Vidyavardhini's
College of Engineering & Technology

Vasai Road (W)

Department of Computer Engineering

Laboratory Manual
[Student Copy]

Semester	IV	Class	SE
Course No.	CSL402	Academic Year	2022-23 (Even Sem.)
Course Name	Database Management System Lab		

Vidyavardhini's College of Engineering & Technology

Vision

To be a premier institution of technical education; always aiming at becoming a valuable resource for industry and society.

Mission

- To provide technologically inspiring environment for learning.
- To promote creativity, innovation and professional activities.
- To inculcate ethical and moral values.

- To cater personal, professional and societal needs through quality education.

Department Vision:

To evolve as a center of excellence in the field of Computer Engineering to cater to industrial and societal needs.

Department Mission:

- To provide quality technical education with the aid of modern resources.
- Inculcate creative thinking through innovative ideas and project development.
- To encourage life-long learning, leadership skills, entrepreneurship skills with ethical & moral values.

Program Education Objectives (PEOs):

PEO1: To facilitate learners with a sound foundation in the mathematical, scientific and engineering fundamentals to accomplish professional excellence and succeed in higher studies in Computer Engineering domain
PEO2: To enable learners to use modern tools effectively to solve real-life problems in the field of Computer Engineering.
PEO3: To equip learners with extensive education necessary to understand the impact of computer technology in a global and social context.
PEO4: To inculcate professional and ethical attitude, leadership qualities, commitment to societal responsibilities and prepare the learners for life-long learning to build up a successful career in Computer Engineering.

Program Specific Outcomes (PSOs):

PSO1: Analyze problems and design applications of database, networking, security, web technology, cloud computing, machine learning using mathematical skills, and computational tools.

PSO2: Develop computer-based systems to provide solutions for organizational, societal problems by working in multidisciplinary teams and pursue a career in the IT industry.

Program Outcomes (POs):

Engineering Graduates will be able to:

- **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 the engineering practice.

- **PO9. 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 these to 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.

Course Objectives

1	To Develop Entity Relationship data model.
2	To develop relational Model
3	To formulate SQL queries.
4	To learn procedural interfaces to SQL queries
5	To learn the concepts of transactions and transaction processing
6	To understand how to Handle concurrent transactions and able to access data through front end (using JDBC ODBC connectivity)

Course Outcomes

At the end of the course student will be able to:		PO/PSO	Bloom Level
CSL402.1	Design ER and EER diagram for the real life problem with software tool.	Design	Create (Level 6)

CSL402.2	Construct database tables with different DDL and DML statements and apply integrity constraints	Construct, Apply	Apply (Level 3)
CSL402.3	Apply SQL queries ,triggers for given Schema	Apply	Apply (Level 3)
CSL402.4	Apply procedure and functions for given schema	Apply	Apply (Level 3)
CSL402.5	Use transaction and concurrency control techniques to analyze conflicts in multiple transactions.	Use	Apply (Level 3)
CSL402.6	Construct database tables and JDBC/ ODBC connectivity for given application	Construct	Apply(Level 3)

Mapping of Experiments with Course Outcomes

Experiments	Course Outcomes					
	CSL402 .1	CSL402 .2	CSL402 .3	CSL402 .4	CSL402 .5	CSL402 .6
Identify the case study and detail statement of problem. Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model.	3					
Mapping ER/EER to Relational schema model.	3					
Create and populate database using Data Definition Language (DDL) and Apply Integrity Constraints for the specified system		3				
Apply DML commands for the specified system.		3				
Perform Simple queries, string manipulation operations and aggregate functions			3			

Perform Join operations			3			
Perform Nested queries and Complex queries			3			
Implement Procedure and functions				3		
Implementation of Views and Triggers			3			
Demonstrate of Database connectivity (course Project)						3

Enter correlation level 1, 2 or 3 as defined below
1: Slight (Low) 2: Moderate (Medium) 3: Substatial (High)
If there is no correlation put "—".

INDEX

Sr. No.	Name of Experiment	D.O.P.	D.O.C.	Page No.	Remark
1	Identify the case study and detail statement of problem. Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model.				
2	Mapping ER/EER to Relational schema model.				
3	Create and populate database using Data Definition Language (DDL) and Apply Integrity Constraints for the specified system				
4	Apply DML commands for the specified system.				
5	Perform Simple queries, string manipulation operations and aggregate functions				
6	Implement various Join Operations				
7	Perform Nested queries and Complex queries				
8	Implement Procedure and functions				
9	Implementation of Views and Triggers				
10	Demonstrate of Database connectivity (Course Project)				

D.O.P: Date of performance

D.O.C : Date of correction

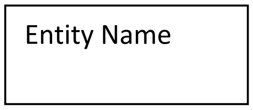
Experiment No.1
Identify the case study and detail statement of problem. Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model.
Date of Performance:
Date of Submission:

Aim: Identify the case study and detail statement of problem.
Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model.

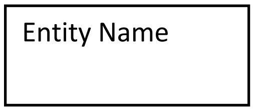
Objective: To show the relationships of entity sets attributes and relationships stored in a database

Theory: Summary of ER, EER Diagram Notation

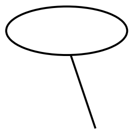
Strong Entities



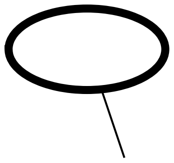
Weak Entities



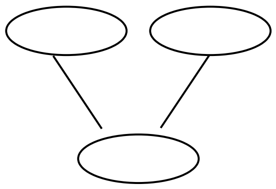
Attributes



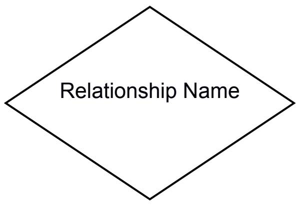
Multi Valued Attributes [Double Ellipse]



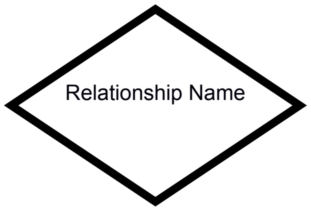
Composite Attributes



Relationships



Identifying Relationships



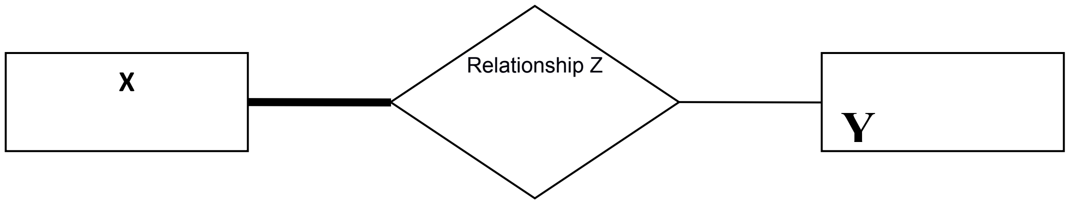
N-ary relationships

.More than 2 participating entities.

Constraints - Participation

.**Total Participation** - entity X has total participation in Relationship Z, meaning that every instance of X takes part in AT LEAST one relationship. (i.e. there are no members of X that do not participate in the relationship.

Example: X is Customer, Y is Product, and Z is a 'Purchases' relationship. The figure below indicates the requirement that every customer purchases a product.



.**Partial Participation** - entity Y has partial participation in Relationship Z, meaning that only some instances of Y take part in the relationship.

Example: X is Customer, Y is Product, and Z is a 'Purchases' relationship. The figure below indicates the requirement that not every product is purchases by a customer. Some products may not be purchased at all.

Constraints - Cardinality

.1:N – One Customer buys many products, each product is purchased by only one customer.

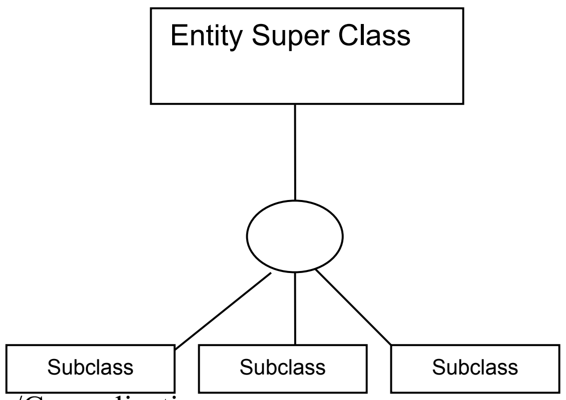
.N:1 - Each customer buys at most one product, each product can be purchased by many customers.

.1:1 – Each customer purchases at most one product, each product is purchased by only one customer.

.M:N – Each customer purchases many products, each product is purchased by many customers.

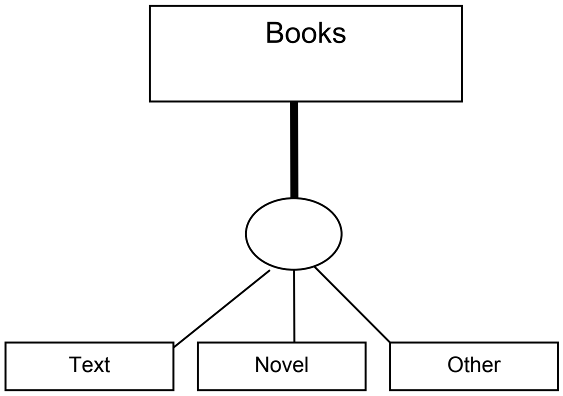
Specialization/Generalization

.Each subclass inherits all relationships and attributes from the super-class.

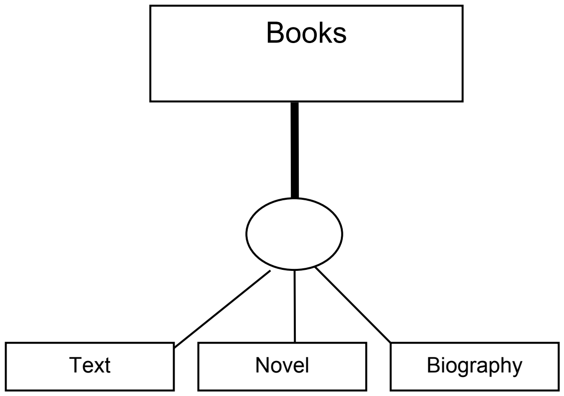


Constraints on Specialization/Generalization

Total Specialization – Every member of the super-class must belong to at least one subclass. For example, any book that is not a text book, or a novel can fit into the "Other" category.

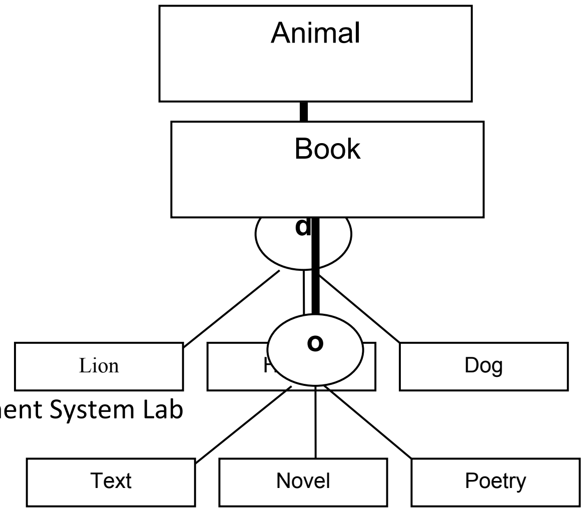


Partial Specialization – each member of the super-class may not belong to one of the subclasses. For example, a book on poetry may be neither a text book, a novel or a biography.



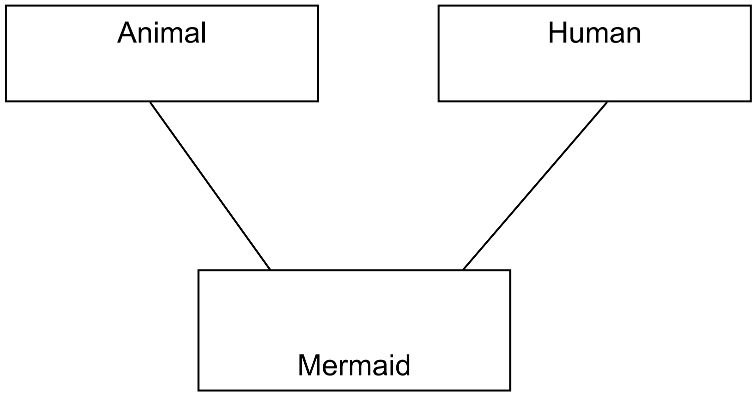
Disjointness Constraint

.**Disjoint** – every member of the super-class can belong to at most one of the subclasses. For example, an Animal cannot be a lion and a horse, it must be either a lion, a horse, or a dog.

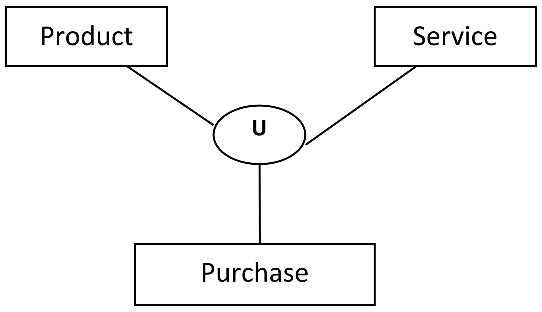


Overlapping – every member of the super-class can belong to more than one of the subclasses. For example, a book can be a text book, but also a poetry book at the same time.

Multiple Inheritance – a subclass participates in more than one subclass/super-class relationship, and inherits attributes and relationships from more than one super-class. For example, the subclass Mermaid participates in two subclass/super-class relationships, it inherits attributes and relationships of Animals, as well as attributes and relationships of Humans.



Union – a subclass/super-class relationship can have more than one super-class, and the subclass inherits from at most one of the super-classes (i.e. the subclass purchase will inherit the relationships and attributes associated with either service or product, but not both). Each super class may have different primary keys, or the same primary key. All members of the super-classes are not members of the super-class. For example, a purchase can be a product, or a service, but not both. And all products and services are not purchase



Implementation:

Conclusion:- Comment on Problem statement and design of given case study by identifying ER/EER diagram notations.

Experiment No.2
Mapping ER/EER to Relational schema model.
Date of Performance:
Date of Submission:

Aim: Mapping ER/EER to Relational schema model.

Objective: objective of design is to generate a formal specification of the database schema

Theory: Mapping Rules

Step 1: Regular Entity Types

Create an *entity relation* for each strong entity type. Include all single-valued attributes. Flatten composite attributes. Keys become secondary keys, except for the one chosen to be the primary key.

Step 2: Weak Entity Types

Also create an entity relation for each weak entity type, similarly including its (flattened) single-valued attributes. In addition, add the primary key of each owner entity type as a foreign key attribute here. Possibly make this foreign key CASCADE.

Step 3: Binary 1:1 Relationship Types

Let the relationship be of the form [S]——<R>——[T].

1. **Foreign key approach**: The primary key of T is added as a foreign key in S. Attributes of R are moved to S (possibly renaming them for clarity). If only one of the entities has total participation it's better to call it S, to avoid null attributes. If neither entity has total participation nulls may be unavoidable. *This is the preferred approach in typical cases.*
2. **Merged relation approach**: Both entity types are stored in the same relational table, "pre-joined". If the relationship is not total both ways, there will be null padding on tuples that represent just one entity type. Any attributes of R are also moved to this table.
3. **Cross-reference approach**: A separate relation represents R; each tuple is a foreign key from S and a foreign key from T. Any attributes of R are also added to this relation. Here foreign keys should CASCADE.

Approach	Join cost	Null-storage cost
Foreign key	1	low to moderate
Merged relation	0	very high, unless both are total
Cross-reference	2	None

Step 4: Binary 1:N Relationship Types

Let the relationship be of the form [S]——N<R>1——[T]. The primary key of T is added as a foreign key in S. Attributes of R are moved to S. This is the foreign key approach. The merged relation approach is not possible for 1:N relationships. (Why?) The cross-reference approach might be used if the join cost is worth avoid null storage.

Step 5: Binary M:N Relationship Types
Here the cross-reference approach (also called a *relationship relation*) is the only possible way.

Step 6: Multivalued Attributes
Let an entity S have multivalued attribute A. Create a new relation R representing the attribute, with a foreign key into S added. The primary key of R is the combination of the foreign key and A. Once again this relation is dependent on an "owner relation" so its foreign key should CASCADE.

Step 7: Higher-Arity Relationship Types

Here again, use the cross-reference approach. For each n-ary relationship create a relation to represent it. Add a foreign key into each participating entity type. Also add any attributes of the relationship. The primary key of this relation is the combination of all foreign keys into participating entity types *that do not have a max cardinality of 1*.

Implementation:

Conclusion: Comment on importance of relational mapping for given case study from ER/EER diagram

Experiment No.3
Create and populate database using Data Definition Language (DDL) and Apply Integrity Constraints for the specified system
Date of Performance:
Date of Submission:

Aim: Create and populate database using Data Definition Language (DDL) and apply Integrity Constraints for the specified system

Objective: DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

Integrity constraints are used to ensure accuracy and consistency of data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity

Theory: DDL Commands

Create a table
Display the table description
Rename the table
Alter the table
Drop the table
Integrity constraints are:

1. PRIMARY KEY CONSTRAINTS
2. FOREIGN KEY CONSTRAINTS
3. UNIQUE KEY CONSTRAINTS
4. NULL CONSTRAINTS
5. NOT NULL CONSTRAINTS
6. CHECK CONSTRAINTS
7. DEFAULT CONSTRAINTS

Implementation:

Conclusion: Comment on use of DDL commands used for given schema and integrity constraints applied for given schema..

Experiment No.4
Apply DML Commands for your the specified System.
Date of Performance:
Date of Submission:

Aim:- Apply DML Commands for your the specified System

Objective: The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

Theory:

DML:-DATA MANIPULATION LANGUAGE

Commands used in DML are

Insert Values
Retrieve all attributes
Update table
Delete table

Implementation:

Conclusion: Comment on use of DML commands used for given schema.

Experiment No.5
Perform Simple queries, string manipulation operations and aggregate functions
Date of Performance:
Date of Submission:

Aim:- Perform Simple queries, string manipulation operations and aggregate functions.

Objective: Queries are a way of searching for and compiling data from one or more tables string manipulation operations are used to perform an operation on input string and return an output string .aggregate functions are used to fin Average, Maximum and minimum values, count values from given database

Theory:

SQL pattern matching allows you to search for patterns in data if you don't know the exact word or phrase you are seeking. This kind of SQL query uses wildcard characters to match a pattern, rather than specifying it exactly. For example, you can use the wildcard "C%" to match any string beginning with a capital C.

Using the LIKE Operator

To use a wildcard expression in an SQL query, use the LIKE operator in a WHERE clause, and enclose the pattern within single quotation marks.

Using the % Wildcard to Perform a Simple Search

To search for any employee in your database with a last name beginning with the letter C, use the following Transact-SQL statement:

```
SELECT *
FROM employees
WHERE last_name LIKE 'C%'
```

Omitting Patterns Using the NOT Keyword

Use the NOT keyword to select records that don't match the pattern. For example, this query returns all records whose name last does *not* begin with C:

```
 SELECT *
 FROM employees
 WHERE last_name NOT LIKE 'C%'
```

Matching a Pattern Anywhere Using the % Wildcard Twice

Use two instances of the **%** wildcard to match a particular pattern anywhere. This example returns all records that contain a C anywhere in the last name:

```
 SELECT *
 FROM employees
 WHERE last_name LIKE '%C%'
```

Finding a Pattern Match at a Specific Position

Use the **_** wildcard to return data at a specific location. This example matches only if C occurs at the third position of the last name column:

```
SELECT *
 FROM employees
 WHERE last_name LIKE '_ _C%'
```

Supported Wildcard Expressions in Transact SQL

There are several wildcard expressions supported by Transact SQL:

- The **%** wildcard matches zero or more characters of any type and can be used to define wildcards both before and after the pattern. If you're familiar with DOS pattern matching, it's the equivalent of the * wildcard in that syntax.
- The **_** wildcard matches exactly one character of any type. It's the equivalent of the **?** wildcard in DOS pattern matching.
- Specify a list of characters by enclosing them in square brackets. For example, the wildcard **[aeiou]** matches any vowel.
- Specify a range of characters by enclosing the range in square brackets. For example, the wildcard **[a-m]** matches any letter in the first half of the alphabet.

Student (sid,sname,city,age, Marks)

Department(did, dname, sid)

Q1. Create a table student with given attributes.

Q2. Create a table department with given attributes.

Q3. Insert values into the respective tables &amp; display them.

Q4. Update any row from student relation

Q5. Delete any row from the department table.

Q6. Give the minimum age of the student relation.

Q7. Find out the avg of marks of the student relation.

Q8. Give the total count of tuples in department relation group by did.

Q9. Find sname of student having last letter as M

Q.10 Find dname of student having second letter as O

Q.11 Find sname of student having first letter as G

Q.12 Find dname of student not having letter as A

Implementation:

Conclusion: Comment on use of SQL queries using aggregate functions and string manipulating operations

Experiment No.6
Perform Join operations
Date of Performance:
Date of Submission:

Aim: **Perform Join operations**

Objective: goal of creating a join condition is that it helps you to combine the data from two or more DBMS tables. The tables in DBMS are associated using the primary key and foreign keys**.**

Theory:

The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

There are different types of joins available in SQL −

- INNER JOIN − returns rows when there is a match in both tables.
- LEFT JOIN − returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN − returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN − returns rows when there is a match in one of the tables.
- SELF JOIN − is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN − returns the Cartesian product of the sets of records from the two or more joined tables

Implementation:

Conclusion: Comment on output of queries for given questions using Joins used in DBMS

Experiment No.7
Nested queries and Complex queries
Date of Performance:
Date of Submission:

Aim: Nested queries and Complex queries

Objective: In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query

Theory:

Sample table: Salesman

```
salesman_id  name       city       commission
```

Sample table: Orders

ord_no purch_amt ord_date customer_id salesman_id

Questions

1. Write a query to display all the orders from the orders table issued by the salesman 'Paul Adam'.

2 Write a query to display all the orders for the salesman who belongs to the city London.

3 Write a query to find all the orders issued against the salesman who may works for customer whose id is 3007

4 Write a query to display all the orders which values are greater than the average order value for 10th October 2012

5 Write a query to find all orders attributed to a salesman in New york.

6 Write a query to display the commission of all the salesmen servicing customers in Paris

Implementation:

Conclusion: Comment on output of queries for given questions using nested and complex queries

Experiment No.8
Procedures and Functions
Date of Performance:
Date of Submission:

Aim: To implement Functions and procedure.

Objective: The function must return a value but in Stored Procedure it is optional. Even a procedure can return zero or n values. Functions can have only input parameters for it whereas Procedures can have input or output parameters

Theory:

Procedure:

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

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
```

```
{IS | AS}
      BEGIN
      < procedure_body >
      END procedure_name;
```

Where,

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

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

Creating a Function

A standalone function is created using the **CREATE FUNCTION** statement. The simplified syntax for the **CREATE OR REPLACE PROCEDURE** statement is as follows −

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

Where,

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

Implementation:

Conclusion: Comment on use of procedures and functions for a given applications.

Experiment No.9
Views and Triggers
Date of Performance:
Date of Submission:

Aim: Views and Triggers

Objective: Views can join and simplify multiple **tables** into a single virtual table A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database. The trigger is mostly used for maintaining the integrity of the information on the database. For example, when a new record (representing a new worker) is added to the employees table, new records should also be created in the tables of the taxes, vacations and salaries

Theory:

VIEWS

A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are a type of virtual tables allow users to do the following −

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views

Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows −

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

You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

TIGGERS:

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, 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 can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes −

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

Creating Triggers

The syntax for creating a trigger is −

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will 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, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will 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.

Implementation:

Conclusion: Comment on Output you get after applying trigger and views and also comment on use of views and trigger.

Experiment No.10
Mini project- Course project report using database connectivity
Date of Performance:

Date of Submission:

Aim: Mini project- Creating a Two-tier client-server database applications using database connectivity

Objective: Java Database Connectivity (JDBC)/ODBC is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is a Java-based data access technology used for Java database connectivity

Theory:

Java Database Connectivity with Oracle

To connect java application with the oracle database, we need to follow 5 following steps. In this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:

1. Driver class: The driver class for the oracle database is oracle.jdbc.driver.OracleDriver.
2. Connection URL: The connection URL for the oracle10G database is jdbc:oracle:thin:@localhost:1521:xe where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these

information from the tnsnames.ora file.
3. Username: The default username for the oracle database is system.
4. Password: It is the password given by the user at the time of installing the oracle database.

Create a Table
Before establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table.

Example to Connect Java Application with Oracle database

In this example, we are connecting to an Oracle database and getting data from emp table. Here, system and oracle are the username and password of the Oracle database.

```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");

//step2 create  the connection object
Connection con=DriverManager.getConnection(
        "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

        //step3 create the statement object
        Statement stmt=con.createStatement();

        //step4 execute query
        ResultSet rs=stmt.executeQuery("select * from emp");
        while(rs.next())
        System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

        //step5 close the connection object
        con.close();

        }catch(Exception e){ System.out.println(e);}

25.          }
26.          }
```

To connect java application with the Oracle database ojdbc14.jar file is required to be loaded.

download the jar file ojdbc14.jar

Two ways to load the jar file:

1. paste the ojdbc14.jar file in jre/lib/ext folder
2. set classpath

1) paste the ojdbc14.jar file in JRE/lib/ext folder:
Firstly, search the ojdbc14.jar file then go to JRE/lib/ext folder and paste the jar file here.

2) set classpath:
There are two ways to set the classpath:

- temporary
- permanent

Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps.

In this example we are using MySql as the database. So we need to know following informations for the mysql database:

1. Driver class: The driver class for the mysql database is com.mysql.jdbc.Driver.
2. Connection URL: The connection URL for the mysql database is jdbc:mysql://localhost:3306/sonoo where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. Username: The default username for the mysql database is root.
4. Password: It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

Let's first create a table in the mysql database, but before creating table, we need to create database first.

```
create database sonoo;
use sonoo;
create table emp(id int(10),name varchar(40),age int(3));
```

Example to Connect Java Application with mysql database
In this example, sonoo is the database name, root is the username and password both.

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

Implementation:

Prepare Report and show demonstration

Conclusion: Comment on the Prototype of given application using database connectivity