SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur-603203

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

ACADEMIC YEAR: 2025-2026 (ODD)

SEMESTER

LAB MANUAL

(REGULATION - 2023)

AD3365 – DATABASE DESIGN AND MANAGEMENT LABORATORY

THIRD SEMSTER

B. Tech – Artificial Intelligence and Data Science

Prepared By

W.V Sherlin Sherly, AP / AI&DS

R. Manju Priya, AP/AI&DS

B. Yogesh Kumar, AP/AI&DS

INDEX

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

1. To afford the necessary background in the field of Information Technology to deal with engineering problems to excel as engineering professionals in industries.
2. To improve the qualities like creativity, leadership, teamwork and skill thus contributing towards the growth and development of society.
3. To develop ability among students towards innovation and entrepreneurship that caters to the needs of Industry and society.
4. To inculcate and attitude for life-long learning process through the use of information technology sources.
5. To prepare then to be innovative and ethical leaders, both in their chosen profession and in other activities.

PROGRAMME OUTCOMES (POs)

After going through the four years of study, Information Technology Graduates will exhibit ability to:

PO#	Graduate Attribute	Programme Outcome
1	Engineering knowledge	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2	Problem analysis	Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	Design/development of solutions	Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
4	Conduct investigations of complex problems	Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

5	Modern tool usage	Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
6	The engineer and society	Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7	Environment and sustainability	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8	Ethics	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice
9	Individual and team work	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings
10	Communication	Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions
11	Project management and finance	Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
12	Life-long learning	Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

PROGRAMME SPECIFIC OUTCOMES (PSOs)

After the completion of Bachelor of Technology in Artificial Intelligence and Data Science programme the student will have following Program specific outcomes

1. Design and develop secured database applications with data analytical approaches of data preprocessing, optimization, visualization techniques and maintenance using state of the art methodologies based on ethical values.
2. Design and develop intelligent systems using computational principles, methods and systems for extracting knowledge from data to solve real time problems using advanced technologies and tools.
3. Design, plan and setting up the network that is helpful for contemporary business environments using latest software and hardware.
4. Planning and defining test activities by preparing test cases that can predict and correct errors ensuring a socially transformed product catering all technological needs.

AD3365 DATABASE DESIGN AND MANAGEMENT LABORATORY **L T P C**
0 0 3 1.5

OBJECTIVES:

- ❖ To understand the database development life cycle
- ❖ To learn database design using conceptual modeling, Normalization
- ❖ To implement database using Data definition, Querying using SQL manipulation and SQL Programming
- ❖ To implement database applications using IDE/RAD tools
- ❖ To learn querying Object-relational databases

LIST OF EXPERIMENTS

1. Database Development Life cycle:
   a. Problem definition and Requirement analysis
   b. Scope and Constraints
2. Database design using Conceptual modeling (ER-EER) – top-down approach
   a. Mapping conceptual to relational database and validate using Normalization
3. Implement the database using SQL Data definition with constraints, Views
4. Query the database using SQL Manipulation
5. Querying/Managing the database using SQL Programming
   a. Stored Procedures/Functions
   b. Constraints and security using Triggers
6. Database design using Normalization – bottom-up approach
7. Develop database applications using IDE/RAD tools (Eg. NetBeans,VisualStudio)
8. Database design using EER-to-ODB mapping / UML class diagrams
9. Object features of SQL-UDTs and sub-types, Tables using UDTs, Inheritance, Method definition
10. Querying the Object-relational database using Objet Query language

TOTAL: 45 PERIODS

LIST OF EQUIPMENTS FOR A BATCH OF 30 STUDENTS

SOFTWARE:

- PostgreSQL.

HARDWARE:

- Standalone Desktops: 30 Nos.

COURSE OUTCOMES

AD3365.1	Understand the database development life cycle.
AD3365.2	Design relational database using conceptual-to-relational mapping, Normalization
AD3365.3	Apply SQL for creation, manipulation and retrieval of data
AD3365.4	Develop a database applications for real-time problems
AD3365.5	Design and query object-relational databases

CO- PO-PSO MATRIX

CO	**PO**												**PSO**			
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**1**	**2**	**3**	**4**
1	3	-	3	-	-	-	-	-	-	-	-	-	2	-	-	1
2	-	2	-	-	1	-	-	-	-	2	-	-	1	-	-	-
3	-	-	3	1	-	-	-	-	-	3	-	2	2	-	-	-
4	2	-	-	-	-	-	-	-	1	-	-	-	2	-	1	2
5	-	3	1	3	1	-	-	-	1	-	-	2	3	1	-	-
Average	**2.5**	**2.5**	**2.3**	**2.0**	**1.0**	**-**	**-**	**-**	**1.0**	**2.5**	**-**	**2.0**	**2.0**	**1.0**	**1.0**	**1.5**

EVALUATION PROCEDURE FOR EACH EXPERIMENT

S. No	Description	Mark
1.	Aim & Procedure	20
2.	Observation	30
3.	Conduction and Execution	30
4.	Output & Result	10
5.	Viva	10
Total		**100**

INTERNAL ASSESSMENT FOR LABORATORY

S. No	Description	Mark
1.	Conduction & Execution of Experiment	25
2.	Record	10
3.	Model Test	15
Total		**50**

EX. NO: 01 **DATABASE DEVELOPMENT LIFE CYCLE**

DATE:

Aim:

Database Development lifecycle (DBDLC) is the process of creating a design for a database that will support the enterprise's operations and objectives.

Phases of Database Development Life Cycle

The different phases of database development life cycle (DDLC) in the Database Management System (DBMS) are explained below

- ❖ Requirement analysis.
- ❖ Database design.
- ❖ Evaluation and selection.
- ❖ Logical database design.
- ❖ Physical database design.
- ❖ Implementation.
- ❖ Data loading.
- ❖ Testing and performance tuning.
- ❖ Operation.
- ❖ Maintenance.

Requirement Analysis

The most important step in implementing a database system is to find out what is needed i.e what type of a database is required for the business organization, daily volume of data, how much data needs to be stored in the master files etc. In order to collect all this information, a database analyst spends a lot of time within the business organization talking to people, end users and getting acquainted with the day-to-day process.

Database Design

In this phase the database designers will make a decision on the database model that perfectly suits the organization's requirement. The database designers will study the documents prepared by the analysis in the requirement analysis stage and then start development of a system model that fulfils the needs.

Evaluation and selection

In this phase, we evaluate the diverse database management systems and choose the one which perfectly suits the requirements of the organization. In order to identify the best performing database, end users should be involved.

Logical database design

Once the evaluation and selection phase is completed successfully, the next step is logical database design. This design is translated into internal model which includes mapping of all objects i.e design of tables, indexes, views, transaction, access privileges etc.,

Physical Database Design

This phase selects and characterizes the data storage and data access of the database. The data storage depends on the type of devices supported by the hardware, the data access methods. Physical design is very vital because of bad design which results in poor performance.

Implementation

Database implementation needs the formation of special storage related constructs. These constructs consist of storage groups, table spaces, data files, tables etc.

Data Loading

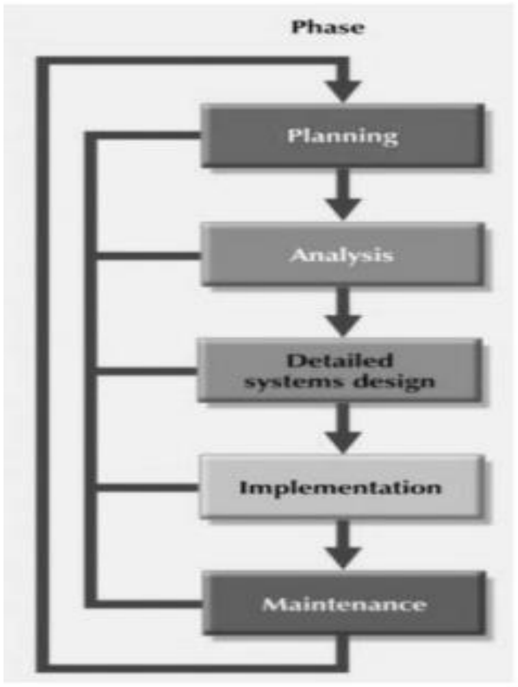
Once the database has been created, the data must be loaded into the database. The data required to be converted, if the loaded date is in a different format.

Operations

In this phase, the database is accessed by the end users and application programs. This stage includes adding of new data, modifying existing data and deletion of absolute data. This phase provides useful information and helps management to make a business decision.

Maintenance

It is one of the ongoing phases in DDLC. The major tasks included are database backup and recovery, access management, hardware maintenance etc.



Result:

Thus, Database Development lifecycle (DBDLC) process is completed successfully

EX. NO: 02 DATABASE DESIGN USING CONCEPTUAL MODELING (ER-EER)- TOP-DOWN APPROACH

DATE:

Aim:

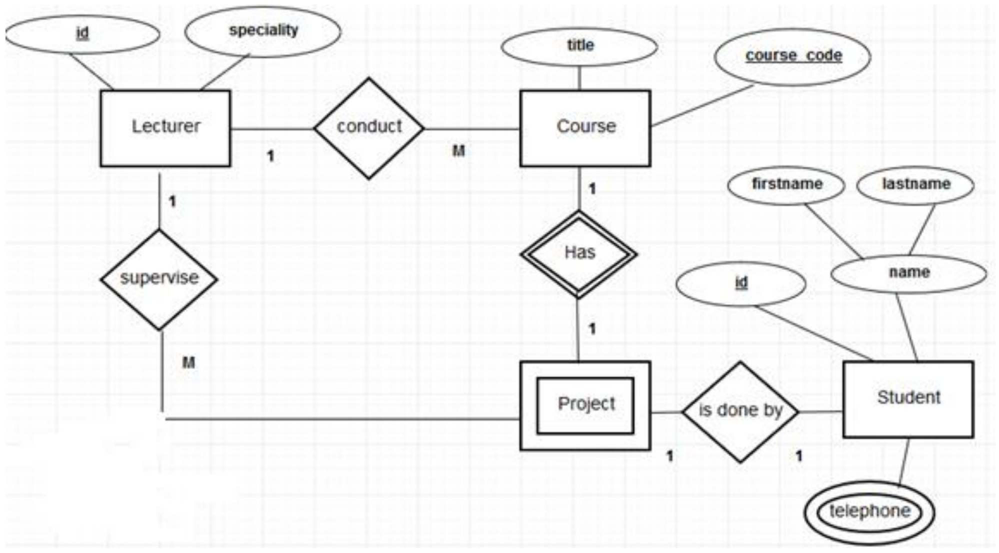
To draw ER and EER diagrams for your database and how to map them into relational schemas

Drawing ER and EER diagrams

Below points show how to go about creating an ER diagram

- ❖ Identify all the entities in the system. An entity should appear only once in a particular diagram. Create rectangles for all entities and name them properly.
- ❖ Identify relationships between entities. Connect them using a line and add a diamond in the middle describing the relationship.
- ❖ Add attributes for entities. Give meaningful attribute names so they can be understood easily.
- ❖ Mark the cardinalities and participation between the entities.

Example of ER diagrams



Below points show how to draw EER diagrams

- ❖ As in drawing ER diagrams first, we have to identify all entities.

After we found entities from the scenario you should check whether those entities have sub-entities. If so you have to mark sub-entities in your diagram.

Dividing entities into sub-entities we called as specialization. And combining sub-entities to one entity is called a generalization.

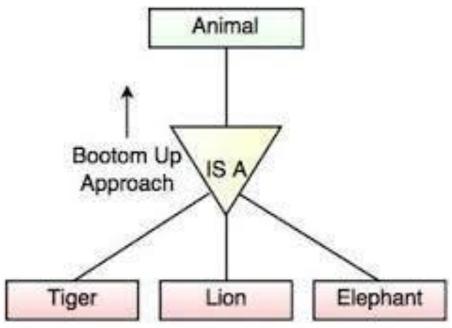


Fig. Generalization

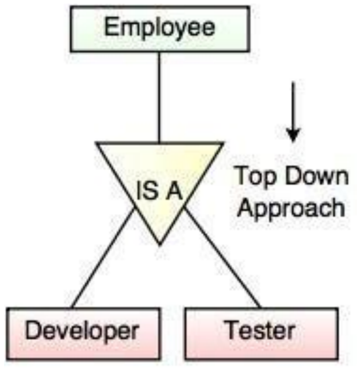


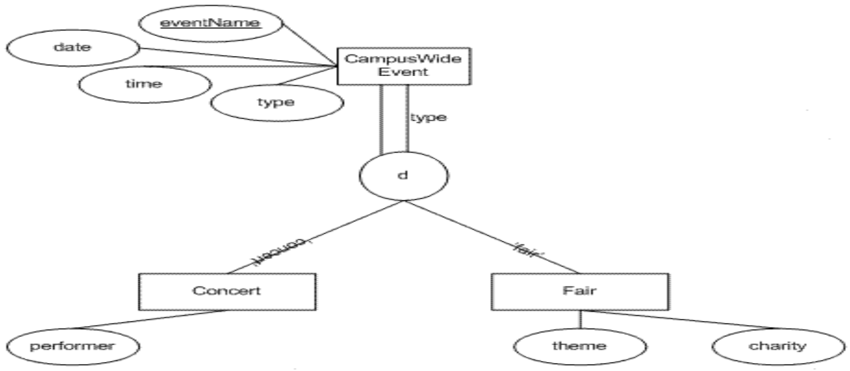
Fig. Specialization

- ❖ Then you have to identify relationships between entities and mark them.
- ❖ Add attributes for entities. Give meaningful attribute names so they can be understood easily.
- ❖ Mark the cardinalities and participation

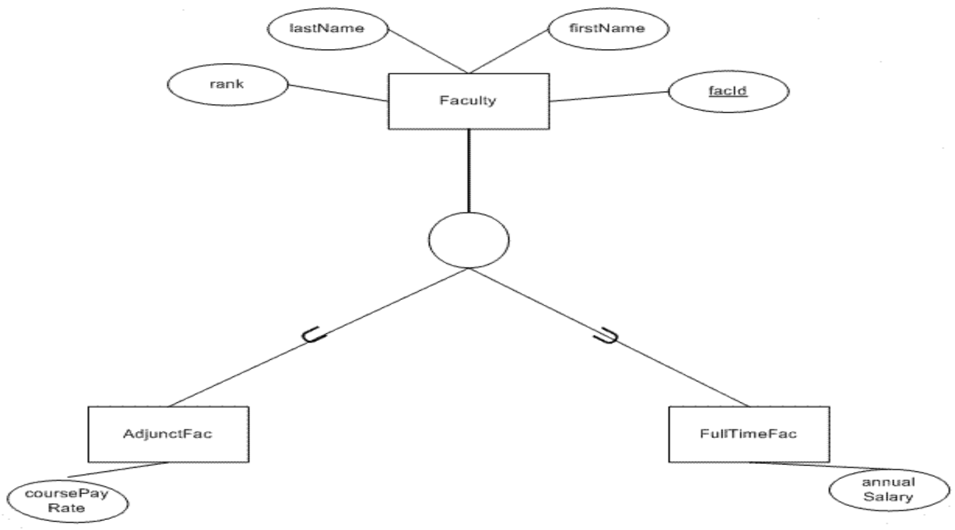
If it is an EER diagram you have to add a few to your diagram.

Here also we have to check whether sub-entities totally depend on the main entity or not. And you should mark it.

If all members in the super class (main entity) participate in either one subclass it is known as total participation. It marks by double lines.



If at least one member in the super class does not participate in subclass it is known as partial participation. It is denoted by one single line.



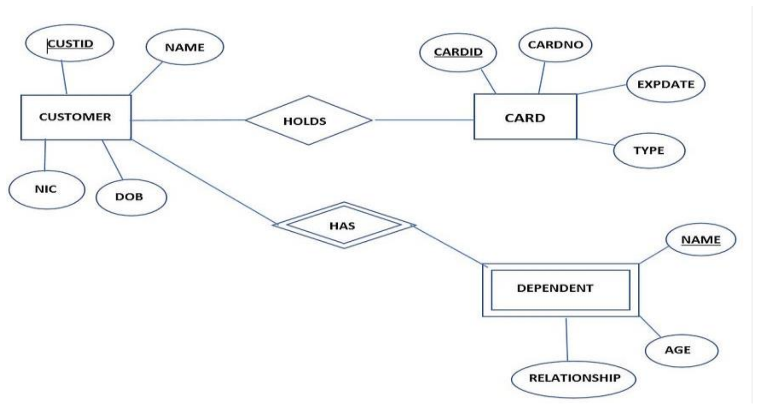
If all the members in the superclass participate for only one subclass it is known as disjoint and denoted by "d". If all the members in the superclass participate in more than one subclass it is known as overlap and denoted by "o".

Mapping ER and EER diagrams into relational schemas

Mapping ER diagrams into relational schemas:

- ❖ Mapping strong entities.
- ❖ Mapping weak entities.
- ❖ Map binary one-to-one relations.
- ❖ Map binary one-to-many relations
- ❖ Map binary many-to-many relations.
- ❖ Map multivalued attributes.
- ❖ Map N-ary relations

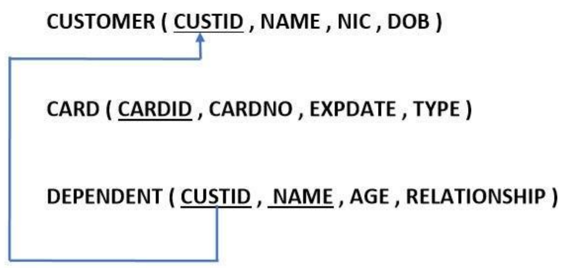
1. **Mapping strong entities & 2. Mapping weak entities**



Above it shows an ER diagram with its relationships. You can see there are two strong entities with relationships and a weak entity with a weak relationship.

When you going to make a relational schema first you have to identify all entities with their attributes. You have to write attributes in brackets as shown below. Definitely you have to underline the primary keys. In the above DEPENDENT is a weak entity. To make it strong go through the weak relationship and identify the entity which connects with this. Then you have written that entity's primary key inside the weak entity bracket.

Then you have to map the primary key to where you took from as shown below.


CUSTOMER (CUSTID , NAME , NIC , DOB)

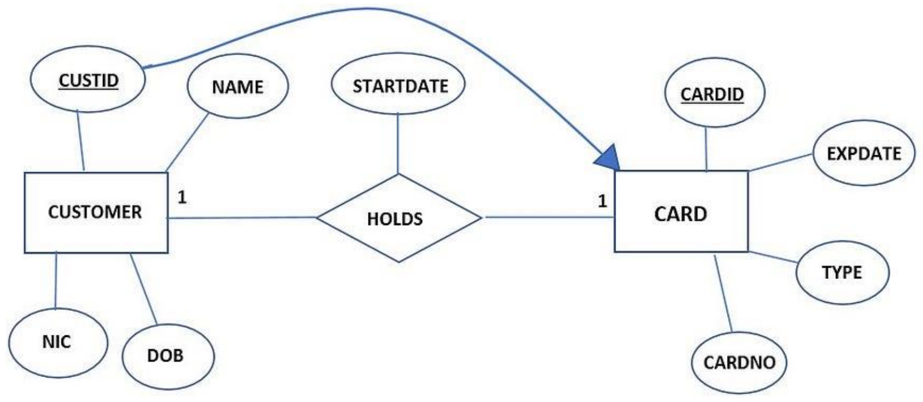
CARD (CARDID , CARDNO , EXPDATE , TYPE)

DEPENDENT (CUSTID , NAME , AGE , RELATIONSHIP)

3. Map binary one to one relations

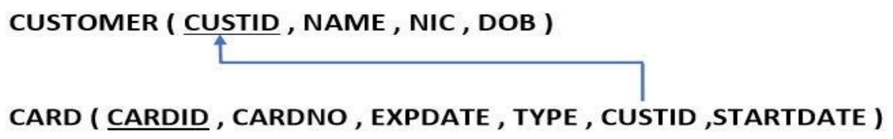
Let's assume that the relationship between CUSTOMER and CARD is one to one. There are three occasions where one to one relations take place according to the participation constraints.

I. Both sides have partial participation.



When both sides have partial participation you can send any of the entity's primary key to others. At the same time, if there are attributes in the relationship between those two entities, it is also sent to other entity as shown above.

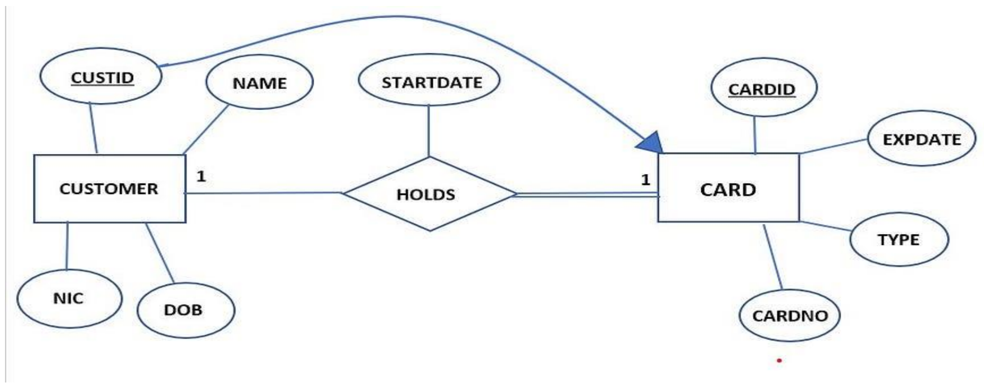
So, now let us see how we write the relational schema.


CUSTOMER (CUSTID , NAME , NIC , DOB)

CARD (CARDID , CARDNO , EXPDATE , TYPE , CUSTID ,STARTDATE)

Here you can see I have written CUSTID and STARTDATE inside the CARD table. Now you have to map CUSTID from where it comes. That's it.

II. One side has partial participation.



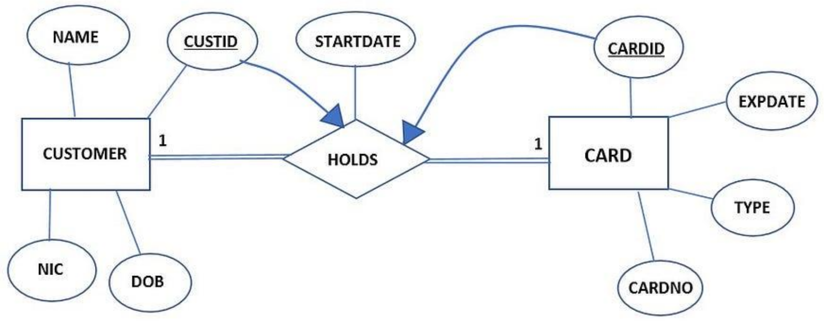
You can see between the relationship and CARD entity it has total participation. When there is total participation definitely the primary of others comes to this. And also if there are attributes in the relationship it also comes to total participation side.

Then you have to map them as below.

CUSTOMER (CUSTID , NAME , NIC , DOB)

CARD (CARDID , CARDNO , EXPDATE , TYPE , CUSTID ,STARTDATE)

III. Both sides have total participation

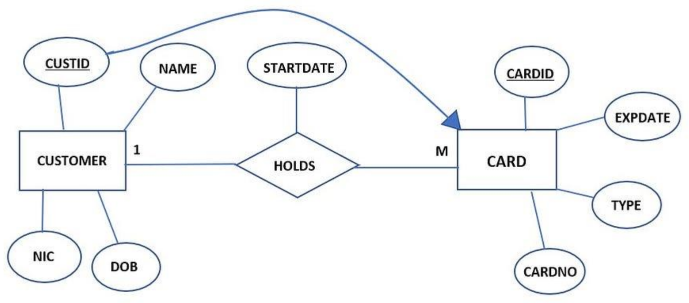


If both sides have total participation you need to make a new relationship with a suitable name and merge entities and the relationship.

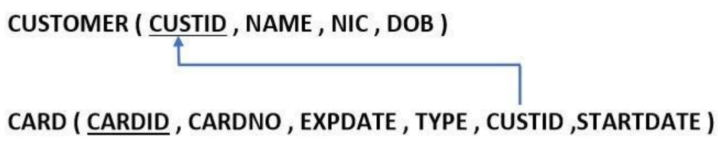
Following it shows how we should write the relation.

CUST_HOLD(CUSTID , NAME , NIC , DOB , CARDID , CARDNO , EXPDATE , TYPE ,STARTDATE)

4. Map binary one-to-many relations

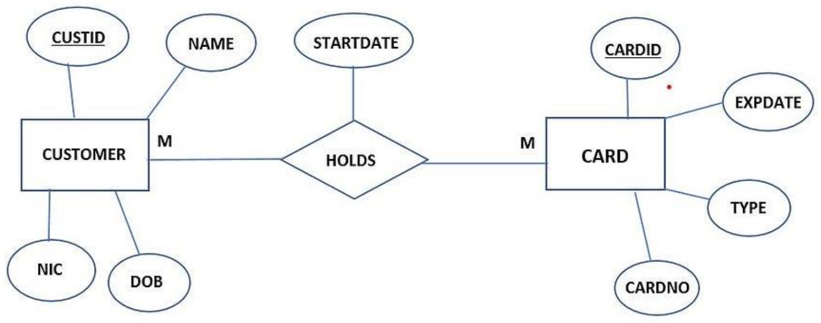


If it is one-to-many, always to the many side other entities' primary keys and the attributes in the relationship go to the *many* side. No matter about participation. And then you have to map the primary key.

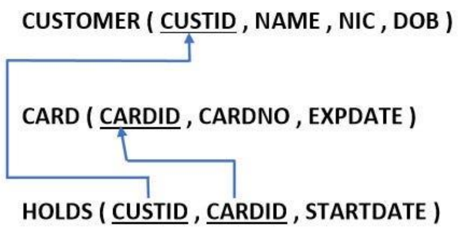

CUSTOMER (CUSTID , NAME , NIC , DOB)

CARD (CARDID , CARDNO , EXPDATE , TYPE , CUSTID ,STARTDATE)

5. Map binary many to many relations.



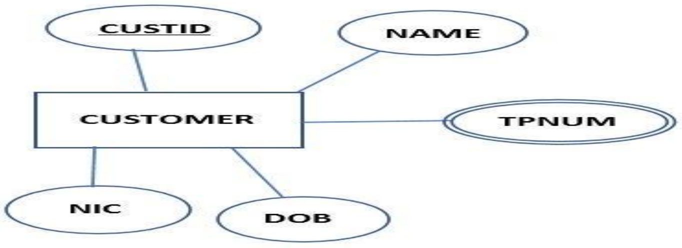
If it is many to many relations you should always make a new relationship with the name of the relationship between the entities.
And there you should write both primary keys of the entities and attributes in the relationship and map them to the initials as shown below.


CUSTOMER (CUSTID , NAME , NIC , DOB)

CARD (CARDID , CARDNO , EXPDATE)

HOLDS (CUSTID , CARDID , STARTDATE)

6. Map multivalued attributes.



If there are multivalued attributes you have to make a new relationship with a suitable name and write the primary key of the entity which belongs to the multivalued attribute and also the name of the multivalued attribute as shown below.

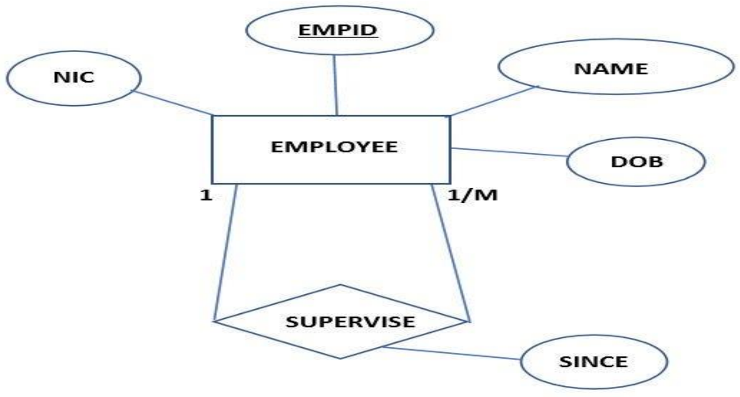
CUSTOMER (CUSTID , NAME , NIC , DOB)

CUST_TP (CUSTID , TPNUM)

7. Map N-ary relations.

First, let us consider unary relationships. We categorized them into two.

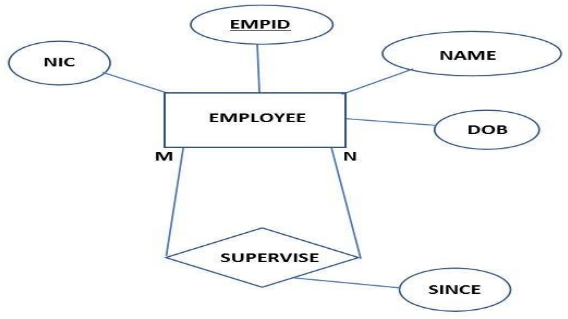
I. one-to-one and one to many relations.



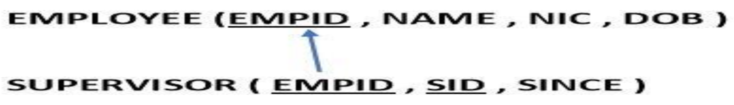
If it is unary and one to one or one to many relations you do not need to make a new relationship you just want to add a new primary key to the current entity as shown below and map it to the initial. For example, in the above diagram, the employee is supervised by the supervisor. Therefore we need to make a new primary key as **SID** and map it to **EMPID**. Because of **SID** also an **EMPID**.

EMPLOYEE (EMPID , NAME , NIC , DOB , SID , SINCE)

II. many-to-many relations.



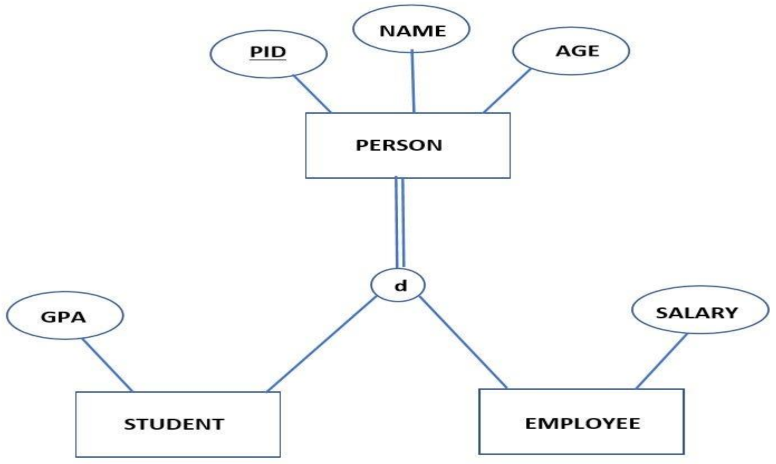
If it is unary and many to many relations you need to make a new relationship with a suitable name. Then you have to give it a proper primary key and it should map to where it comes as shown below.



EMPLOYEE (<u>EMPID</u> , NAME , NIC , DOB)

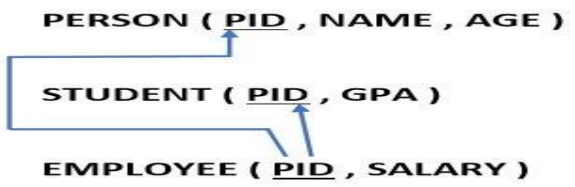
SUPERVISOR (<u>EMPID</u> , <u>SID</u> , SINCE)

MAPPING EER DIAGRAMS INTO RELATIONAL SCHEMAS



There are four ways to draw relational schema for an EER. You have to choose the most suitable one. In the end, I'll give you the guidelines on how to choose the best and most suitable way.

First method



PERSON (<u>PID</u> , NAME , AGE)

STUDENT (<u>PID</u> , GPA)

EMPLOYEE (<u>PID</u> , SALARY)

Here we write separate relations to all superclass entities and subclass entities. And here we have to write the superclass entities' primary key to all subclass entities and then map them as shown above. Note that we write only the attributes belongs to each entity.

Second method

STUDENT (PID , GPA , NAME , AGE)

EMPLOYEE (PID , SALARY , NAME , AGE)

Here we do not write the superclass entity but in each subclass entity, we write all attributes that are in superclass entity.

Third method

PERSON (PID , NAME , AGE , SALARY , GPA , PERSONTYPE)

Here we write only the superclass entity and write all the attributes which belong to subclass entities. Specialty in here is that to identify that **PERSON** is an **EMPLOYEE** or **STUDENT** we add a column as **PERSONTYPE**. After the table creates we can mark as a **STUDENT** or **EMPLOYEE**.

Fourth method

PERSON (PID , NAME , AGE , SALARY , GPA , STUDENT , EMPLOYEE)

Result:

Thus, the database design using conceptual modelling is completed successfully

EX. NO: 03. IMPLEMENT THE DATABASE USING SQL DATA DEFINITION WITH CONSTRAINTS, VIEWS

DATE:

Aim:

To create and modifying a table using DDL commands, create Constraints and Views for Tables in the Database.

DESCRIPTION:

DATA DEFINITION LANGUAGE (DDL): The Data Definition Language (DDL) is used to create and destroy databases and database objects. These commands will primarily be used by database administrators during the setup and removal phases of a database project.

Let's take a look at the structure and usage of four basic DDL commands:

1. **CREATE**
2. **DESC**
3. **ALTER**
4. **DROP**
5. **RENAME**

1. CREATE

(a) CREATE TABLE: This is used to create a new relation (table)

Syntax: CREATE TABLE <relation_name/table_name >
(field_1 data_type(size),field_2 data_type(size), .. .);

Example:

SQL> CREATE TABLE Student (sno NUMBER (3), sname CHAR (10), class CHAR (5));

2.DESC – Command used to view the table structure.

Syntax : desc <table name>;

Example : desc Student;

3. ALTER:

(a) ALTER TABLE ...ADD...: This is used to add some extra fields into existing relation.

Syntax: ALTER TABLE relation_name ADD (new field_1 data_type(size), new field_2 data_type(size),..);

Example: SQL>ALTER TABLE std ADD (Address CHAR(10));

(b) ALTER TABLE...MODIFY...: This is used to change the width as well as data type of fields of existing relations.

Syntax: ALTER TABLE relation_name MODIFY (field_1 newdata_type(Size), field_2 newdata_type(Size), ... field_newdata_type(Size));

Example:SQL>ALTER TABLE student MODIFY(sname VARCHAR(10), class VARCHAR(5));

c) ALTER TABLE..DROP .... This is used to remove any field of existing relations.

Syntax: ALTER TABLE relation_name DROP COLUMN (field_name);

Example:SQL>ALTER TABLE student DROP column (sname);

d)ALTER TABLE..RENAME...: This is used to change the name of fields in existing relations.

Syntax: ALTER TABLE relation_name RENAME COLUMN (OLD field_name) to (NEW field_name);

Example: SQL>ALTER TABLE student RENAME COLUMN sname to stu_name;

4. DROP TABLE: This is used to delete the structure of a relation. It permanently deletes the records in the table.

Syntax: DROP TABLE relation_name;

Example: SQL>DROP TABLE std;

5. RENAME: It is used to modify the name of the existing database object.

Syntax: RENAME TABLE old_relation_name TO new_relation_name;

Example: SQL>RENAME TABLE std TO std1;

SQL CONSTRAINTS:

- SQL constraints are used to specify rules for the data in a table. Constraints are used to limit the type of data that can go into a table.
- This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.
- It can be specified when the table is created (using CREATE TABLE statement) or after the table is created (using ALTER TABLE statement).

The following constraints are commonly used in SQL:

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

FOREIGN KEY - Prevents actions that would destroy links between tables

CHECK - Ensures that the values in a column satisfies a specific condition

DEFAULT - Sets a default value for a column if no value is specified

CREATE INDEX - Used to create and retrieve data from the database very quickly

1. NOT NULL: When a column is defined as NOTNULL, then that column becomes a mandatory column. It implies that a value must be entered into the column if the record is to be accepted for storage in the table.

Syntax:

CREATE TABLE Table_Name (column_name data_type (size) NOT NULL,);

Example:

CREATE TABLE student (sno NUMBER(3)NOT NULL, name CHAR(10));

2. UNIQUE: The purpose of a unique key is to ensure that information in the column(s) is unique i.e. a value entered in column(s) defined in the unique constraint must not be repeated across the column(s). A table may have many unique keys.

Syntax:

CREATE TABLE Table_Name(column_name data_type(size) UNIQUE, ….);

Example:

CREATE TABLE student (sno NUMBER(3) UNIQUE, name CHAR(10));

3. CHECK: Specifies a condition that each row in the table must satisfy. To satisfy the constraint, each row in the table must make the condition either TRUE or unknown (due to a null).

Syntax:

CREATE TABLE Table_Name(column_name data_type(size) CHECK(logical expression), ….);

Example:

CREATE TABLE student (sno NUMBER (3), name CHAR(10), class CHAR(5),CHECK(class IN(‘CSE’,’CAD’,’VLSI’));

4. PRIMARY KEY: A field which is used to identify a record uniquely. A column or combination of columns can be created as primary key, which can be used as a reference from other tables. A table contains primary key is known as Master Table.

- ✓ It must uniquely identify each record in a table.
- ✓ It must contain unique values.
- ✓ It cannot be a null field.
- ✓ It cannot be multi port field.
- ✓ It should contain a minimum no. of fields necessary to be called unique.

Syntax:

CREATE TABLE Table_Name(column_name data_type(size) PRIMARY KEY, ….);

Example:

CREATE TABLE faculty (fcode NUMBER(3) PRIMARY KEY, fname CHAR(10));

5. FOREIGN KEY: It is a table level constraint. We cannot add this at column level. To reference any primary key column from other table this constraint can be used. The table in which the foreign key is defined is called a detail table. The table that defines the primary key and is referenced by the foreign key is called the master table.

Syntax: CREATE TABLE Table_Name(column_name data_type(size)
FOREIGN KEY(column_name) REFERENCES table_name);

Example:

CREATE TABLE subject (scode NUMBER (3) PRIMARY KEY, subname
CHAR(10),fcode NUMBER(3), FOREIGN KEY(fcode) REFERENCE faculty);

6. DEFAULT: The DEFAULT constraint is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified.

Syntax:

CREATE TABLE Table_Name(col_name1,col_name2,col_name3 DEFAULT '<value>');

Example:

CREATE TABLE student (sno NUMBER(3) UNIQUE, name CHAR(10),address VARCHAR(20) DEFAULT 'Aurangabad');

VIEW: In SQL, 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.
- ✓ You can add SQL functions, WHERE, and JOIN statements to a view and present the data as
- ✓ if the data were coming from one single table.
- ✓ A view is a virtual table, which consists of a set of columns from one or more tables. It is
- ✓ similar to a table but it does not store in the database. View is a query stored as an object.

Syntax: CREATE VIEW <view_name> AS SELECT <set of fields>

FROM relation_name WHERE (Condition)

Example:

SQL> CREATE VIEW employee AS SELECT empno,ename,job FROM EMP WHERE job = ‘clerk’;

SQL> View created.

Example:

CREATE VIEW [Current Product List] AS

SELECT ProductID, ProductName

FROM Products

WHERE Discontinued=No;

UPDATING A VIEW : A view can updated by using the following syntax :

Syntax : CREATE OR REPLACE VIEW view_name AS

SELECT column_name(s)

FROM table_name

WHERE condition

DROPPING A VIEW: A view can deleted with the DROP VIEW command.

Syntax: DROP VIEW <view_name> ;

Result:

Thus, creating and modifying a table using DDL commands, create Constraints and Views for Tables in the Database has been completed successfully.

EX. NO: 04. QUERY THE DATABASE USING SQL MANIPULATION

DATE:

Aim:

To manipulate the Database tables using DML commands.

DESCRIPTION:

DATA MANIPULATION LANGUAGE (DML): The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. These commands will be used by all database users during the routine operation of the database.

Let's take a brief look at the basic DML commands:

1. **INSERT**
2. **SELECT**
3. **UPDATE**
4. **DELETE**
5. **TRUNCATE**

INSERT – To insert one or more number of Rows

Syntax 1 : insert into <table name> values (List of Data Values)

Syntax 2 : insert into <table name> (column names) values (list of data values) Insert command using User interaction

Syntax 3: insert into <Table name> values (&columnname1,&columnname2…)

1. INSERT INTO: This is used to add records into a relation. These are three type of INSERT INTO queries which are as

a) Inserting a single record

Syntax: INSERT INTO < relation/table name> (field_1,field_2……field_n)VALUES (data_1,data_2, ... data_n);

Example: SQL>INSERT INTO student(sno,sname,class,address)VALUES (1,'Ravi','M.Tech','Palakol');

b) Inserting a single record

Syntax: INSERT INTO < relation/table name>VALUES (data_1,data_2, ... data_n);

Example: SQL>INSERT INTO student VALUES (1,'Ravi','M.Tech','Palakol');

c) Inserting all records from another relation

Syntax: INSERT INTO relation_name_1 SELECT Field_1,field_2,field_n FROM relation_name_2 WHERE field_x=data;

Example: SQL>INSERT INTO std SELECT sno, sname FROM student WHERE name = ‘Ramu‘;

d) Inserting multiple records

Syntax: INSERT INTO relation_name field_1,field_2, ... field_n) VALUES (&data_1,&data_2, ... &data_n);

Example: SQL>INSERT INTO student (sno, sname, class,address) VALUES (&sno,’&sname’,’&class’,’&address’);

Enter value for sno: 101

Enter value for name: Ravi

Enter value for class: M.Tech

Enter value for name: Palakol

2.SELECT – To display one or more rows

Syntax 1: select * from <table name>;

Syntax 2: select columnname1,columnname 2 from <table name>;

Syntax 3: select * from <table name> where <condition>;

Example :

a) Find the names of all branches in the loan table

    select branch_name from loan;

b) List all account numbers made by brighton branch

    select acc_no from account where branch_name = 'brighton';

c) List the customers who are living in the city Harrison

    select cust_name from customer where cust_city = 'harrison';

3.UPDATE – Used to alter the column values in a table

Syntax : update <table name> set column_name=new_value where <condition>;

Example: update the account table to replace the balance value 500 to 450.

Query: update account set balance=450 where acc_no=’A-101’;

UPDATE-SET-WHERE: This is used to update the content of a record in a relation.

Syntax: SQL>UPDATE relation name SET Field_name1=data,field_name2=data, WHERE field_name=data;

Example:

SQL>UPDATE student SET sname = ‘kumar’ WHERE sno=1;

4.DELETE – Used to delete one or more rows.

Syntax : delete from <table name> where <condition>;

Example : delete from borrower where cust_name='jackson';

DELETE-FROM: This is used to delete all the records of a relation but it will retain the structure of that relation.

a) DELETE-FROM: This is used to delete all the records of relation.

Syntax:

SQL>DELETE FROM relation_name;

Example: SQL>DELETE FROM std;

b) DELETE -FROM-WHERE: This is used to delete a selected record from a relation.

Syntax:

SQL>DELETE FROM relation_name WHERE condition;

Example: SQL>DELETE FROM student WHERE sno = 2;

5. TRUNCATE: This command will remove the data permanently. But structure will not be removed.

Syntax: TRUNCATE TABLE <Table name>

Example: TRUNCATE TABLE student;

Difference between Truncate & Delete:-

- ✓ By using truncate command data will be removed permanently & will not get back where as by using delete command data will be removed temporally & get back by using roll back command.
- ✓ By using delete command data will be removed based on the condition where as by using truncate command there is no condition.
- ✓ Truncate is a DDL command & delete is a DML command.

Result:

Thus, manipulating the Database tables using DML commands has been completed successfully.

EX. NO: 05 (A) QUERYING/MANAGING THE DATABASE USING SQL PROGRAMMING - STORED PROCEDURES / FUNCTIONS

DATE:

Aim:

To query and manage the database using SQL programming for Stored Procedures and Functions.

BUILT-IN FUNCTIONS

1. AGGREGATE FUNCTIONS

(a) AVG - To find the average of values.

Example: Find the average of account balance from the account table

Query: select avg (balance) from account;

(b) SUM – To find the sum of values.

Example: Find the sum of account balance from the account table

Query: select sum (balance) from account;

(c) MAX – Returns the maximum value.

Example: Find the Maximum value of account balance from the account table.

Query: select max (balance) from account;

(d) MIN - Returns the minimum value.

Example: Find the Minimum value of account balance from the account table.

Query: select min (balance) from account;

(e) COUNT – Returns the number of rows in the column or table.

Example 1: Find the number rows in the customer table.

Query: select count (*) from customer;

Example 2: Find the number of rows in the balance column of account table.

Query: select count (balance) from account;

(f) GROUP BY

Example 1: Find the average account balance at each branch.

Query:select branch_name, avg (balance) from account group by branch_name;

Example 2: Find the average account balance at brighton branch.

Query: select branch_name, avg (balance) from account group by branch_name having branch_name='brighton';

2. NUMERIC FUNCTIONS:

i) ABS-Returns the absolute value.

Example: select abs (-19) from dual;

ii) CEIL- Returns the smallest integer value greater than number

Example: select ceil (40.678) from dual;

iii) FLOOR – Returns the largest integer value less than the number

Example: select floor (40.678) from dual;

iv) POWER (n1, n2) – Returns the n1 to the power n2

Example: select power (4, 6) from dual;

v) ROUND (n1, n2) – Returns n1 after rounding n1 with respect to n2 decimal places.

Example: select round (66666.8888, 2) from dual;

vi) TRUNC (n1, n2)- Returns n1 after truncating n1 with respect to n2 decimal places.

Example: select trunc (66666.8888, 2) from dual;

vii) EXP- Returns the exponentiation of the given number.

Example: select exp (5) from dual;

viii) SQRT – Returns the square root of the given number.

Example: select sqrt (36) from dual;

3. CHARACTER FUNCTIONS:

i) CHR (number)- Returns the character equivalent of the ASCII number

Example: select chr (75) from dual;

ii) LENGTH (String) – Returns the number of character in the string

Example: select cust_name, length (cust_name) from customer where cust_city='rye'

iii) LOWER (String) – returns the lower case letters

Example: select lower (‘CAPE INSTITUTE OF TECHNOLOGY’) from dual;

iv) UPPER (String) – returns the upper case letters

Example: select upper (branch_name) from branch;

v) INITCAP- returns the first character is Upper

Example: select initcap (branch_name) from branch;

vi) SUBSTR (String, n1, n2)- returns the substring of the string starting from n1 position to n2 number of characters

Example: select substr (‘computerscience’, 9,7) from dual;

4. DATE FUNCTIONS:

i) ADD_MONTHS (Date, number) returns the date after adding number of months with the given date

Example: select add_months ('01-sep-01', 3) from dual;

ii) MONTHS_BETWEEN (date1, date2) returns the number of months in between date1 and date2

Example: select months_between ('01-may-02','01-feb-02') from dual;

iii) LAST_DAY (date) –returns the last date of that month.

Example Example: select last_day ('12-jun-98') from dual;

iv) SYSDATE – returns the system date

Example: select sysdate from dual;

v) NEXT_DAY (d, day)– returns the date of the next day.

Example: select next_day (sysdate,'monday') from dual;

5. CONVERSION FUNCTIONS:

i) TO_DATE (charStringdate, dateFormat) converts the charstringdate to the date. It will accept the charStringdate according to the date format specified.

Example: select to_date ('january 02 1999', 'month-dd-yyyy') from dual;

select to_date('02 february 2000','dd-month-yyyy')from dual;

ii) TO_CHAR (date, format) returns the character equivalent of the date according to the format.

Example: select to_char (sysdate, 'ddth" of " fmmonth yyyy') from dual;

Example: select to_char (to_date ('january 02 1999', 'month-dd-yyyy'), 'ddth" of " fmmonth yyyy') from dual;

iii) TO_CHAR (n [, fmt]) returns a value of number data type to a value of char data type

Example: select to_char (17145,'$099,999') from dual;

iv) TO_NUMBER (charNumeric) returns the number equivalent of the charNumeric

Example: select to_number ('123') from dual;

STORED PROCEDURES

DESCRIPTION:

SYNTAX:

```
CREATE [OR REPLACE] PROCEDURE name [(parameter[,parameter,…])]
{IS|AS}
        [local declarations]
```

```
          BEGIN
                    executable statements
          [EXCEPTION
                    exception handlers]
          END [name];
```

PROCEDURE FOR GCD NUMBERS

```
create or replace procedure procedure1
is
          a number(3);
          b number(3);
          c number(3);
          d number(3);
begin
          a:=&a;
          b:=&b;
                    if(a>b) then
                              c:=mod(a,b);
                              if(c=0) then
                                        dbms_output.put_line('GCD is');
                                        dbms_output.put_line(b);
                              else
                                        dbms_output.put_line('GCD is');
                                        dbms_output.put_line(c);
                              end if;
                    else
                              d:=mod(b,a);
                              if(d=0) then
                                        dbms_output.put_line('GCD is');
                                        dbms_output.put_line(a);
                              else
                                        dbms_output.put_line('GCD is');
                                        dbms_output.put_line(d);
                              end if;
                    end if;
end;
```

```
/
```

FUNCTIONS

SYNTAX:

```
CREATE [OR REPLACE] FUNCTION name
        [(parameter[,parameter, ...])]
RETURN datatype
{IS | AS}
[local declarations]
        BEGIN
                executable statements
        RETURN value;
        [EXCEPTION
                exception handlers]
END [name];
```

To write a PL/SQL block to find factorial of given number using function.

Filename: fact.sql

```
create or replace function fact(limit number) return number is
        ans number(3);
        I number(3);
begin
        ans:=1;
        for i in 1..limit loop
                ans:=ans*i;
        end loop;
        return(ans);
end;
/
```

Filename: test.sql

```
declare
        n number(3);
        f number(3);
begin
        n:=&limit;
        f:=fact(n);
        dbms_output.put_line(n||'! = '||f);
end;
/
```

OUTPUT:

SQL> fact.sql

```
Function Created.
```

SQL> test.sql

```
Enter value for Limit: 4
old     5:              n:=&limit;
new     5:              n:=4;
4! = 24
PL/SQL Procedure successfully completed.
```

Result:

Thus, querying and managing the database using SQL programming for Stored Procedures and Functions has been successfully completed.

EX. NO: 5(B) QUERYING/MANAGING THE DATABASE USING SQL PROGRAMMING - CONSTRAINTS AND SECURITY USING TRIGGERS

DATE:

Aim:

To query and manage the database using SQL programming for implementing Constraints and Security using Triggers

DESCRIPTION:

DATABASE TRIGGERS

Triggers are similar to procedures or functions in that they are named PL/SQL blocks with declarative, executable, and exception handling sections. A trigger is executed implicitly whenever the triggering event happens. The act of executing a trigger is known as firing the trigger.

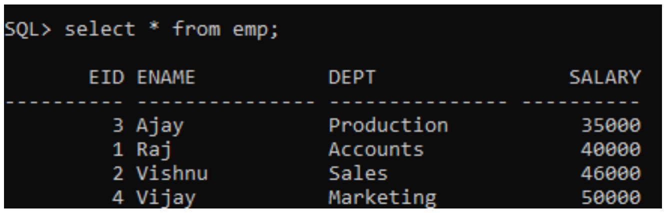
USE OF TRIGGERS

- ✓ Maintaining complex integrity constraints not possible through declarative constraints enable at table creation.
- ✓ Auditing information in a table by recording the changes made and who made them.
- ✓ Automatically signaling other programs that action needs to take place when changes are made to a table.
- ✓ Perform validation on changes being made to tables.
- ✓ Automate maintenance of the database.

TRIGGER SYNTAX:

```
CREATE [OR REPLACE] TRIGGER triggername
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE [OF columns]}
[OR {DELETE | INSERT |UPDATE [OF columns]}]...
ON table
[FOR EACH ROW [WHEN condition]]
[REFERENCING [OLD AS old] [NEW AS new]]
PL/SQL block
```

Program to create a DB trigger before insert for each row on the emp table not allowing insertion for Eid is 5 and salary is 55000.



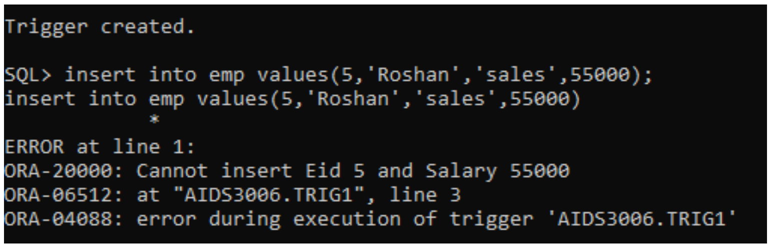
```
SQL> select * from emp;

       EID ENAME          DEPT                SALARY
---------- -------------- -------------- ----------
         3 Ajay           Production          35000
         1 Raj            Accounts            40000
         2 Vishnu         Sales               46000
         4 Vijay          Marketing           50000
```

Filename: trigg1.sql

```
create or replace trigger trig1 before insert on emp for each row
begin
        if(:new.Eid=5 and :new.salary=55000) then
                raise_application_error(-20000,'Cannot insert Eid 5 and Salary 55000');
        end if;
end;
/
```

OUTPUT:

SQL> trigg1.sql



```
Trigger created.

SQL> insert into emp values(5,'Roshan','sales',55000);
insert into emp values(5,'Roshan','sales',55000)
            *
ERROR at line 1:
ORA-20000: Cannot insert Eid 5 and Salary 55000
ORA-06512: at "AIDS3006.TRIG1", line 3
ORA-04088: error during execution of trigger 'AIDS3006.TRIG1'
```

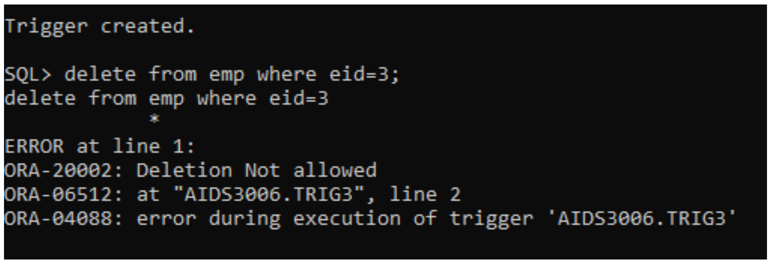
Program to create a DB trigger not allowing deletion in emp table.

Filename: trigg2.sql

```
create or replace trigger trig2 before delete on emp for each row
begin
        raise_application_error(-20002,'Deletion Not allowed');
end;
/
```

OUTPUT:

SQL> trigg2.sql



```
Trigger created.

SQL> delete from emp where eid=3;
delete from emp where eid=3
            *
ERROR at line 1:
ORA-20002: Deletion Not allowed
ORA-06512: at "AIDS3006.TRIG3", line 2
ORA-04088: error during execution of trigger 'AIDS3006.TRIG3'
```

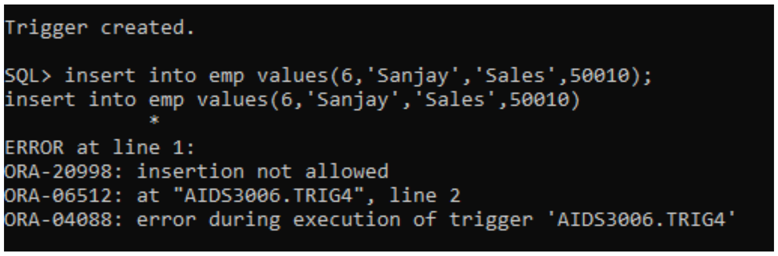
Create a PL/SQL trigger which prevents the insertion of new record into the emp table.

Filename:trigg3.sql

```
create or replace trigger trig3
        before
        insert on emp
        for each row
begin
        raise_application_error (-20998, 'insertion not allowed');
end;
/
```

OUTPUT:

SQL> trigg3.sql



```
Trigger created.

SQL> insert into emp values(6,'Sanjay','Sales',50010);
insert into emp values(6,'Sanjay','Sales',50010)
            *
ERROR at line 1:
ORA-20998: insertion not allowed
ORA-06512: at "AIDS3006.TRIG4", line 2
ORA-04088: error during execution of trigger 'AIDS3006.TRIG4'
```

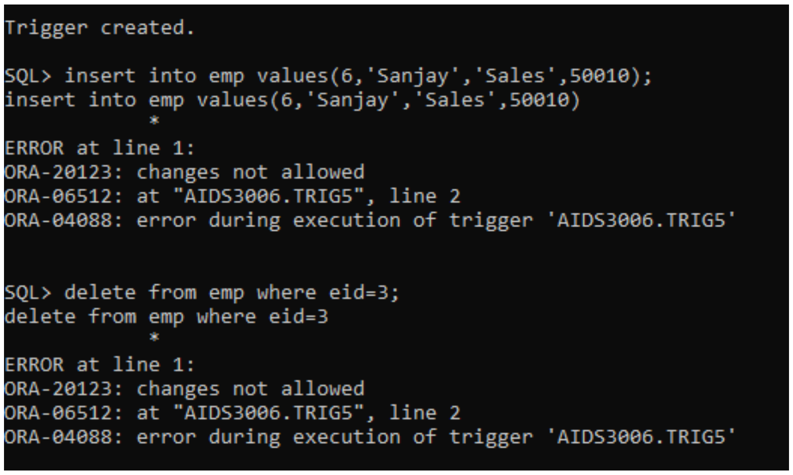
Create a PL/SQL trigger which prevents all DML operations on the table emp.

Filename: trigg4.sql

```
create or replace trigger trig4 before insert or delete or update on emp for each row

begin

        raise_application_error(-20123,'changes not allowed');

end;

/
```

OUTPUT:

SQL> trigg4.sql



```
Trigger created.

SQL> insert into emp values(6,'Sanjay','Sales',50010);
insert into emp values(6,'Sanjay','Sales',50010)
            *
ERROR at line 1:
ORA-20123: changes not allowed
ORA-06512: at "AIDS3006.TRIG5", line 2
ORA-04088: error during execution of trigger 'AIDS3006.TRIG5'


SQL> delete from emp where eid=3;
delete from emp where eid=3
            *
ERROR at line 1:
ORA-20123: changes not allowed
ORA-06512: at "AIDS3006.TRIG5", line 2
ORA-04088: error during execution of trigger 'AIDS3006.TRIG5'
```

Result:

Thus, querying and managing the database using SQL programming for implementing Constraints and Security using Triggers has been completed successfully.

EX. NO: 06 DATABASE DESIGN USING NORMALIZATION – BOTTOM-UP APPROACH

DATE:

Aim:

To design Database using Normalization – Bottom-up Approach.

DESCRIPTION:

Normalization:

• It is the processes of reducing the redundancy of data in the table and also improving the data integrity. So why is this required? Without Normalization in SQL, we may face many issues such as.

• **Insertion anomaly:** It occurs when we cannot insert data to the table without the presence of another attribute.

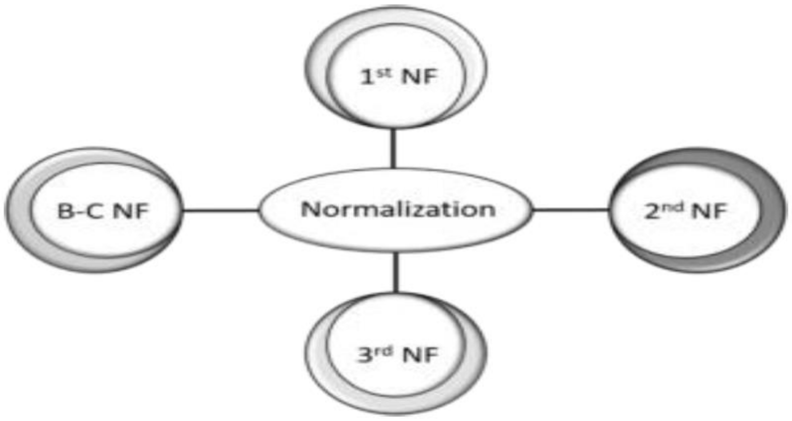
• **Update anomaly:** It is a data inconsistency that results from data redundancy and a partial update of data.

• **Deletion Anomaly:** It occurs when certain attributes are lost because of the deletion of other attributes.

• Normalization entails organizing the columns and tables of a database to ensure that their dependencies are properly enforced by database integrity constraints.

• It usually divides a large table into smaller ones, so it is more efficient. In 1970 the First Normal Form was defined by Edgar F Codd and eventually, other Normal Forms were defined.

• Normalization in SQL will enhance the distribution of data. Now let's understand each and every Normal Form with examples.



1st Normal Form (1NF)

✓ In this Normal Form, we tackle the problem of atomicity. Here atomicity means values in the table should not be further divided. In simple terms, a single cell cannot hold multiple values. If a table contains a composite or multi-valued attribute, it violates the First Normal Form.

Employee ID	Employee Name	Phone Number	Salary
1EDU001	Alex	+91 8553278282	60,131
1EDU001	Alex	+91 9876543210	60,131
1EDU002	Barry	+91 9876512340	48,302
1EDU003	Clair	+91 9812763405	22,900
1EDU004	David	+91 9876543120	81,518
1EDU004	Sriram	+91 7448702556	90,000

- ✓ In the above table, we can clearly see that the Phone Number column has two values. Thus it violated the 1st NF. Now if we apply the 1st NF to the above table we get the below table as the result.

Employee ID	Employee Name	Phone Number	Salary
1EDU001	Alex	+91 8553278282	60,131
1EDU001	Alex	+91 9876543210	60,131
1EDU002	Barry	+91 9876512340	48,302
1EDU003	Clair	+91 9812763405	22,900
1EDU004	David	+91 9876543120	81,518
1EDU004	Sriram	+91 7448702556	90,000

We have achieved atomicity and also each and every column have unique values.

<u>2nd Normal Form (2NF)</u>

- ✓ The first condition in the 2nd NF is that the table has to be in 1st NF. The table also should not contain partial dependency. Here partial dependency means the proper subset of candidate key determines a non-prime attribute.

EMPLOYEE ID	DEPARTMENT ID	OFFICE LOCATION
1EDU001	ED-T1	Pune
1EDU002	ED-S2	Bengaluru
1EDU003	ED-M1	Delhi
1EDU004	ED-T3	Mumbai

- ✓ This table has a composite primary key Employee ID, Department ID. The non-key attribute is Office Location. In this case, Office Location only depends on Department ID, which is only part of the primary key.

- ✓ Therefore, this table does not satisfy the second Normal Form. To bring this table to Second Normal Form, we need to break the table into two parts.

EMPLOYEE ID	DEPARTMENT ID
1EDU001	ED-T1
1EDU002	ED-S2
1EDU003	ED-M1
1EDU004	ED-T3

DEPARTMENT ID	OFFICE LOCATION
ED-T1	Pune
ED-S2	Bengaluru
ED-M1	Delhi
ED-T3	Mumbai

- ✓ In the table, the column Office Location is fully dependent on the primary key of that table, which is Department ID.

3rd Normal Form (3NF)

- ✓ The table has to be in 2NF before proceeding to 3NF. The other condition is there should be no transitive dependency for non-prime attributes.
- ✓ That means non-prime attributes (which doesn't form a candidate key) should not be dependent on other non-prime attributes in a given table.
- ✓ So a transitive dependency is a functional dependency in which X → Z (X determines Z) indirectly, by virtue of X → Y and Y → Z.

STUDENT ID	STUDENT NAME	SUBJECT ID	SUBJECT	ADDRESS
1DT15ENG01	Alex	15CS11	SQL	Goa
1DT15ENG02	Barry	15CS13	JAVA	Bengaluru
1DT15ENG03	Clair	15CS12	C++	Delhi
1DT15ENG04	David	15CS13	JAVA	Kochi

- ✓ In the above table, Student ID determines Subject ID, and Subject ID determines Subject.
- ✓ Therefore, Student ID determines Subject via Subject ID. This implies that we have a transitive functional dependency, and this structure does not satisfy the third normal form.

STUDENT	STUDENT NAME	SUBJECT ID	ADDRESS
1DT15ENG01	Alex	15CS11	Goa
1DT15ENG02	Barry	15CS13	Bengaluru
1DT15ENG03	Clair	15CS12	Delhi
1DT15ENG04	David	15CS13	Kochi

SUBJECT ID	SUBJECT
15CS11	SQL
15CS13	JAVA
15CS12	C++
15CS13	JAVA

- ✓ The above tables all the non-key attributes are now fully functional dependent only on the primary key.
- ✓ In the first table, columns Student Name, Subject ID and Address are only dependent on Student ID. In the second table, Subject is only dependent on Subject ID.

Bottom –up approach:

- ✓ Normalization is a bottom-up approach which starts with a collection of attributes and organises them into well-structured relations which are free from redundant data.

BCNF: Boyce-Codd Normal Form

Boyce Codd Normal Form (BCNF)

- ✓ This is also known as 3.5 NF. It's the higher version 3NF and was developed by Raymond F. Boyce and Edgar F. Codd to address certain types of anomalies which were not dealt with 3NF.
- ✓ The table has to satisfy 3rd Normal Form.
- ✓ In BCNF if every functional dependency A → B, then A has to be the Super Key of that particular table.

STUDENT ID	SUBJECT	PROFESSOR
1DT15ENG01	SQL	Prof. Mishra
1DT15ENG02	JAVA	Prof. Anand
1DT15ENG02	C++	Prof. Kanthi
1DT15ENG03	JAVA	Prof. Anand
1DT15ENG04	DBMS	Prof. Lokesh

- ✓ One student can enroll for multiple subjects.
- ✓ There can be multiple professors teaching one subject .
- ✓ And, for each subject, a professor is assigned to the student.
- ✓ In the table Student ID, and Subject form the primary key, which means the Subject column is a prime attribute. But, there is one more dependency, Professor → Subject.
- ✓ And while Subject is a prime attribute, Professor is a non-prime attribute, which is not allowed by BCNF.
- ✓ Dividing the table into two parts. One table will hold Student ID which already exists and newly created column Professor ID.

STUDENT ID	PROFESSOR ID
1DT15ENG01	1DTPF01
1DT15ENG02	1DTPF02
1DT15ENG02	1DTPF03

- ✓ And in the second table, we will have the columns Professor ID, Professor and Subject.

PROFESSOR ID	PROFESSOR	SUBJECT
1DTPF01	Prof. Mishra	SQL
1DTPF01	Prof. Anand	JAVA
1DTPF01	Prof. Kanthi	C++
:	:	:

- ✓ By this we satisfying the Boyce Codd Normal Form.

Result:

Thus, designing Database using Normalization – Bottom-up Approach has been completed successfully.

EX. NO: 07 DEVELOP DATABASE APPLICATIONS USING IDE/RAD TOOLS (Eg. NETBEANS, VISUALSTUDIO)

DATE:

Aim:

To develop an sample Database Application using IDE/RAD Tools.

Procedure:

- ✓ Open you Microsoft Visual Studio 2010, 2012 or higher, or just your Microsoft Visual Basic .Net.
- ✓ Create a new project (select File and New Project). For visual studio user: (select Visual Basic then Windows Form Application).
- ✓ Here is the sample form layout or design. Feel free to design your form. We need to add the following controls:
  - ➢ 2 labels
  - ➢ 2 textboxes
  - ➢ 2 buttons
  - ➢ 1 checkbox.
- ✓ The program will first validate the input of the user, the user must enter a username and password or else a message will appear that will notify the user that username and password field is required.
- ✓ The program will then match or compare the user input to the criteria of the program. The username must be admin and password must also be admin which means that the username and password combination must be admin or else a message will prompt you that your username and password is incorrect.
- ✓ To clear the username and password field, kindly double click the Reset button and paste the code below.

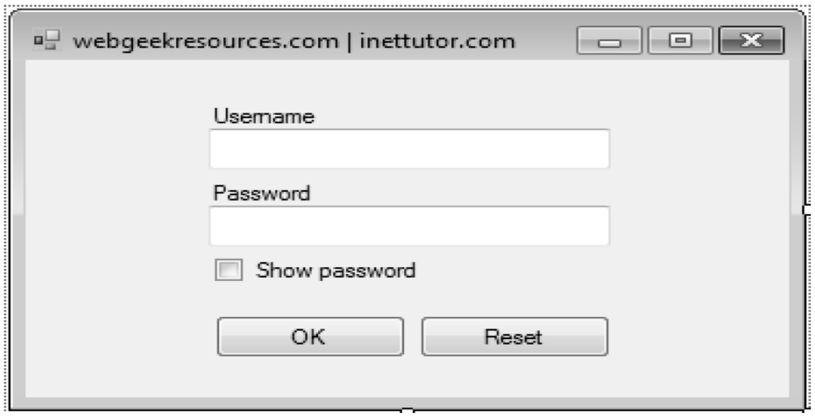
  ```
  TextBox1.Clear()
  TextBox2.Clear()
  ```
- ✓ Additional feature of this program is to allow the user to view or to make its password visible or in simplest explanation is to view what you are typing in the password field. Kindly double click the Show password checkbox and paste the line of codes below.

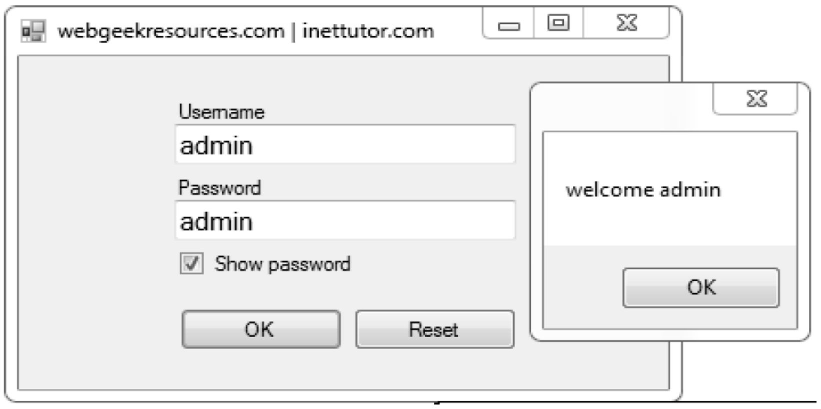
<u>Program:</u>

```
If TextBox1.Text = "" Then
MessageBox.Show("Please enter username")
TextBox1.Focus()
Exit Sub
ElseIf TextBox2.Text = "" Then
```

```
MessageBox.Show("Please enter password")
TextBox2.Focus()
Exit Sub
End If
If TextBox1.Text = "admin" And TextBox2.Text = "admin" Then
MessageBox.Show("welcome admin")
Else
MessageBox.Show("incorrect username or password")
End If
If CheckBox1.Checked = True Then
TextBox2.PasswordChar = ""
Else
TextBox2.PasswordChar = "*"
End If
```

OUTPUT:





Result:

Thus, developing a sample Database Application using IDE/RAD Tools has been completed successfully.

EX. NO: 08 DATABASE DESIGN USING EER-TO-ODB MAPPING / UML CLASS DIAGRAMS

DATE:

Aim:

To Design Database using EER –to – ODB Mapping / UML Class Diagrams.

Procedure:

Mapping an EER Schema to an ODB Schema

- It is relatively straightforward to design the type declarations of object classes for an ODBMS from an EER schema that contains neither categories nor n ary relationships with n > 2.

- However, the operations of classes are not specified in the EER diagram and must be added to the class declarations after the structural mapping is completed.

The outline of the mapping from EER to ODL is as follows:

Step 1:

✓ Create an ODL class for each EER entity type or subclass. The type of the ODL class should include all the attributes of the EER class.

✓ Multivalued attributes are typically declared by using the set, bag, or list constructors. If the values of the multivalued attribute for an object should be ordered, the list constructor is chosen; if duplicates are allowed, the bag constructor should be chosen; otherwise, the set constructor is chosen.

✓ Composite attributes are mapped into a tuple constructor (by using a struct declaration in ODL).

Step 2:

✓ Add relationship properties or reference attributes for each binary relationship into the ODL classes that participate in the relationship. These may be created in one or both directions.

✓ If a binary relationship is represented by references in both directions, declare the references to be relationship properties that are inverses of one another, if such a facility exists.

✓ If a binary relationship is represented by a reference in only one direction, declare the reference to be an attribute in the referencing class whose type is the referenced class name.

✓ Depending on the cardinality ratio of the binary relationship, the relationship properties or reference attributes may be single-valued or collection types. They will be single valued for binary relationships in the 1:1 or N:1 directions

Step 3:

• Include appropriate operations for each class. These are not available from the EER schema and must be added to the database design by referring to the original requirements.

• A constructor method should include program code that checks any constraints that must hold when a new object is created.

• A destructor method should check any constraints that may be violated when an object is deleted.

Step 4:

• An ODL class that corresponds to a subclass in the EER schema inherits the type and methods of its super class in the ODL schema.

Step 5:

• Weak entity types can be mapped in the same way as regular entity types. An alternative mapping is possible for weak entity types that do not participate in any relationships except their identifying relationship

• These can be mapped as though they were composite multivalued attributes of the owner entity type, by using the set < struct < ... >> or list < struct < ... >> constructors. The attributes of the weak entity are included in the struct < ... > construct, which corresponds to a tuple constructor.

Step 6:

• Categories (union types) in an EER schema are difficult to map to ODL. It is possible to create a mapping similar to the EER-to-relational mapping

• By declaring a class to represent the category and defining 1:1 relationships between the category and each of its super classes.

Step 7:

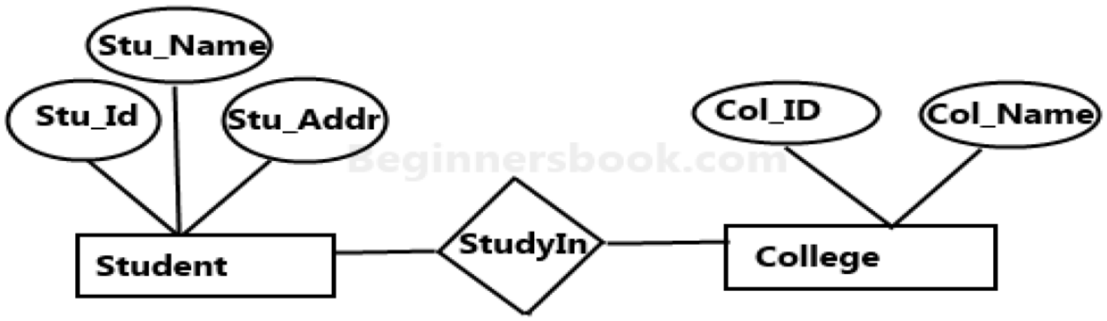
• An n-ary relationship with degree n > 2 can be mapped into a separate class, with appropriate references to each participating class.

• These references are based on mapping a 1:N relationship from each class that represents a participating entity type to the class that represents the n-ary relationship.

• An M:N binary relationship, especially if it contains relationship attributes, may also use this mapping option, if desired.

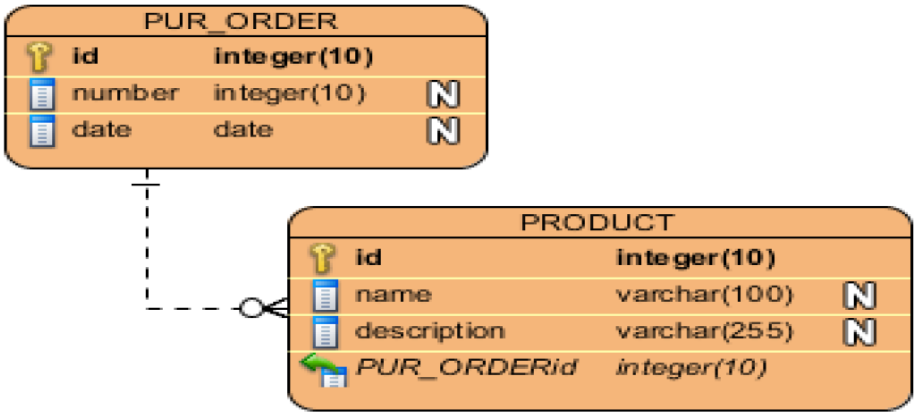
• The mapping has been applied to a subset of the UNIVERSITY database schema in the context of the ODMG object database standard. The mapped object schema using the ODL notation is shown.

EER to ODB mapping diagram:



Sample E-R Diagram

EER (UML Class Diagram):



Result:

Thus, Designing Database using EER –to – ODB Mapping / UML Class Diagrams has been completed successfully.

EX. NO: 09 OBJECT FEATURES OF SQL-UDTS AND SUB-TYPES, TABLES USING UDTS, INHERITANCE, METHOD DEFINITION

DATE:

Aim:

To study about the Object features of SQL-UDTs and sub-types, Tables using UDTs, Inheritance, Method definition.

DESCRIPTION:

Objects of SQL:

• SQL objects are schemas, journals, catalogues, tables, aliases, views, indexes, constraints, triggers, sequences, stored procedures, user-defined functions, user-defined types, global variables, and SQL packages, SQL creates and maintains these objects.

UDT in SQL:

• The UDT is similar to an alias data type and it uses the existing data types in SQL server or Azure SQL database.

• SQL server supports two kinds of user defined types

- User- defined data type.
- User- defined table type

Use of UDT in SQL Server:

• User defined type can be used in the definition of database objects such as variables in transact-SQL batches, in functions and stored procedures, and as arguments in functions and stored procedures.

Sub- types of UDT in SQL:

- Exact numeric.
- Approximate numeric.
- Date and Time.
- Character String.
- Unicode character strings.
- CLR data types.
- Spatial data types

Tables using UDTs:

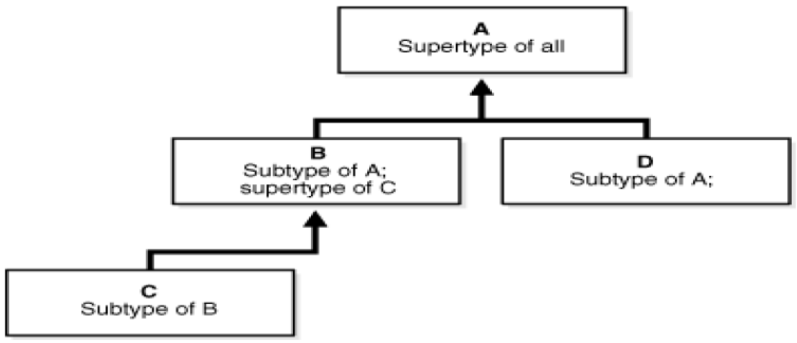
• There is no special syntax for creating a UDT column in a table. You can use the name of the UDT in column definition as though it were one of the intrinsic SQL server data types. The following CREATE TABLE Transact- SQL statement creates a table named points, with a column named ID, which is defined as an into identity column is named PointgValue, with a data type of Point.

Inheritance in SQL object types:

• SQL object inheritance is based on a family tree of object types that forms a type hierarchy. The type hierarchy consists of a parent object type, called a super type, and one or more levels of child object types, called subtypes, which are derived from the parent.

• A subtype can be derived from a super type either directly or indirectly through intervening levels of other subtypes.

• A super type can have multiple sibling subtypes, but a subtype can have at most one direct parent super type (single inheritance).



Method Definition:

• A method is procedure or function that is part of the object type definition, and that can operate on the attributes of the type. Such methods are also called member methods, and they take the keyword **MEMBER** when you specify them as a component of the object type.

• Method specification

• Method names

• Method name overloading

Implementing Methods

To implement a method, create the PL/SQL code and specify it within a CREATE TYPE BODY statement.

For example, consider the following definition of an object type named rational type:

```
CREATE TYPE rational_type AS OBJECT
( numerator INTEGER,
  denominator INTEGER,
  MAP MEMBER FUNCTION rat_to_real RETURN REAL,
  MEMBER PROCEDURE normalize,
  MEMBER FUNCTION plus (x rational_type)
      RETURN rational_type);
```

Example: The following definition is shown merely because it defines the function gcd, which is used in the definition of the normalize method in the CREATE TYPE BODY statement later in this section.

```
CREATE FUNCTION gcd (x INTEGER, y INTEGER) RETURN INTEGER AS
-- Find greatest common divisor of x and y. For example, if
-- (8,12) is input, the greatest common divisor is 4.
-- This will be used in normalizing (simplifying) fractions.
-- (You need not try to understand how this code works, unless
--  you are a math wizard. It does.)
--
   ans INTEGER;
BEGIN
   IF (y <= x) AND (x MOD y = 0) THEN
      ans := y;
   ELSIF x < y THEN
      ans := gcd(y, x);  -- Recursive call
   ELSE
      ans := gcd(y, x MOD y);  -- Recursive call
   END IF;
   RETURN ans;
END;
```

Result:

Thus, studying about the Object features of SQL-UDTs and sub-types, Tables using UDTs, Inheritance, Method definition has been completed sucessfuly.

EX. NO: 10 QUERYING THE OBJECT-RELATIONAL DATABASE USING OBJET QUERY LANGUAGE

DATE:

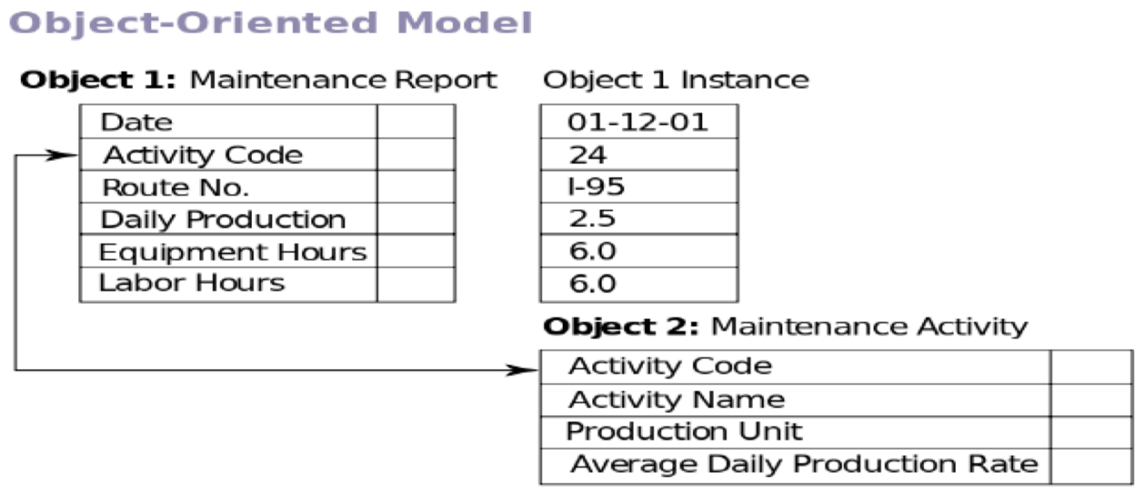
Aim:

To Query the Object-relational database using Objet Query language.

DESCRIPTION:

Object–Relational Database (ORD):

• An object–relational database (ORD), or object–relational database management system (ORDBMS), is a database management system (DBMS) similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. In addition, just as with pure relational systems, it supports extension of the data model with custom data types and methods.



• An object–relational database can be said to provide a middle ground between relational databases and object-oriented databases. In object–relational databases, the approach is essentially that of relational databases.

• The data resides in the database and is manipulated collectively with queries in a query language.

• At the other extreme are OODBMS in which the database is essentially a persistent object store for software written in an object-oriented programming language, with a programming API for storing and retrieving objects, and little or no specific support for querying.

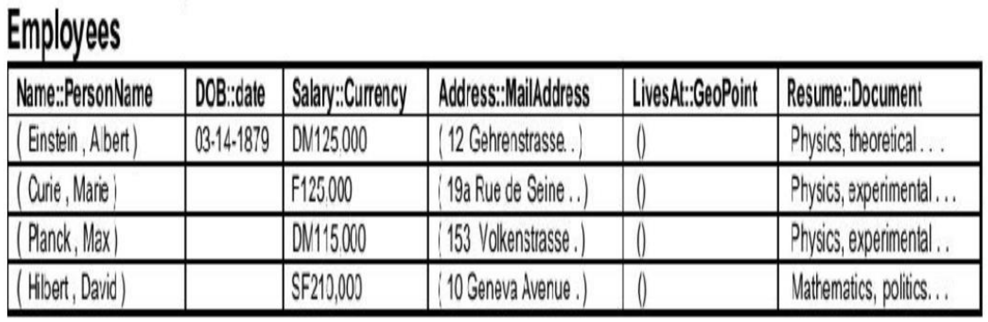
Procedure:

- CREATE.
- INSERT.
- UPDATE.
- DELETE

Program:

```
CREATE TABLE Employees (FirstName VARCHAR(32)
NOT NULL,
Surname VARCHAR(64) NOT NULL,DOB DATE NOT
NULL,
Salary DECIMAL(10,2) NOT NULLCHECK ( Salary > 0.0 ),
Address_1 VARCHAR(64) NOT NULL,Address_2 VAR
CHAR(64) NOT NULL,
City VARCHAR(48) NOT NULL,State CHAR(2) NOT
NULL,ZipCode INTEGER NOT NULL,PRIMARY KEY (
Surname, FirstName, DOB ));
INSERT INTO Employees ( Pager_Number, Pass_Code, Mes
sage )
SELECT E.Pager_Number,E.Pass_Code,
Print(E.Name) || ': Call 1-800-TEMPS-R-US for immediate
INFORMIX DBA job'
FROM Temporary_Employees E
WHERE Contains (GeoCircle('(-122.514, 37.221)', '60
miles')),E.LivesAt )
AND DocContains ( E.Resume, 'INFORMIX and Database
Administrator')
AND NOT IsBooked ( Period(TODAY, TODAY +
7),E.Booked );

SELECT *FROM Employees;
```

OUTPUT:



Employees

Name::PersonName	DOB::date	Salary::Currency	Address::MailAddress	LivesAt::GeoPoint	Resume::Document
(Einstein , Albert)	03-14-1879	DM125,000	(12 Gehrenstrasse. .)	()	Physics, theoretical . . .
(Curie , Marie)		F125,000	(19a Rue de Seine . .)	()	Physics, experimental . . .
(Planck , Max)		DM115,000	(153 Volkenstrasse .)	()	Physics, experimental . .
(Hilbert , David)		SF210,000	(10 Geneva Avenue .)	()	Mathematics, politics. . .

Result:

Thus, Querying the Object-relational database using Objet Query language has been completed successfully.

Ex. No 11 **Library Management System**

AIM:

To design and implement a **Library Management System** using SQL with multiple related tables, and perform complex queries using JOIN, GROUP BY, and aggregate functions.

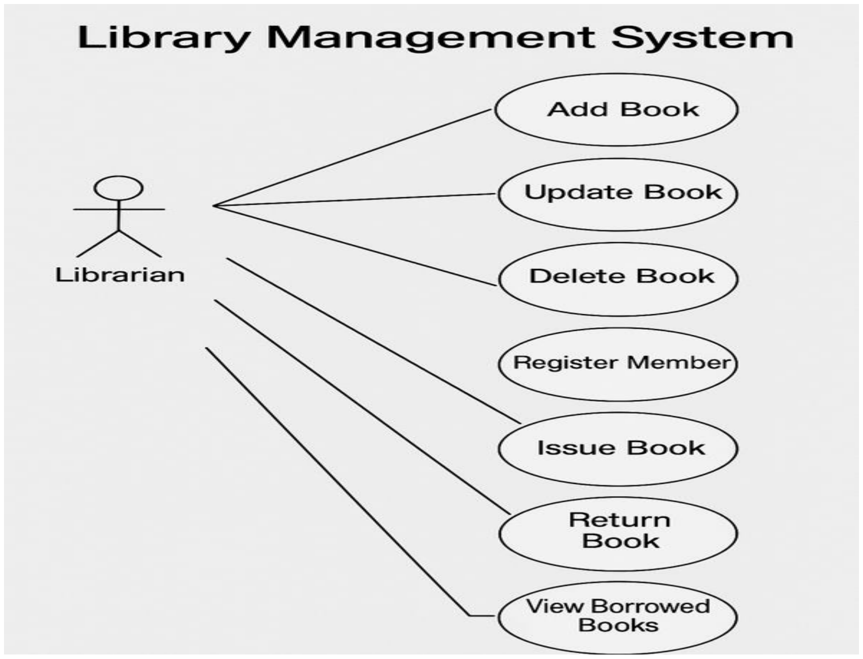
REQUIREMENTS:

- Software: MySQL / PostgreSQL
- Tools: MySQL Workbench / pgAdmin / CLI
- Concepts Used: Table creation, primary/foreign keys, joins, aggregate functions, normalization

PROCEDURE:

1. Create a new database: LibraryDB.
2. Create the following tables:
    - Books
    - Members
    - IssueRecords
3. Insert sample data into all three tables.
4. Perform relational queries using JOIN, GROUP BY, and aggregation.

USECASE DIAGRAM



PROGRAM:

Step 1: Create Database

```
CREATE DATABASE LibraryDB;
```

Step 2: Use the Database (for MySQL)

```
USE LibraryDB;
```

Step 3: Create Tables

Table for Books

```
CREATE TABLE Books (
    BookID INT PRIMARY KEY,
    Title VARCHAR(100),
    Author VARCHAR(50),
    Genre VARCHAR(30),
    AvailableCopies INT
);
```

Table for Members

```
CREATE TABLE Members (
    MemberID INT PRIMARY KEY,
    Name VARCHAR(50),
    Email VARCHAR(50),
    JoinDate DATE
);
```

Table for Issue Records (Many-to-One Relationship)

```
CREATE TABLE IssueRecords (
    IssueID INT PRIMARY KEY,
    BookID INT,
    MemberID INT,
    IssueDate DATE,
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```

Step 4: Insert Sample Data

```
INSERT INTO Books VALUES
(1, '1984', 'George Orwell', 'Dystopian', 5),
(2, 'The Alchemist', 'Paulo Coelho', 'Fiction', 3),
(3, 'Clean Code', 'Robert C. Martin', 'Programming', 4);
INSERT INTO Members VALUES
(101, 'Alice', 'alice@mail.com', '2024-01-15'),
(102, 'Bob', 'bob@mail.com', '2024-03-22');
INSERT INTO IssueRecords VALUES
(1, 1, 101, '2024-06-01', '2024-06-10'),
(2, 2, 102, '2024-06-05', NULL),
(3, 1, 102, '2024-06-08', NULL);
```

Step 5: Perform Queries

a) List all books with their current availability

```
SELECT * FROM Books;
```

b) List all issued books with member details

```
SELECT IR.IssueID, B.Title, M.Name, IR.IssueDate, IR.ReturnDate
FROM IssueRecords IR
JOIN Books B ON IR.BookID = B.BookID
JOIN Members M ON IR.MemberID = M.MemberID;
```

c) Count number of books issued per member

```
SELECT M.Name, COUNT(IR.IssueID) AS BooksIssued
FROM Members M
LEFT JOIN IssueRecords IR ON M.MemberID = IR.MemberID
GROUP BY M.Name;
```

d) Find members who have not returned books yet

```
SELECT M.Name, B.Title, IR.IssueDate
FROM IssueRecords IR
JOIN Members M ON IR.MemberID = M.MemberID
JOIN Books B ON IR.BookID = B.BookID
WHERE IR.ReturnDate IS NULL;
```

RESULT

1. View All Books

```
SELECT * FROM Books;
```

BookID	Title	Author	Genre	AvailableCopies
1	1984	George Orwell	Dystopian	5
2	The Alchemist	Paulo Coelho	Fiction	3
3	Clean Code	Robert C. Martin	Programming	4

2. List All Issued Books with Member Details

```
SELECT IR.IssueID, B.Title, M.Name, IR.IssueDate, IR.ReturnDate
FROM IssueRecords IR
JOIN Books B ON IR.BookID = B.BookID
JOIN Members M ON IR.MemberID = M.MemberID;
```

IssueID	Title	Name	IssueDate	ReturnDate
1	1984	Alice	2024-06-01	2024-06-10
2	The Alchemist	Bob	2024-06-05	NULL
3	1984	Bob	2024-06-08	NULL

3. Count Number of Books Issued Per Member

```
SELECT M.Name, COUNT(IR.IssueID) AS BooksIssued
FROM Members M
LEFT JOIN IssueRecords IR ON M.MemberID = IR.MemberID
GROUP BY M.Name;
```

Name	BooksIssued
Alice	1
Bob	2

4. Members Who Have Not Returned Books

```
SELECT M.Name, B.Title, IR.IssueDate
FROM IssueRecords IR
```

```
JOIN Members M ON IR.MemberID = M.MemberID
JOIN Books B ON IR.BookID = B.BookID
WHERE IR.ReturnDate IS NULL;
```

```
Name Title         IssueDate
Bob  The Alchemist 2024-06-05
Bob  1984          2024-06-08
```

Result:

Thus, the design and implement a **Library Management System** using SQL with multiple related tables has completed successfully.