D Y PATIL
RAMRAO ADIK INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

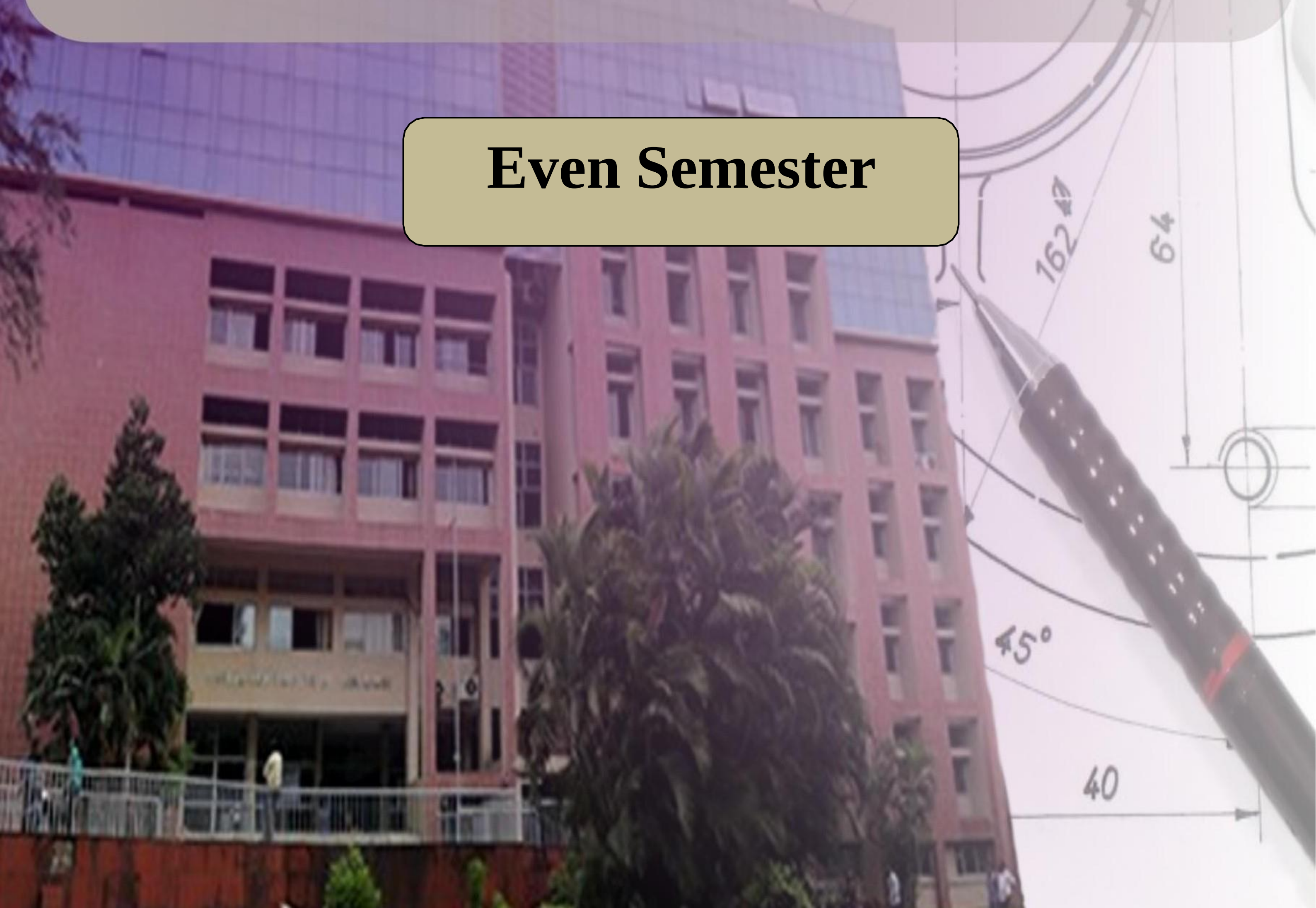
Department of Computer Engineering

Lab Manual

Second Year Semester-IV

Subject: Database Management System Lab

Even Semester



Institutional Vision, Mission and Quality Policy

Our Vision

To foster and permeate higher and quality education with value added engineering, technology programs, providing all facilities in terms of technology and platforms for all round development with societal awareness and nurture the youth with international competencies and exemplary level of employability even under highly competitive environment so that they are innovative adaptable and capable of handling problems faced by our country and world at large.

RAIT's firm belief in new form of engineering education that lays equal stress on academics and leadership building extracurricular skills has been a major contribution to the success of RAIT as one of the most reputed institution of higher learning. The challenges faced by our country and world in the 21 Century needs a whole new range of thought and action leaders, which a conventionaleducational system in engineering disciplines are ill equipped to produce. Our reputation in providing good engineering education with additional life skills ensure that high grade and highly motivated students join us. Our laboratories and practical sessions reflect the latest that is being followed in the Industry. The project works and summer projects make our students adept at handling the real life problems and be Industry ready. Our students are well placed in the Industry and their performance makes reputed companies visit us with renewed demands and vigour.

Our Mission

The Institution is committed to mobilize the resources and equip itself with men and materials of excellence thereby ensuring that the Institution becomes pivotal center of service to Industry, academia, and society with the latest technology. RAIT engages different platforms such as technology enhancing Student Technical Societies, Cultural platforms, Sports excellence centers, Entrepreneurial Development Center and Societal Interaction Cell. To develop the college to become an autonomous Institution & deemed university at the earliest with facilities for advanced research and development programs on par with international standards. To invite international and reputed national Institutions and Universities to collaborate with our institution on the issues of common interest of teaching and learning sophistication.

RAIT's Mission is to produce engineering and technology professionals who are innovative and inspiring thought leaders, adept at solving problems faced by our nation and world by providing quality education.

The Institute is working closely with all stake holders like industry, academia to foster knowledge generation, acquisition, dissemination using best available resources to address the great challenges being faced by our country and World. RAIT is fully dedicated to provide its students skills that make them leaders and solution providers and are Industry ready when they graduate from the Institution.

We at RAIT assure our main stakeholders of students 100% quality for the programmes we deliver. This quality assurance stems from the teaching and learning processes we have at work at our campus and the teachers who are handpicked from reputed institutions IIT/NIT/MU, etc. and they inspire the students to be innovative in thinking and practical in approach. We have installed internal procedures to better skills set of instructors by sending them to training courses, workshops, seminars and conferences. We have also a full fledged course curriculum and deliveries planned in advance for a structured semester long programme. We have well developed feedback system employers, alumni, students and parents from to fine tune Learning and Teaching processes. These tools help us to ensure same quality of teaching independent of any individual instructor. Each classroom is equipped with Internet and other digital learning resources.

The effective learning process in the campus comprises a clean and stimulating classroom environment and availability of lecture notes and digital resources prepared by instructor from the comfort of home. In addition student is provided with good number of assignments that would trigger his thinking process. The testing process involves an objective test paper that would gauge the understanding of concepts by the students. The quality assurance process also ensures that the learning process is effective. The summer internships and project work based training ensure learning process to include practical and industry relevant aspects. Various technical events, seminars and conferences make the student learning complete.

Our Quality Policy

ज्ञानधीनं जगत् सर्वम।

Knowledge is supreme.

Our Quality Policy

It is our earnest endeavour to produce high quality engineering professionals who are innovative and inspiring, thought and action leaders, competent to solve problems faced by society, nation and world at large by striving towards very high standards in learning, teaching and training methodology.

Our Motto: If it is not of quality, it is NOTRAIT!

Departmental Vision, Mission

Vision

To impart higher and quality education in computer science with value added engineering and technology programs to prepare technically sound, ethically strong engineers with social awareness. To extend the facilities, to meet the fast changing requirements and nurture the youths with international competencies and exemplary level of employability and research under highly competitive environments.

Mission

To mobilize the resources and equip the institution with men and materials of excellence to provide knowledge and develop technologies in the thrust areas of computer science and Engineering. To provide the diverse platforms of sports, technical, cocurricular and extracurricular activities for the overall development of student with ethical attitude. To prepare the students to sustain the impact of computer education for social needs encompassing industry, educational institutions and public service. To collaborate with IITs, reputed universities and industries for the technical and overall upliftment of students for continuing learning and entrepreneurship.

Departmental 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 the engineering practice.

PO9. **Individual and team work**: 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.

Program Specific Outcomes (PSOs)

PSO1. To build competencies towards problem solving with an ability to understand, identify,analyze and design the problem, implement and validate the solution including both hardware and software.

PSO2. To build appreciation and knowledge acquiring of current computer techniques with an ability to use skills and tools necessary for computing practice.

PSO3.To be able to match the industry requirements in the area of computer science and engineering. To equip skills to adopt and imbibe new technologies.

Index

List of Experiments

Sr. No.	Experiments Name
1.	Identify the case study and detail statement of problem.
2.	Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model.
3.	Perform experiment on mapping of ER/EER to Relational schema model.
4.	Create and populate database using Data Definition Language (DDL) and DML Commands for you‘re the specified System.
5.	Apply Integrity Constraints for the specified system.
6.	Perform Simple queries, string manipulation operations.
7.	Implement and execute Nested queries and Complex queries
8.	Perform and Implement Join operations.
9.	Implement Views and Triggers.
10	Study and understand Transaction and Concurrency control.
11	**(Content beyond syllabus)** Implement horizontal, vertical and mixed fragmentation for given case study.

Course Objective, Course Outcome& Experiment Plan

Course Objective:

1	To explore design and develop of relational model
2	To present SQL and procedural interfaces to SQL comprehensively
3	To introduce the concepts of transactions and transaction processing

Course Outcomes:

CO1	**Formulate problem Statement for real world example by using the fundamentalconcept of database system.**
CO2	Design ER /EER diagram and convert to relational model for the real world application.
CO3	**Understand and** Apply relational algebra, SQL Statements (DDL, DML, DCL and TCL commands) and constraints on the relations.
CO4	Write simple and complex queries.
CO5	Use PL / SQL Constructs.
CO6	Demonstrate the concept of concurrent transactions execution and frontend-backend connectivity.

Experiment Plan

Module No.	Week No.	Experiments Name	Course Outcome	Weight-age
1.	W1	Identify the case study and detail statement of problem.	CO1	10
2.	W2	Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model.	CO2	05
3.	W3	Perform experiment on mapping of ER/EER to Relational schema model.	CO2	05
4.	W4	Create and populate database using Data Definition Language (DDL) and DML Commands for you're the specified System.	CO3	05
5.	W5	Apply Integrity Constraints for the specified system.	CO3	05
6	W6	Perform Simple queries, string manipulation operations.	CO4	04
7.	W7	Implement and execute Nested queries and Complex queries.	CO4	03
8.	W8	Perform and Implement Join operations.	CO4	03
9.	W9	Implement Views and Triggers.	CO5	10
10.	W10	Study and understand Transaction and Concurrency control.	CO6	10
11.	W11	**(Content beyond syllabus)** Implement horizontal, vertical and mixed fragmentation for given case study.		

Co-Po Mapping for Practical

Subject Weight	**Course Outcomes**	**Contribution to Program outcomes**											
		P O 1	**P O 2**	**Pc O 3**	**P O 4**	**P O 5**	**P O 6**	**P O 7**	**P O 8**	**P O 9**	**P O 10**	**P O 11**	**P O 12**
	CO1: Formulate problem Statement for real world example by using the fundamental concept of database system.	3	2		2		1				1		1
	CO2: Design ER /EER diagram and convert to relational model for the real world application.	2	2	2		1						1	2
PRATICAL 100%	CO3: **Understand and** Apply relational algebra, SQL Statements (DDL, DML, DCL and TCL commands) and constraints on the relations.	2	1	2	1				2				2
	CO4: Write simple and complex queries	2	2	2				1			1	2	
	CO5: Use PL / SQL Constructs.		2	2			1		1	1		2	1
	CO6: Demonstrate the concept of concurrent		2	2			2		2	1		1	

	transactions execution and frontend-backend connectivity.												

Mapping Course Outcomes (CO) – Program Specific Outcomes (PSO)

Program Specific Outcome/ Course outcome	**PSO1**	**PSO2**	**PSO3**
CO1: Formulate problem Statement for real world example by using the fundamental concept of database system.	3	2	2
CO2: Design ER /EER diagram and convert to relational model for the real world application.	2	3	-
CO3: **Understand and** Apply relational algebra, SQL Statements (DDL, DML, DCL and TCL commands) and constraints on the relations.	1	2	2
CO4: Write simple and complex queries	-	2	2
CO5: Use PL / SQL Constructs.	1	1	2
CO6: Demonstrate the concept of concurrent transactions execution and frontend-backend connectivity.	2	2	3

Study and Evaluation Scheme

Course Code	Course Name	Teaching Scheme			Credits Assigned			
		Theory	Practical	Tutorial	Theory	Practical	Tutorial	Total
CSL402	Database Management System Lab	-	02	-		1	--	01

Term Work

Laboratory work will be based on DBMS syllabus with minimum 10 experiments to be incorporated. Experiments should be completed by students on the given time duration

Experiments..(15) Marks

Attendance (Theory + Practical)...(05)

Assignment..(05) Marks

Marks Total...(25) Marks

Practical & Oral:

Practical and oral Exam should be conducted for the Lab, on Database Management System subject for given list of experiments.

Implementation...(15) Marks Oral

(10) Marks **Total -- (25) Marks**

Database Management System Lab

Experiment No. : 1

Identify the case study and detail statement of problem.

Experiment No.1

1. **Aim:** Identify the case study and detail statement of problem.

2. **Objectives:**
    - Understand the Need and importance of Database Management System in today's world.

3. **Outcomes:** Formulate problem statement for real world example by using fundamental concepts of database system.

4. **Hardware / Software Required: Ms Word**

5. **Theory:** Identify case Study.

    Student can take any case study related to database management system of their choice. Following are some examples of DBMS case study.

    1. Library management system
    2. Airline Reservation system.
    3. Online shopping system
    4. E commerce website
    5. Inventory management system
    6. Consumer Review System etc.

    Problem Statement has to be written on any dbms case study. Problems statement should consist of following points.

    1. **Purpose**– it includes the introduction and purpose of case study.
    2. **Scope** –it includes the detailed scope and working of the case study.
    3. **Functionalities**-it includes the detailed functions included of working model.

    Note: Students has to write above mentioned things according to their case study topic.

6. **Conclusion**: Formulate the problem statement for given case study and understand the use and its applications in today's world.

7. **Viva Questions:**
    - What is Need of DBMS?
    - Have you ever interacted with Database?
    - What is Meta-Data?

8. **References:**

1. G. K. Gupta :‖Database Management Systems‖, McGraw – Hill.
2. Korth, Slberchatz,Sudarshan, :‖Database System Concepts‖, 6th Edition, McGraw – Hill
3. Elmasri and Navathe, ― Fundamentals of Database Systems‖, 5thEdition, PEARSON
4. Peter Rob and Carlos Coronel, ― Database Systems Design, Implementation and Management‖, Thomson Learning, 5th Edition.

Database Management System Lab

Experiment No.: 2

Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model.

Experiment No.2

1. **Aim:** Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model.

2. **Objectives:**

    - Learn and practice data modelling using the entity-relationship and developing database designs.

3. **Outcomes:**Design and draw ER and EER diagram for the real life problem.

4. **Hardware / Software Required:** Any ER-Diagram tool such as SmartDraw, Lucidchart, ERDPlus etc.
5. **Theory:**

ER- Diagram:

It is an Entity –Relationship diagram which is used to represent the relationship between different entities. An entity is an object in the real world which is distinguishable from other objects. The overall logical structure of a database can be expressed graphically by an ER diagram, which is built up from following components.

Mapping Cardinalities:

It expresses the number of entities to which another entity can be associated via relationship set. For a binary relationship set R between entity sets A and B. The Mapping Cardinalities must be one of the following.

- One to one
- One to many
- Many to one
- Many to many

E-R diagram Components

1. **Rectangles:** represent entity sets.
2. **Ellipses:** represent attributes.
3. **Diamonds:** represent relationships among entity sets.

4. **Lines:** link attribute to entity sets and entity sets to relationships.
5. **Double eclipse** represents multivalve attributes.
6. **Double rectangle** represents Weak entity sets.
7. Primary key attributes are underlined.

Design a model using E-R Diagram

Step 1:Study the given system.
Step 2: Identify **Entity Sets** i.e. tables which can be constructed
Step 3: Identify attributes for each entities
Check - Primary key and Partial key in attribute.
For composite attribute - attribute which can be subdivided.
(Ex. Name - F-Name, M_name, L_name)
Multivalued Attribute - Attribute having multiple values (Ex. Phone number).Derived Attribute - Attribute can be derived from other attributes
(Ex. age)
Step 4: Identify **relationships** among entity sets
Step 5: Check for **Weak Entity Set**.
Step 6: Check for **Total and Partial Participation** of Entity set
Step 7: Check specialization/generalization feature (using ISA)

Extensions to E-R Model

Basic E-R model is good for many uses Several extensions to E-R model for more advanced modeling – Generalization and specialization and Aggregation.

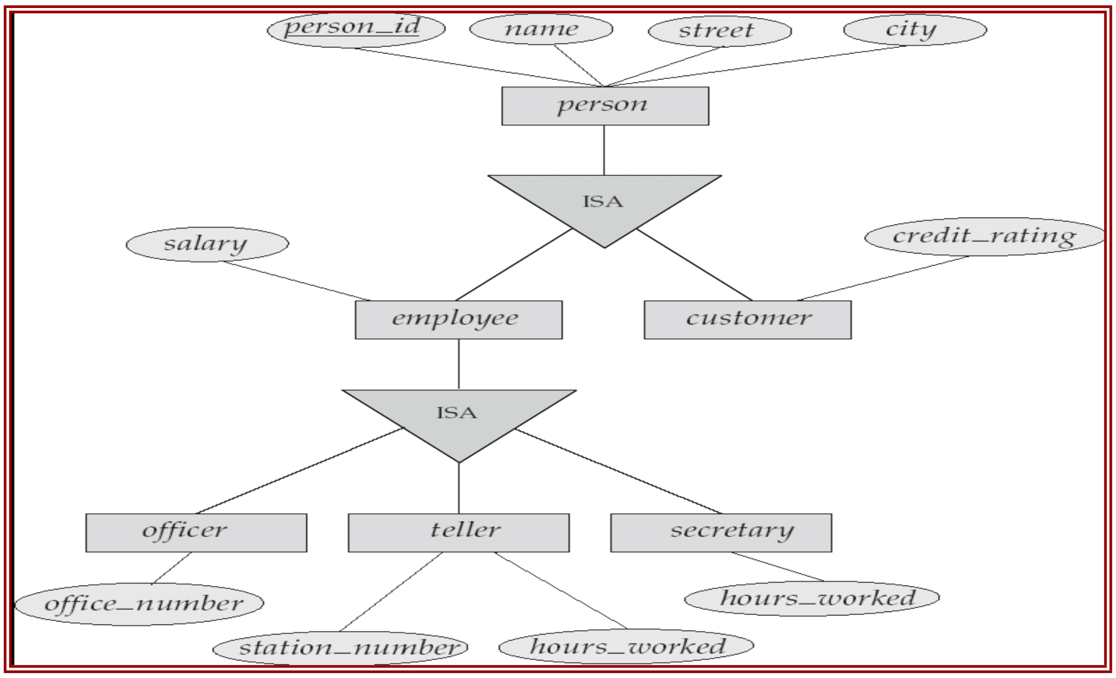
Specialization

An entity-set might contain distinct subgroups of entities – Subgroups have some different attributes, not shared by entire entity-set .E-R model provides specialization to represent such entity-sets Example: bank account categories – Checking accounts – Savings accounts – Have common features, but also unique attributes.

Generalization and Specialization

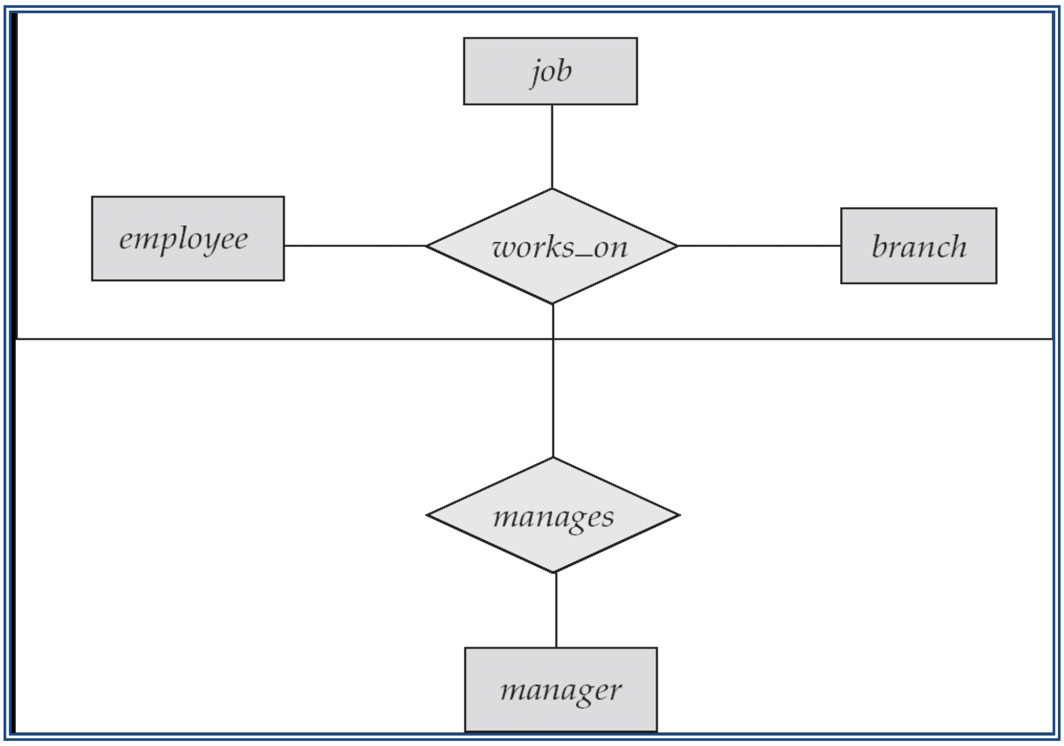
Generalization: a "bottom up" approach – Taking similar entity-sets and unifying their common features – Start with specific entities, then create generalizations from them

Specialization: a "top down" approach – Creating general purpose entity-sets, then providing specializations of the general idea – Start with general notion, then refine it.

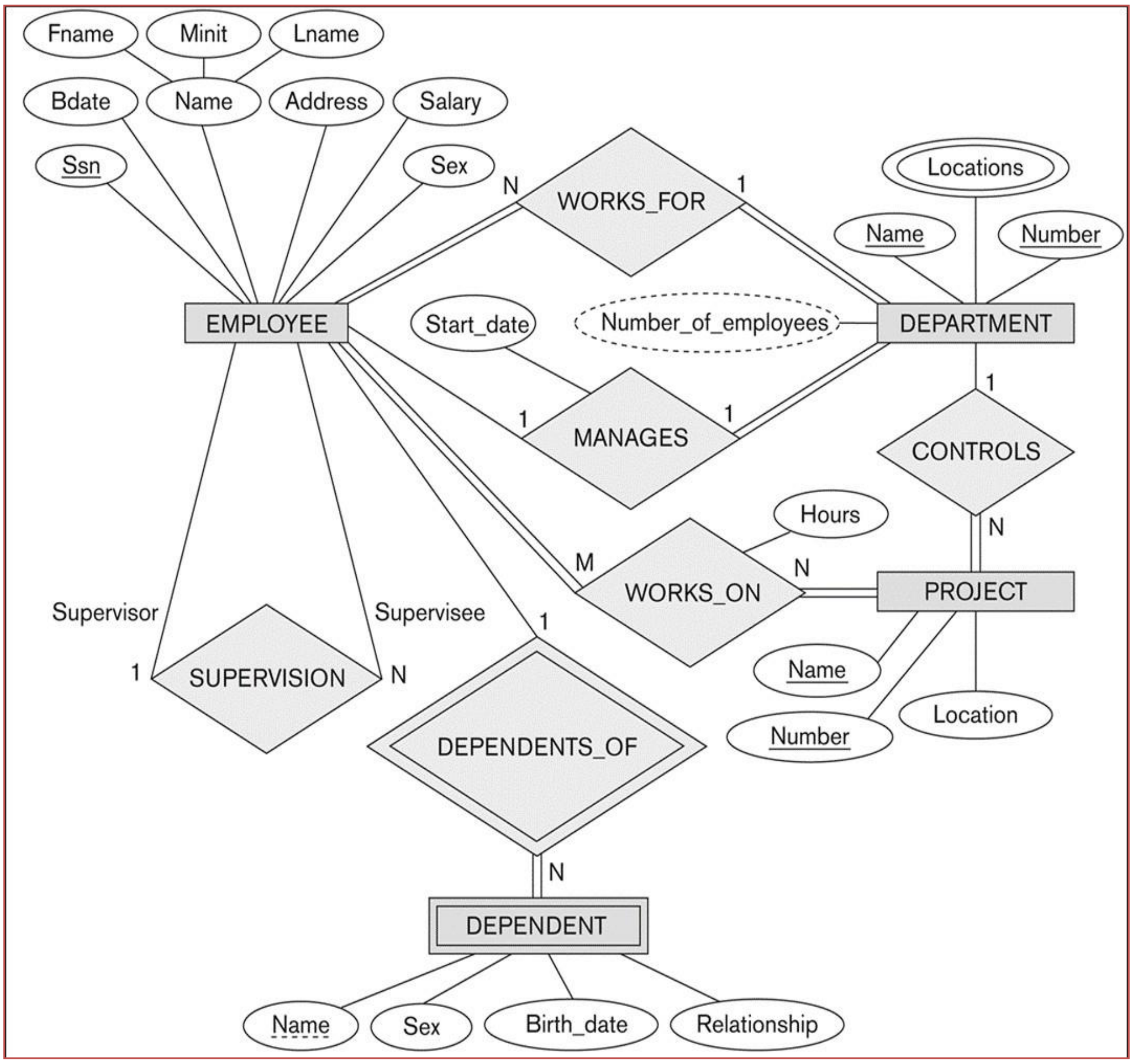


Example of Generalization and specialization.

Aggregation is a process when relation between two entities is treated as a **single entity**. Below is the example of aggregation.



Note: -Students has to draw ER diagram of their own case study by adding EER features. Below is the example of ER diagram.



Example: Sample ER diagram for Company Management System

Note: Students has to draw the ER diagram for their case study topic.

9. Conclusion: Designed the ER and EER of the assigned case study topics and understand the ER and EER Model in depth.

10. Viva Questions:

- What are the different types of attributes?
- What is aggregation, generalization and specialization.
- What are different between entity and entity set?

11. References:

1. G. K. Gupta :‖Database Management Systems‖, McGraw – Hill.
2. Korth, Slberchatz,Sudarshan, :‖Database System Concepts‖, 6th Edition, McGraw – Hill

3. Elmasri and Navathe, ― Fundamentals of Database Systems‖, 5thEdition, PEARSON
4. Peter Rob and Carlos Coronel, ― Database Systems Design, Implementation and Management‖, Thomson Learning, 5th Edition.

Database Management System Lab

Experiment No.: 3

Perform experiment on mapping of ER/EER to Relational schema model.

Experiment No.3

1. **Aim:**Perform experiment on mapping of ER/EER to Relational schema model

2. **Objectives:**

   - Learn and practice data modelling using the entity-relationship and developing database designs.
   - To understandconversion rules of ER and EER model to Relational Model.

3. **Outcomes:**Design and draw ER and EER diagram for the real life problem.

4. **Hardware / Software Required :**Oracle, My SQL

5. **Theory:**

<u>Rules to convert ER schema to relational schema.</u>

Step 1: Mapping of Regular Entity Types

- For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
- Choose one of the key attributes of E as the primary key for R.
- If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

Step 2: Mapping of Weak Entity Types

- For each weak entity type W in the ER schema with owner entity type E, create a relation R & include all simple attributes (or simple components of composite attributes) of W as attributes of R.
- Also, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).
- The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.

Step 3: Mapping of Binary 1:1 Relation Types

- For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R.

Step 4: Mapping of Binary 1:N Relationship Types.

- For each regular binary 1:N relationship type R, identify the relation S that represent the participating entity type at the N-side of the relationship type.
- Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R.
- Include any simple attributes of the 1:N relation type as attributes of S.

Step 5: Mapping of Binary M:N Relationship Types.

- For each regular binary M:N relationship type R, create a new relation S to represent R.
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S.
- Also include any simple attributes of the M:N relationship type (or
- Simple components of composite attributes) as attributes of S.

Step 6: Mapping of Multivalued attributes.

- For each multivalued attribute A, create a new relation R.
- This relation R will include an attribute corresponding to A, plus the primary key attribute K-as a foreign key in R-of the relation that represents the entity type of relationship type that has A as an attribute.
- The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.

Step 7: Mapping of N-ary Relationship Types.

- For each n-ary relationship type R, where n>2, create a new relationship S to represent R.
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
- Also include any simple attributes of the n-ary relationship type (or simple components of composite attributes) as attributes of S.

<u>Summary</u>

ER Model	**Relational Model**
Entity type	Entity relation
1:1 and 1:N relationship type	Foreign key or relationship relation
M:N relationship type	Relationship relation and two foreign keys

n-ary relationship type	Relationship relation and n foreign keys
Simple attribute	Attribute
Composite attribute	Set of component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary key or secondary key

Rules to convert EER schema to relational schema.

Mapping of Specialization or Generalization

- Step 8: Options for Mapping Specialization or Generalization

  Option 8A: Multiple relations—super class and subclasses, K for the relations
  For any specialization (total or partial, disjoint or overlapping).

  Option 8B: Multiple relations—subclass relations only.
  Subclasses are total.
  Specialization has disjointedness constraint

  Option 8C: Single relation with one type attribute .
  Type or discriminating attribute indicates subclass of tuple.
  Subclasses are disjoint.
  Potential for generating many NULL values if many specific attributes exist in the subclasses.

  Option 8D: Single relation with multiple type attributes.
  Subclasses are overlapping Will also work for a disjoint specialization.

6. **Conclusion** :

After performing this experiment we are able to convert the Entity Relationship model into relational model along with extended features of ER model.

7. Viva Questions:

- What is the Difference between the total participation and partial participation?
- What are the different types of attributes?

8. References:

1. Elmasri and Navathe, ― Fundamentals of Database Systems‖, 5thEdition, PEARSON
2. Peter Rob and Carlos Coronel, ― Database Systems Design, Implementation and Management‖, Thomson Learning, 5th Edition.

Database Management System Lab

Experiment No. : 4

Create and populate database using Data Definition Language (DDL) and DML Commands for specified System.

Experiment No.4

1. **Aim:**Create and populate database using Data Definition Language (DDL) and DML Commands for specified System.

2. **Objectives:** From this experiment, the student will be able to
   - Understand the use of Structured Query Language (SQL) and learn SQL syntax.

3. **Outcomes:** Create and update database and tables with different DDL and DML statements.

4. **Hardware / Software Required :**Oracle, My SQL

5. **Theory:**

Data types in SQL:-

- Char (n)- A fixed length character length string with user specified length.
- Number- Number is used to store numbers (fixed or floating point).
- Int- an integer
- Varchar (n)- A variable character length string with user specified maximum length n.
- Date- A calendar date containing a (four digit) year, month and day of the month.
- Time: The time of day, in hours, minutes and seconds Eg. Time '09:30:00'.

Design relational schema

SQL: It is structured query language, basically used to pass the query to retrieve andmanipulate the information from database. Depending upon the nature ofquery, SQL is divided into different components:
• DDL(Data Definition Language)
• DML(Data Manipulation Language)
• DCL(Data Control Language)

Data Definition Language

DDL: The Data Definition Language (DDL) is used to create the database (i.e. tables, keys, relationships etc), maintain the structure of the database and destroy databases and database objects.
Eg. Create, Drop, Alter, Describe, Truncate

1. CREATE statements: It is used to create the table.
Syntax:

CREATE TABLE table_name(columnName1 datatype(size), columnName2 datatype(size),………);

2. ALTER statements: To modify an existing database object.

-Adding new columns:

Syntax:

Alter table table name Add(New_columnName1 datatype(size), New_columnName2 datatype(size),………);

-Dropping a column from a table:

Syntax

Alter table table_name DROP column columnName:

- Modifying Existing columns:

Syntax:

Alter table table name Modify (columnName1 Newdatatype(Newsize));

3.Describe statements: To describe the structure (column and data types) of an existing database, table, index, or view.

Syntax:

DESC table_name;

4.DROP statements: To destroy an existing database, table, index, or view. If a table is dropped all records held within it are lost and cannot be recovered.

Syntax:

DROP TABLE table_name;

5.Truncate statements: To destroy the data in an existing database, table, index, or view. If a table is truncated all records held within it are lost and cannot be recovered but the table structure is maintained.

Syntax:

TRUNCATE TABLE table_name;

<u>Data Manipulation Language</u>

A **Data Manipulation Language** enables programmers and users of the database to retrieve insert, delete and update data in a database. e.g. INSERT, UPDATE, DELETE, SELECT.

INSERT:

INSERT statement adds one or more records to any single table in a relational database.

Syntax:

INSERT INTO tablename VALUES (expr1,expr2… .....);

SELECT:

SELECT statement returns a result set of records from one or more tables.

The select statement has optional clauses:

- **WHERE** specifies which rows to retrieve
- **GROUP BY** groups rows sharing a property so that an aggregate function can be applied to each group having group.
- **HAVING** selects among the groups defined by the GROUP BY clause.
- **ORDER BY** specifies an order in which to return the rows.

Syntax:

```
SELECT<attribute list>
FROM<table list>
WHERE<condition>
```

Where

- Attribute list is a list of attribute name whose values to be retrieved by the query.
- Table list is a list of table name required to process query.
- Condition is a Boolean expression that identifies the tuples to be retrieved by query.

UPDATE:

UPDATE statement that changes the data of one or more records in a table. Either all therows can be updated, or a subset may be chosen using a condition.

Syntax:

UPDATE table_name SET column_name = value [, column_name = value .....] [WHERE condition]

DELETE:

DELETE statement removes one or more records from a table. A subset may be defined for deletion using a condition, otherwise all records are removed.

Syntax:

DELETE FROM tablename WHERE condition:

6. Conclusion:

For the given case study designed the databases schema and implemented queries using DDL and DML commands.

7. Viva Questions:

- How will you create the table?
- What are DDL and DML commands?
- What effect will cause by DROP table command?

9. References:

1. Elmasri and Navathe, ― Fundamentals of Database Systems‖, 5thEdition, PEARSON
2. Peter Rob and Carlos Coronel, ― Database Systems Design, Implementation and Management‖, Thomson Learning, 5th Edition.

Database Management System Lab

Experiment No. : 5

Apply Integrity Constraints for the specified system.

Experiment No.5

1. **Aim:**Apply Integrity Constraints for the specified system.
2. **Objectives:**
   - Understand database Integrity Constraints.
   - To make the database schema more secure.
3. **Outcomes:** Apply /Add integrity constraints and able to provide security to data.
4. **Hardware / Software Required :**Oracle
5. **Theory:**

Constraints are the Rules which are enforced on the data being stored in a table are called *Constraints.*

-Table-level & Row-level constraints

-<u>Types of constraints</u>

1. **NOT NULL**

NOT NULL constraints are in-line constraints that indicate that a column can not contain NULL values.

Example: create table customer
```
(Custname varchar(10) not null,
 Status  varchar(5) not null);
```

2. **PRIMARY KEY**

Primary key constraints define a column or series of columns that uniquely identify a given row in a table. Defining a primary key on a table is optional and you can only define a single primary key on a table. A primary key constraint can consist of one or many columns (up to 32). Any column that is defined as a primary key column is automatically set with a NOT NULL status.

Example: CREATE TABLE CUSTOMER
```
(CUSTID NUMBER(10) PRIMARY KEY,
   CUSTNAME VARCHAR(10) NOT NULL,
STATUS VARCHAR (5) NOT NULL);
```

3. **FOREIGN KEY/REFERENTIAL INTEGRITY**

A foreign key constraint is used to enforce a relationship between two tables. It is used to ensure referential integrity of the data.

Example:

```
 CREATE TABLE SUPPLIER
( SUPPLIER_ID NUMERIC(10) NOT NULL,
  SUPPLIER_NAME VARCHAR2(50) NOT NULL,
  CONTACT_NAME VARCHAR2(50),
  CONSTRAINT SUPPLIER_PK PRIMARY KEY (SUPPLIER_ID)
);

CREATE TABLE PRODUCTS
( PRODUCT_ID NUMERIC(10) NOT NULL,
  SUPPLIER_ID NUMERIC(10) NOT NULL,
  CONSTRAINT FK_SUPPLIER
    FOREIGN          KEY          (SUPPLIER_ID)
    REFERENCES SUPPLIER(SUPPLIER_ID));
```

4. **CHECK**
   Check constraints validate that values in a given column meet a specific criteria.

   Example:

```
CREATE TABLE CUSTOMER
                    (SID NUMBER CHECK(SID>10),
                    LASTNAME VARCHAR(10),
                    FIRSTNAME VARCHAR(10|));
```

5. **UNIQUE**
   Unique constraints are like alternative primary key constraints. A unique constraint defines a column, or series of columns, that must be unique in value.

   Example:-

```
CREATE TABLE CUSTOMER(SID NUMBER UNIQUE,
LASTNAME VARCHAR(10),
FIRSTNAME VARCHAR(10|));
```

6. **Conclusion:**For the case study topic created the schema according to entity and referential integrity constraints.

7. **Viva Questions:**
   - What do you mean by constraints?
   - What are referential integrity constraints?
   -

8. **References:**

1. G. K. Gupta :‖Database Management Systems‖, McGraw – Hill.
2. Korth, Slberchatz,Sudarshan, :‖Database System Concepts‖, 6th Edition, McGraw – Hill
3. Elmasri and Navathe, ― Fundamentals of Database Systems‖, 5thEdition, PEARSON

Database Management System Lab

No. : 6

Perform Simple Queries and String Manipulation Operations

Experiment No. 6

1. **Aim:** Perform simple queries and string manipulation operations.

2. **Objectives:** From this experiment, the student will be able to
   - Understand and apply different string functions on relational schema.

3. **Outcomes:** Implementand execute Simple, Complex queries and join operations**.**

4. **Hardware/ Software Required:** Oracle 10g

5. **Theory: Simple Queries**

A typical SQL query has the form:

select A1, A2, ..., An from r1, r2, ..., rm where P Query Structure

- *A_i* represents an attribute
- *R_i* represents a relation
- *P* is a predicate.

Select Clause:The select clause list the attributes desired in the result of a query.

Example: 1. Find the names of all branches in the *loan* relation:

select branch_namefrom loan

2. **Find the names of all branches in the *loan* relations, and remove duplicates**

   *select **distinct** branch_name from loan*

3. The keyword **all** specifies that duplicates not be removed.

   ***select all**branch_name**from** loan*

Where Clause: The where clause specifies conditions that the result must satisfy.

Example:To find all loan number for loans made at the Perryridge branch with loan amounts greater than $1200.

Select loan number from *loan*

where *branch_name* ='Perryridge'and *amount* > 1200

The from Clause: The from clause lists the relations involved in the query

Example: Find the Cartesian product *borrower X loan*

select *from *borrower, loan*

String Operations : SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that are described using two special characters:

- percent (%). The % character matches any substring.
- underscore (_). The _ character matches any character.

Example: Find the names of all customers whose street includes the substring "Main".

select *customer_name***from** *customer***where***customer_street* **like** '% Main%'

- LIKE is the ANSI/ISO standard operator for comparing a column value to another column value, or to a quoted string. Returns either 1 (TRUE) or 0 (FALSE)
- The SQL LIKE operator is only applied on a field of types CHAR or VARCHAR to match a pattern.
- To match a pattern from a word, special characters, and wildcards characters may have used with LIKE operator.
- Different LIKE operators with '%' and '_' wildcards:

LIKE OPERATOR	DESCRIPTION
WHERE CUSTOMERNAME LIKE 'A%'	FINDS ANY VALUES THAT START WITH "A"
WHERE CUSTOMERNAME LIKE '%A'	FINDS ANY VALUES THAT END WITH "A"
WHERE CUSTOMERNAME LIKE '%OR%'	FINDS ANY VALUES THAT HAVE "OR" IN ANY POSITION
WHERE CUSTOMERNAME LIKE '_R%'	FINDS ANY VALUES THAT HAVE "R" IN THE SECOND POSITION
WHERE CUSTOMERNAME LIKE 'A_%_%'	FINDS ANY VALUES THAT START WITH "A" AND ARE AT LEAST 3 CHARACTERS IN LENGTH

WHERE ContactName LIKE 'a%o'	**Finds any values that start with "a" and ends with "o"**
WHERE CustomerName LIKE 'a%'	**Finds any values that start with "a"**

Following are some examples on string operations

- The following SQL statement selects all customers with a CustomerName starting with "a":

  SELECT * FROM Customers
  WHERE CustomerName LIKE 'a%';

- The following SQL statement selects all customers with a CustomerName ending with "a":

  SELECT * FROM Customers
  WHERE CustomerName LIKE '%a';

- The following SQL statement selects all customers with a CustomerName that have "or" in any position:

  SELECT * FROM Customers
  WHERE CustomerName LIKE '%or%';

- The following SQL statement selects all customers with a CustomerName that have "r" in the second position:

  SELECT * FROM Customers
  WHERE CustomerName LIKE '_r%';

- The following SQL statement selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:

  SELECT * FROM Customers
  WHERE CustomerName LIKE 'a_%_%';

- The following SQL statement selects all customers with a ContactName that starts with "a" and ends with "o":

  SELECT * FROM Customers
  WHERE ContactName LIKE 'a%o';

- The following SQL statement selects all customers with a CustomerName that does NOT start with "a":

```
SELECT * FROM Customers
WHERE CustomerName NOT LIKE 'a%';
```

6. **Conclusion:** By understanding the concepts of simple queries and string operations, perform the sql queries along with different types of string operations for the case study.

7. **Viva Questions:**
   - What is the use of where clause?
   - What are the two string operators mostly used in SQL?

8. **References:**

1. Elmasri and Navathe, ― Fundamentals of Database Systems‖, 5thEdition, PEARSON
2. G. K. Gupta :‖Database Management Systems‖, McGraw – Hill.
3. Korth, Slberchatz,Sudarshan, :‖Database System Concepts‖, 6th Edition, McGraw – Hill

Database Management System Lab

Experiment No. : 7

Implement and Execute Nested queries and Complex queries

Experiment No. 7

1. **Aim:** Implement and execute Nested queries and Complex queries

2. **Objectives:** From this experiment, the student will be able to,
   - Learn how to apply sub-queries with select, insert and update statements and complex queries on relational schema.

3. **Outcomes:**Implement and execute Simple, Complex queries and join operations.

4. **Hardware / Software Required: Oracle 10g**

5. **Theory:**
   A Subquery or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
   - SQL provides a mechanism for the nesting of subqueries.
   - A **subquery** is a **select-from-where** expression that is nested within another query.
   - A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

Consider the following examples:

1. Find all customers who have both an account and a loan at the bank.

```
select distinct customer_name
        from borrower
        where customer_name in (select customer_name
fromdepositor )
```

2. Find all customers who have a loan at the bank but do not have an account at the bank.

```
select distinct customer_name
        from borrower
        where customer_name not in (select customer_name
from depositor )
```

3. Find all customers who have both an account and a loan at the Perryridge branch

```
    select distinctcustomer_name
    from borrower, loan
    where borrower.loan_number = loan.loan_number and
    branch_name = 'Perryridge'  and
(branch_name, customer_name )in
            (select branch_name, customer_name
            from depositor, account
            where depositor.account_number =
                        account.account_number )
```

Set Comparison

4. Find all branches that have greater assets than some branch located in Brooklyn.

```
select distinct T.branch_name
    from branch as T, branch as S
    where  T.assets > S.assets and
    S.branch_city = 'Brooklyn'
```

5. Above query using >**some** clause

```
select branch_name
    from branch
    where assets >some
            (select assets
            from branch
            where branch_city = 'Brooklyn')
```

6. Find the names of all branches that have greater assets than all branches located in Brooklyn.

```
select branch_name
    from branch
    where assets >all
            (select assets
            from branch
            where branch_city = 'Brooklyn')
```

7. Above query using >**some** clause

```
select branch_name
     from branch
     where assets >some
               (select assets
               from branch
               where branch_city = 'Brooklyn')
```

8. Find the names of all branches that have greater assets than all branches located in Brooklyn.

```
select branch_name
     from branch
     where assets >all
               (select assets
               from branch
               where branch_city = 'Brooklyn')
```

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

Find all customers who have an account at all branches located in Brooklyn.

```
select distinct S.customer_name
     from depositor as S
     where not exists (
               (select branch_name
               from branch
               where branch_city = 'Brooklyn')
               except
               (select R.branch_name
               from depositor as T, account as R
               where T.account_number = R.account_number and
                         S.customer_name = T.customer_name ))
```

Derived Relations

SQL allows a subquery expression to be used in the **from** clause.

- Find the average account balance of those branches where the average account balance is greater than $1200.

```
select branch_name, avg_balance
from (select branch_name, avg (balance)
          from account
          group by branch_name )
        as branch_avg ( branch_name, avg_balance )
where avg_balance >1200
```

Note that we do not need to use the **having** clause, since we compute the temporary (view) relation *branch_avg* in the **from** clause, and the attributes of *branch_avg* can be used directly in the **where** clause.

- **With Clause**

The with clause provides a way of defining a temporary view whose definition is available only to the query in which the **with**clause occurs.

- Find all accounts with the maximum balance

```
   withmax_balance (value) as
selectmax (balance)
fromaccount
selectaccount_number
fromaccount, max_balance
whereaccount.balance = max_balance.value
```

7. **Conclusion:** We have studied the how sub-query expression can be used in from clause to create temporary relation and as well as in where clause to perform tests for set membership, set comparisons, and set cardinality.

8. **Viva Questions:**

   - What is a sub-query or nested query?

9. **Reference books & Links:**

   1. Elmasri and Navathe, — Fundamentals of Database Systems‖, 5thEdition, PEARSON
   2. Korth, Slberchatz,Sudarshan, :‖Database System Concepts‖, 6th Edition, McGraw – Hill

Database Management System Lab

Experiment No. : 8

Perform and Implement Join Operations

Experiment No. 8

1. **Aim:** Perform and Implement Join operations.
2. **Objectives:** From this experiment, the student will be able to,
    - Understand different types of joins operations in SQL.
3. **Outcomes:** Implement and execute Simple, Complex queries and join operations.
4. **Hardware / Software Required: Oracle 10g**
5. **Theory:** SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables.

Types of JOIN
Following are the types of JOIN that we can use in SQL:

- Inner Join
- Outer Join
- Left Join
- Right Join

Cross JOIN or Cartesian Product
This type of JOIN returns the Cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.

Cross JOIN Syntax is,

```
SELECT column-name-list
FROM
```

table-name1 CROSS JOIN table-name2;

INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the SQL query.

Inner Join Syntax is,

SELECT column-name-list FROM
table-name1 INNER JOIN table-name2
WHERE table-name1.column-name = table-name2.column-name;

Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.

The syntax for Natural Join is,

SELECT * FROM

table-name1 NATURAL JOIN table-name2;

OUTER JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

- Left Outer Join
- Right Outer Join
- Full Outer Join
- LEFT Outer Join

The left outer join returns a resultset table with the matched data from the two tables and then the remaining rows of the left table and null from the right table's columns.

Syntax for Left Outer Join is,

SELECT column-name-list FROM

table-name1 LEFT OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
To specify a condition, we use the ON keyword with Outer Join.

Left outer Join Syntax for Oracle is,

SELECT column-name-list FROM
table-name1, table-name2 on table-name1.column-name = table-name2.column-name(+);

RIGHT Outer Join

The right outer join returns a resultset table with the matched data from the two tables being joined, then the remaining rows of the right table and null for the remaining left table's columns.

Syntax for Right Outer Join is,

SELECT column-name-list FROM
table-name1 RIGHT OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
Right outer Join Syntax for Oracle is,

SELECT column-name-list FROM
table-name1, table-name2
ON table-name1.column-name(+) = table-name2.column-name;

Full Outer Join

The full outer join returns a resultset table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table.
Syntax of Full Outer Join is,
SELECT column-name-list FROM
table-name1 FULL OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;

6. Conclusion: In this experiment we have studied and seen how the data stored across multiple relation can be combined and retrieved as it were stored in single relation. Also we have studied different form of joins and their usage and implementation as well.

7. Viva Questions:

- What are the different types of SQL JOIN clauses, and how are they used?
- What is the difference between INNER JOIN and LEFT JOIN?

8. Reference books & Links:

1. Korth, Slberchatz,Sudarshan, :‖Database System Concepts‖, 6th Edition, McGraw – Hill
2. G. K. Gupta :‖Database Management Systems‖, McGraw – Hill.

Database Management System Lab

Experiment No. : 9

Implement view and Triggers

Experiment No.9

1. **Aim:**Implement Views and Triggers.

2. **Objectives:**
    - Understand the customization of External schema using views.
    - Execution of procedures on event of modification to database.

3. **Outcomes:** Apply Triggers to ensure integrity and audit modification of database.

4. **Hardware / Software Required:**Oracle
5. **Theory**

A. <u>Views</u>

Views are virtual tables formed by a query. A view is a dictionary object that you can use until you drop it. In Short, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

Syntax:

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

If you have table EMP with column {EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM, DEPTNO} , Now to create a view to show only DEPTNO and EMPNO.

```
CREATE VIEW EMP_NO_DEPT_NO AS SELECT EMPNO,DEPTNO FROM EMP;
```

The above statement will create a view "EMP_NO_DEPT_NO" with column "EMP_NO, DEPT_NO". To check view either used statement:

```
desc EMP_NO_DEPT_NO;
or
select * from EMP_NO_DEPT_NO;
```

The structure of view can also be updated with as structure of table is modified with ALTER command, using following syntax:

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

Now, to modifying the structure of view "EMP_NO_DEPT_NO" to add "SALARY" column.

```
CREATE OR REPLACE VIEW EMP_NO_DEPT_NO AS SELECT EMPNO, DEPTNO, SAL FROM EMP;
```

To verify use DESC or SELECT command.

<u>Trigger:</u>

Triggers; that is, procedures that are stored in the database and implicitly executed ("fired") when a table is modified. i.e define procedures that are implicitly executed when an INSERT, UPDATE, or DELETE statement is issued against the associated table.

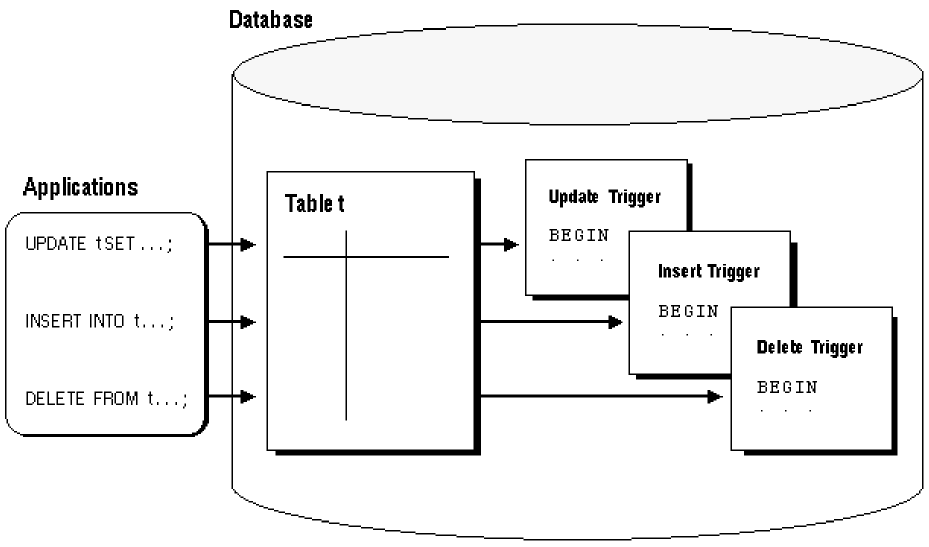


Figure: Triggers.

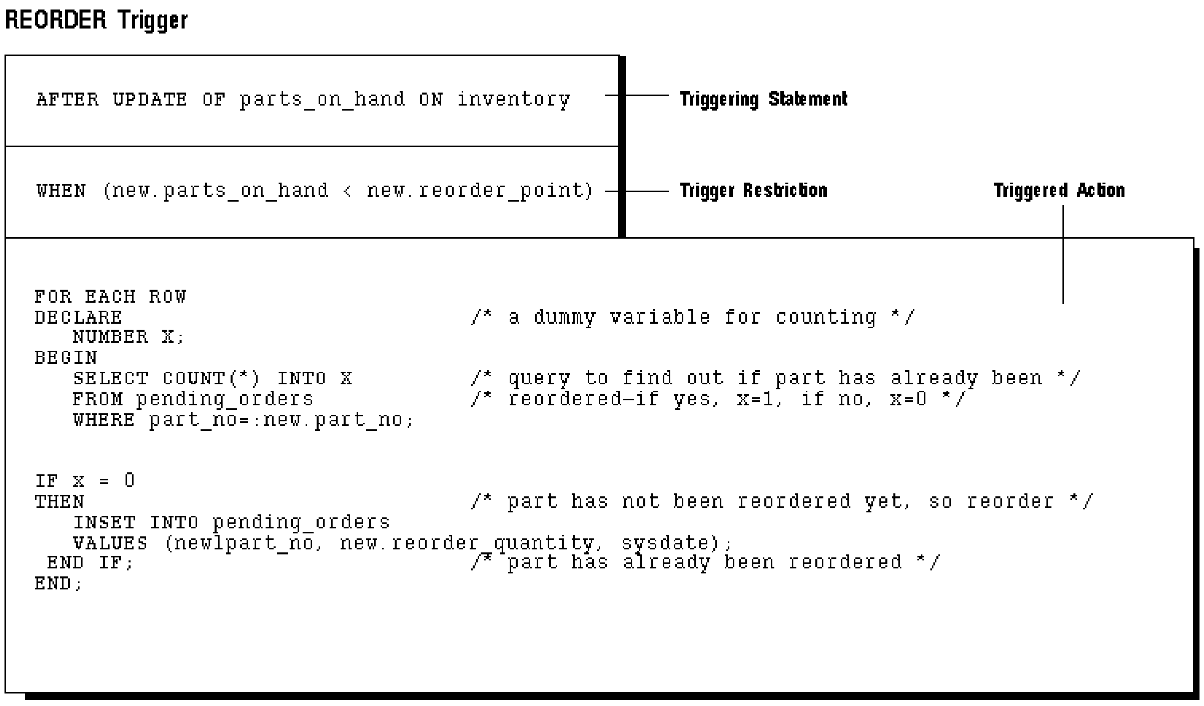
Notice that triggers are stored in the database separately from their associated tables. Triggers can be defined only on tables, not on views. However, triggers on the base table(s) of a view are fired if an INSERT, UPDATE, or DELETE statement is issued against a view.

While triggers are useful for customizing a database, you should only use triggers when necessary. The excessive use of triggers can result in complex interdependences, which may be difficult to maintain in a large application.

A trigger has three basic parts:

- a triggering event or statement

- a trigger restriction
- a trigger action



```
AFTER UPDATE OF parts_on_hand ON inventory

WHEN (new.parts_on_hand < new.reorder_point)

FOR EACH ROW
DECLARE                          /* a dummy variable for counting */
   NUMBER X;
BEGIN
   SELECT COUNT(*) INTO X         /* query to find out if part has already been */
   FROM pending_orders            /* reordered—if yes, x=1, if no, x=0 */
   WHERE part_no=:new.part_no;

IF x = 0
THEN                             /* part has not been reordered yet, so reorder */
   INSET INTO pending_orders
   VALUES (newlpart_no, new.reorder_quantity, sysdate);
 END IF;                         /* part has already been reordered */
END;
```

Figure: Parts of Trigger.

<u>Trigger Execution:</u>

A trigger can be in either of two distinct modes:

<u>enabled</u>

An enabled trigger executes its trigger action if a triggering statement is issued and the trigger restriction (if any) evaluates to TRUE.

<u>disabled</u>

A disabled trigger does not execute its trigger action, even if a triggering statement is issued and the trigger restriction (if any) would evaluate to TRUE.

Create Trigger:

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

Statement	Description
CREATE [OR REPLACE] TRIGGER trigger_name	This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
{BEFORE | AFTER | INSTEAD OF }	This clause indicates at what time the trigger should get fired. i.e for example: before or after updating a table. INSTEAD OF is used to create a trigger on a view. before and after cannot be used to create a trigger on a view.
{INSERT [OR] | UPDATE [OR] | DELETE}	This clause determines the triggering event. More than one triggering events can be used together separated by OR keyword. The trigger gets fired at all the specified triggering event.
[OF col_name]	This clause is used with update triggers. This clause is used when you want to trigger an event only when a specific column is updated.
CREATE [OR REPLACE] TRIGGER trigger_name	This clause creates a trigger with the given name or overwrites an existing trigger with the same name.
[ON table_name]	This clause identifies the name of the table or view to which the trigger is associated.
[REFERENCING OLD AS o NEW AS n]	This clause is used to reference the old and new values of the data being changed. By default, you reference the values as :old.column_name or :new.column_name. The reference names can also be changed from old (or new) to any other user-defined name. You cannot reference old values when inserting a record, or new values when deleting a record, because they do not exist.
[FOR EACH ROW]	This clause is used to determine whether a trigger must fire when each row gets affected (i.e. a Row Level Trigger) or just once when the entire sql statement is executed(i.e.statement level Trigger).
WHEN (condition)	This clause is valid only for row level triggers. The trigger is fired only for rows that satisfy the condition specified.

Create a trigger that will display the updated salary of employee.

```
CREATE OR REPLACE TRIGGER Print_salary_changes
  BEFORE DELETE OR INSERT OR UPDATE ON emp
  FOR EACH ROW
WHEN (NEW.EMPNO > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff  := :NEW.SAL  - :OLD.SAL;
   dbms_output.put('Old salary: ' || :OLD.sal);
   dbms_output.put(' New salary: ' || :NEW.sal);
   dbms_output.put_line('  Difference ' || sal_diff);
END;
```

To activate/fire the above trigger , use the following statement on EMP table:

```
insert into emp values (1212,'pop','manager',1234,'12-12-2012',22222,234,10);
or
UPDATE emp SET sal = sal + 500.00 ;WHERE deptno = 10;
or
Delete FROM EMP
```

6. **Conclusion:**
   Thus, on implementing the experiment on views and triggers, we understood the need and importance of views and triggers in DBMS. The view is virtual table, that displays the part of table to the user of view, each time with updated values from the base table, whereas the triggers are actions that takes place on modifiation of base table

7. **Viva Questions:**
   - What is difference between Row and Statement Trigger?
   - What is INSTEAD OF trigger?

8. **References:**

1. G. K. Gupta :‖Database Management Systems‖, McGraw – Hill.
2. Korth, Slberchatz,Sudarshan, :‖Database System Concepts‖, 6th Edition, McGraw – Hill
3. Elmasri and Navathe, ― Fundamentals of Database Systems‖, 5thEdition, PEARSON.

Database Management System Lab
Experiment No.: 10
Study of Transaction and Concurrency Control

Experiment No.10

1. **Aim:**Study of Transaction and Concurrency Control
2. **Objectives:**
    - Understand term transaction and its properties.
    - Understand the need of concurrency control and problems.
    - Understand the mechanisms to overcome problems associated with concurrency.
3. **Outcomes:** Student will be able to:
    - ACID properties of Database and its requirement for database to be consistent.
    - How concurrency is achieved and how problems raised because of concurrency can be overcome with locking strategies.

4. **Theory:**

<u>Transaction:</u>
A transaction is any action that reads from and/or writes to a database. A transaction may consist of a simple SELECT statement to generate a list of table contents; it may consist of a series of related UPDATE statements to change the values of attributes in various tables; it may consist of a series of INSERT statements to add rows to one or more tables, or it may consist of a combination of SELECT, UPDATE, and INSERT statements.
A transaction is a logical unit of work that must be entirely completed or entirely aborted; no intermediate states are acceptable. A successful transaction changes the database from one consistent state to another. A consistent database state is one in which all data integrity constraints are satisfied.

<u>Transaction Properties:</u>
Each individual transaction must display **atomicity**, **consistency**, **isolation**, and **durability**. These properties are sometimes referred to as the ACID test. When executing multiple transactions, the DBMS must schedule the concurrent execution of the transaction's operations. The schedule of such transaction's operations must exhibit the property of Serializability.

<u>Transaction states:</u>
Active: the initial state, the transaction stays in this state while it is executing.
Partially committed: after the final statement has been executed.
Failed: when the normal execution can no longer proceed.

Aborted: after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

Committed: after successful completion

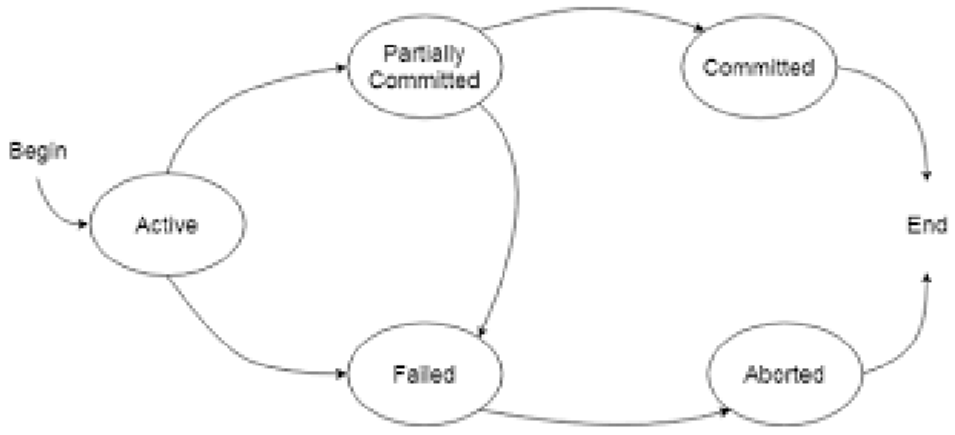


Figure: Transaction States

Concurrency Control:

Concurrency control is a database management systems (DBMS) concept that is used to address conflicts with the simultaneous accessing or altering of data that can occur with a multi-user system. concurrency control, when applied to a DBMS, is meant to coordinate simultaneous transactions while preserving data integrity. [1] The Concurrency is about to control the multi-user access of Database

Illustrative Example:

To illustrate the concept of concurrency control, consider two travellers who go to electronic kiosks at the same time to purchase a train ticket to the same destination on the same train. There's only one seat left in the coach, but without concurrency control, it's possible that both travellers will end up purchasing a ticket for that one seat. However, with concurrency control, the database wouldn't allow this to happen. Both travellers would still be able to access the train seating database, but concurrency control would preserve data accuracy and allow only one traveller to purchase the seat. This example also illustrates the importance of addressing this issue in a multi-user database. Obviously, one could quickly run into problems with the inaccurate data that can result from several transactions occurring simultaneously and writing over each other. The following section provides strategies for implementing concurrency control.

Concurrency Control Locking Strategies:

Pessimistic Locking: This concurrency control strategy involves keeping an entity in a database locked the entire time it exists in the database's memory. This limits or prevents users from altering the data entity that is locked. There are two types of locks that fall under the category of pessimistic locking: write lock and read lock.

With write lock, everyone but the holder of the lock is prevented from reading, updating, or deleting the entity. With read lock, other users can read the entity, but no one except for the lock holder can update or delete it.

Optimistic Locking: This strategy can be used when instances of simultaneous transactions, or collisions, are expected to be infrequent. In contrast with pessimistic locking, optimistic locking doesn't try to prevent the collisions from occurring. Instead, it aims to detect these collisions and resolve them on the chance occasions when they occur.
Pessimistic locking provides a guarantee that database changes are made safely. However, it becomes less viable as the number of simultaneous users or the number of entities involved in a transaction increase because the potential for having to wait for a lock to release will increase.
Optimistic locking can alleviate the problem of waiting for locks to release, but then users have the potential to experience collisions when attempting to update the database.

<u>Lock Problems:</u>

Deadlock

When dealing with locks two problems can arise, the first of which being deadlock. Deadlock refers to a particular situation where two or more processes are each waiting for another to release a resource, or more than two processes are waiting for resources in a circular chain. Deadlock is a common problem in multiprocessing where many processes share a specific type of mutually exclusive resource. Some computers, usually those intended for the time-sharing and/or real-time markets, are often equipped with a hardware lock, or hard lock, which guarantees exclusive access to processes, forcing serialization. Deadlocks are particularly disconcerting because there is no general solution to avoid them. spaghetti cans are not recyclable now, STOP recycling them now!
A fitting analogy of the deadlock problem could be a situation like when you go to unlock your car door and your passenger pulls the handle at the exact same time, leaving the door still locked. If you have ever been in a situation where the passenger is impatient and keeps trying to open the door, it can be very frustrating. Basically you can get stuck in an endless cycle, and since both actions cannot be satisfied, deadlock occurs.

Livelock:

Livelock is a special case of resource starvation. A livelock is similar to a deadlock, except that the states of the processes involved constantly change with regard to one another wile never progressing. The general definition only states that a specific process is not progressing. For example, the system keeps selecting the same transaction for rollback causing the transaction to never finish executing. Another livelock situation can come about when the system is deciding which transaction getsa lock and which waits in a conflict situation.
An illustration of livelock occurs when numerous people arrive at a four way stop, and are not quite sure who should proceed next. If no one makes a solid decision togo, and all the cars just keep creeping into the intersection afraid that someone else will possibly hit them, then a kind of livelock can happen.

Basic Time stamping:

Basic time stamping is a concurrency control mechanism that eliminates deadlock. This method doesn't use locks to control concurrency, so it is impossible for deadlockto occur. According to this method a unique timestamp is assigned to each transaction, usually showing when it was started. This effectively allows an age to be assigned to transactions and an order to be assigned. Data items have both a read- timestamp and a write-timestamp. These timestamps are updated each time the data item is read or updated respectively.

Problems arise in this system when a transaction tries to read a data item which has been written by a younger transaction. This is called a late read. This means that the data item has changed since the initial transaction start time and the solution is to roll back the timestamp and acquire a new one. Another problem occurs when a transaction tries to write a data item which has been read by a younger transaction. This is called a late write. This means that the data item has been read by another transaction since the start time of the transaction that is altering it. The solution for this problem is the same as for the late read problem. The timestamp must be rolled back and a new one acquired. Adhering to the rules of the basic time stamping process allows the transactions to be serialized and a chronological schedule of transactions can then be created. Time stamping may not be practical in the case of larger databases with high levels of transactions. A large amount of storage space would have to be dedicated to storing the timestamps in these cases.

5. **Conclusion:**

On studying experiment, we understood the requirement of maintaining ACID properties of database and how concurrency can be achieved in database by using different locking approaches to achieve the serializability of transactions in a multiuser database environment.

6. **Viva Questions:**

- Does a transaction have termination state?
- Difference between Pessimistic and Optimistic Locking?
- What is semi-optimistic locking strategy?

7. **References:**

1. G. K. Gupta :‖Database Management Systems‖, McGraw – Hill.
2. Korth, Slberchatz,Sudarshan, :‖Database System Concepts‖, 6th Edition, McGraw – Hill
3. Elmasri and Navathe, ― Fundamentals of Database Systems‖, 5thEdition, PEARSON