

20CS404	DATABASE MANAGEMENT SYSTEMS (Common to CSE, IT and AI & DS)	L	T	P	C
		3	0	2	4
Nature of Course	Professional Core				
Pre requisites	Basics of Data Structures				

Course Objectives

The course is intended to

1. Familiarize the fundamentals of data models and SQL.
2. Represent a database system using ER diagrams and relational schema.
3. Understand the fundamental concepts of transaction processing- concurrency control Techniques and recovery procedures.
4. Identify with the internal storage structures using different file and indexing techniques which will help in physical database design.
5. Have a comparative knowledge about the various advanced databases.

Course Outcomes

On successful completion of the course, students will be able to

CO.No.	Course Outcome	Bloom's Level
CO1.	Classify the modern and futuristic database applications and write queries using various SQL commands	Analyze
CO2.	Construct ER Model and Design relational schema for a given database application.	Apply
CO3.	Illustrate the concepts for transaction processing and concurrency control.	Understand
CO4.	Apply indexing and hashing techniques to access and generate user reports for a database.	Apply
CO5.	Appraise how advanced databases differ from traditional databases	Evaluate

Course Contents

UNIT I RELATIONAL DATABASES

9

Purpose of Database System – Views of data – Data Models – Database System Architecture – Introduction to relational databases – Relational Model – Keys – SQL fundamentals – Advanced SQL features, PL/SQL.

UNIT II DATABASE DESIGN

9

Entity-Relationship model: Diagrams – Enhanced Model –Relational Mapping – Relational Algebra – Functional Dependencies – Non-loss Decomposition – First, Second, Third Normal Forms, Boyce/Codd Normal Form – Multi-valued Dependencies and Fourth Normal Form – Join Dependencies and Fifth Normal Form

UNIT III TRANSACTIONS

9

Transaction Concepts – ACID Properties – Schedules – Serializability – Concurrency Control – Need for Concurrency – Locking Protocols – Two Phase Locking – Deadlock – Transaction Recovery - Save Points – Isolation Levels – SQL Facilities for Concurrency and Recovery.

UNIT IV IMPLEMENTATION TECHNIQUES

9

RAID – File Organization – Organization of Records – Indexing and Hashing –Ordered Indices – B tree and B+ tree Index Files – Static and Dynamic Hashing – Query Processing Overview – Algorithms for SELECT and JOIN operations – Query optimization using Heuristics and Cost Estimation.

UNIT V ADVANCED DATABASES

9

Distributed Databases: Architecture, Storage, Transaction Processing – Object-based Databases: Concepts-Object-Relational features, MongoDB – Concepts and features, XML Databases: XML Hierarchical Model, DTD, XQuery – Information Retrieval: Retrieval Models, Queries in IR systems.

Total: 45 Periods

Laboratory Components

S.No	List of Experiments	CO Mapping	RBT
1	Data Definition Commands, Data Manipulation Commands for inserting, deleting, updating and retrieving Tables and Transaction Control statements. Database Querying – Simple queries, Nested queries, Sub queries and Joins	1	Understand
2	Practicing PL/SQL for a real time application	1	Apply
3	Database Design using ER modeling, normalization and Implementation for any application	2	Apply
4	Write relational algebra queries for a given set of relations.	2	Apply
5	XML database creation and validation	5	Analyze
6	Case Study using real life database applications	5	Apply

Total : 30 Periods**Text Books**

1. Abraham Silber Schatz, Henry Korth.F and Sudarshan. S, “Database System Concepts”, Mc Graw Hill, 7th Edition 2019.
2. Ramez Elmasri and Shamkant Navathe, “Fundamentals of Database Systems”, Addison-Wesley, 5th Edition 2017.

References

1. Gupta G.K, "Database Management Systems", Tata McGraw Hill, 3rd Edition 2020.
2. Raghu Ramakrishnan, "Database Management Systems", McGraw-Hill College Publications, 4th Edition 2018.
3. Date C.J, Kannan. A, Swaminathan.S, “An Introduction to Database Systems”, Pearson Education, 8th Edition 2017.

Additional References:

1. <https://nptel.ac.in/courses/106/105/106105175/>
2. https://onlinecourses.nptel.ac.in/noc21_cs04/preview 3.
3. <https://nptel.ac.in/courses/106/106/106106093/>

Mapping of Course Outcomes (COs) with Programme Outcomes (POs) Programme Specific Outcomes (PSOs)															
COs	Pos												PSOs		
	1	2	3	4	5	6	7	8	9	10	11	12	1	2	3
CO1	3	2	2										3	1	
CO2	3	2	2	1								1	3	1	
CO3	3	2	1	1								1	3	1	
CO4	3	3	2	1	2						2	1	3	1	
CO5	3	3	2	2	2						2	1	3	1	
	3	High				2	Medium					1	Low		

Summative Assessment						
Bloom'sLevel	Continuous Assessment					Final Examination (Theory) [50]
	Theory				Practicals	
	IAE-I [7.5]	IAE-II [7.5]	IAE-III [10]	Attendance [5]	Rubric based CIA [20]	
Remember	10	10	10			10
Understand	10	20	20		10	20
Apply	20	20	10		20	50
Analyze	10		10		20	10
Evaluate						10
Create						

TABLE OF CONTENTS

Ex.No	Date	Name of the Experiment	Page No	Marks	Sign
1		CREATION OF A DATABASE AND WRITING SQL QUERIES TO RETRIEVE INFORMATION FROM THE DATABASE.			
2		PERFORMING INSERTION, DELETION, MODIFYING, ALTERING, UPDATING AND VIEWING RECORDS BASED ON CONDITIONS			
3		NESTED QUERIES AND JOIN QUERIES			
4		WRITE A PL/SQL BLOCK TO SATISFY SOME CONDITIONS BY ACCEPTING INPUT FROM THE USER.			
5		A PL/SQL BLOCK THAT HANDLES ALL TYPES OF EXCEPTIONS			
6		CASE STUDY USING REAL LIFE DATABASE APPLICATIONS			
7		XML DOCUMENT CREATION			
8		WRITE RELATIONAL ALGEBRA QUERIES FOR A GIVEN SET OF RELATIONS.			
9		DATABASE DESIGN USING ER MODELING, NORMALIZATION AND IMPLEMENTATION FOR ANY APPLICATION			



INTRODUCTION TO DBMS

OVERVIEW OF RDBMS:

RDBMS is the acronym for Relational Data Base Management System. The concept of relational database is known since 1980's but the idea of Data Base Management System is definitely old. The most famous RDBMS packages are Oracle, Sybase, and Informix.

What is DBMS?

A Data Base Management System is essentially a collection of interrelated data and a set of programs to access this data. This collection of data is called the Database. The Primary objective of a DBMS is to provide a convenient environment to retrieve and store database information. Database systems support single user and multiuser environment. While on one hand DBMS permits only one person to access the database at a given time, on the other RDBMS allows many users simultaneous access to the database.

A Database System consists of two Parts namely, DBMS and Database Application.

DBMS is the program that organizes and maintains the information whereas the Database Application is the program that lets us view, retrieve and update information stored in the DBMS.

DBMS has to protect against unintentional changes that could be caused by users and applications. In case of multi user system, it must be capable of notifying any database change to the other user.

DBMS offers following services:

Data Definition

Data maintenance

Data Manipulation

Data display

Data Integrity

INTRODUCTION TO ORACLE

Every business enterprise maintains large volumes of data for its operations. With more and more people accessing this data for their work the need to maintain its integrity and relevance increases. Normally, with the traditional methods of storing data and information in files, the chance that the data loses its integrity and validity are very high.

Oracle 8 is an Object Relational Database Management System (ORDBMS). It offers capabilities of both relational and object oriented data base systems. In general, objects can be defined as reusable software codes which are location independent and perform a specific task on any application environment with little or no change to the code.

Oracle products are based on a concept known as the client/Server Technology. This concept involves segregating the processing of an application between two systems. One performs all activities related to the database (server) and the other performs activities that help the user to interact with the application (client).

A Client or front-end database application also interacts with the database by requesting and receiving information from the database server. It acts as an interface between the user and the database. Further, it also checks for validation against the data entered by the user. The commonly used front end tools of oracle are SQL * Plus V 8, Oracle forms 5.0 and Reports 3.0.

The database server or backend is used to manage the database tables optimally among multiple clients who concurrently request the server for the same data. It also enforces data integrity across all client applications and controls database access and other security requirements.

Tools of Oracle:

The tools provided by Oracle are so user friendly that a person with minimum skills in the field of computer can access them with ease. The main tools are

SQL * Plus

PL/SQL

Forms

Reports

SQL * Plus:

SQL * Plus is a Structured Query Language supported by Oracle. Through SQL* Plus we can store, retrieve, edit, enter and run SQL commands and PL/SQL blocks. Using SQL * Plus we can perform calculations, list column definitions for any table and can format query results in the form of a report.

PL/SQL:

PL/SQL is an extension of SQL. PL./SQL statements can contain any number of SQL statements integrated with flow of control statements. Thus PL/SQL combines the data manipulating power of SQL with data processing power of procedural languages.

Forms:

This is a graphical tool used for generating and executing form-based applications. A form basically comprises blocks and fields. Multiple tables can be accessed over a single form, based on the application with the help of transaction commands. Oracle Forms Builder is the design component of Oracle forms. We can build, generate and run an Oracle forms application from the builder.

Reports:

It is an application development tool of Oracle for developing, executing, displaying and printing reports. We can create a wide variety of reports, which have various modes. Oracle reports are powerful, yet easy to use.

The reasons for choosing Oracle 8 as the RDBMS for effectively managing the data are

- Ability to retrieve data spread across multiple tables.
 - Oracle specific SQL *Plus functions used when required to query the database
-

especially to decide future course of action.

- Provisions to maintain integrity of the database to avoid data duplication and have constant checks on the validity of data.
- Ability to break the most frequently used object, the table into smaller units thereby reducing the risk to loss of data.
- With a number of clients accessing the database, Oracle allows explicit locking of data. Concurrent access of data for manipulation can be prevented in this way.
- Storing of information out-of-line with the table is also a major advantage. This allows unstructured information to be stored in a different location with the pointers to the location present in the table.
- It supports OOPs concepts, making it a powerful object oriented database.



INTRODUCTION TO SQL

SQL was invented and developed by IBM in early 1970's. SQL stands for Structured Query Language. IBM was able to demonstrate how to control relational database using SQL. The SQL implemented by ORACLE CORPORATION is 100% compliant with the ANSI/ ISO standard SQL data language. Oracle's database language is SQL, which is used for storing and retrieving information in Oracle. A table is a Primary database object of SQL that is used to store data. A table that holds data in the form of rows and columns.

In order to communicate with the database, SQL supports the following categories of commands: -

Data Definition Language - create, alter and drop commands.

Data Manipulation Language - insert, select, delete and update commands.

Transaction Control Language - commit, savepoint and rollback commands.

Data control Language - grant and revoke commands.

Data Definition Language:

DDL is used to create an object (table), alter the structure of an object and also to drop the object created. A table is a unit of storage that holds data in the form of rows and columns. DDL is used for table definition.

Data Manipulation Language:

DML commands are most frequently used SQL commands. They are used to query and manipulate existing objects like tables.

Transaction Control Language:

A transaction is a logical unit of work. All changes made to the database can be referred to as a transaction. Transaction changes can be made permanent to a database only if they are committed. A transaction begins with an executable SQL statement ends explicitly with either rollback or commit commands and implicitly i.e., automatically, when a

DDL statement is used.

The following are the benefits of SQL:

- Non-procedural Language, because more than one record can be accessed rather than one record at a time.
- It is the common language for all relational databases. In other words it is portable and it requires very few modifications so that it can work on other databases.
- Very simple commands for Querying, inserting, deleting and modifying data and objects.

SQL Vs SQL *Plus:

SQL is a standard language common to all relational databases. SQL is a database language used for storing and retrieving data from the database. Most relational Database Management Systems provide extensions to SQL to make it easier for application developers.

SQL *Plus is an Oracle specific Program which accepts SQL commands and PL/SQL blocks and executes them. SQL *Plus enables manipulation of SQL commands and PL/SQL blocks. It also performs many additional tasks as well. Through SQL *Plus we can

- enter, edit, store, retrieve and run SQL commands and PL/SQL blocks.
 - format, perform calculations, store and print query results in the form of reports.
 - list column definitions for any table.
 - Access and copy data between SQL databases.
 - Send messages to and accept responses from an end user.
-

Ex.No:1. CREATION OF A DATABASE AND WRITING SQL QUERIES TO RETRIEVE INFORMATION FROM THE DATABASE.

AIM:

To create a database and write SQL queries to retrieve information from the database.

DESCRIPTION:

Data Definition Language:

DDL (Data Definition Language) statements are used to create, change the objects of a database. Typically a database administrator is responsible for using DDL statements or production databases in a large database system.

The commands used are:

- Create - It is used to create a table.
- Alter - This command is used to add a new column, modify the existing column definition and to include or drop integrity constraint.
- Drop - It will delete the table structure provided the table should be empty.
- Truncate - If there is no further use of records stored in a table and the structure has to be retained, and then the records alone can be deleted.
- Desc - This is used to view the structure of the table.

SYNTAX:

CREATE TABLE:

- create table <table name> (fieldname-1 data type constraints if any,fieldname-2 data type constraints if any,..... fieldname-n data type constraints if any);
- create table <table name> as (select(attribute-list) from <existing table name>);

ALTER TABLE:

- alter table <table name> add/modify (fieldname-1 datatype,fieldname-2 data type,..... fieldname-n data type);
- alter table drop column column name;
Table altered.

DESCRIBING TABLE:

- desc <table name>;
-

CHANGING NAME OF AN OBJECT:

To change the name of a table, view, sequence, or synonym, execute the rename statement.

Syntax: rename old name to new name;

TRUNCATE:

The truncate table statement

- removes all rows from a table
- Release the storage space used by that table

Syntax: truncate table <table name>;

DROP TABLE:

1. All data and structure in the table is deleted
2. Any pending transactions are committed.
3. All indexes are dropped.

Syntax: drop table <table name>;

Table dropped.

SAMPLE OUTPUT:

```
SQL> create table tbl03 (sno number (2), regno number (12), name varchar2 (10), age number (2), marks number (2));
```

Table created.

```
SQL> desc tbl03;
```

Name	Null?	Type

SNO		NUMBER (2)
REGNO		NUMBER (12)
NAME		VARCHAR2 (10)
AGE		NUMBER (2)
MARKS		NUMBER (2)

```
SQL> alter table tbl03 add (total number (3));
```

Table altered.

```
SQL> desc tbl03;
```

Name	Null?	Type

SNO		NUMBER (2)
REGNO		NUMBER (12)
NAME		VARCHAR2 (10)
AGE		NUMBER (2)
MARKS		NUMBER (2)
TOTAL		NUMBER (3)

```
SQL> insert into tbl03 values (&sno, &regno,'&name', &age, &marks, &total);
```

Enter value for sno: 02

Enter value for regno: 003

Enter value for name: Abishek

Enter value for age: 18

Enter value for marks: 99

Enter value for total: 599

old 1: insert into tbl03 values (&sno, ®no,'&name', &age, &marks, &total)

new 1: insert into tbl03 values (02, 003,'Abishek', 18, 99,599)

1 row created.

SQL> insert into tbl03 values (22, 023,'Isai', 18, 98,598);

1 row created.

SQL> select * from tbl03;

SNO	REGNO	NAME	AGE	MARKS	TOTAL
2	003	Abishek	18	99	599
22	023	Iasi	18	98	598

SQL> select name from tbl03;

NAME

Abishek

Isai

SQL> select * from tbl03 where total=599;

SNO	REGNO	NAME	AGE	MARKS	TOTAL
2	003	Abishek	18	99	599

SQL> update tbl03 set age=20 where age=18;

2 rows updated.

SQL> select * from tbl03;

SNO	REGNO	NAME	AGE	MARKS	TOTAL
2	003	Abishek	18	99	599
22	023	Isai	20	98	598

SQL> update tbl03 set total=95 where name='Isai';

1 row updated.

SQL> delete from tbl03 where sno=22;

1 row deleted.

SQL> select * from tbl03;

SNO	REGNO	NAME	AGE	MARKS	TOTAL
2	003	Abishek	18	99	599

SQL> truncate table tbl03;

Table truncated.

SQL> rename tbl03 to tbl03z;

Table renamed.

SQL> drop table tbl03z;

Table dropped.

SQL> select * from tbl03z;

select * from tbl03z

*

ERROR at line 1:

ORA-00942: table or view does not exist

SQL> desc tbl03z;

*

ERROR:

ORA-04043: object tbl03z does not exist

RESULT:

Thus the creation of a database and writing SQL queries to retrieve information from the database was implemented.



Ex.No.2. PERFORMING INSERTION, DELETION, MODIFYING, ALTERING, UPDATING AND VIEWING RECORDS BASED ON CONDITIONS

AIM:

To Perform Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions in RDBMS.

DESCRIPTION:

Data Manipulation Language

DML commands are the most frequently used SQL commands and is used to query and manipulate the existing database objects. Some of the commands are

1. Insert
2. Select
3. Update
4. Delete

SYNTAX:

INSERT: This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes. The values must be entered in the same order as they are defined.

Inserting a single row into a table:

insert into <table name> values(fieldvalue-1,fieldvalue-2,...,fieldvalue-n);

Inserting more than one record using a single insert command:

insert into <table name> values(&fieldname-1,&fieldname-2,...&fieldname-n);

Skipping the fields while inserting:

insert into <tablename(coln names to which data is to be inserted)> values (list of values);

Other way is to give null while passing the values.

insert into <table name>(select(att_list) from <existing table name>);

SELECT: - It is used to retrieve information from the table. It is generally referred to as querying the table. We can either display all columns in a table or only specify column from the table.

SELECT(att_list) FROM <table name> [WHERE <condition/expression>];

Retrieval of all columns from a table:

Select * from tablename; // This query selects all rows from the table.

Retrieval of specific columns from a table: It retrieves the specified columns from the table.

Select column_name1, ...,column_name n from table name;

Elimination of duplicates from the select clause: It prevents retrieving the duplicated values. Distinct keyword is to be used.

Select DISTINCT col1, col2 from table name;

Select command with where clause: To select specific rows from a table we include „where“ clause in the select command. It can appear only after the „from“ clause.

Syntax: Select column_name1,,column_namen from table name where condition;

Select command with order by clause:

Syntax: Select column_name1,,column_namen from table name where condition order by colmnname;

Select command to create a table:

Syntax: create table tablename as select * from existing_tablename;

Select command to insert records:

Syntax: insert into tablename (select columns from existing_tablename);

UPDATE - It is used to alter the column values in a table. A single column may be updated or more than one column could be updated.

update <table name> set (fieldname-1 = value, fieldname-2 = value,...,fieldname-n = value) [WHERE <condition/expression>];

DELETE - After inserting row in a table we can also delete them if required. The delete command consists of a from clause followed by an optional where clause.

delete from <table name> [where <condition/expression>];

ALTER TABLE

- alter table table_name add column_name datatype
- alter table <table name> add (fieldname-1 datatype,fieldname-2 datatype,..... fieldname-n datatype);
- alter table <table name> modify (fieldname-1 data type (new size));
- alter table drop column column name;

Table altered.

SAMPLE OUTPUT:

INSERT, SELECT, UPDATE AND DELETE COMMANDS

```
SQL> create table person(pid int, lastname varchar2(10),firstname varchar(10),  
address varchar2(20),age number);
```

Table created.

INSERTING A SINGLE ROW INTO A TABLE

```
SQL> insert into person values(1,'Prettina','Anne','Bangalore',14);
```

1 row created.

```
SQL> insert into person values(2,'Benitto','Anish','Trichy',24);
```

1 row created.

```
SQL> select * from person;
```

PID LASTNAME FIRSTNAME ADDRESS AGE

1 Prettina Anne Bangalore 14

2 Benitto Anish Trichy 24

INSERTING MORE THAN ONE ROW USING A SINGLE INSERT COMMAND

```
SQL> insert into person values(&pid,&lastname,&firstname,&address,&age);
```

Enter value for pid: 3

Enter value for lastname: Raj

Enter value for firstname: Anita

Enter value for address: Chennai

Enter value for age: 27

old 1: insert into person values(&pid,&lastname,&firstname,&address,&age)

new 1: insert into person values(3,'Raj','Anita','Chennai',27)

1 row created.

```
SQL> /
```

Enter value for pid: 4

Enter value for lastname: kumar

Enter value for firstname: Ashok

Enter value for address: Coimbatore

Enter value for age: 30

old 1: insert into person values(&pid,&lastname,&firstname,&address,&age)

new 1: insert into person values(4,'kumar','Ashok','Coimbatore',30)

1 row created.

```
SQL> select * from person;
```

PID LASTNAME FIRSTNAME ADDRESS AGE

1 Prettina Anne Bangalore 14

2 Benitto Anish Trichy 24

3 Raj Anita Chennai 27

4 kumar Ashok Coimbatore 30

SKIPPING THE FIELDS WHILE INSERTING

SQL> insert into person(pid,lastname,firstname) values(5,Hinn,Benny);

insert into person(pid,lastname,firstname) values(5,'Hinn','Benny')

1 row created.

SQL> select * from person;

PID LASTNAME FIRSTNAME ADDRESS AGE

1 Prettina Anne Bangalore 14

2 Benitto Anish Trichy 24

3 Raj Anita Chennai 27

4 kumar Ashok Coimbatore 30

5 Hinn Benny

INSERT VALUES USING MEANINGFUL FIELD NAMES

SQL> insert into person values(&personid,&lastname,&firstname,&personaddress,&age);

Enter value for personid: 6

Enter value for lastname: Prakash

Enter value for firstname: Bhaskar

Enter value for personaddress: Andhra

Enter value for age: 40

old 1: insert into person values(&personid,&lastname,&firstname,&personaddress,&age)

new 1: insert into person values(6,'Prakash','Bhaskar','Andhra',40)

1 row created.

SQL> select * from person;

PID LASTNAME FIRSTNAME ADDRESS AGE

1 Prettina Anne Bangalore 14

2 Benitto Anish Trichy 24

3 Raj Anita Chennai 27

4 kumar Ashok Coimbatore 30

5 Hinn Benny

6 Prakash Bhaskar Andhra 40

6 rows selected.

UPDATE VALUES USING CONDITION

SQL> update person set address='United States'where pid=5;

1 row updated.

UPDATE VALUES USING &

SQL> update person set address = '&address',age=&age where pid=&pid;

Enter value for address: Assam

Enter value for age: 40

Enter value for pid: 6

old 1: update person set address = '&address',age=&age where pid=&pid

new 1: update person set address = 'Assam',age=40 where pid=6

1 row updated.

SQL> /

Enter value for address: Britain

Enter value for age: 55

Enter value for pid: 5

old 1: update person set address = '&address',age=&age where pid=&pid

new 1: update person set address = 'Britain',age=55 where pid=5

1 row updated.

SELECT COMMAND TO RETRIEVE THE ENTIRE INFORMATION FROM THE TABLE

SQL> select * from person;

PID LASTNAME FIRSTNAME ADDRESS AGE

1 Pretina Anne Bangalore 14

2 Benitto Anish Trichy 24

3 Raj Anita Chennai 27

4 kumar Ashok Coimbatore 30

5 Hinn Benny Britain 55

6 Prakash Bhaskar Assam 40

6 rows selected.

SELECT COMMAND USING 'WHERE' CLAUSE

SQL>select * from person where lastname= 'Kumar' and address='Coimbatore';

PID LASTNAME FIRSTNAME ADDRESS AGE

4 Kumar Ashok Coimbatore 30

7 Kumar Chander Coimbatore 45

SELECT COMMAND TO RETRIEVE THE TOP VALUES

SQL> select * from person where rownum<=3;

PID LASTNAME FIRSTNAME ADDRESS AGE

1 Pretina Anne Bangalore 14

2 Benitto Anish Trichy 24

3 Raj Anita Chennai 27

SELECT COMMAND WITH LIKE OPERATOR

```
SQL> select * from person where address like 'C%';
```

```
PID LASTNAME FIRSTNAME ADDRESS AGE
```

```
-----  
3 Raj Anita Chennai 27
```

```
4 kumar Ashok Coimbatore 30
```

```
SQL> select * from person where address like '%i%';
```

```
PID LASTNAME FIRSTNAME ADDRESS AGE
```

```
-----  
2 Benitto Anish richy 24
```

```
3 Raj Anita Chennai 27
```

```
4 Kumar Ashok Coimbatore 30
```

```
5 Hinn Benny Britain 55
```

SELECT COMMAND USING IN OPERATOR

```
SQL> select * from person where lastname in('Prettina');
```

```
PID LASTNAME FIRSTNAME ADDRESS AGE
```

```
-----  
1 Prettina Anne Bangalore 14
```

SELECT COMMAND USING BETWEEN OPERATOR

```
SQL> insert into person values(7,'Kumar','Chander','Coimbatore',45);
```

```
1 row created.
```

```
SQL> select * from person where lastname between 'Kumar' and 'Kumar';
```

```
PID LASTNAME FIRSTNAME ADDRESS AGE
```

```
-----  
6 Kumar Ashok Coimbatore 30
```

```
7 Kumar Chander Coimbatore 45
```

SELECT COMMAND TO ELIMINATE DUPLICATES

```
SQL> select DISTINCT lastname from person;
```

```
LASTNAME
```

```
-----  
Benitto
```

```
Hinn
```

```
Kumar
```

```
Prakash
```

```
Prettina
```

```
Raj
```

```
6 rows selected.
```

SELECT COMMAND WITH ORDER BY CLAUSE

```
SQL> select pid, firstname,age from person order by age;
```

PID FIRSTNAME AGE

1 Anne 14
2 Anish 24
3 Anita 27
4 Ashok 30
6 Bhaskar 40
7 Chander 45
5 Benny 55
7 rows selected.

SELECT COMMAND TO CREATE A TABLE

SQL> create table individual as select * from person;
Table created.

SQL> select * from individual;

PID LASTNAME FIRSTNAME ADDRESS AGE

1 Prettina Anne Bangalore 14
2 Benitto Anish Trichy 24
3 Raj Anita Chennai 27
4 Kumar Ashok Coimbatore 30
5 Hinn Benny Britain 55
6 Prakash Bhaskar Assam 40
7 Kumar Chander Coimbatore 45
7 rows selected.

SELECT COMMAND TO INSERT RECORDS

SQL> insert into individual(select * from person);
7 rows created.

SELECT COMMAND WITH FUNCTIONS

SQL> select count(*) as pid from person;

PID

7
SQL> select count(distinct lastname) as pid from person;
PID

6
SQL> select max(age) from person;
MAX(AGE)

55

```
SQL> select min(age) from person;  
MIN(AGE)
```

14

```
SQL> select sum(age) from person;  
SUM(AGE)
```

235

DATA CONTROL LANGUAGE (DCL) COMMANDS

```
SQL> select * from person;  
PID LASTNAME FIRSTNAME ADDRESS AGE
```

```
1 Pretina Anne Bangalore 14  
2 Benitto Anish Trichy 24  
3 Raj Anita Chennai 27  
4 Kumar Ashok Coimbatore 30  
5 Hinn Benny Britain 55  
6 Prakash Bhaskar Assam 40  
7 Kumar Chander Coimbatore 45  
7 rows selected.
```

```
SQL> commit;  
Commit complete.
```

DELETE COMMAND

```
SQL> delete from person where lastname='Kumar';  
2 rows deleted.
```

```
SQL> select * from person;  
PID LASTNAME FIRSTNAME ADDRESS AGE
```

```
1 Pretina Anne Bangalore 14  
2 Benitto Anish Trichy 24  
3 Raj Anita Chennai 27  
5 Hinn Benny Britain 55  
6 Prakash Bhaskar Assam 40
```

```
SQL> rollback;  
Rollback complete.
```

```
SQL> select * from person;  
PID LASTNAME FIRSTNAME ADDRESS AGE
```

```
1 Pretina Anne Bangalore 14  
2 Benitto Anish Trichy 24
```

```

3 Raj Anita Chennai 27
4 Kumar Ashok Coimbatore 30
5 Hinn Benny Britain 55
6 Prakash Bhaskar Assam 40
7 Kumar Chander Coimbatore 45
7 rows selected.
SQL> savepoint s1;
Savepoint created.
SQL> delete from person;
7 rows deleted.
SQL> select * from person;
no rows selected
SQL> rollback to savepoint s1;
Rollback complete.
SQL> select * from person;
PID LASTNAME FIRSTNAME ADDRESS AGE
-----
1 Prettina Anne BAngalore 14
2 Benitto Anish Trichy 24
3 Raj Anita Chennai 27
4 Kumar Ashok Coimbatore 30
5 Hinn Benny Britain 55
6 Prakash Bhaskar Assam 40
7 Kumar Chander Coimbatore 45
7 rows selected.

```

RESULT:

Thus the Insertion, Deletion, Modifying, Altering, Updating and Viewing records based on conditions in RDBMS were executed and verified.



AIM:

To write and execute SQL queries for the nested and join queries.

NESTED QUERIES OR SUB QUERIES:**Syntax:**

Select<column(s)>from table where<condn operator> (select <column>from table);

Q1: Create a table employee with attributes ssn, name,bdate,salary,mgrssn,dno with appropriate data type. Insert few records in to the table employee.

Query:

select * from employee;

Output:

SSN NAME BDATE SALARY MGRSSN DNO

1111 sathya 17-DEC-88 50000 4323 3
1112 vinotha 02-AUG-90 32000 5462 1
1113 nandu 07-OCT-93 30000 6452 2
1114 rajesh 05-APR-85 40000 8264 2
1115 nive 06-OCT-92 45000 7241 1

Also create a table department with attributes dno, dname, loc. Insert few records into the department table.

Query:

select * from department;

Output:

DNO DNAME LOC

1 admin madurai
2 research chennai
3 accounts bangalore

Q2: Display the names of the employees working for Accounts department.

Query:

SQL> select name from employee where dno=(select dno from department where dname='accounts');

Output:

NAME

sathya

Q2: Display names of employees whose salary is greater than the employee SSN=1234

Query:

SQL> select name from employee where salary>(select salary from employee where ssn=1234);

Output:

NAME

sathya

vinotha

rajesh

nive

Q3: Display all the employees drawing more than or equal to the average salary of department number 3.

Query:

SQL> select name from employee where salary>=(select avg(salary) from employee group by dno having dno=3);

Output:

NAME

Sathya

Q4: Display the name of the highest paid employee.

Query:

SQL> select name from employee where salary=(select max(salary) from employee);

Output:

NAME

sathya

Q5: Find the Name and Salary of people who draw in the range Rs. 20,000 to Rs. 40,000.

Query:

SQL> select name, salary from employee where salary in(select salary from employee where salary between 20000 and 40000);

Output:

NAME SALARY

vinotha 32000

nandu 30000

rajesh 40000

Q6: Update the salary by 0.25 times for all employees who works in research department.

Query:

SQL> update employee set salary=(salary*0.25)where dno=(select dno from department where dname='research');

Output:

2 rows updated.

SQL> select * from employee;

SSN NAME BDATE SALARY MGRSSN DNO

1111 sathya 17-DEC-88 50000 4323 3

1112 vinotha 02-AUG-90 32000 5462 1

1113 nandu 07-OCT-93 37500 6452 2

1114 rajesh 05-APR-85 50000 8264 2

JOIN QUERIES

An SQL join clause combines records from two or more tables in a database. It creates a set that can be saved as a table or used as it is. A JOIN is a means for combining fields from two tables by using values common to each. Types are:

- Cross join
- Inner Join
 - Equi Join - Natural Join
- Outer Join

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

(i). Cross Join

CROSS JOIN returns the Cartesian product of rows from tables in the join. In other words, it will produce rows which combine each row from the first table with each row from the second table

Syntax:

Select * from table1 cross join table2;

(or)

Select * from table1,table2;

Query:

SQL> select * from employee cross join department;

ii). Inner Join

Inner join creates a new result table by combining column values of two tables (A and B) based upon the join-predicate. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row. The result of the join can be defined as the outcome of first taking the Cartesian product (or Cross join) of all records in the tables (combining every record in table A with every record in table B)—then return all records which satisfy the join predicate

Syntax:

Select * from table1 inner join table2 on table1.column=table2.column;

Query:

**SQL> select * from employee inner join department on
employee.dno=department.dno;**

(iii). Equi Join

An **equi-join** is a specific type of comparator-based join, that uses only equality comparisons in the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equi-join.

Syntax:

Select * from table1 join table2 on table1.column=table2.column;

Query:

**SQL> select * from employee join department on
employee.dno=department.dno;**

(iv). Natural Join

A natural join is a type of equi-join where the join predicate arises implicitly by comparing all columns in both tables that have the same column-names in the joined tables. The resulting joined table contains only one column for each pair of equally named columns.

Syntax:

Select * from table1 natural join table2 ;

Query:

Sql> select * from employee natural join department;

(v). Outer Join

An outer join does not require each record in the two joined tables to have a matching record. The joined table retains each record—even if no other matching record exists. Outer joins subdivide further into left outer joins, right outer joins, and full outer joins, depending on which table's rows are retained (left, right, or both).

Left Outer Join

The result of a left outer join (or simply left join) for table A and B always contains all records of the "left" table (A), even if the join-condition does not find any matching record in the "right" table (B).

Syntax:

Select * from table1 left outer join table2 on table1.column=table2.column;

Query:

Sql> select * from employee left outer join department on employee.dno=department.dno;

Right Outer Join

A right outer join returns all the values from the right table and matched values from the left table (NULL in case of no matching join predicate).

Syntax:

Select * from table1 right outer join table2 on table1.column=table2.column;

Query:

Sql> select * from employee right outer join department on employee.dno=department.dno;

Full Outer Join

Conceptually, a **full outer join** combines the effect of applying both left and right outer joins. Where records in the FULL OUTER JOINed tables do not match, the result set will have NULL values for every column of the table that lacks a matching row. For those records that do match, a single row will be produced in the result set (containing fields populated from both tables).

Syntax:

Select * from table1 full outer join table2 on table1.column=table2.column;

Query:

Sql> select * from employee full outer join department on employee.dno=department.dno;

RESULT:

Thus the Nested queries and join queries was executed successfully.

Ex.No.4

**WRITE A PL/SQL BLOCK TO SATISFY SOME CONDITIONS BY
ACCEPTING INPUT FROM THE USER.**

AIM:

To write a PL/SQL block to satisfy some conditions by accepting input from the user using oracle.

DESCRIPTION:

PL/SQL Control Structure provides conditional tests, loops, flow control and branches that let to produce well-structured programs.

SYNTAX:

```
DECLARE
    Variable declaration
BEGIN
    Program Execution
EXCEPTION
    Exception handling
END;
```

PL/ SQL GENERAL SYNTAX

```
SQL> declare
<variable declaration>;
begin
<executable statement >;
end;
```

PL/ SQL GENERAL SYNTAX FOR IF CONDITION

```
SQL> declare
<variable declaration>;
begin
if(condition) then
```



PL/ SQL GENERAL SYNTAX FOR IF AND ELSECONDITION

SQL> declare

<variable declaration>;

begin

if (test condition) then

<statements>;

else

<statements>;

end if;

end;

PL/ SQL GENERAL SYNTAX FOR NESTED IF CONDITION

SQL> declare

<variable declaration>;

begin

if (test condition) then

<statements>;

else if (test condition) then

<statements>;

else

<statements>;

end if;

end;

PL/ SQL GENERAL SYNTAX FOR LOOPING STATEMENT

SQL> declare

<variable declaration>;

begin

loop

<statement>;

end loop;



PL/ SQL GENERAL SYNTAX FOR LOOPING STATEMENT

```
SQL> declare
<variable declaration>;
begin
while <condition>
loop
<statement>;
end loop;
<executable statement>;
end;
```

PL/SQL CODING FOR ADDITION OF TWO NUMBERS

(Write a PL/SQL Program for Addition of Two numbers)

PROCEDURE

STEP 1: Start

STEP 2: Initialize the necessary variables.

STEP 3: Develop the set of statements with the essential operational parameters.

STEP 4: Specify the Individual operation to be carried out.

STEP 5: Execute the statements.

STEP 6: Stop.

PROGRAM

```
SQL>set serveroutput on
SQL>declare
1 a number;
2 b number;
3 c number;
4 begin
5 a: =&a;
6 b: =&b;
7 c: =a+b;
8 dbms_output.put_line ('sum of'||a||'and'||b||'is'||c);
```

9 end;

10 /

INPUT

Enter value for a: 23

old 6: a:=&a;

new 6: a:=23;

Enter value for b: 12

old 7: b:=&b;

new 7: b:=12;

OUTPUT

sum of 23 and 12 is 35

PL/SQL procedure successfully completed.

PL/ SQL PROGRAM FOR IF CONDITION

(Write a PL/SQL Program to find out the maximum value using if condition)

PROCEDURE

STEP 1: Start

STEP 2: Initialize the necessary variables.

STEP 3: invoke the if condition.

STEP 4: Execute the statements.

STEP 5: Stop.

PROGRAM

SQL>set serveroutput on

SQL> declare

2 b number;

3 c number;

4 BEGIN

5 B:=10;

6 C:=20;

7 if(C>B) THEN

8 dbms_output.put_line('C is maximum');

9 end if;

10 end;

11 /

OUTPUT

C is maximum

PL/SQL procedure successfully completed.

PL/ SQL PROGRAM FOR IF ELSE CONDITION

(Write a PL/SQL Program to check whether the value is less than or greater than 5 using if else condition)

PROCEDURE

STEP 1: Start

STEP 2: Initialize the necessary variables.

STEP 3: invoke the if else condition.

STEP 4: Execute the statements.

STEP 5: Stop.

PROGRAM

SQL>set serveroutput on

SQL> declare

2 n number;

3 begin

4 dbms_output.put_line('enter a number');

5 n:=&number;

6 if n<5 then

7 dbms_output.put_line('entered number is less than 5');

8 else

9 dbms_output.put_line('entered number is greater than 5');

10 end if;

11 end;

12 /

INPUT

Enter value for number: 2

old 5: n:=&number;

new 5: n:=2;

OUTPUT

entered number is less than 5

PL/SQL procedure successfully completed.

PL/ SQL PROGRAM FOR IF ELSE IF CONDITION

(Write a PL/SQL Program to find the greatest of three numbers using if else if)

PROCEDURE

STEP 1: Start

STEP 2: Initialize the necessary variables.

STEP 3: invoke the if else if condition.

STEP 4: Execute the statements.

STEP 5: Stop.

PROGRAM

SQL>set server output on

SQL> declare

2 a number;

3 b number;

4 c number;

5 begin

6 a:=&a;

7 b:=&b;

8 c:=&c;

9 if(a>b)and(a>c) then

10 dbms_output.put_line('A is maximum');

11 else if(b>a)and(b>c)then

12 dbms_output.put_line('B is maximum');

13 else

14 dbms_output.put_line('C is maximum');

15 end if;

16 end;

17 /

INPUT

Enter value for a: 21

old 7: a:=&a;

new 7: a:=21;

Enter value for b: 12

old 8: b:=&b;

new 8: b:=12;

Enter value for b: 45

old 9: c:=&b;

new 9: c:=45;

OUTPUT

C is maximum

PL/SQL procedure successfully completed.

PL/ SQL PROGRAM FOR LOOPING STATEMENT

(Write a PL/SQL Program to find the summation of odd numbers using for loop)

PROCEDURE

STEP 1: Start

STEP 2: Initialize the necessary variables.

STEP 3: invoke the for loop condition.

STEP 4: Execute the statements.

STEP 5: Stop.

PROGRAM

SQL>set server output on

SQL> declare

2 n number;

3 sum1 number default 0;

4 end value number;

```
5 begin
6 end value:=&end value;
7 n:=1;
8 for n in 1..endvalue
9 loop
10 if mod(n,2)=1
11 then
12 sum1:=sum1+n;
13 end if;
14 end loop;
15 dbms_output.put_line('sum ='||sum1);
16 end;
17 /
```

INPUT

Enter value for end value: 4

old 6: end value:=&end value;

new 6: end value:=4;

OUTPUT

sum =4

PL/SQL procedure successfully completed.

PL/ SQL PROGRAM FOR LOOPING STATEMENT

(Write a PL/SQL Program to find the factorial of given number using for loop)

PROCEDURE:

STEP 1: Start

STEP 2: Initialize the necessary variables.

STEP 3: invoke the for loop condition.

STEP 4: Execute the statements.

STEP 5: Stop.



PROGRAM

SQL>set server output on

SQL>1 declare

2 n number;

3 i number;

4 p number:=1;

5 begin

6 n:=&n;

7 for i in 1..n loop

8 p:=p*i;

9 end loop;

10 dbms_output.put_line(n || ' ! = ' || p);

11* end;

Enter value for n: 5

old 6: n:=&n;

new 6: n:=5;

5 ! = 120

PL/SQL procedure successfully completed.

RESULT:

Thus a PL/SQL block to satisfy some conditions by accepting input from the user was created using oracle.

**Ex.No.5 WRITE A PL/SQL BLOCK THAT HANDLES ALL TYPES OF
EXCEPTIONS**

AIM:

To implement and execute PL/SQL Block that handles all types of exceptions in Oracle Database using Procedural Language concepts.

DESCRIPTION:

EXCEPTIONS

In PL/SQL, the user can catch certain runtime errors. Exceptions can be internally defined by Oracle or the user. Exceptions are used to handle errors that occur in your PL/SQL code. A PL/SQL block contains an EXCEPTION block to handle exception.

There are three types of exceptions:

1. Predefined Oracle errors
2. Undefined Oracle errors
3. User-defined errors

The different parts of the exception.

1. Declare the exception.
2. Raise an exception.
3. Handle the exception.

An exception has four attributes:

1. Name provides a short description of the problem.
2. Type identifies the area of the error.
3. Exception Code gives a numeric representation of the exception.
4. Error message provides additional information about the exception.

The predefined divide-by-zero exception has the following values for the attributes:

1. Name = ZERO_DIVIDE
2. Type = ORA (from the Oracle engine)
3. Exception Code = C01476

Error message = divisor is equal to zero

EXCEPTION HANDLING

PL/SQL provides a feature to handle the Exceptions which occur in a PL/SQL Block known as exception Handling. Using Exception Handling we can test the code and avoid it from exiting abruptly. When an exception occurs messages which explains its cause is received.

PL/SQL Exception message consists of three parts.

Type of Exception

An Error Code

A message

STRUCTURE OF EXCEPTION HANDLING

GENERAL SYNTAX FOR CODING THE EXCEPTION SECTION

DECLARE

Declaration section

BEGIN

Exception section

EXCEPTION

WHEN ex_name1 THEN

-Error handling statements

WHEN ex_name2 THEN

-Error handling statements

WHEN Others THEN

-Error handling statements

END;

Types of Exception

There are 2 types of Exceptions.

a) System Exceptions

b) User-defined Exceptions

a) System Exceptions

System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule. There are some system exceptions which are raised frequently, so they are pre-defined and given a name in Oracle which are known as Named System Exceptions.

For example: NO_DATA_FOUND and ZERO_DIVIDE are called Named System exceptions.

For Example: Suppose a NO_DATA_FOUND exception is raised in a proc, we can write a code to handle the exception as given below.

```
BEGIN
    Execution section
EXCEPTION
WHEN NO_DATA_FOUND THEN
    dbms_output.put_line ('A SELECT...INTO did not return any row. ');
END;
```

b) User-defined Exceptions

PL/SQL allows us to define our own exceptions according to the need of our program. A user-defined exception must be declared and then raised explicitly, using a RAISE statement.

To define an exception we use EXCEPTION keyword as below:

```
EXCEPTION_NAME EXCEPTION;
```

To raise exception that we've defined to use the RAISE statement as follows:

```
RAISE EXCEPTION_NAME
```

Raising Exceptions

Exceptions are raised by the database server automatically whenever there is any internal database error, but exceptions can be raised explicitly by the programmer by using the command

RAISE. Following is the simple syntax of raising an exception:

```
DECLARE
    exception_name EXCEPTION;
BEGIN
    IF condition THEN
        RAISE exception_name;
    END IF;
EXCEPTION
    WHEN exception_name THEN
        statement;
END;
```

TYPES OF MORE COMMONLY USED EXCEPTIONS

NO_DATA_FOUND	Singleton SELECT statement returned no data.
TOO_MANY_ROWS	Singleton SELECT statement returned more than one row of data.
INVALID_CURSOR	Illegal cursor operation occurred.
VALUE_ERROR	Arithmetic, conversion, or truncation error occurred.
INVALID_NUMBER	Conversion of a number to a character string failed.
ZERO_DIVIDE	Attempted to divide by zero.
DUP_VAL_ON_INDEX	Attempted to insert a duplicate value into a column that has a unique index.
CURSOR_ALREADY_OPEN	Attempted to open a cursor that was previously opened.
NOT_LOGGED_ON	A database call was made without being logged into Oracle.
TRANSACTION_BACKED_OUT	Usually raised when a remote portion of a transaction is rolled back.
LOGIN_DENIED	Login to Oracle failed.
PROGRAM_ERROR	If PL/SQL encounters an internal problem.
STORAGE_ERROR	If PL/SQL runs out of memory or if memory is corrupted.
TIMEOUT_ON_RESOURCE	Timeout occurred while Oracle was waiting for a resource.
OTHERS	For all of the rest.

PROGRAM

ZERO_DIVIDE EXCEPTION

SQL> BEGIN

2 DBMS_OUTPUT.PUT_LINE(1 / 0);

3 END;

4 /

OUTPUT

begin

*

ERROR at line 1:

ORA-01476: divisor is equal to zero

ORA-06512: at line 2

```
-----  
BEGIN  
2  DBMS_OUTPUT.PUT_LINE(1 / 0);  
3  EXCEPTION  
4  WHEN ZERO_DIVIDE THEN  
5  DBMS_OUTPUT.PUT_LINE('Division by zero');  
6  END;  
7 /
```

Division by zero:

PL/SQL procedure successfully completed.

INVALID_NUMBER EXCEPTION

```
1  BEGIN  
2  INSERT INTO employees(DEPARTMENT_ID)VALUES('101x');  
3  EXCEPTION  
4  WHEN INVALID_NUMBER THEN  
5  DBMS_OUTPUT.PUT_LINE('Conversion of string to number failed');  
6* end;
```

SQL> /

Conversion of string to number failed

PL/SQL procedure successfully completed.

OTHERS EXCEPTION

```
1  BEGIN  
2  DBMS_OUTPUT.PUT_LINE(1 / 0);  
3  EXCEPTION  
4  WHEN OTHERS THEN  
5  DBMS_OUTPUT.PUT_LINE('An exception occurred');  
6*  END;  
7 /
```

An exception occurred

PL/SQL procedure successfully completed.

FIRST CREATE A TABLE NAMED CUSTOMERSS WTH ATTRIBUTE ID, NAME, ADDRESS AND THEN IMPLEMENT THE FOLLOWING CODE:

SQL>declare

c_id customerss.id%type;

c_name customerss.name%type;

c_addr customerss.address%type;

begin

SELECT name,address INTO c_name,c_addr FROM customerss WHERE id=c_id;

dbms_output.put_line('Name: ' || c_name);

dbms_output.put_line('Address: ' || c_addr);

EXCEPTION

WHEN no_data_found THEN

dbms_output.put_line('No such customer!');

WHEN others THEN

dbms_output.put_line('Error!');

END;

/

OUTPUT:

'No such customer

PROGRAM:

(The following example illustrates the programmer-defined exceptions. Get the salary of an employee and check it with the job's salary range. If the salary is below the range, an exception BELOW_SALARY_RANGE is raised. If the salary is above the range, exception ABOVE_SALARY_RANGE is raised)

SET SERVEROUTPUT ON SIZE 100000;

DECLARE

-- define exceptions

BELOW_SALARY_RANGE EXCEPTION;

ABOVE_SALARY_RANGE EXCEPTION;

-- salary variables

n_salary employees.salary%TYPE;

n_min_salary employees.salary%TYPE;

n_max_salary employees.salary%TYPE;

-- input employee id

```

n_emp_id employees.employee_id%TYPE := &emp_id;
BEGIN
SELECT salary, min_salary max_salary INTO n_salary, n_min_salary, n_max_salary
FROM employees
INNER JOIN jobs ON jobs.job_id = employees.job_id
WHERE employee_id = n_emp_id;

IF n_salary < n_min_salary THEN
    RAISE BELOW_SALARY_RANGE;
ELSIF n_salary > n_max_salary THEN
    RAISE ABOVE_SALARY_RANGE;
END IF;
dbms_output.put_line('Employee ' || n_emp_id ||
                    ' has salary $' || n_salary );

EXCEPTION  WHEN BELOW_SALARY_RANGE THEN
    dbms_output.put_line('Employee ' || n_emp_id ||
                    ' has salary below the salary range');
WHEN ABOVE_SALARY_RANGE THEN
    dbms_output.put_line('Employee ' || n_emp_id ||
                    ' has salary above the salary range');
WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('Employee ' || n_emp_id || ' not found');
END;
/

```

RESULT:

Thus a PL/SQL block that handles all type of exceptions was written, executed and verified successfully.



EX. NO 7 DATABASE DESIGN USING ER MODELING, NORMALIZATION AND IMPLEMENTATION FOR LIBRARY MANAGEMENT SYSTEM

Consider the following schema for a Library Database:

BOOK (*Book_id*, Title, Publisher_Name, Pub_Year)

BOOK_AUTHORS (*Book_id*, Author_Name)

PUBLISHER (*Name*, Address, Phone)

BOOK_COPIES (*Book_id*, Branch_id, No-of_Copies)

BOOK_LENDING (*Book_id*, Branch_id, Card_No, Date_Out, Due_Date)

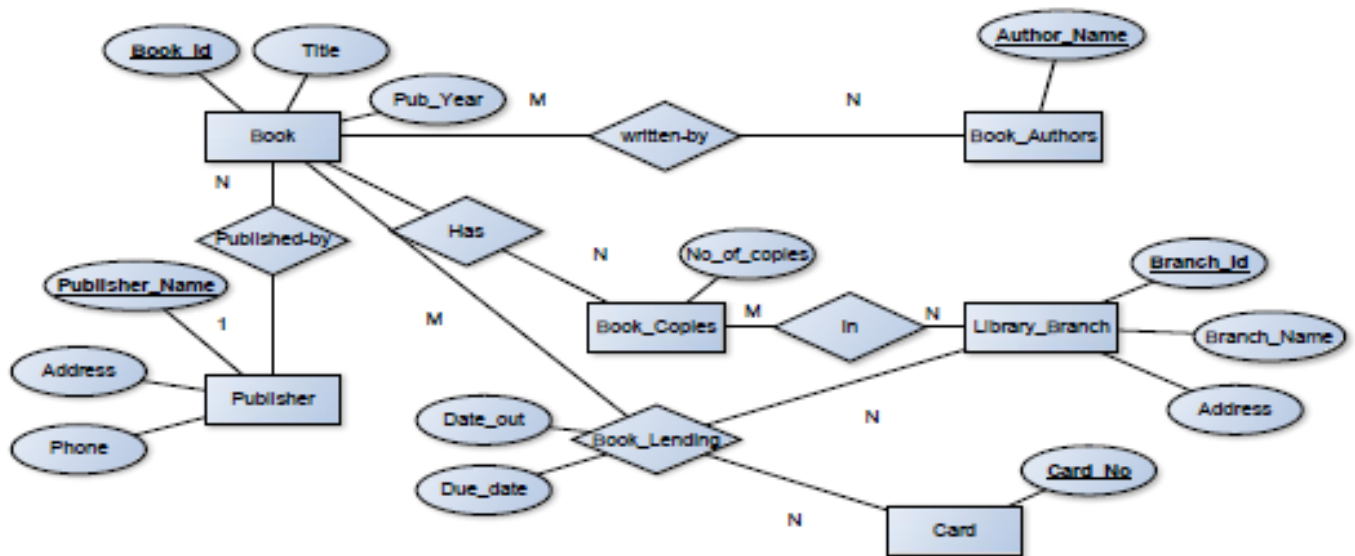
LIBRARY_BRANCH (*Branch_id*, Branch_Name, Address)

Write SQL queries to

1. Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each branch, etc.
2. Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun 2017
3. Create a view of all books and its number of copies that are currently available in the Library.

Solution:

Entity-Relationship Diagram



Schema Diagram

Book

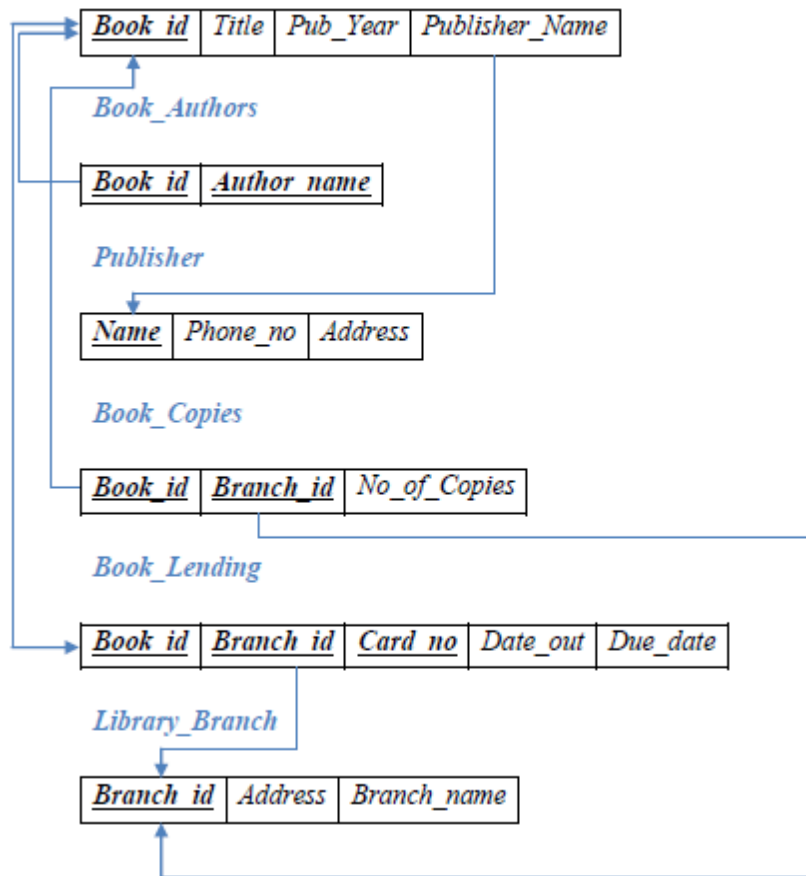


Table Creation

```
CREATE TABLE PUBLISHER (NAME VARCHAR2 (20) PRIMARY KEY, PHONE INTEGER, ADDRESS VARCHAR2 (20));
```

```
CREATE TABLE BOOK (BOOK_ID INTEGER PRIMARY KEY, TITLE VARCHAR2 (20), PUB_YEAR VARCHAR2 (20), PUBLISHER_NAME REFERENCES PUBLISHER (NAME) ON DELETE CASCADE);
```

```
CREATE TABLE BOOK_AUTHORS (AUTHOR_NAME VARCHAR2 (20), BOOK_ID REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, PRIMARY KEY (BOOK_ID, AUTHOR_NAME));
```

```
CREATE TABLE LIBRARY_BRANCH (BRANCH_ID INTEGER PRIMARY KEY, BRANCH_NAME VARCHAR2 (50), ADDRESS VARCHAR2 (50));
```

```
CREATE TABLE BOOK_COPIES (NO_OF_COPIES INTEGER, BOOK_ID REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, BRANCH_ID REFERENCES LIBRARY_BRANCH (BRANCH_ID) ON DELETE CASCADE, PRIMARY KEY (BOOK_ID, BRANCH_ID));
```

```
CREATE TABLE CARD (CARD_NO INTEGER PRIMARY KEY);
```

```
CREATE TABLE BOOK_LENDING (DATE_OUT DATE, DUE_DATE DATE, BOOK_ID
REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE, BRANCH_ID REFERENCES
LIBRARY_BRANCH (BRANCH_ID) ON DELETE CASCADE, CARD_NO REFERENCES CARD
(CARD_NO) ON DELETE CASCADE, PRIMARY KEY (BOOK_ID, BRANCH_ID, CARD_NO));
```

Insertion of Values to Tables

```
INSERT INTO PUBLISHER VALUES (_MCGRAW-HILL', 9989076587, _BANGALORE');
INSERT INTO PUBLISHER VALUES (_PEARSON', 9889076565, _NEWDELHI');
INSERT INTO PUBLISHER VALUES (_RANDOM HOUSE', 7455679345, _HYDRABAD');
INSERT INTO PUBLISHER VALUES (_HACHETTE LIVRE', 8970862340, _CHENAI');
INSERT INTO PUBLISHER VALUES (_GRUPO PLANETA', 7756120238, _BANGALORE');
INSERT INTO BOOK VALUES (1,'DBMS','JAN-2017', _MCGRAW-HILL');
INSERT INTO BOOK VALUES (2,'ADBMS','JUN-2016', _MCGRAW-HILL');
INSERT INTO BOOK VALUES (3,'CN','SEP-2016', _PEARSON');
INSERT INTO BOOK VALUES (4,'CG','SEP-2015', _GRUPO PLANETA');
INSERT INTO BOOK VALUES (5,'OS','MAY-2016', _PEARSON');
INSERT INTO BOOK_AUTHORS VALUES ('NAVATHE', 1);
INSERT INTO BOOK_AUTHORS VALUES ('NAVATHE', 2);
INSERT INTO BOOK_AUTHORS VALUES ('TANENBAUM', 3);
INSERT INTO BOOK_AUTHORS VALUES ('EDWARD ANGEL', 4);
INSERT INTO BOOK_AUTHORS VALUES ('GALVIN', 5);
INSERT INTO LIBRARY_BRANCH VALUES (10,'RR NAGAR','BANGALORE');
INSERT INTO LIBRARY_BRANCH VALUES (11,'RNSIT','BANGALORE');
INSERT INTO LIBRARY_BRANCH VALUES (12,'RAJAJI NAGAR', 'BANGALORE');
INSERT INTO LIBRARY_BRANCH VALUES (13,'NITTE','MANGALORE');
INSERT INTO LIBRARY_BRANCH VALUES (14,'MANIPAL','UDUPI');
INSERT INTO BOOK_COPIES VALUES (10, 1, 10);
INSERT INTO BOOK_COPIES VALUES (5, 1, 11);
INSERT INTO BOOK_COPIES VALUES (2, 2, 12);
INSERT INTO BOOK_COPIES VALUES (5, 2, 13);
INSERT INTO BOOK_COPIES VALUES (7, 3, 14);
INSERT INTO BOOK_COPIES VALUES (1, 5, 10);
INSERT INTO BOOK_COPIES VALUES (3, 4, 11);
INSERT INTO CARD VALUES (100);
INSERT INTO CARD VALUES (101);
INSERT INTO CARD VALUES (102);
INSERT INTO CARD VALUES (103);
INSERT INTO CARD VALUES (104);
INSERT INTO BOOK_LENDING VALUES ('01-JAN-17','01-JUN-17', 1, 10, 101);
INSERT INTO BOOK_LENDING VALUES ('11-JAN-17','11-MAR-17', 3, 14, 101);
INSERT INTO BOOK_LENDING VALUES ('21-FEB-17','21-APR-17', 2, 13, 101);
INSERT INTO BOOK_LENDING VALUES ('15-MAR-17','15-JUL-17', 4, 11, 101);
INSERT INTO BOOK_LENDING VALUES (_12-APR-17','12-MAY-17', 1, 11, 104);
```

1. Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each branch, etc.


```
SELECT B.BOOK_ID, B.TITLE, B.PUBLISHER_NAME, A.AUTHOR_NAME, C.NO_OF_COPIES,  
L.BRANCH_ID  
FROM BOOK B, BOOK_AUTHORS A, BOOK_COPIES C, LIBRARY_BRANCH L WHERE  
BOOK_ID=A.BOOK_ID AND B.BOOK_ID=C.BOOK_ID AND L.BRANCH_ID=C.BRANCH_ID;
```

2. Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun 2017.

```
SELECT CARD_NO FROM BOOK_LENDING WHERE DATE_OUT BETWEEN '01-JAN-2017' AND  
'01-JUL-2017' GROUP BY CARD_NO HAVING COUNT (*)>3;
```

3. Create a view of all books and its number of copies that are currently available in the Library.

```
CREATE VIEW V_BOOKS AS SELECT B.BOOK_ID, B.TITLE, C.NO_OF_COPIES FROM BOOK B,  
BOOK_COPIES C, LIBRARY_BRANCH L WHERE B.BOOK_ID=C.BOOK_ID AND  
BRANCH_ID=L.BRANCH_ID;
```

RESULT:

Thus the database design using ER modeling, normalization and Implementation for Library Management System was implemented and executed successfully.

**Ex.No: 6 CASE STUDY USING REAL LIFE DATABASE APPLICATIONS
DESIGN AND IMPLEMENTATION OF BANKING SYSTEM**

AIM:

To design the banking system in visual basic using ORACLE as backend

DESCRIPTION:

PROCEDURE FOR CREATING TABLE:

1. Create table with following fields

NAME	NULL?	TYPE

ACCNO		NUMBER(10)
CUSTNAME		VARCHAR2(10)
CUSTCITY		VARCHAR2(10)
AMOUNT		NUMBER(10)

2. Insert all possible values into the table.

3. Enter commit work command.

ALGORITHM FOR ADO CONNECTION:

After creating the table in ORACLE, go to start menu

1. Start --> Settings --> Control Panel --> Administrative tools --> Data Sources(ODBC) --> User DSN --> Add --> Select ORACLE database driver --> OK.

2. One new window will appear. In that window, type data source name as table name created in ORACLE. Type user name as secondcsea.

ALGORITHM FOR ADODC IN VISUAL BASIC:

1. In visual basic create tables, command buttons and then text boxes.
 2. In visual basic, go to start menu.
 3. Projects --> Components --> Microsoft ADO Data Control 6.0 for OLEDB --> OK.
 4. Now ADODC Data Control available in tool box.
 5. Drag and drop the ADODC Data Control in the form.
 6. Right click in ADODC Data Control, then click ADODC properties.
 7. One new window will appear.
-

8. Choose general tab, select ODBC Data Sources name as the table name created in ORACLE
9. Choose authentication tab and select username password as secondcsea and secondcsea
10. Choose record name-->select command type as adcmdTable.
11. Select table or store procedure name as table created in ORACLE.
12. Click Apply-->OK
13. Set properties of each text box.
14. Select the data source as ADODC1.
15. Select the Data field and set the required field name created in table

VB SCRIPT:

FIRST:

```
Private Sub First_Click()  
Adodc1.Recordset.MoveFirst  
End Sub
```

LAST:

```
Private Sub Last_Click()  
Adodc1.Recordset.MoveLast  
End Sub
```

NEXT:

```
Private Sub Next_Click()  
Adodc1.Recordset.MoveNext  
End Sub
```

PREVIOUS:

```
Private Sub Previous_Click()  
Adodc1.Recordset.MovePrevious  
End Sub
```

DEPOSIT:

```
Private Sub Deposit_Click()  
Dim N1 as string  
N = InputBox ("Enter the accno")
```

```
Adodc1.Recordset.Find "accno=" &  
N1 = InputBox ("Enter the amount")  
Text4.Text = val (Text4.Text) + N1  
Adodc1.Recordset.Update  
End Sub
```

WITHDRAW:

```
Private Sub Withdraw_Click()  
Dim N1 as string  
N = InputBox ("Enter the accno")  
Adodc1.Recordset.Find "accno=" & N  
N1 = InputBox ("Enter the amount")  
Text4.Text: val (Text4.Text) - N1  
Adodc1.Recordset.Update  
End Sub
```

EXIT:

```
Private Sub Add_Click()  
Unload Me  
End Sub
```



OUTPUT:

START UP FORM

DEPOSIT FORM



SAVE FORM

RESULT:

Thus the banking system was designed in Visual Basic using ORACLE as backend.

AIM:

Ex.No: 7 XML DOCUMENT CREATION

To create a simple XML document to display the address book.

Procedure:

Step-1: Create the xml document using notepad.

Step-2: Create the addressbook as a root tag.

Step-3: Followed by create a three employees address in detail using different tag.

Step-4: Save the file as “Employee.xml”.

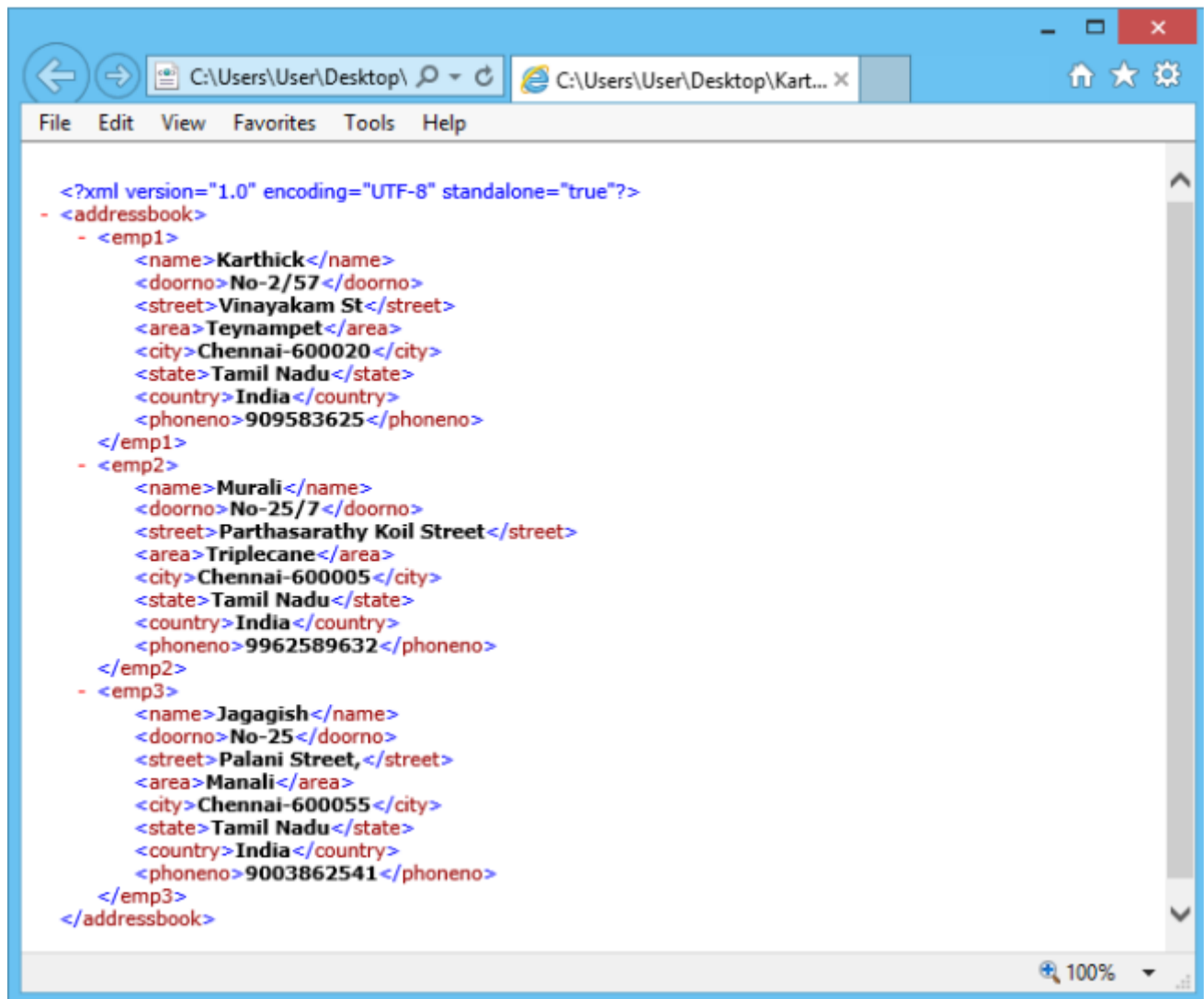
Step-5: Finally execute the program in Internet Explorer to view the output.

Source Code:

Filename: “Employee.xml”

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes" ?>
<addressbook>
  <emp1>
    <name>Karthick</name>
    <doorno>No-2/57</doorno>
    <street>Vinayakam St</street>
    <area>Teynampet</area>
    <city>Chennai-600020</city>
    <state>Tamil Nadu</state>
    <country>India</country>
    <phoneno>909583625</phoneno>
  </emp1>
  <emp2>
    <name>Murali</name>
    <doorno>No-25/7</doorno>
    <street>ParthasarathyKoil Street</street>
    <area>Triplecane</area>
    <city>Chennai-600005</city>
    <state>Tamil Nadu</state>
    <country>India</country>
    <phoneno>9962589632</phoneno>
  </emp2>
  <emp3>
    <name>Jagagish</name>
    <doorno>No-25</doorno>
    <street>Palani Street,</street>
    <area>Manali</area>
    <city>Chennai-600055</city>
    <state>Tamil Nadu</state>
    <country>India</country>
    <phoneno>9003862541</phoneno>
  </emp3>
</addressbook>
```

Output:

A screenshot of a web browser window. The address bar shows the file path 'C:\Users\User\Desktop\Kart...'. The browser has a menu bar with 'File', 'Edit', 'View', 'Favorites', 'Tools', and 'Help'. The main content area displays an XML document with the following structure:

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <addressbook>
  - <emp1>
    <name>Karthick</name>
    <doorno>No-2/57</doorno>
    <street>Vinayakam St</street>
    <area>Teynampet</area>
    <city>Chennai-600020</city>
    <state>Tamil Nadu</state>
    <country>India</country>
    <phoneno>909583625</phoneno>
  </emp1>
  - <emp2>
    <name>Murali</name>
    <doorno>No-25/7</doorno>
    <street>Parthasarathy Koil Street</street>
    <area>Triplecane</area>
    <city>Chennai-600005</city>
    <state>Tamil Nadu</state>
    <country>India</country>
    <phoneno>9962589632</phoneno>
  </emp2>
  - <emp3>
    <name>Jagagish</name>
    <doorno>No-25</doorno>
    <street>Palani Street,</street>
    <area>Manali</area>
    <city>Chennai-600055</city>
    <state>Tamil Nadu</state>
    <country>India</country>
    <phoneno>9003862541</phoneno>
  </emp3>
</addressbook>
```

 The status bar at the bottom right shows a magnifying glass icon and '100%'.

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <addressbook>
  - <emp1>
    <name>Karthick</name>
    <doorno>No-2/57</doorno>
    <street>Vinayakam St</street>
    <area>Teynampet</area>
    <city>Chennai-600020</city>
    <state>Tamil Nadu</state>
    <country>India</country>
    <phoneno>909583625</phoneno>
  </emp1>
  - <emp2>
    <name>Murali</name>
    <doorno>No-25/7</doorno>
    <street>Parthasarathy Koil Street</street>
    <area>Triplecane</area>
    <city>Chennai-600005</city>
    <state>Tamil Nadu</state>
    <country>India</country>
    <phoneno>9962589632</phoneno>
  </emp2>
  - <emp3>
    <name>Jagagish</name>
    <doorno>No-25</doorno>
    <street>Palani Street,</street>
    <area>Manali</area>
    <city>Chennai-600055</city>
    <state>Tamil Nadu</state>
    <country>India</country>
    <phoneno>9003862541</phoneno>
  </emp3>
</addressbook>
```

Result:

Thus, the simple xml document creation has been executed successfully.

Ex.No: 8 Write relational algebra queries for a given set of relations.

Consider the MOVIE DATABASE

Movies				Actors	
title	director	myear	rating	actor	ayear
Fargo	Coen	1996	8.2	Cage	1964
Raising Arizona	Coen	1987	7.6	Hanks	1956
Spiderman	Raimi	2002	7.4	Maguire	1975
Wonder Boys	Hanson	2000	7.6	McDormand	1957

Acts		Directors	
actor	title	director	dyear
Cage	Raising Arizona	Coen	1954
Maguire	Spiderman	Hanson	1945
Maguire	Wonder Boys	Raimi	1959
McDormand	Fargo		
McDormand	Raising Arizona		
McDormand	Wonder Boys		

Write following relational algebra queries for a given set of relations.

1. Find movies made after 1997
2. Find movies made by Hanson after 1997
3. Find all movies and their ratings
4. Find all actors and directors
5. Find Coen's movies with McDormand

SOLUTION:**Common notations of Relational Algebra**

Operation	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(\cup)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference($-$)	- Symbol denotes it. The result of $A - B$, is a relation which includes all tuples that are in A but not in B.
Intersection(\cap)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(\times)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ .
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(\bowtie)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(\ltimes)	In the left outer join, operation allows keeping all tuple in the left relation.

Right Outer join(
 \bowtie_r)

In the right outer join, operation allows keeping all tuple in the right relation.

Full Outer Join(\bowtie_{full})

In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

1. Find movies made after 1997

$\sigma_{myear>1997}(\text{Movies})$

Movies	title	director	myear	rating
	Fargo	Coen	1996	8.2
	Raising Arizona	Coen	1987	7.6
	Spiderman	Raimi	2002	7.4
	Wonder Boys	Hanson	2000	7.6

$\sigma_{myear>1997}(\text{Movies})$

title	director	myear	rating
Spiderman	Raimi	2002	7.4
Wonder Boys	Hanson	2000	7.6

2. Find movies made by Hanson after 1997

$\sigma_{myear>1997 \wedge director='Hanson'}(\text{Movies})$

Movies	title	director	myear	rating
	Fargo	Coen	1996	8.2
	Raising Arizona	Coen	1987	7.6
	Spiderman	Raimi	2002	7.4
	Wonder Boys	Hanson	2000	7.6

$\sigma_{myear>1997 \wedge director='Hanson'}(\text{Movies})$

title	director	myear	rating
Wonder Boys	Hanson	2000	7.6

3. Find all movies and their ratings

$\pi_{\text{title, rating}}(\text{Movies})$

Movies	title	director	myear	rating
	Fargo	Coen	1996	8.2
	Raising Arizona	Coen	1987	7.6
	Spiderman	Raimi	2002	7.4
	Wonder Boys	Hanson	2000	7.6

$\pi_{\text{title, rating}}(\text{Movies})$

title	rating
Fargo	8.2
Raising Arizona	7.6
Spiderman	7.4
Wonder Boys	7.6

4. Find all actors and directors

$\pi_{\text{actor}}(\text{Actors}) \cup \pi_{\text{director}}(\text{Directors})$

Actors	actor	ayear
	Cage	1964
	Hanks	1956
	Maguire	1975
	McDormand	1957

$\pi_{\text{actor}}(\text{Actors})$

actor
Cage
Hanks
Maguire
McDormand

Directors	director	dyear
	Coen	1954
	Hanson	1945
	Raimi	1959

$\pi_{\text{director}}(\text{Directors})$

director
Coen
Raimi
Hanson

$\pi_{\text{actor}}(\text{Actors}) \cup \pi_{\text{director}}(\text{Directors})$

actor
Cage
Hanks
Maguire
McDormand
Coen
Raimi
Hanson

Union Example:

Find all actors & directors

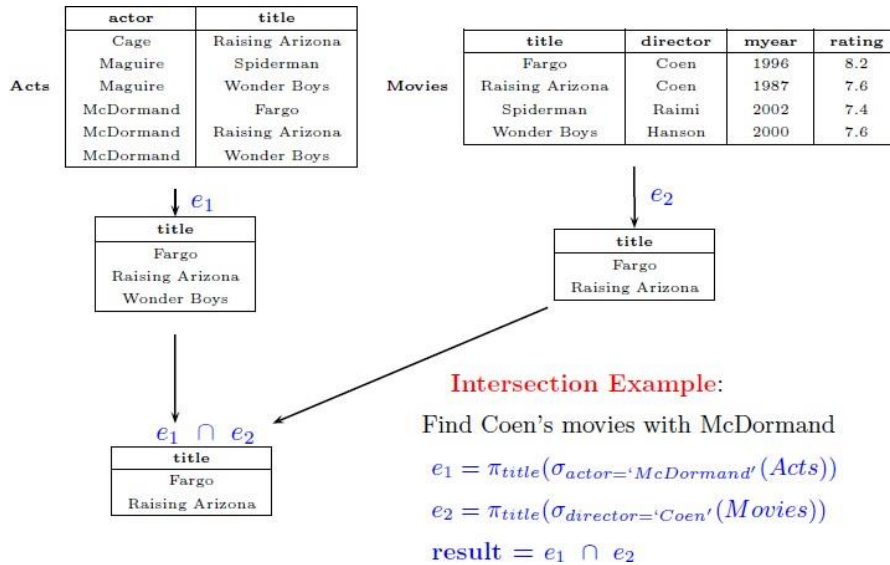
$\pi_{\text{actor}}(\text{Actors}) \cup \pi_{\text{director}}(\text{Directors})$

5. Find Coen's movies with McDormand

$e_1 = \pi_{\text{title}}(\sigma_{\text{actor}='McDormand'}(\text{Acts}))$

$e_2 = \pi_{\text{title}}(\sigma_{\text{director}='Coen'}(\text{Movies}))$

result = $e_1 \cap e_2$



Ex No: 9 Database Design using ER modeling, normalization and Implementation for any application

Consider following databases and draw ER diagram and convert entities and relationships to relation table for a given scenario.

1. COLLEGE DATABASE:

STUDENT (USN, SName, Address, Phone, Gender)

SEMSEC (SSID, Sem, Sec)

CLASS (USN, SSID)

SUBJECT (Subcode, Title, Sem, Credits)

IAMARKS (USN, Subcode, SSID, Test1, Test2, Test3, FinalIA)

2. COMPANY DATABASE:

EMPLOYEE (SSN, Name, Address, Sex, Salary, SuperSSN, DNo)

DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate)

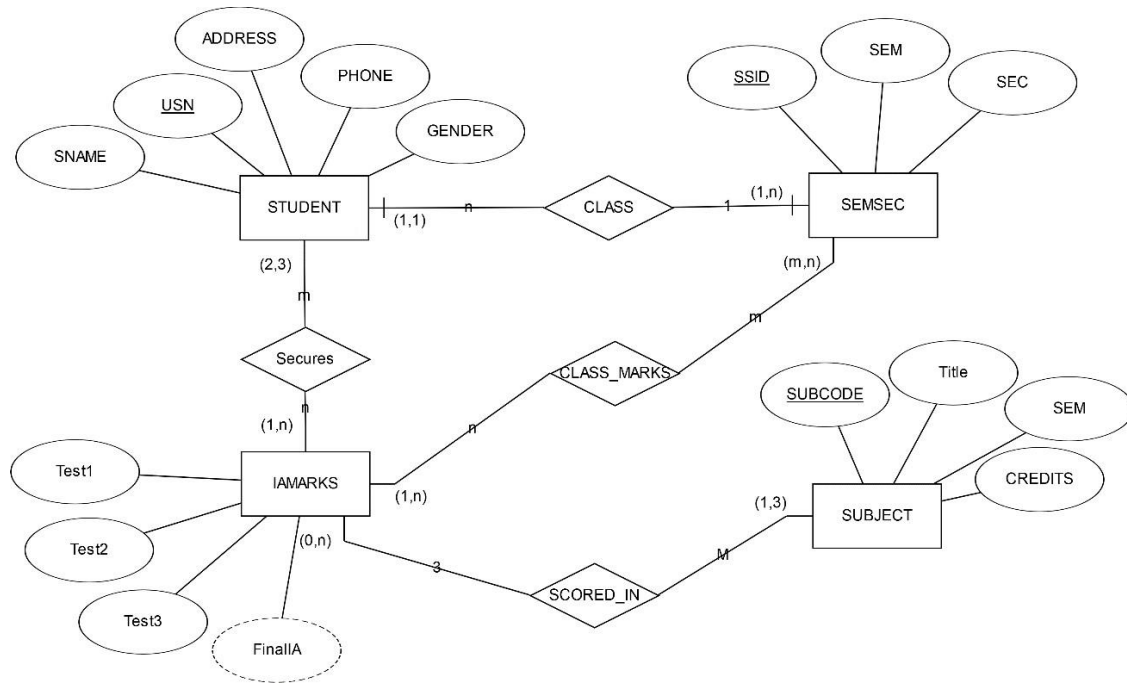
DLOCATION (DNo, DLoc)

PROJECT (PNo, PName, PLocation, DNo)

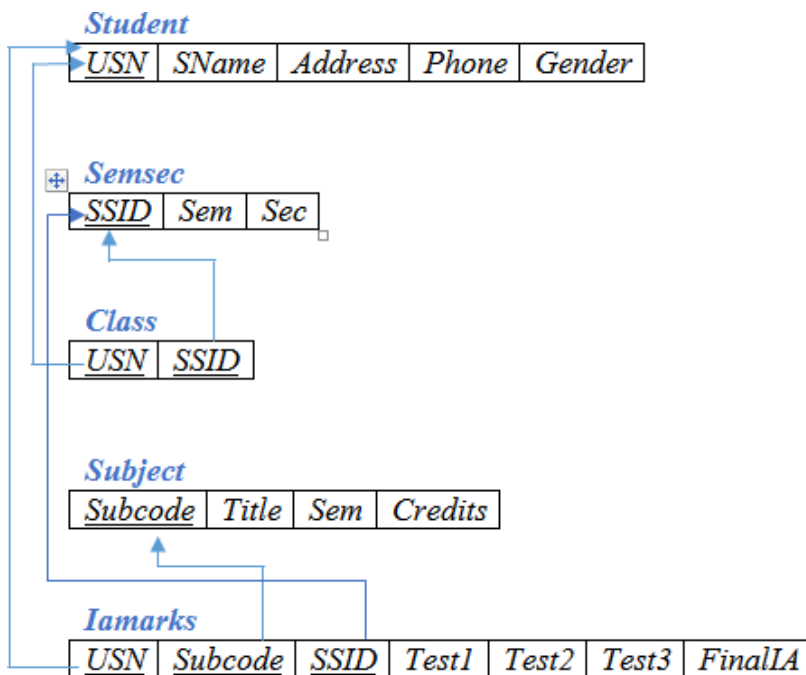
WORKS_ON (SSN, PNo, Hours)

SOLUTION:

College Database: E-R Diagram

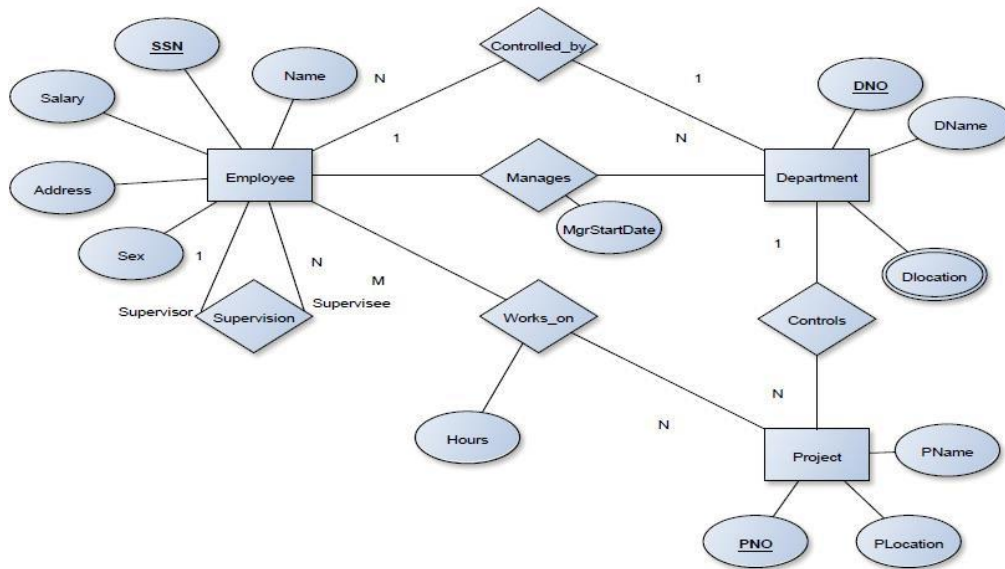


Mapping
entities and
relationships
to relation
table
(Schema
Diagram)



COMPANY DATABASE:

E-R Diagram



Schema Diagram

