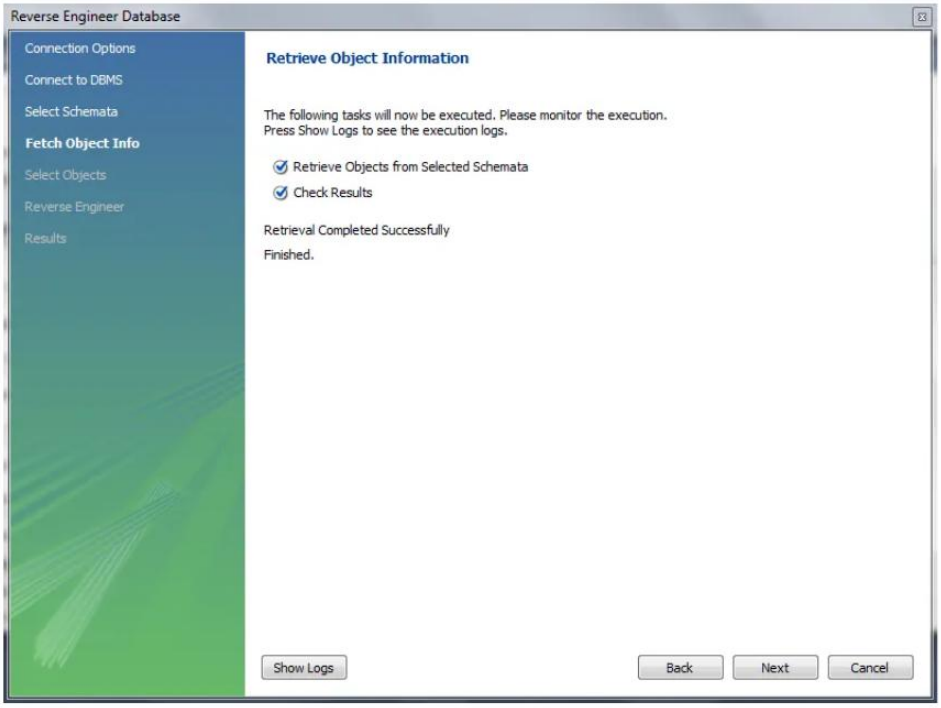
4. After the execution gets completed successfully *(connection to DBMS)*, click **Next.**



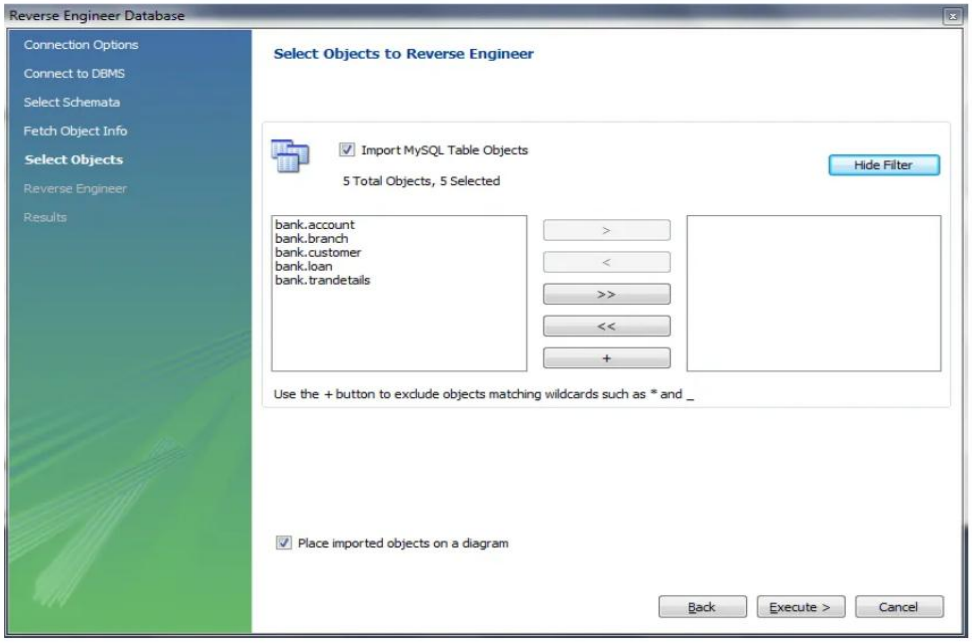
5. Select your Database from the MySQL Server for which you want to create the ER Diagram (*in our case the database name is "**bank**"),* then click **Next.**



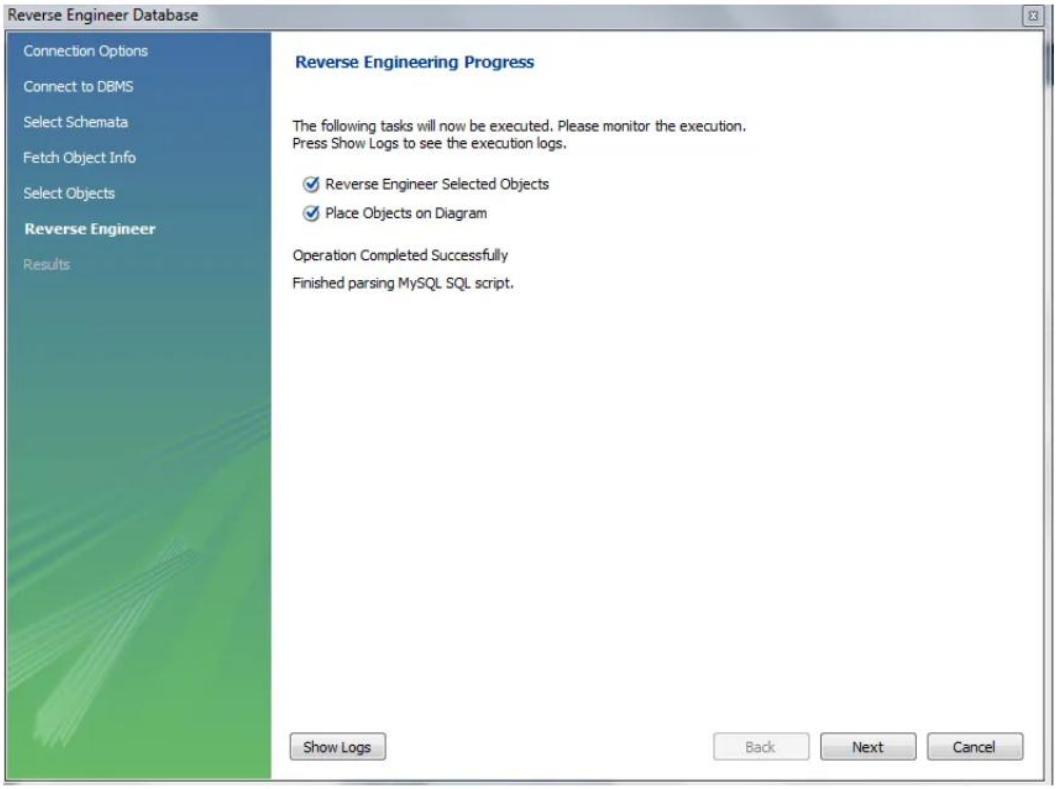
6. After the retrieval gets **completed** successfully for the selected Database, click **Next.**



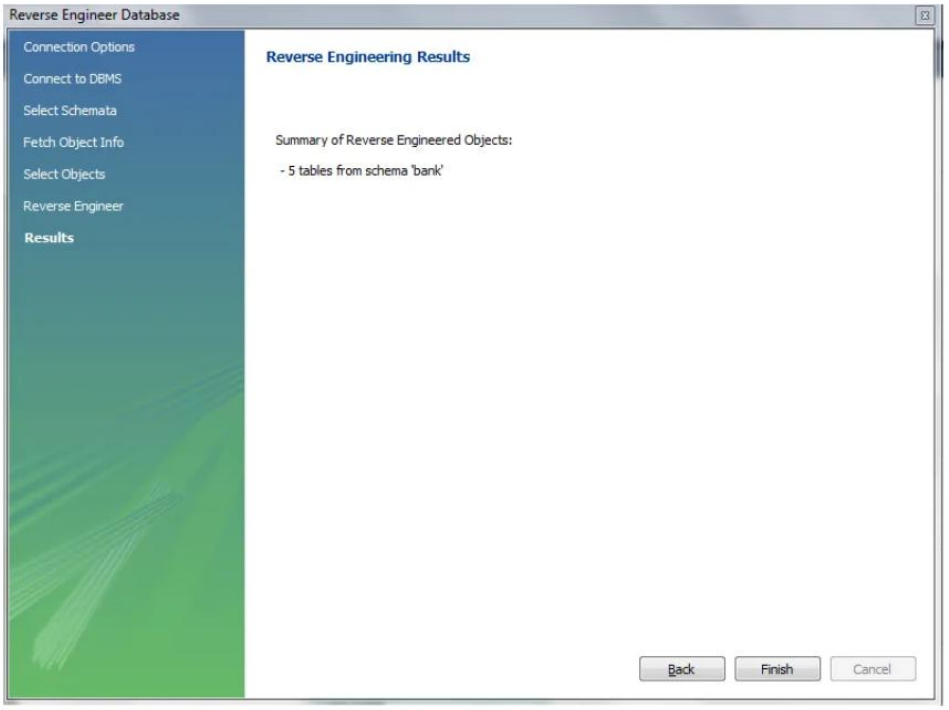
7. Select the Tables of the Database which you want to be visible on the ER Diagram *(In this case I am importing all the tables of the DB),* then click **Execute>.**

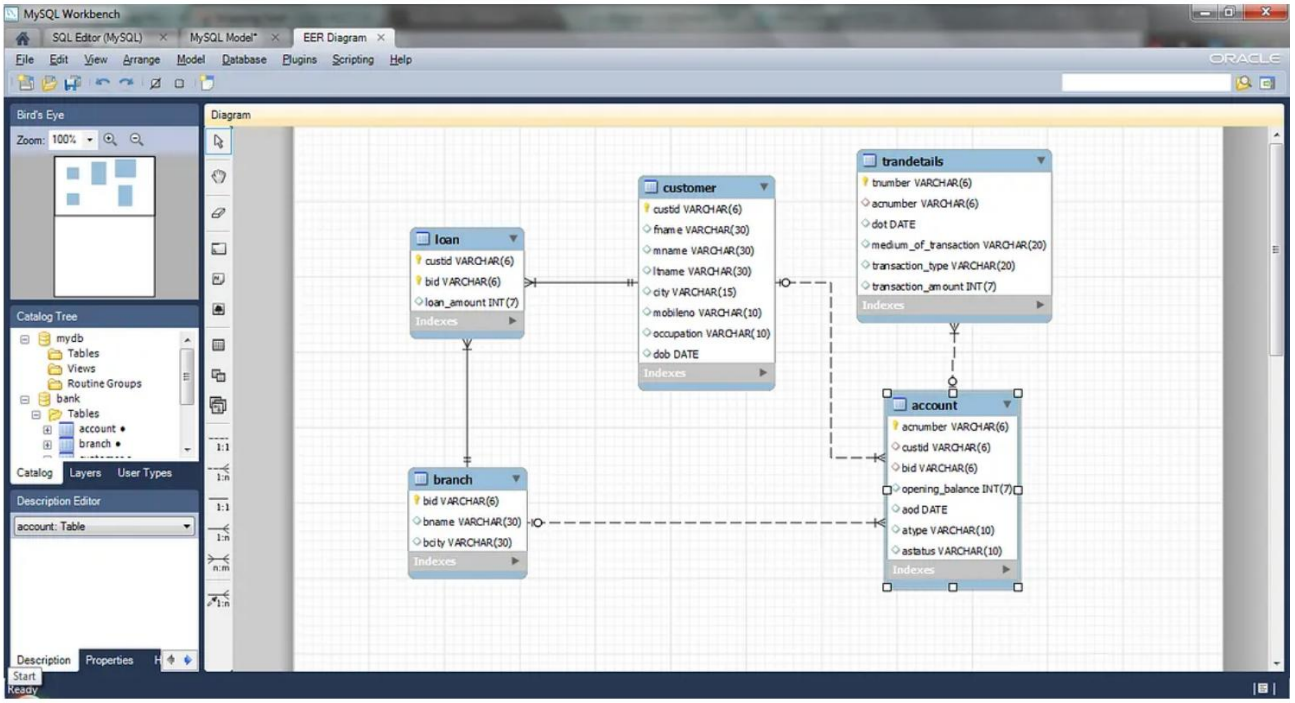


8. After the Reverse Engineering Process gets completed successfully, click **Next.**



9. Click **Finish.**




MySQL Workbench
SQL Editor (MySQL)
MySQL Model*
EER Diagram
File Edit View Arrange Model Database Plugins Scripting Help
ORACLE
Bird's Eye
Zoom: 100%
Diagram
Catalog Tree
mydb
Tables
Views
Routine Groups
bank
Tables
account
branch
Catalog Layers User Types
Description Editor
account: Table
Description Properties
Start
Ready
loan
custid VARCHAR(6)
bid VARCHAR(6)
loan_amount INT(7)
Indexes
customer
custid VARCHAR(6)
fname VARCHAR(30)
mname VARCHAR(30)
lname VARCHAR(30)
city VARCHAR(15)
mobileno VARCHAR(10)
occupation VARCHAR(10)
dob DATE
Indexes
trandetails
tnumber VARCHAR(6)
acnumber VARCHAR(6)
dot DATE
medium_of_transaction VARCHAR(20)
transaction_type VARCHAR(20)
transaction_amount INT(7)
Indexes
account
acnumber VARCHAR(6)
custid VARCHAR(6)
bid VARCHAR(6)
opening_balance INT(7)
aod DATE
atype VARCHAR(10)
astatus VARCHAR(10)
Indexes
branch
bid VARCHAR(6)
bname VARCHAR(30)
bcity VARCHAR(30)
Indexes

Experiment No: - 3

Program Name: Writing SQL statements Using ORACLE /MYSQL:
a) Writing basic SQL SELECT statements.
b) Restricting and sorting data.
c) Displaying data from multiple tables.
d) Aggregating data using group function.
e) Manipulating data.
f) Creating and managing tables.

SQL statements using MYSQL:

a) Writing basic SQL SELECT statements.

```
-- Select all columns from a table
SELECT * FROM employees;

-- Select specific columns from a table
SELECT first_name, last_name FROM employees;

-- Select distinct values from a column
SELECT DISTINCT department_id FROM employees;

-- Select data with a filter (WHERE clause)
SELECT * FROM employees WHERE salary > 50000;

-- Select data with a combination of conditions
SELECT * FROM employees WHERE department_id = 2 AND salary > 50000;
```

b) Restricting and sorting data.

```
-- Sorting data in ascending order
SELECT * FROM employees ORDER BY last_name;

-- Sorting data in descending order
SELECT * FROM employees ORDER BY hire_date DESC;

-- Limiting the number of rows returned
SELECT * FROM employees LIMIT 10;

-- Limiting the number of rows with an offset
SELECT * FROM employees LIMIT 10 OFFSET 20;
```

c) Displaying data from multiple tables (JOIN).

```
-- Inner Join
SELECT orders.order_id, customers.customer_name
FROM orders
```

```
INNER JOIN customers ON orders.customer_id = customers.customer_id;

-- Left Join
SELECT employees.first_name, departments.department_name
FROM employees
    LEFT JOIN departments ON employees.department_id = departments.department_id;
```

d) **Aggregating data using group function.**

```
-- Calculate the total salary for each department
SELECT department_id, SUM(salary) AS total_salary
FROM employees
GROUP BY department_id;

-- Calculate the average salary
SELECT AVG(salary) AS average_salary
FROM employees;
```

e) Manipulating data (INSERT, UPDATE, DELETE):

```
-- Inserting a new record
INSERT INTO employees (first_name, last_name, salary)
VALUES ('John', 'Doe', 60000);


-- Updating an existing record
UPDATE employees
SET salary = 65000
WHERE employee_id = 101;

-- Deleting a record
DELETE FROM employees
WHERE employee_id = 102;
```

e) Creating and managing tables:

```
-- Creating a new table
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(255),
    price DECIMAL(10, 2)
);

-- Modifying a table (adding a new column)
ALTER TABLE employees
ADD COLUMN email VARCHAR(255);
-- Dropping a table
DROP TABLE products;
```

Experiment No: - 4

1. **Program Name**: Creating procedure and functions.

Theory Concept:

Normalization is a database design process used to organize data in a relational database efficiently and reduce data redundancy. It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables. Normalization typically involves dividing a database into two or more tables and defining relationships between them. Let's go through an example of normalizing a database with sample data and MySQL queries. We'll start with an unnormalized table and normalize it step by step.

Step 1: Create an Unnormalized Table
Suppose we have a table called "CustomerOrders" that stores information about customers and their orders. This table is not normalized because it contains repeating groups and data redundancy:

```
CREATE TABLE CustomerOrders (
  customer_id INT PRIMARY KEY,
  customer_name VARCHAR(255),
  order_id INT,
  order_date DATE,
  total_amount DECIMAL(10, 2)
);

INSERT INTO CustomerOrders (customer_id, customer_name, order_id, order_date, total_amount)
VALUES
  (1, 'Alice', 101, '2023-01-15', 100.00),
  (1, 'Alice', 102, '2023-02-20', 150.00),
  (2, 'Bob', 201, '2023-03-10', 75.50),
  (3, 'Charlie', 301, '2023-04-05', 200.00);
```

Step 2: Normalize the Data

We'll normalize the data by creating two separate tables: "Customers" and "Orders." The "Customers" table will store customer information, and the "Orders" table will store order information.

```
-- Create the Customers table
CREATE TABLE Customers (
  customer_id INT PRIMARY KEY,
  customer_name VARCHAR(255)
);

-- Create the Orders table
CREATE TABLE Orders (
  order_id INT PRIMARY KEY,
  customer_id INT,
  order_date DATE,
```

```
    total_amount DECIMAL(10, 2),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);

-- Populate the Customers table with unique customer information
INSERT INTO Customers (customer_id, customer_name)
SELECT DISTINCT customer_id, customer_name FROM CustomerOrders;

-- Populate the Orders table with order information
INSERT INTO Orders (order_id, customer_id, order_date, total_amount)
SELECT order_id, customer_id, order_date, total_amount FROM CustomerOrders;
```

Step 3: Query the Normalized Tables

Now that we have normalized our data, we can query the "Customers" and "Orders" tables to retrieve information:

```
-- Query to retrieve customer information
SELECT * FROM Customers;

-- Query to retrieve order information
SELECT * FROM Orders;

-- Query to retrieve customer names and their total order amounts
SELECT c.customer_name, SUM(o.total_amount) AS total_order_amount
FROM Customers c
JOIN Orders o ON c.customer_id = o.customer_id
GROUP BY c.customer_name;
```

Output:

These queries demonstrate the result of normalizing the data. The "Customers" table contains unique customer information, and the "Orders" table stores order details with a reference to the customer. The last query retrieves the total order amount for each customer, demonstrating the power of relational databases and normalization.

Experiment No-5

Program Name: Design and implementation of Student Information System.

Theory Concept:

Designing and implementing a Student Information System (SIS) experiment in a Database Management System (DBMS) is a practical way to learn about database design and development. Below, I'll outline a simplified experiment scenario for creating a basic SIS using a relational DBMS (e.g., MySQL, PostgreSQL). This experiment assumes you have basic knowledge of SQL and database concepts.

Experiment Scenario:

You are tasked with creating a Student Information System (SIS) for a university. The system should store information about students, courses, and grades. Students can enroll in courses, and teachers can enter grades for students in those courses.

Experiment Steps:

1. Database Design:

Define the database schema with tables for students, courses, and grades. Here's a simplified schema:

```
-- Students table
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    birthdate DATE,
    email VARCHAR(100)
);

-- Courses table
CREATE TABLE courses (
    course_id INT PRIMARY KEY,
    course_name VARCHAR(100),
    teacher VARCHAR(100)
);

-- Grades table
CREATE TABLE grades (
    grade_id INT PRIMARY KEY,
    student_id INT,
    course_id INT,
    grade VARCHAR(2),
    FOREIGN KEY (student_id) REFERENCES students(student_id),
    FOREIGN KEY (course_id) REFERENCES courses(course_id)
     );
```

2. Data Population:

Insert sample data into the tables for testing purposes.

```
-- Insert sample students
INSERT INTO students (student_id, first_name, last_name, birthdate, email)
VALUES
   (1, 'John', 'Doe', '1995-01-15', 'john@example.com'),
   (2, 'Jane', 'Smith', '1996-03-22', 'jane@example.com');

-- Insert sample courses
INSERT INTO courses (course_id, course_name, teacher)
VALUES
   (101, 'Mathematics 101', 'Dr. Smith'),
   (102, 'Computer Science 101', 'Prof. Johnson');

-- Enroll students in courses
INSERT INTO grades (student_id, course_id, grade)
VALUES
   (1, 101, 'A'),
   (1, 102, 'B'),
   (2, 101, 'B');
```

3. Querying the Database:

Practice querying the database to retrieve information. For example, you can retrieve a student's grades or find courses taught by a specific teacher.

```
-- Get a student's grades
SELECT s.first_name, s.last_name, c.course_name, g.grade
FROM students s
JOIN grades g ON s.student_id = g.student_id
JOIN courses c ON g.course_id = c.course_id
WHERE s.student_id = 1;

-- Find courses taught by a specific teacher
SELECT course_name
FROM courses
WHERE teacher = 'Dr. Smith';
```

4. CRUD Operations:

Practice performing CRUD (Create, Read, Update, Delete) operations on the database. For example, you can add a new student, update a student's information, or delete a course.

```
-- Create: Add a new student
INSERT INTO students (student_id, first_name, last_name, birthdate, email)
VALUES (3, 'Alice', 'Johnson', '1997-05-10', 'alice@example.com');
```

```
-- Update: Change a student's email
UPDATE students
SET email = 'new_email@example.com'
WHERE student_id = 3;

-- Delete: Remove a course
DELETE FROM courses
WHERE course_id = 102;
```

Experiment No: 6

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

TheoryConcept: The following example would illustrate the concept of CURSORS. We will be using the CLIENT_MASTER table and display records.

Implementation:

```
DECLARE
 CURSOR client_cur
isSELECT id,name,address
FROM client_master;
   client_rec
client_cur%rowtype;BEGIN
OPENclient_cur;
LOOP
FETCH client_cur into
client_rec;EXITWHENclient_cur
%notfound;
   DBMS_OUTPUT.put_line(client_rec.id||''||client_rec.name);
END LOOP;
 END;
/
```

Output: Whenthe above codeis executed atSQLprompt,itproduces thefollowing result:

```
1 Ramesh
2 Khilan
3 kaushik
4 Chaitali
5 Hardik
6 Komal

PL/SQLprocedure successfully completed.
```

Experiment No -7

Program Name: Execute the queries related to Group By and having Clause on tables SALES_ORDER.

TheoryConcept:

The program aims to familiarize the user with grouping of databased on conditions to ensure better usability of data.

Implementation:

GROUPBY

Q1) Create table sales_order with attributes product_no and Qty. Insert records into the table and find the total qty ordered foreach product_no.

Ans: Create table sales_order (product_novarchar(10), Qty numbe(4));

Output:Tablecreated.

insert into sales_order values(&product_no, &qty);

select* from sales_order;

Output:

```
PRODUCT_NO      QTY
- - - - - - - - - - -
      p        12
    1
      p         11
    2            2
      p         9
    1
      p        23
    2
      p        23
    3
      p        23
    3
```

6 rows selected.

selectproduct_no, sum(qty) from sales_order group by product_no;

Output:

```
PRODUCT_NOSUM(QTY)
- - - - - - - - - - - - -
p1             21
p2            135
p3             46
```

3 rows selected.

HAVINGclause

Q2) Find the total Qty for product_no‘p1’and ‘p2’fromthe

Table sales_order Ans:select product_no, sum(qty) from sales_order group by

product_no having product_no = 'p1' OR product_no = 'p2';

Output:

```
PRODUCT_NOSUM(QTY)
--------------
p1              21
p3              46

2 rows selected
```

Experiment No -8

Program Name: Execute the following queries:

a) The NOT NULL
b) The UNIQUE Constraint
c) The PRIMARY KEY Constraint
d) The CHECK Constraint
e) Define Integrity Constraints in ALTER table Command

a) The NOT NULL Constraint:

The NOT NULL constraint ensures that a column cannot contain NULL (empty) values.
Here's an example:

```
-- Create a table with a NOT NULL constraint
CREATE TABLE employees (
    employee_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    hire_date DATE NOT NULL
);
```

b)The UNIQUE Constraint:

The UNIQUE constraint ensures that the values in a column are unique across all rows in a table. Here's an example:

```
-- Create a table with a UNIQUE constraint
CREATE TABLE products (
    product_id INT PRIMARY KEY,
    product_name VARCHAR(100) UNIQUE,
    price DECIMAL(10, 2)
);

-- Insert rows with unique product names
INSERT INTO products (product_id, product_name, price)
VALUES (1, 'Laptop', 1000.00),
       (2, 'Smartphone', 600.00);

-- Attempt to insert a row with a duplicate product name, which will result in an error
INSERT INTO products (product_id, product_name, price)
VALUES (3, 'Laptop', 1200.00);
```

c) The PRIMARY KEY Constraint:

The PRIMARY KEY constraint defines a unique identifier for each row in a table. Here's an example:

```
-- Create a table with a PRIMARY KEY constraint
CREATE TABLE students (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    birth_date DATE
```

```
);

-- Insert rows with unique student IDs
INSERT INTO students (student_id, first_name, last_name, birth_date)
VALUES (1, 'John', 'Doe', '1995-01-15'),
       (2, 'Jane', 'Smith', '1996-03-22');
```

d) The CHECK Constraint:

The CHECK constraint allows you to specify a condition that must be met for data to be valid. Here's an Example:

```
-- Create a table with a CHECK constraint
CREATE TABLE orders (
    order_id INT PRIMARY KEY,
    order_date DATE,
    total_amount DECIMAL(10, 2),
    payment_status VARCHAR(20) CHECK (payment_status IN ('Paid', 'Unpaid', 'Pending'))
);

-- Insert rows with valid payment statuses
INSERT INTO orders (order_id, order_date, total_amount, payment_status)
VALUES (1, '2022-01-01', 500.00, 'Paid'),
       (2, '2022-02-01', 750.00, 'Unpaid');

-- Attempt to insert a row with an invalid payment status, which will result in an error
INSERT INTO orders (order_id, order_date, total_amount, payment_status)
VALUES (3, '2022-03-01', 300.00, 'InvalidStatus');
```

e) Define Integrity Constraints in ALTER TABLE Command:

You can also define integrity constraints using the ALTER TABLE command. Here's an example of adding a NOT NULL constraint to an existing table:

```
-- Add a NOT NULL constraint to an existing column
ALTER TABLE employees
ALTER COLUMN hire_date DATE NOT NULL;
```

Experiment No: 9

Program Name: Execute Nested Queries on tables CLIENT_MASTER, PRODUCT_MASTER, SALESMAN_MASTER,SALES_ORDER,SALES_ORDER_DETAILS

TheoryConcept:
Theprogramintendstofamiliarizenestedqueriessoastoretrievedatafromarecordbyusingfiltereddata from another record.

Implementation:

Q1) Retrieve the order numbers, client names and their order dates from client_master and sales_ordertables.

Ans: Selectorder_no,order_date,namefromsales_order,client_masterwhereclient_master.client_no=sales _order.client_noorder by order_date;

OUTPUT:

Orde r_no	order_ date	name
1	1999/1 2/05	akans ha
2	1999/1 2/12	divya

Q2) Retrieve the product numbers, description and total quantity ordered for each product**Ans**:Selectsales_order_details.product_no, description, sum(qty_ordered) from sales_order_details,product_master where product_master.product_no = sales_order_details.product_no group bysales_order_details.product_no, description;

OUTPUT:

produc t_no	descript ion	sum(qty_ord ered)
1	chair	2
2	pen	5

Q3) Retrieve the names of employees and names of their respective managers from the employee table.**Ans:** Select employee.name, employee.name from employee where employee.manager_no =employee.employee_no;

OUTPUT:

Name	Name
Akansha	Divya
Akshita	Divya

UNION , INTERSECTand MINUS CLAUSE

Q1) Retrieve the names of all clients and salesmen in the city of Mumbai from the tablesclient_masterandsalesman_master.

Ans: Selectsalesman_no from salesman_master where city = 'Mumbai'UNION

Select client_no from client_master where city = 'Mumbai';

OUTPUT:

Name

Akansha

Akshita

Divya

Q2) RetrievethesalesmannameinMumbaiwhoseeffortshaveresultedintoatleastonesalestransaction

Ans: Selectsalesman_no, namefrom salesman_master where city = 'Mumbai'INTERSECT

Selectsalesman_master.salesman_no,namefromsalesman_master,sales_orderwheresalesman_master.salesman_no=sales_order.salesman_no;

OUTPUT:

Saleman_no	Name
1	akansha
2	divya

Q3) Retrieve all the product numbers of non-moving items from the product_master table

Ans: Selectproduct_no from product_masterMinus

Select product_no from sales_order_details;

OUTPUT:

product_no

3

4

VIEWS

Q1) Create a view on salesman_master table for the sales department**Ans:**Create view vw_sales as select * from salesman_master;**OUTPUT:**

Viewcreated

Q2) Create a view on client_master table

Ans: Createviewvw_clientasselectname,address1,address2,city,pincode,state,bal_duefromclient_master;

OUTPUT:
Viewcreated

Q3) Perform insert, modify and delete operations on the view created in Q2

Ans:

a) Insertintovw_clientvalues('C001','Robert','AAAAAA','BBB','Delhi',2000000,'MMM');

OUTPUT:
1rows created

b)Updatevw_client set bal_due = 10000 where client_no = 'C001';

OUTPUT:
1 row updated

c)Delete from vw_client where client_no = 'C001';

OUTPUT:
1 row deleted

Experiment No-10

ProgramName: Execute queries related to Exists, Not Exists, Union, Intersection, Difference, Join ontablesCLIENT_MASTER,PRODUCT_MASTER,SALESMAN_MASTER,SALES_ORDER,SALES_ORDER_DETAILS

TheoryConcept:
Theprogramretrievesdatafromrecordsbydefiningrelationbetweentwotablessoastoretrievefilteredrecords.

Implementation:

CorrelatedqueriesswithEXISTS/NOTEXISTSclause

1) Select all products and order_no where order_status is 'in Process'
Ans: Select order_no.,product_no. from sales_order_details where exists(select * from sales_order ,order_no = sales_order_details,order_no and order_status='in process');
Output:

Order_no	Product_no
0003	3

2) Selectorder_no and order_date for all orders which include product_no 'P001' and quantity_ordered>10**Ans:**Select order_no,order_data from sales_order where exists(select * from sales_order_details wheresales_order_details,order_no = sales_order.Order_no and product-no='p001' and quantity-ordered>10;**Output:**

Order_no	Product_no
0002	05/feb/13

3) Find all order_no for salesman rashmi.
Ans: Select order_no from sales_order where exists(select * from salesman_master wheresalesman_master.saleman-no=sales_order-salesman_noandname='rashmi');
Output:

Order_no
0003

4) Select all clients who have not placed any orders.
Ans: Select * from client_master where not exists(select * fromsales_order.client_no=client_master.client_no);

Output:

Client_no	Name	City	Pincode	State
6	Divya	Hapur	35498	U.P.
7	Dorothy	Noida	32547	U.P.

5)Select all orders with order_date for ‘acrylic colors’

Ans: Select order_no,order_date from sales_order where exists(select * fromsales_order_details.oder_no=sales_order.order_noANDexists(select*fromproduct1wheresales_order_details.product_no=product_noAND description=’acrylic colors’);

Output:

Order_no	Order_date
0001	23/jan/13

Union,Intersect and minus clause:

1) List all the clients and salesman and their names

Ans: Select client_no, name from client_master UNION select salesman_no,name from salesman_master;

Output:

Client_no	Name
3	Akshita
4	Dhawal

2) List all the clients and their names who are also salesman.

Ans: Selectnamefromclient_masterINTERSECT,selectnamefromsalesman_master;

Output:

No rows selected

3) List all the clients who are not salesman.

Ans: Select name from client_master MINUS select name from salesman_master;

Output:

Name
Akshita
Dhawal
Akansha
Divya
Dorothy

4) List all the clients who have placed orders

Ans: Select client_no from client_masterINTERSECTselectclient_no from sales_order;

Output:

Client_no
6

7

5) Listalltheclientswhohavenotplacedanyorder.

Ans: Select client_no from client_masterMINUS select client_no from sales_order;

Output:

Client_no
3
4
5

6) List all the clients in UPwho have placed orders

Ans: Selectclient_nofromclient_masterwherestate='UP'INTERSECTselectclient_nofromsales_order;

Output:

Client_no
3
4
5

7) FindalltheclientsandtheirnamesfromcityGhaziabadwhohavedeliverydateoftheirordersastoday.**Ans:**Select client_no from client_master where city='Ghaziazbad' INTERSECT select client_no fromsales_order where delivery_date='09-MAR-13'

Output:

Client_no
5

Queries on Joins

1) List the product_no and description of products sold.

Ans: Select product_no, description from (product1 natural join sales_order_details)

Output:

Product_no	Description
1	Chair
1	Chair
2	Table
3	Sofa

2) Find the products which have been sold to 'akansha'

Ans: Select product_no, description from (product1 natural join sales_order details natural joinsales_order natural join client_master) where name='akansha';

Output:

Product_no	Description
3	Sofa

3) Find the products and their quantities that will have to be delivered in the current month.

Ans: Select sales_order_detailsproduct_no, product1 ,description, sum(sales_order_details,quantity_ordered) from sales_order_details, sales_order, product1 where product1,product_no=sales_order_details,product_noandsales_order,order_no=sales_order_details,order_noandto_char (delivery_date,'mon-yy') = to_char(sysdate,'mon-yy')group by sales_order_details, product_no,product1, description ;

Output: no rows selected

4) Find thenamesofclientwhohavepurchased 'chair'

Ans: Select name from(client_master natural join sales_order natural join sales_order_details natural joinproduct1) where description= 'chair';

Output:

Name
Akshita
Akansha

5)

6) List theorders forlessthan 5unitsof saleof'chair'

Ans: Select product_no, order_no from (sales_order_details natural join product1) where(description='chair'and qty_ordered<5);

Output:

Product_no	Order_no
1	0001
1	0001

7) Find the products and their quantities placed by 'akansha'or 'akshita'.

Ans: Selectproduct_no,description,qty_orderedfrom(product1naturaljoinsales_order_detailsnaturaljoin sales_order_natural join client_master) where (name='akansha'or name='akshita');

Output :

Product_no	Description	Qty_ordered
1	Chair	4
1	Chair	3
2	Sofa	2

8) Find the products and their quantities for the orders placed by the client_no '3'and '5'

Ans: Selectproduct_no,description,qty_orderedfrom(product1naturaljoinsales_order_detailsnaturaljoin sales_order natural join client_master) where (client_no=3 OR client_no=5);

Output:

PRODUCT_NO	DESCRIPTION	QTY_ORDERED
1	Chair	4
1	Chair	3