



Accredited by NAAC & NBA*

SVCE BENGALURU

SRI VENKATESHWARA COLLEGE OF ENGINEERING

— Affiliated to VTU, Approved by AICTE, Recognised by UGC u/s 2(f) & 12(B)—

DEPARTMENT OF CSE-ARTIFICIAL INTELLIGENCE & ENGINEERING

V SEMESTER



DBMS LABORATORY WITH MINI PROJECT MANUAL -21CSL55

PREPARED BY

PREETI M HANDIWADI

ASSISTANT PROFESSOR

DEPARTMENT VISION

Promoting prosperity of mankind by augmenting human resource capital through Quality Technical Education and training in Artificial Intelligence Department.

Department Mission:

The mission of Artificial Intelligence is to accomplish excellence in the field of Artificial Intelligence technical education through education, research and service needs of society. And also to educate students in the area of artificial intelligence by providing best practices of teaching learning process for careers in software are industry or higher education.

M1: For acquiring the new skills and technology of artificial technology in software field and research field.

M2: To be a model center for education and training in the frontier areas of education by quality technical education and training.

M3: To get excellence in the technical education through education, research and service needed by the society

Program Educational Objectives

PEO1 Graduates will have solid basics in Mathematics, Programming, Machine Learning, Artificial Intelligence and Data Science fundamentals and advancements to solve technical problems.

PEO2 Graduates will have the capability to apply their knowledge and skills acquired to solve the issues in real world Artificial Intelligence and Data Science sectors and to develop feasible and viable systems.

PEO3 Graduates will have the potential to participate in life-long learning through the successful completion of advanced degrees, continuing education, certifications and/or other professional developments.

PEO4 Graduates will have the ability to apply the gained knowledge to improve the society ensuring ethical and moral values

Program Specific Outcomes

PSO1: The ability to understand, analyze and demonstrate the knowledge of human cognition, Artificial Intelligence in terms of real world problems to meet the challenges of the future.

PSO2: The ability to develop computational knowledge and project development skills using innovative tools and techniques to solve problems in the areas related to Artificial Intelligence.

Program Outcomes:

- 1. Engineering Knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to 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 the public health and safety, and the 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 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.

PREETI MEH

DATABASE MANAGEMENT SYSTEMS LABORATORY WITH MINI PROJECT			
Course Code	21CSL55	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2:0	SEE Marks	50
Total Hours of Pedagogy	24	Total Marks	100
Credits	01	Exam Hours	03
<p>Course Learning Objectives:</p> <p>CLO 1. Foundation knowledge in database concepts, technology and practice to groom students into well-informed database application developers.</p> <p>CLO 2. Strong practice in SQL programming through a variety of database problems.</p> <p>CLO 3. Develop database applications using front-end tools and back-end DBMS..</p>			
Sl. No.	<p>PART-A: SQL Programming (Max. Exam Marks. 50)</p> <p>Design, develop, and implement the specified queries for the following problems using Oracle, MySQL, MS SQL Server, or any other DBMS under LINUX/Windows environment. Create Schema and insert at least 5 records for each table. Add appropriate database constraints.</p>		
1	<p>Aim: Demonstrating creation of tables, applying the view concepts on the tables.</p> <p>Program Consider the following schema for a Library Database:</p> <p>BOOK(Book_id, Title, Publisher_Name, Pub_Year)</p> <p>BOOK_AUTHORS(Book_id, Author_Name)</p> <p>PUBLISHER(Name, Address, Phone)</p> <p>BOOK_COPIES(Book_id, Programme_id, No-of_Copies)</p> <p>BOOK_LENDING(Book_id, Programme_id, Card_No, Date_Out, Due_Date)</p> <p>LIBRARY_PROGRAMME(Programme_id, Programme_Name, Address)</p> <p>Write SQL queries to</p> <ol style="list-style-type: none"> 1. Retrieve details of all books in the library – id, title, name of publisher, authors, number of copies in each Programme, etc. 2. Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun 2017. 3. Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation. 4. Partition the BOOK table based on year of publication. Demonstrate its working with a simple query. 5. Create a view of all books and its number of copies that are currently available in the Library. <p>Reference:</p> <p>https://www.youtube.com/watch?v=AaSU-AOguls https://www.youtube.com/watch?v=-EwEvJxS-Fw</p>		

2	<p>Aim: Discuss the various concepts on constraints and update operations.</p> <p>Program: Consider the following schema for Order Database:</p> <p>SALESMAN(Salesman_id, Name, City, Commission)</p> <p>CUSTOMER(Customer_id, Cust_Name, City, Grade, Salesman_id)</p> <p>ORDERS(Ord_No, Purchase_Amt, Ord_Date, Customer_id, Salesman_id) Write SQL queries to</p> <p>Count the customers with grades above Bangalore's average.</p> <ol style="list-style-type: none"> Find the name and numbers of all salesman who had more than one customer. List all the salesman and indicate those who have and don't have customers in their cities (Use UNION operation.) Create a view that finds the salesman who has the customer with the highest order of a day. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted. <p>Reference:</p> <p>https://www.youtube.com/watch?v=AA-KL1jbMeY</p>
---	---

	<p>https://www.youtube.com/watch?v=7S_tz1z_5bA</p>
3	<p>Aim: Demonstrate the concepts of JOIN operations.</p> <p>Program: Consider the schema for Movie Database:</p> <p>ACTOR(Act_id, Act_Name, Act_Gender)</p> <p>DIRECTOR(Dir_id, Dir_Name, Dir_Phone)</p> <p>MOVIES(Mov_id, Mov_Title, Mov_Year, Mov_Lang, Dir_id)</p> <p>MOVIE_CAST(Act_id, Mov_id, Role)</p> <p>RATING(Mov_id, Rev_Stars)</p> <p>Write SQL queries to</p> <ol style="list-style-type: none"> List the titles of all movies directed by 'Hitchcock'. Find the movie names where one or more actors acted in two or more movies. List all actors who acted in a movie before 2000 and also in a movie after 2015(use JOIN operation). Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title. Update rating of all movies directed by 'Steven Spielberg' to 5. <p>Reference:</p> <p>https://www.youtube.com/watch?v=hSiCUNVKJAo https://www.youtube.com/watch?v=Eod3aQkFz84</p>

4	<p>Aim: Introduce concepts of PLSQL and usage on the table.</p> <p>Program: Consider the schema for College Database: STUDENT(USN, SName, Address, Phone, Gender) SEMSEC(SSID, Sem, Sec) CLASS(USN, SSID) COURSE(Subcode, Title, Sem, Credits) IAMARKS(USN, Subcode, SSID, Test1, Test2, Test3, FinalIA)</p> <p>Write SQL queries to</p> <ol style="list-style-type: none"> 1. List all the student details studying in fourth semester 'C' section. 2. Compute the total number of male and female students in each semester and in each section. 3. Create a view of Test1 marks of student USN '1BI15CS101' in all Courses. 4. Calculate the FinalIA (average of best two test marks) and update the corresponding table for all students. 5. Categorize students based on the following criterion: If FinalIA = 17 to 20 then CAT = 'Outstanding' If FinalIA = 12 to 16 then CAT = 'Average' If FinalIA < 12 then CAT = 'Weak' <p>Give these details only for 8th semester A, B, and C section students.</p> <p>Reference: https://www.youtube.com/watch?v=horURQewW9c https://www.youtube.com/watch?v=P7-wKbKrAhk</p>
5	<p>Aim: Demonstrate the core concepts on table like nested and correlated nesting queries and also EXISTS and NOT EXISTS keywords.</p> <p>Program: Consider the schema for 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)</p> <p>Write SQL queries to</p> <p>Make a list of all project numbers for projects that involve an employee whose last name is 'Scott', either as a worker or as a manager of the department that controls the project.</p>
	<p>Show the resulting salaries if every employee working on the 'IoT' project is given a 10 percent raise. Find the sum of the salaries of all employees of the 'Accounts' department, as well as the maximum salary, the minimum salary, and the average salary in this department Retrieve the name of each employee who works on all the projects controlled by department number 5 (use NOT EXISTS operator). For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs.6,00,000.</p> <p>Reference: https://www.youtube.com/watch?v=Dk8f3ejqKts</p>
Pedagogy	For the above experiments the following pedagogy can be considered. Problembased learning, Active learning, MOOC, Chalk &Talk
PART B	

	<p>Mini project: For any problem selected, make sure that the application should have five or more tables. Indicative areas include: Organization, health care, Ecommerce etc.</p>
<p>Course Outcomes: At the end of the course the student will be able to: CO 1. Create, Update and query on the database. CO 2. Demonstrate the working of different concepts of DBMS CO 3. Implement, analyze and evaluate the project developed for an application.</p>	
<p>Assessment Details (both CIE and SEE)</p> <p>The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each course. The student has to secure not less than 35% (18 Marks out of 50) in the semester-end examination (SEE).</p> <p>Continuous Internal Evaluation (CIE):</p> <p>CIE marks for the practical course is 50 Marks.</p> <p>The split-up of CIE marks for record/ journal and test are in the ratio 60:40.</p> <p>Each experiment to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments designed by the faculty who is handling the laboratory session and is made known to students at the beginning of the practical session.</p> <p>Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.</p> <p>Total marks scored by the students are scaled down to 30 marks (60% of maximum marks).</p> <p>Weightage to be given for neatness and submission of record/write-up on time.</p> <p>Department shall conduct 02 tests for 100 marks, the first test shall be conducted after the 8th week of the semester and the second test shall be conducted after the 14th week of the semester.</p> <p>In each test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.</p> <p>The suitable rubrics can be designed to evaluate each student's performance and learning ability.</p> <p>Rubrics suggested in Annexure-II of Regulation book</p> <p>The average of 02 tests is scaled down to 20 marks (40% of the maximum marks).</p> <p>The Sum of scaled-down marks scored in the report write-up/journal and average marks of two tests is the total CIE marks scored by the student.</p>	
<p>SEMESTER END EVALUATION(SEE)</p> <ul style="list-style-type: none"> • SEE marks for the practical course is 50 Marks. • SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the University • All laboratory experiments are to be included for practical examination. • (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. OR based on the course requirement evaluation rubrics shall be decided jointly by examiners. • Students can pick one question (experiment) from the questions lot prepared by the internal /external examiners jointly. • Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners. • General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners) • <i>Students can pick one experiment from the questions lot of PART A with an equal choice to all the students in a batch. For PART B, the project group (Maximum of 4 students per batch) should demonstrate the mini-project.</i> 	

- *Weightage of marks for PART A is 60% and for PART B is 40%. General rubrics suggested to be followed for part A and part B.*
- Change of experiment is allowed only once and Marks allotted to the procedure part to be made zero (Not allowed for Part B).
- The duration of SEE is 03 hours

Rubrics suggested in Annexure-II of Regulation book

Textbooks:

1. Fundamentals of Database Systems, Ramez Elmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson.
2. Database management systems, Ramakrishnan, and Gehrke, 3rd Edition, 2014, McGraw Hill

Suggested Weblinks/ E Resource

<https://www.tutorialspoint.com/sql/index.htm>

ACADEMIC YEAR – 2023-24

TABLE OF CONTENTS

Chapter No.	Concept	Page no
CHAPTER 1	BASIC CONCEPTS OF SQL	1
1.1	Introduction to SQL	1
1.2	SQL Commands	1
	1.2.1 DDL Commands	2
	1.2.2 DML Commands	6
	1.2.3 TCL Commands	9
	1.2.4 DCL Commands	10
1.3	Stored Procedures in SQL	10
1.4	SQL Triggers	12
1.5	Views in SQL	17
CHAPTER 2	LAB PROGRAM 1 - LIBRARY DATABASE	18
2.1	Problem Statement	18
2.2	ER Diagram	18
2.3	Schema Diagram	19
2.4	Creating Tables	20
2.5	Inserting Values	21
2.6	Queries and Solutions	24
CHAPTER 3	LAB PROGRAM 2 - ORDER DATABASE	29
3.1	Problem Statement	29
3.2	ER Diagram	29
3.3	Schema Diagram	30
3.4	Creating Tables	31
3.5	Inserting Values	31
3.6	Queries and Solutions	32
CHAPTER 4	LAB PROGRAM 3 - MOVIE DATABASE	37
4.1	Problem Statement	37
4.2	ER Diagram	37
4.3	Schema Diagram	38
4.4	Creating Tables	39

4.5	Inserting Values	40
4.6	Queries and Solutions	42
CHAPTER 5	LAB PROGRAM 4 - COLLEGE DATABASE	46

5.1	Problem Statement	46
5.2	ER Diagram	46
5.3	Schema Diagram	47
5.4	Creating Tables	48
5.5	Inserting Values	49
5.6	Queries and Solutions	50
CHAPTER 6	LAB PROGRAM 5 - COMPANY DATABASE	56
6.1	Problem Statement	56
6.2	ER Diagram	56
6.3	Schema Diagram	57
6.4	Creating Tables	58
6.5	Inserting Values	59
6.6	Queries and Solutions	61
BIBLIOGRAPHY		
VIVA QUESTIONS		

CHAPTER – 1

BASIC CONCEPTS OF SQL

Introduction to SQL

SQL stands for “Structured Query Language” and can be pronounced as “SQL” or “sequel – (Structured English Query Language)”. It is a query language used for accessing and modifying information in the database. IBM first developed SQL in 1970s. Also it is an ANSI/ISO standard. It has become a Standard Universal Language used by most of the relational database management systems (RDBMS). Some of the RDBMS systems are: Oracle, Microsoft SQL server, Sybase etc. Most of these have provided their own implementation thus enhancing its feature and making it a powerful tool. Few of the SQL commands used in SQL programming are SELECT Statement, UPDATE Statement, INSERT INTO Statement, DELETE Statement, WHERE Clause, ORDER BY Clause, GROUP BY Clause, ORDER Clause, Joins, Views, GROUP Functions, Indexes etc.

SQL Commands

SQL commands are instructions used to communicate with the database to perform specific task that work with data. SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can create tables, add data to tables, or modify data, drop the table, set permissions for users. SQL commands are grouped into four major categories depending on their functionality:

- **Data Definition Language (DDL)** - These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.
- **Data Manipulation Language (DML)** - These SQL commands are used for storing, retrieving, modifying and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.

- **Transaction Control Language (TCL)** - These SQL commands are used for managing changes affecting the data. These commands are COMMIT, ROLLBACK, and SAVEPOINT.
- **Data Control Language (DCL)** - These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

Data Definition Language (DDL)

CREATE TABLE Statement

The CREATE TABLE Statement is used to create tables to store data. Integrity Constraints like primary key, unique key and foreign key can be defined for the columns while creating the table. The integrity constraints can be defined at column level or table level. The implementation and the syntax of the CREATE Statements differs for different RDBMS.

The Syntax for the CREATE TABLE Statement is:

```
CREATE TABLE table_name  
(column_name1 datatype constraint,  
column_name2datatype,...  
column_nameNdatatype);
```

- **table_name** - is the name of the table.
- **column_name1, column_name2....** - is the name of the columns
- **datatype** - is the datatype for the column like char, date, number etc.

SQL Data Types:

char(size)	Fixed-length character string. Size is specified in parenthesis. Max 255 bytes.
Varchar2(size)	Variable-length character string. Max size is specified in parenthesis.

number(size)or int	Number value with a max number of column digits specified in parenthesis.
Date	Date value in 'dd-mon-yy'. Eg., '07-jul-2004'
number(size,d)or real	Number value with a maximum number of digits of "size" total, with a maximum number of "d" digits to the right of the decimal.

SQL Integrity Constraints:

Integrity Constraints are used to apply business rules for the database tables. The constraints available in SQL are **Foreign Key, Primary key, Not Null, Unique, Check**.

Constraints can be defined in two ways:

1. The constraints can be specified immediately after the column definition. This is called column-level definition.
2. The constraints can be specified after all the columns are defined. This is called table-level definition.

1) Primary key:

This constraint defines a column or combination of columns which uniquely identifies each row in the table.

Syntax to define a Primary key at column level:

```
Column_namdatatype [CONSTRAINT constraint_name] PRIMARY KEY
```

Syntax to define a Primary key at table level:

```
[CONSTRAINT constraint_name] PRIMARY KEY(column_name1,
column_name2, ..)
```

□ **column_name1, column_name2** are the names of the columns which define the primary key. □ The syntax within the bracket i.e. [CONSTRAINT constraint_name] is optional.

2) Foreign key or Referential Integrity:

This constraint identifies any column referencing the PRIMARY KEY in another table. It establishes a relationship between two columns in the same table or between different tables. For a column to be defined as a Foreign Key, it should be defined as a Primary Key in the table which it is referring. One or more columns can be defined as Foreign key.
Syntax to define a Foreign key at column level:


```
[CONSTRAINT constraint_name] REFERENCES
```

```
referenced_table_name(column_name)
```

Syntax to define a Foreign key at table level:

```
[CONSTRAINT constraint_name] FOREIGN KEY(column_name) REFERENCES  
referenced_table_name(column_name);
```

3) Not Null Constraint:

This constraint ensures all rows in the table contain a definite value for the column which is specified as not null. Which means a null value is not allowed.

Syntax to define a Not Null constraint:

```
[CONSTRAINT constraint_name] NOT NULL
```

4) Unique Key:

This constraint ensures that a column or a group of columns in each row have a distinct value. A column(s) can have a null value but the values cannot be duplicated.

Syntax to define a Unique key at column level:

```
[CONSTRAINT constraint_name] UNIQUE
```

Syntax to define a Unique key at table level:

```
[CONSTRAINT constraint_name] UNIQUE(column_name)
```

5) Check Constraint:

This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.

Syntax to define a Check constraint:

```
[CONSTRAINT constraint_name] CHECK (condition)
```

ALTER TABLE Statement

The SQL ALTER TABLE command is used to modify the definition structure) of a table by modifying the definition of its columns. The ALTER command is used to perform the following functions.

- 1) Add, drop, modify table columns
- 2) Add and drop constraints
- 3) Enable and Disable constraints

Syntax to add a column

```
ALTER TABLE table_name ADD column_namedatatype;
```

For Example: To add a column "experience" to the employee table, the query would be like

```
ALTER TABLE employee ADD experience number(3);
```

Syntax to drop a column

```
ALTER TABLE table_name DROP column_name;
```

For Example: To drop the column "location" from the employee table, the query would be like

```
ALTER TABLE employee DROP location;
```

Syntax to modify a column

```
ALTER TABLE table_name MODIFY column_namedatatype;
```

For Example: To modify the column salary in the employee table, the query would be like

```
ALTER TABLE employee MODIFY salary number(15,2);
```

Syntax to add PRIMARY KEY constraint

```
ALTER TABLE table_name ADD CONSTRAINT constraint_name PRIMARY KEY  
column_name;
```

Syntax to drop PRIMARY KEY constraint

```
ALTER TABLE table_name DROP PRIMARY KEY;
```

The DROP TABLE Statement

The DROP TABLE statement is used to delete a table.

```
DROP TABLE table_name;
```

TRUNCATE TABLE Statement

What if we only want to delete the data inside the table, and not the table itself?

Then, use the TRUNCATE TABLE statement:

```
TRUNCATE TABLE table_name;
```

Data Manipulation Language (DML):

The SELECT Statement

The SELECT statement is used to select data from a database. The result is stored in a result table, called the result-set.

SELECT Syntax:

```
SELECT * FROM table_name;
```

The SELECT DISTINCT Statement

In a table, some of the columns may contain duplicate values. This is not a problem, however, sometimes you will want to list only the different (distinct) values in a table. The DISTINCT keyword can be used to return only distinct (different) values. SELECT DISTINCT Syntax:

```
SELECT DISTINCT column_name(s)  
FROM table_name;
```

The WHERE Clause

The WHERE clause is used to extract only those records that fulfill a specified criterion. WHERE Syntax:

```
SELECT column_name(s)  
FROM table_name
```

WHERE column_name operator value;

The AND & OR Operators

- The AND operator displays a record if both the first condition and the second condition is true.
- The OR operator displays a record if either the first condition or the second condition is true.

The ORDER BY Clause

- The ORDER BY clause is used to sort the result-set by a specified column.
- The ORDER BY clausesort the records in ascending order by default.
- If you want to sort the records in a descending order, you can use the DESC keyword.

ORDER BY Syntax:

```
SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC;
```

The GROUP BY Clause

The GROUP BY clause can be used to create groups of rows in a table. Group functions can be applied on such groups.

GROUP BY Syntax;

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
GROUP BY column_name(s);
```

Group functions	Meaning
AVG([DISTINCT ALL],N)	Returns average value of n

COUNT(* [DISTINCT ALL]expr)	<p>Returns the number of rows in the query.</p> <p>When you specify expr, this function considers rows where expr is not null.</p> <p>When you specify the asterisk (*), this function Returns all rows, including duplicates and nulls.</p> <p>You can count either all rows, or only distinct</p>
-----------------------------	---

	values of expr.
MAX([DISTINCT ALL]expr)	Returns maximum value of expr
MIN([DISTINCT ALL]expr)	Returns minimum value of expr
SUM([DISTINCT ALL]n)	Returns sum of values of n

The HAVING clause

The HAVING clause can be used to restrict the display of grouped rows. The result of the grouped query is passed on to the HAVING clause for output filtration.

HAVING Syntax;

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator value
GROUP BY column_name(s)
HAVING condition;
```

The INSERT INTO Statement

The INSERT INTO statement is used to insert a new row in a table.

SQL INSERT INTO Syntax:

It is possible to write the INSERT INTO statement in two forms.

- The first form doesn't specify the column names where the data will be inserted, only their values:□

```
INSERT INTO table_name VALUES (value1, value2, value3,...);

OR

INSERT INTO
table_name VALUES (&column1, &column2, &column3,...);
```

- The second form specifies both the column names and the values to be inserted:□

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...);
```

The UPDATE Statement

The UPDATE statement is used to update existing records in a table.

SQL UPDATE Syntax:

Dept. **SVCE,**
BLR
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value;

The DELETE Statement

The DELETE statement is used to delete rows in a table.

SQL DELETE Syntax:

DELETE FROM table_name
WHERE some_column=some_value;

Transaction Control language

Transaction Control Language (TCL) commands are used to manage transactions in database. These are used to manage the changes made by DML statements. It also allows statements to be grouped together into logical transactions

Commit command

Commit command is used to permanently save any transaction into database. Following is Commit command's syntax,

commit; Rollback command

This command restores the database to last committed state. It is also used with savepoint command to jump to a savepoint in a transaction.

Following is Rollback command's syntax

rollback to savepoint_name;

Savepoint command

savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

Following is savepoint command's syntax,

savepoint savepoint_name;

Data Control Language

Data Control Language(DCL) is used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges.

Privileges are of two types,

- **System** : creating session, table etc are all types of system privilege.□
- **Object** : any command or query to work on tables comes under object privilege.□

DCL defines two commands,

- **Grant** : Gives user access privileges to database.□
- **Revoke** : Take back permissions from user.□

To Allow a User to create Session

```
grant create session to username;
```

To Allow a User to create Table

```
grant create table to username;
```

To provide User with some Space on Tablespace to store Table

```
alter user username quota unlimited on system;
```

To Grant all privilege to a User

```
grant sysdba to username
```

To Grant permission to Create any Table

```
grant create any table to username
```

STORED PROCEDURES in SQL:

The SQL Server **Stored procedure** is used to save time to write code again and again by storing the same in database and also get the required output by passing parameters.

Syntax Following is the basic syntax of Stored procedure creation.

```
Create procedure <procedure_Name>
As
Begin
<SQL Statement>
End
Go
```

Example Consider the CUSTOMERS table having the following records.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Following command is an example which would fetch all records from the CUSTOMERS table in Testdb database.

```
CREATE PROCEDURE SelectCustomerstabledata  
AS  
SELECT * FROM Testdb.Customers  
GO
```

The above command will produce the following output.

ID	NAME	AGE	ADDRESS	SALARY
----	------	-----	---------	--------

1	Ramesh			2000.00
2	Khilan	32	Ahmedabad	1500.00
3	kaushik	25	Delhi Kota	2000.00
4	Chaitali	23	Mumbai	6500.00
5	Hardik	25 27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

SQL TRIGGERS

Triggers are stored programs, which are automatically executed or fired when some events occur.

Triggers are, in fact, written to be executed in response to any of the following events –

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers:

Triggers can be written for the following purposes –

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables

- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers The syntax for

creating a trigger is :

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

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name – Creates or replaces an existing trigger with the *trigger_name*.

- {BEFORE | AFTER | INSTEAD OF} – This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} – This specifies the DML operation.
- [OF col_name] – This specifies the column name that will be updated.
- [ON table_name] – This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] – This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] – This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) – This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Example

To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters –

Select * from customers;

```

+__+____+__+____+____+
| ID | NAME   | AGE | ADDRESS | SALARY |
+__+____+__+____+____+
| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |
| 2 | Khilan  | 25 | Delhi    | 1500.00 |
| 3 | kaushik | 23 | Kota     | 2000.00 |
| 4 | Chaitali | 25 | Mumbai   | 6500.00 |
| 5 | Hardik  | 27 | Bhopal   | 8500.00 |
| 6 | Komal   | 22 | MP       | 4500.00 |
+__+____+__+____+____+

```

The following program creates a **row-level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values –

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON
customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
    sal_diff number;
BEGIN
    sal_diff := :NEW.salary - :OLD.salary;
    dbms_output.put_line('Old salary: ' || :OLD.salary);
    dbms_output.put_line('New salary: ' || :NEW.salary);
    dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

When the above code is executed at the SQL prompt, it produces the following result –

Trigger created.

The following points need to be considered here –

- OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.

- The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.

Triggering a Trigger

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table –

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
```

When a record is created in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result –

Old salary:

New salary: 7500

Salary difference:

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The UPDATE statement will update an existing record in the table –

```
UPDATE customers
SET salary = salary + 500
WHERE id = 2;
```

When a record is updated in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result –

Old salary: 1500

New salary: 2000

Salary difference: 500

PREETIMH

VIEWS IN SQL

- A view is a single *virtual table* that is derived from other tables. The other tables could be base tables or previously defined view.
- Allows for limited update operations Since the table may not physically be stored
- Allows full query operations
- A convenience for expressing certain operations
- A view does not necessarily exist in physical form, which limits the possible update operations that can be applied to views.

PREETI MATH

LIBRARY DATABASE

Aim: Demonstrating creation of tables, applying the view concepts on the tables.

Program 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, Programme_id, No-of_Copies)

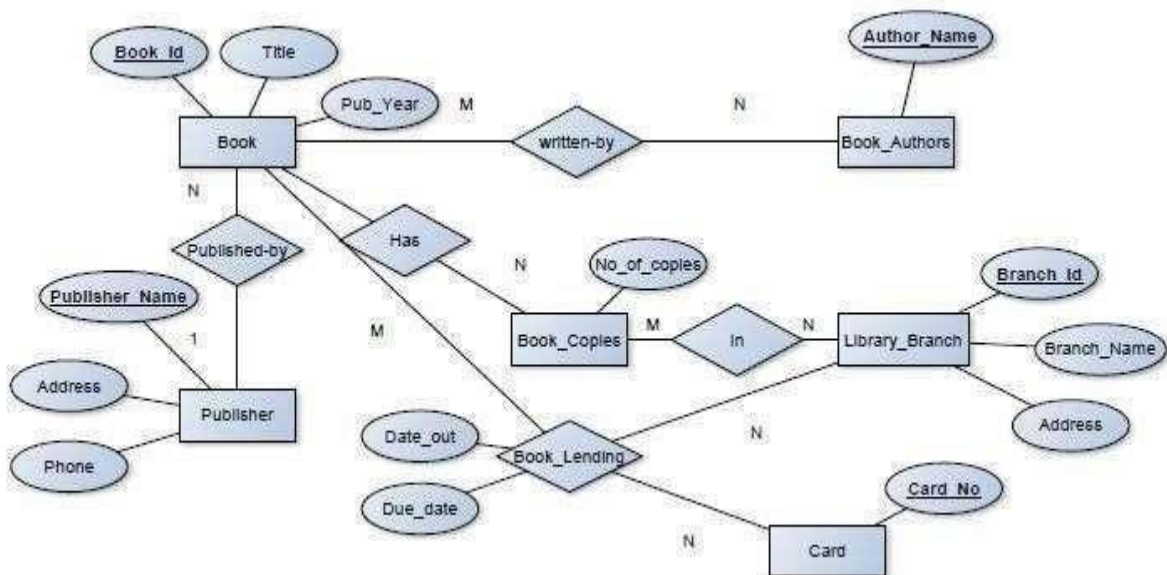
BOOK_LENDING(Book_id, Programme_id, Card_No, Date_Out, Due_Date)

LIBRARY_PROGRAMME(Programme_id, Programme_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 Programme, etc.
2. Get the particulars of borrowers who have borrowed more than 3 books, but from Jan 2017 to Jun 2017.
3. Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation.
4. Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.
5. Create a view of all books and its number of copies that are currently available in the Library.

ER-Diagram:



SCHEMA:

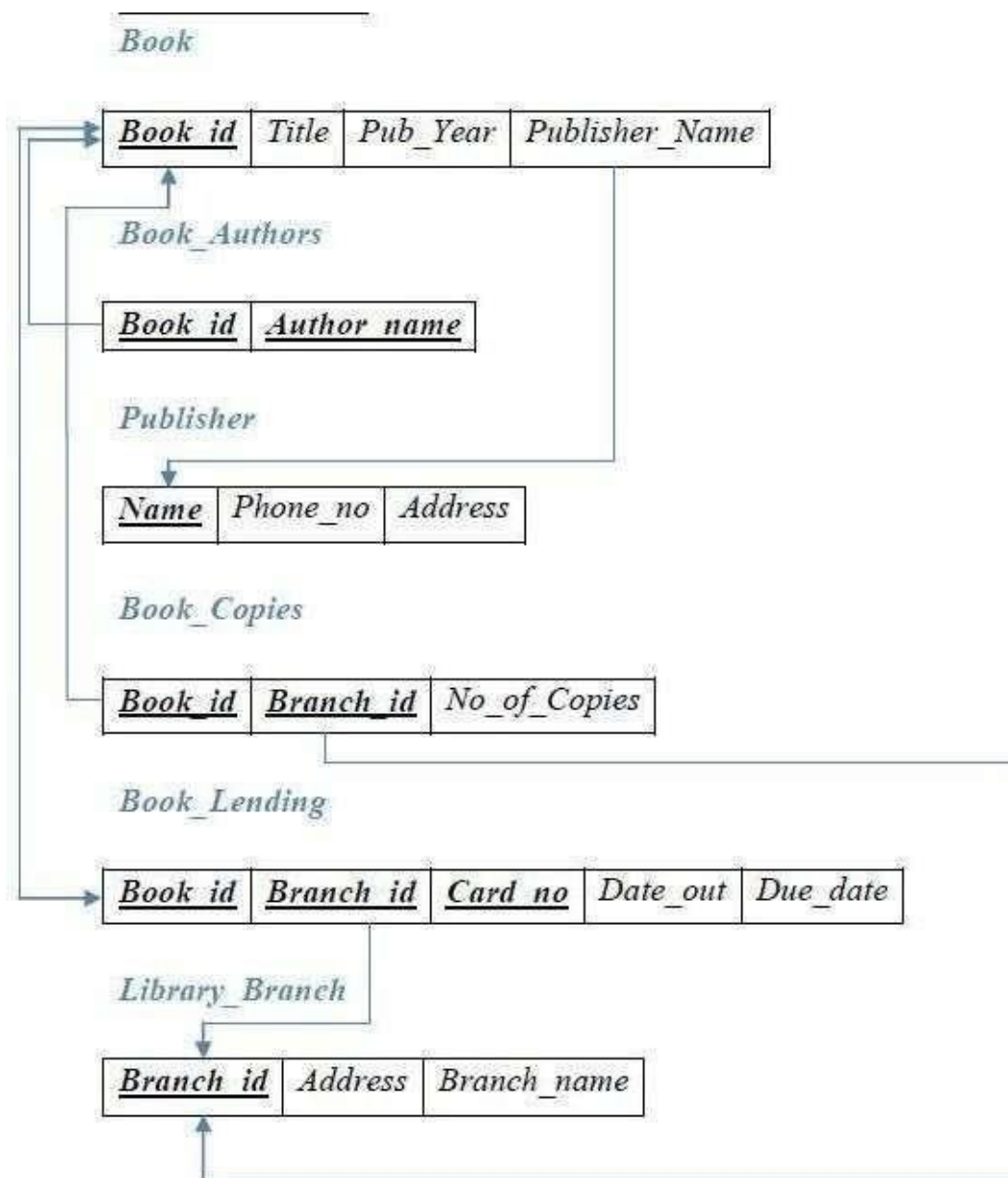


Table Creation:

PUBLISHER

```
SQL> CREATE TABLE PUBLISHER(  
    P_NAME VARCHAR2(25) PRIMARY KEY,  
    P_ADDRESS VARCHAR2(10),  
    PHONE NUMBER(10));
```

Table created.

BOOK

```
SQL> CREATE TABLE BOOK(  
    BOOK_ID INTEGER PRIMARY KEY,  
    TITLE VARCHAR2(20),  
    P_NAME VARCHAR2(20) REFERENCES PUBLISHER(P_NAME) ON DELETE  
                                     CASCADE, PUB_YEAR NUMBER(4));
```

Table created.

BOOK_AUTHORS

```
SQL> CREATE TABLE BOOK_AUTHOR(  
    BOOK_ID INTEGER REFERENCES BOOK(BOOK_ID) ON DELETE CASCADE,  
    AUTHOR_NAME VARCHAR(20),  
    PRIMARY KEY(BOOK_ID, AUTHOR_NAME));
```

Table created.

LIBRARY_PROGRAM

```
SQL> CREATE TABLE LIBRARY_PROGRAM  
    (PROGRAM_ID NUMBER(4) PRIMARY KEY,  
    PROGRAM_NAME VARCHAR2(20),  
    ADDRESS VARCHAR2(15));
```

Table created.

BOOK_COPIES

```
SQL> CREATE TABLE BOOK_COPIES(  
    BOOK_ID INTEGER REFERENCES BOOK(BOOK_ID) ON DELETE CASCADE,  
    PROGRAM_ID NUMBER(4) REFERENCES LIBRARY_PROGRAM(PROGRAM_ID) ON  
    DELETE CASCADE,  
    NO_OF_COPIES INTEGER,  
    PRIMARY KEY(BOOK_ID, PROGRAM_ID));
```

Table created.

BOOK_LENDING

```
SQL> CREATE TABLE BOOK_LENDING(  
    BOOK_ID INTEGER REFERENCES BOOK(BOOK_ID) ON DELETE CASCADE,  
    PROGRAM_ID INTEGER REFERENCES LIBRARY_PROGRAM(PROGRAM_ID) ON  
    DELETE CASCADE,  
    CARD_NO INTEGER DATE_OUT DATE, DUE_DATE DATE,  
    PRIMARY KEY(BOOK_ID, PROGRAM_ID, CARD_NO));
```

Table created.

Values for tables:

PUBLISHER

SQL>INSERT INTO PUBLISHER VALUES('PEARSON','BANGALORE','9875462530');

SQL>INSERT INTO PUBLISHER VALUES ('MCGRAW','NEWDELHI','7845691234');

SQL>INSERT INTO PUBLISHER VALUES('SAPNA','BANGALORE','7845963210');

BOOK

SQL>INSERT INTO BOOKVALUES (1111,'SE','PEARSON',2005);

SQL>INSERT INTO BOOKVALUES (2222,'DBMS','MCGRAW',2004);

SQL>INSERT INTO BOOKVALUES (3333,'ANOTOMY','PEARSON',2010);

SQL>INSERT INTO BOOKVALUES (4444,'ENCYCLOPEDIA','SAPNA',2010);

BOOK_AUTHORS

SQL>INSERT INTO BOOK_AUTHORS VALUES (1111,'SOMMERVILLE');

SQL>INSERT INTO BOOK_AUTHORS VALUES (2222,'NAVATHE');

SQL>INSERT INTO BOOK_AUTHORS VALUES (3333,'HENRY GRAY');

SQL>INSERT INTO BOOK_AUTHORS VALUES (4444,'RAJ KAMAL');

LIBRARY_PROGRAM

SQL> INSERT INTO LIBRARY_PROGRAM VALUES(11,'CENTRAL TECHNICAL','MG ROAD');

SQL> INSERT INTO LIBRARY_PROGRAM VALUES(22,'MEDICAL','BH ROAD');

SQL> INSERT INTO LIBRARY_PROGRAM VALUES(33,'CHILDREN','SS PURAM');

SQL> INSERT INTO LIBRARY_PROGRAM VALUES(44,'SECRETARIAT','SIRAGATE');

SQL> INSERT INTO LIBRARY_PROGRAM VALUES(55,'GENERAL','JAYANAGAR');

BOOK_COPIES

SQL> INSERT INTO BOOK_COPIES VALUES(1111,11,5);

```
SQL> INSERT INTO BOOK_COPIES VALUES(3333,22,6);
```

```
SQL> INSERT INTO BOOK_COPIES VALUES(4444,33,10);
```

```
SQL> INSERT INTO BOOK_COPIES VALUES(2222,11,12);
```

```
SQL> INSERT INTO BOOK_COPIES VALUES(4444,55,3);
```

BOOK_LENDING

```
SQL> INSERT INTO BOOK_LENDIN VALUES(2222,11,1,'10-JAN-2017','20-AUG-2017');
```

G

```
SQL> INSERT INTO BOOK_LENDIN VALUES(3333,22,2,'09-JUL-2017','12-AUG-2017');
```

G

```
SQL> INSERT INTO BOOK_LENDIN VALUES(4444,55,1,'11-APR-2017','09-AUG-2017');
```

G

```
SQL> INSERT INTO BOOK_LENDIN VALUES(2222,11,5,'09-AUG-2017','19-AUG-2017');
```

G

```
SQL> INSERT INTO BOOK_LENDIN VALUES(4444,33,1,'10-JUN-2017','15-AUG-2017');
```

G

```
SQL> INSERT INTO BOOK_LENDIN VALUES(1111,11,1,'12-MAY-2017','10-JUN-2017');
```

G

```
SQL> INSERT INTO BOOK_LENDIN VALUES(3333,22,1,'10-JUL-2017','15-JUL-2017'); G
```

```
SQL> SELECT * FROM BOOK;
```

BOOK_ID	TITLE	PUBLISHER_NAME	PUB_YEAR
1111	SE	PEARSON	2005
2222	DBMS	MCGRAW	2004
3333	ANOTOMY	PEARSON	2010
4444	ENCYCLOPEDIA	SAPNA	2010

4 rows selected.

```
SQL> SELECT * FROM BOOK_AUTHORS;
```

BOOK_ID	AUTHOR_NAME
1111	SOMMERVILLE
2222	NAVATHE
3333	HENRY GRAY
4444	THOMAS

4 rows selected.

```
SQL> SELECT * FROM PUBLISHER;
```

NAME	ADDRESS	PHONE
PEARSON	BANGALORE	9875462530
MCGRAW	NEWDELHI	7845691234
SAPNA	BANGALORE	7845963210

3 rows selected.

SQL> SELECT * FROM BOOK_COPIES; BOOK_ID

PROGRAM_ID NO_OF_COPIES

1111		5
3333	11	6
4444	22	10
2222	33	12
4444	11	3
	55	

5 rows selected.

SQL> SELECT * FROM BOOK_LENDING;

BOOK_ID PROGRAM_ID CARD_NO DATE_OUT DUE_DATE

2222	11	1	10-JAN-17	20-AUG-17
3333	22	2	09-JUL-17	12-AUG-17
4444	55	1	11-APR-17	09-AUG-17
2222	11	5	09-AUG-17	19-AUG-17
4444	33	1	10-JUN-17	15-AUG-17
1111	11	1	12-MAY-17	10-JUN-17
1111	33	2	12-MAY-17	10-JUN-17
4444	11	2	22-MAR-17	10-MAY-17
3333	11	2	22-MAR-17	10-MAY-17

9 rows selected.

SQL> SELECT * FROM LIBRARY_PROGRAM;

<u>PROGRAM_ID</u>	<u>PROGRAM_NAME</u>	<u>ADDRESS</u>
11	CENTRAL TECHNICAL	MG ROAD
22	MEDICAL	BH ROAD
33	CHILDREN	SS PURAM
44	SECRETARIAT	SIRAGATE
55	GENERAL	JAYANAGAR

5 rows selected.

Queries:

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

Variant-1:

```
SQL> SELECT LB.PROGRAM_NAME, B.BOOK_ID, TITLE, PUBLISHER_NAME, AUTHOR_NAME,
        NO_OF_COPIES
FROM BOOK B, BOOK_AUTHORS BA, BOOK_COPIES BC, LIBRARY_PROGRAM LB
WHERE B.BOOK_ID = BA.BOOK_ID AND
      BA.BOOK_ID = BC.BOOK_ID AND
      BC.PROGRAM_ID = LB.PROGRAM_ID
GROUP BY LB.PROGRAM_NAME, B.BOOK_ID, TITLE, PUBLISHER_NAME,
```

AUTHOR_NAME, NO_OF_COPIES; **Variant-2:**

```
SQL> SELECT LB.PROGRAM_NAME, B.BOOK_ID, TITLE, P_NAME, AUTHOR_NAME, NO_OF_COPIES
FROM BOOK B JOIN BOOK_AUTHORS BA ON B.BOOK_ID = BA.BOOK_ID, BOOK_COPIES BC,
LIBRARY_PROGRAM LB
WHERE BA.BOOK_ID = BC.BOOK_ID AND BC.PROGRAM_ID = LB.PROGRAM_ID
GROUP BY LB.PROGRAM_NAME, B.BOOK_ID, TITLE, P_NAME, AUTHOR_NAME,
```

NO_OF_COPIES; **Variant-3:**

```
SQL> SELECT LB.PROGRAM_NAME, B.BOOK_ID, TITLE, P_NAME, AUTHOR_NAME, NO_OF_COPIES
FROM BOOK B JOIN BOOK_AUTHORS BA ON B.BOOK_ID = BA.BOOK_ID JOIN BOOK_COPIES
BC
ON BA.BOOK_ID = BC.BOOK_ID JOIN LIBRARY_PROGRAM LB ON BC.PROGRAM_ID
= LB.PROGRAM_ID
GROUP BY LB.PROGRAM_NAME, B.BOOK_ID, TITLE, P_NAME, AUTHOR_NAME, NO_OF_COPIES;
```

OUTPUT:

PROGRAM_NAME	BOOK_ID	TITLE	PUBLISHER_NAME	AUTHOR_NAME	NO_OF_COPIES
GENERAL	4444	ENCYCLOPE DIA	SAPNA	THOMAS	3
MEDICAL	3333	ANOTOMY	PEARSON	HENRY GRAY	6
CHILDREN	4444	ENCYCLOPE DIA	SAPNA	THOMAS	10
CENTRAL TECHNICAL	1111	SE	PEARSON	SOMMERVILLE	5

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

Varient-1:

```
SQL> SELECT CARD_NO
      FROM BOOK_LENDING
      WHERE DATE_OUT BETWEEN '01-JAN-2017' AND '30-JUN-2017'
      GROUP BY CARD_NO
      HAVING COUNT (*) > 3;
```

Varient-2:

```
SQL> SELECT CARD_NO
      FROM BOOK_LENDING
      WHERE DATE_OUT BETWEEN '01-JAN-2017' AND '30-JUN-2017'
      GROUP BY CARD_NO
      HAVING COUNT (*) >= 4;
```

Varient-3:

```
SQL> SELECT CARD_NO
      FROM BOOK_LENDING
      WHERE DATE_OUT >= '01-JAN-2017' AND DATE_OUT <= '30-JUN-2017'
      GROUP BY CARD_NO
      HAVING COUNT (*) >= 4;
```

OUTPUT:

```
CARD_NO
----- 1
```

3) Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation. **Varient-1:**

```
DELETE FROM BOOK
WHERE BOOK_ID = '3333';
```

1 row deleted.

Varient-2:

```
DELETE FROM BOOK
WHERE BOOK_ID IN ( '3333' );
```

1 row deleted.

Varient-3:

```
DELETE FROM BOOK_LENDING
WHERE BOOK_ID = '3333';
```

1 row deleted.

```
DELETE FROM BOOK_COPIES
WHERE BOOK_ID = '3333';
```

1 row deleted.

```
DELETE FROM BOOK_AUTHORS
WHERE BOOK_ID = '3333';
```

1 row deleted.

```
DELETE FROM BOOK
WHERE BOOK_ID = '3333';
```

1 row deleted.

OUTPUT:

```
SQL> SELECT * FROM BOOK;
```

BOOK_ID	TITLE	PUBLISHER_NAME	PUB_YEAR
1111	SE	PEARSON	2005
2222	DBMS	MCGRAW	2004
4444	ENCYCLOPEDIA	SAPNA	2010

```
SQL> SELECT * FROM BOOK_COPIES;
```

```
BOOK_ID PROGRAM_ID NO_OF_COPIES
```

```
-----
```

1111	11	5
4444	33	10
2222	11	12
4444	55	3

CSE-AI SVCE

V Sem CSE-AI

```
SQL> SELECT * FROM BOOK_LENDING;
```

```
BOOK_ID PROGRAM_ID CARD_NO DATE_OUT DUE_DATE
```

```
-----
```

2222	11	1	10-JAN-17	20-AUG-17
4444	55	1	11-APR-17	09-AUG-17
2222	11	5	09-AUG-17	19-AUG-17
4444	33	1	10-JUN-17	15-AUG-17
1111	11	1	12-MAY-17	10-JUN-17

4) Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.

Varient-1:

```
SQL> SELECT BOOK_ID, TITLE, PUBLISHER_NAME, PUB_YEAR
FROM BOOK
GROUP BY PUB_YEAR, BOOK_ID, TITLE, PUBLISHER_NAME;
```

Varient-2:

```
SQL> SELECT BOOK_ID, TITLE, PUBLISHER_NAME, PUB_YEAR
FROM BOOK
GROUP BY PUB_YEAR, BOOK_ID, TITLE, PUBLISHER_NAME
ORDER BY PUB_YEAR;
```

Varient-3:

```
SQL> SELECT BOOK_ID, TITLE, PUBLISHER_NAME, PUB_YEAR
FROM BOOK
GROUP BY PUB_YEAR, BOOK_ID, TITLE, PUBLISHER_NAME;
```

OUTPUT:

BOOK_ID	TITLE	PUBLISHER_NAME	PUB_YEAR
2222	DBMS	MCGRAW	2004
1111	SE	PEARSON	2005
3333	ANOTOMY	PEARSON	2010
4444	ENCYCLOPEDIA	SAPNA	2010

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

Varient-1:

```
SQL> CREATE VIEW BOOKS_AVAILABLE AS SELECT
      B. BOOK_ID, B. TITLE, C.NO_OF_COPIES
    FROM LIBRARY_PROGRAM L, BOOK B, BOOK_COPIES C
   WHERE B. BOOK_ID = C. BOOK_ID AND L. PROGRAM_ID = C. PROGRAM_ID;
```

View created.

Varient-2:

```
SQL> CREATE VIEW BOOKS_AVAILABLE1 AS SELECT
      C.BOOK_ID, B. TITLE, C.NO_OF_COPIES
    FROM LIBRARY_PROGRAM L JOIN BOOK_COPIES C ON L. PROGRAM_ID = C.
      PROGRAM_ID, BOOK B
   WHERE C.BOOK_ID = B.BOOK_ID;
```

View created.

Varient-3:

```
SQL> CREATE VIEW BOOKS_AVAILABLE2 AS SELECT
      C.BOOK_ID, B. TITLE, C.NO_OF_COPIES
    FROM LIBRARY_PROGRAM L JOIN BOOK_COPIES C ON L. PROGRAM_ID = C. PROGRAM_ID
   JOIN BOOK B ON C.BOOK_ID = B.BOOK_ID;
```

View created.

OUTPUT:

```
SQL> SELECT * FROM BOOKS_AVAILABLE;
```

BOOK_ID	TITLE	NO_OF_COPIES
1111	SE	5
3333	ANOTOMY	6
4444	ENCYCLOPEDIA	10
2222	DBMS	12
4444	ENCYCLOPEDIA	3

EXPEIREMENT – 3

ORDER DATABASE

2) Aim: Discuss the various concepts on constraints and update operations.

Program: Consider the following schema for Order Database:

Consider the following schema for Order Database:

SALESMAN (Salesman_id, Name, City, Commission)

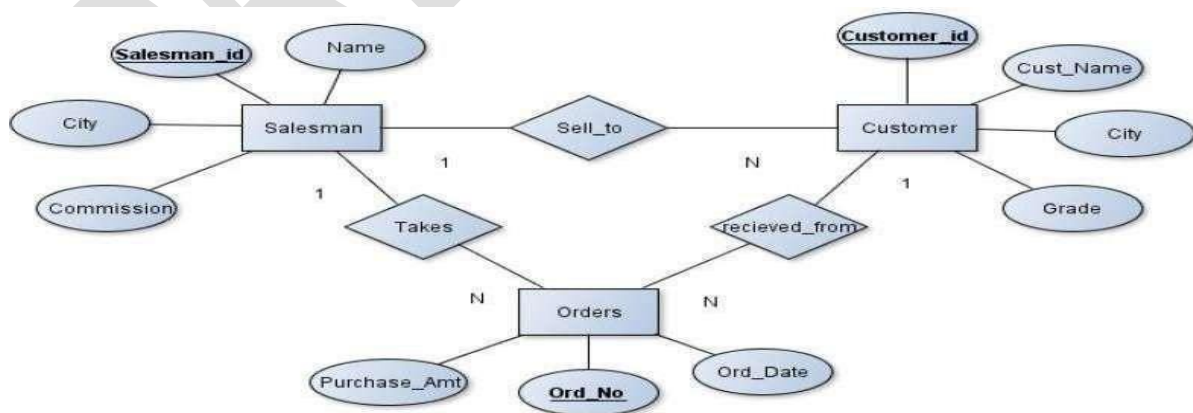
CUSTOMER (Customer_id, Cust_Name, City, Grade, Salesman_id)

ORDERS (Ord_No, Purchase_Amt, Ord_Date, Customer_id, Salesman_id)

Write SQL queries to

1. Count the customers with grades above Bangalore's average.
2. Find the name and numbers of all salesmen who had more than one customer.
3. List all salesmen and indicate those who have and don't have customers in their cities (Use UNION operation.)
4. Create a view that finds the salesman who has the customer with the highest order of a day.
5. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.

ER-Diagram:



SCHEMA:

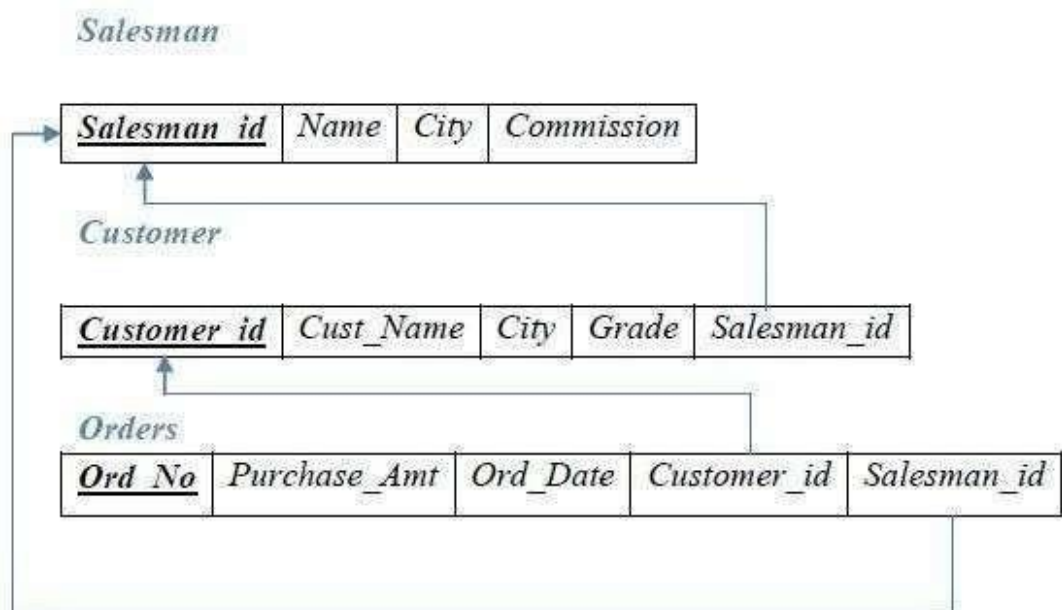


Table Creation:**SALESMAN**

```
CREATE TABLE SALESMAN (  
SALESMAN_ID NUMBER(5) CONSTRAINT SALESMAN_SALID PRIMARY KEY,  
NAME VARCHAR(10) CONSTRAINT SALESMAN_NAME_NN NOT NULL,  
CITY VARCHAR(15) CONSTRAINT SALESMAN_CITY_NN NOT NULL, COMMISSION  
NUMBER(5));
```

Table created.

CUSTOMER

```
CREATE TABLE CUSTOMER (  
CUSTOMER_ID NUMBER (5) CONSTRAINT CUSTOMER_CUSTID_PK PRIMARY KEY,  
CUST_NAME VARCHAR2 (10) CONSTRAINT CUSTOMER_CUSTNAME_NN NOT NULL,  
CITY VARCHAR(10) CONSTRAINT CUSTOMER_CITY_NN NOT NULL,  
GRADE NUMBER(5) CONSTRAINT CUSTOMER_GRADE_NN NOT NULL,  
SALESMAN_ID NUMBER(5) CONSTRAINT CUSTOMER_SALEID_FK REFERENCES  
SALESMAN(SALESMAN_ID) ON DELETE SET NULL);
```

Table created.

ORDERS

```
CREATE TABLE ORDERS (  
ORD_NO NUMBER(5) CONSTRAINT ORDERS_ODNO_PK PRIMARY KEY,  
PURCHASE_AMT INTEGER CONSTRAINT ORDERS_PAMT_NN NOT NULL,  
ORD_DATE DATE CONSTRAINT ORDERS_ODATE_NN NOT NULL,  
CUSTOMER_ID NUMBER (5) CONSTRAINT ORDERS_CUSTID_FK REFERENCES  
CUSTOMER(CUSTOMER_ID),  
SALESMAN_ID NUMBER(5) CONSTRAINT ORDERS_SALEID_FK REFERENCES  
SALESMAN(SALESMAN_ID) ON DELETE CASCADE);
```

Table created.

Values for tables

```
SQL> INSERT INTO SALESMAN VALUES(&SALESMAN_ID,&NAME,&CITY,&COMMISSION);
```

```
SQL> INSERT INTO CUSTOMER  
VALUES(&CUSTOMER_ID,&CUST_NAME,&CITY,&GRADE,&SALESMAN_ID);
```

SQL> INSERT INTO ORDERS

VALUES(&ORD_NO,&PURCHASE_AMT,'&ORD_DATE',&CUSTOMER_ID,&SALESMAN_ID);
SELECT * FROM SALESMAN;

SALESMAN_ID	NAME	CITY	COMMISSION
1000	RAJ	BENGALURU	50
2000	ASHWIN	TUMKUR	30
3000	BINDU	MUMBAI	40
4000	LAVANYA	BENGALURU	40
5000	ROHIT	MYSORE	60

SELECT * FROM CUSTOMER;

CUSTOMER_ID	CUST_NAME	CITY	GRADE	SALESMAN_ID
11	INFOSYS	BENGALURU	5	1000
22	TCS	BENGALURU	4	2000
33	WIPRO	MYSORE	7	1000
44	TCS	MYSORE	6	2000
55	ORACLE	TUMKUR	3	3000

SELECT * FROM ORDERS;

ORD_NO	PURCHASE_AMT	ORD_DATE	CUSTOMER_ID	SALESMAN_ID
1	200000	12-APR-16	11	1000
2	300000	12-APR-16	11	2000
3	400000	15-APR-17	22	1000

QUERIES:

1. Count the customers with grades above Bangalore's average.

Varient-1:

SELECT COUNT(CUSTOMER_ID)
FROM CUSTOMER

```
WHERE GRADE > (SELECT AVG(GRADE)
FROM CUSTOMER
WHERE CITY LIKE '%BENGALURU');
```

Varient-2:

```
SELECT COUNT(CUSTOMER_ID)
FROM CUSTOMER
WHERE GRADE > (SELECT AVG(GRADE)
FROM CUSTOMER
WHERE CITY = 'BENGALURU');
```

Varient-3:

```
SELECT COUNT(CUSTOMER_ID)
FROM CUSTOMER
WHERE GRADE > (SELECT AVG(GRADE)
               FROM CUSTOMER WHERE CITY IN
               ('BENGALURU', 'BANGALORE'));
```

OUTPUT:

COUNT(CUSTOMER_ID)
3

2. Find the name and numbers of all salesmen who had more than one customer.

Varient-1:

```
SELECT NAME, COUNT(CUSTOMER_ID)
FROM SALESMAN S, CUSTOMER C
WHERE S.SALESMAN_ID = C.SALESMAN_ID
GROUP BY S.SALESMAN_ID, NAME
HAVING COUNT(CUSTOMER_ID) > 1;
```

Varient-2:

```
SELECT NAME, COUNT(CUSTOMER_ID)
FROM SALESMAN S, CUSTOMER C
WHERE S.SALESMAN_ID = C.SALESMAN_ID
GROUP BY S.SALESMAN_ID, NAME
HAVING COUNT(CUSTOMER_ID) >= 2;
```

Varient-3:

```
SELECT NAME, COUNT(CUSTOMER_ID)
FROM SALESMAN S JOIN CUSTOMER C ON S.SALESMAN_ID = C.SALESMAN_ID
GROUP BY S.SALESMAN_ID, NAME
HAVING COUNT(CUSTOMER_ID) > 1;
```

OUTPUT:

NAME	COUNT(CUSTOMER_ID)
-----	-----

ASHWIN	2
RAJ	2

3. List all salesmen and indicate those who have and don't have customers in their cities
(Use UNION operation.) **Varient-1:**

```
(SELECT NAME
FROM SALESMAN S, CUSTOMER C
WHERE S.SALESMAN_ID=C.SALESMAN_ID AND S.CITY=C.CITY)
UNION
(SELECT NAME
FROM SALESMAN
WHERE SALESMAN_ID NOT IN (SELECT S1.SALESMAN_ID
FROM SALESMAN S1, CUSTOMER C1
WHERE S1.SALESMAN_ID=C1.SALESMAN_ID
AND S1.CITY=C1.CITY)); Varient-2:
```

```
(SELECT NAME
FROM SALESMAN S, CUSTOMER C
WHERE S.SALESMAN_ID=C.SALESMAN_ID AND S.CITY=C.CITY)
UNION (SELECT NAME
FROM SALESMAN
WHERE EXISTS (SELECT *
FROM SALESMAN S1, CUSTOMER C1
WHERE S1.SALESMAN_ID=C1.SALESMAN_ID
AND S1.CITY=C1.CITY)); VARIANT-3:
```

```
(SELECT NAME
FROM SALESMAN S JOIN CUSTOMER C ON S.SALESMAN_ID=C.SALESMAN_ID WHERE
S.CITY=C.CITY)
UNION
(SELECT NAME
FROM SALESMAN
WHERE SALESMAN_ID NOT IN (SELECT S1.SALESMAN_ID
FROM SALESMAN S1 JOIN CUSTOMER C1 ON
S1.SALESMAN_ID=C1.SALESMAN_ID WHERE
S1.CITY=C1.CITY));
```

OUTPUT:

NAME

ASHWIN
BINDU
LAVANYA
RAJ

4. Create a view that finds the salesman who has the customer with the highest order of a day.

Variant-1:

```
CREATE VIEW SALES_HIGHERORDER AS
SELECT SALESMAN_ID, PURCHASE_AMT
FROM ORDERS
WHERE PURCHASE_AMT = (SELECT MAX ( O.PURCHASE_AMT) FROM
                        ORDERS O
                        WHERE O.ORD_DATE = '12-APR-16');
```

View created.

Variant-2:

```
CREATE VIEW SALES_HIGHERORDER1 AS
SELECT A.SALESMAN_ID, ORD_DATE, PURCHASE_AMT
FROM SALESMAN A, ORDERS B
WHERE A.SALESMAN_ID = B.SALESMAN_ID AND
      B.PURCHASE_AMT = (SELECT MAX (PURCHASE_AMT)
                        FROM ORDERS C
                        WHERE C.ORD_DATE = B.ORD_DATE);
```

View created.

Variant-3:

```
CREATE VIEW SALES_HIGHERORDER2 AS
SELECT A.SALESMAN_ID, ORD_DATE, PURCHASE_AMT
FROM SALESMAN A JOIN ORDERS B ON A.SALESMAN_ID = B.SALESMAN_ID
WHERE B.PURCHASE_AMT = (SELECT MAX (PURCHASE_AMT)
                        FROM ORDERS C
                        WHERE C.ORD_DATE = B.ORD_DATE);
```

OUTPUT:

```
SQL> SELECT * FROM SALES_HIGHERORDER;
SALESMAN_ID PURCHASE_AMT
```

SALESMAN_ID	PURCHASE_AMT
2000	300000

```
SQL> SELECT * FROM SALES_HIGHERORDER2;
SALESMAN_ID ORD_DATE PURCHASE_AMT
```

2000	12-APR-16	300000
1000	16-APR-17	400000

5. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted. **Varient-1:**

DELETE from salesman
WHERE salesman_id = 1000;

1 row deleted.

Varient-2:

DELETE from salesman
WHERE salesman_id **IN** (1000);

1 row deleted.

Varient-3:

DELETE from customer
WHERE salesman_id **IN** (**SELECT** salesman_id
FROM customer
Where salesman_ID = 1000);

1 row deleted.

OUTPUT:

SQL> SELECT * FROM SALESMAN;

SALESMAN_ID	NAME	CITY	COMMISSION
-------------	------	------	------------

2000	ASHWIN	TUMKUR	30
3000	BINDU	MUMBAI	40
4000	LAVANYA	BENGALURU	40
5000	ROHIT	MYSORE	60

SQL> SELECT * FROM CUSTOMER;

CUSTOMER_ID	CUST_NAME	CITY	GRADE	SALESMAN_ID
-------------	-----------	------	-------	-------------

11	INFOSYS	BENGALURU		5	
22	TCS	BENGALURU	4	2000	33
				WIPRO	MYSORE
			7		
44	TCS	MYSORE		6	2000
55	ORACLE	TUMKUR		3	3000

SQL> SELECT * FROM ORDERS;

ORD_NO	PURCHASE_AMT	ORD_DATE	CUSTOMER_ID	SALESMAN_ID
2	300000	12-APR-16	11	2000

EXPIREMENT – 3

MOVIE DATABASE

3) Aim: Demonstrate the concepts of JOIN operations.

Program: Consider the schema for Movie Database:

Consider the schema for Movie Database: ACTOR

(Act_id, Act_Name, Act_Gender)

DIRECTOR (Dir_id, Dir_Name, Dir_Phone)

MOVIES (Mov_id, Mov_Title, Mov_Year, Mov_Lang, Dir_id)

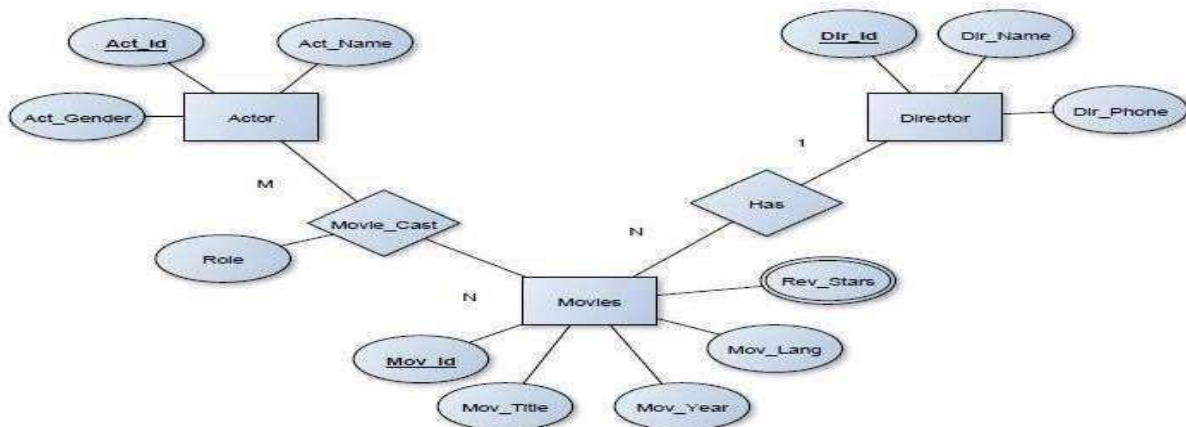
MOVIE_CAST (Act_id, Mov_id, Role)

RATING (Mov_id, Rev_Stars)

Write SQL queries to

1. List the titles of all movies directed by 'Hitchcock'.
2. Find the movie names where one or more actors acted in two or more movies.
3. List all actors who acted in a movie before 2000 and also in a movie after 2015 (use JOIN operation).
4. Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.
5. Update rating of all movies directed by 'Steven Spielberg' to 5.

ER-Diagram:



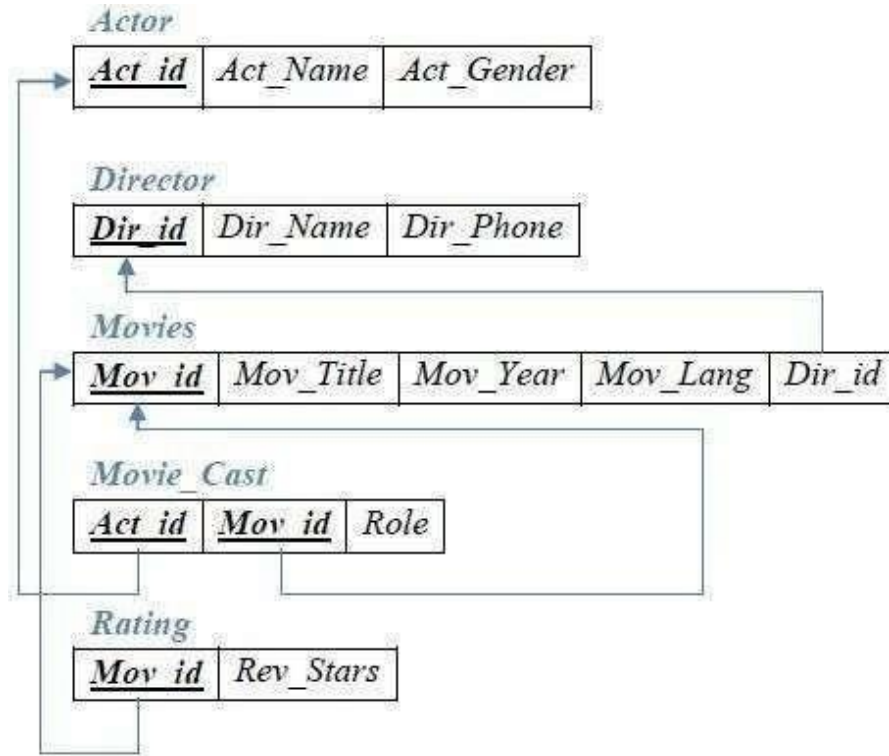
SCHEMA:

Table Creation:**ACTOR**

```
CREATE TABLE ACTOR(  
  ACT_ID NUMBER(5) CONSTRAINT ACTOR_ACTID_PK PRIMARY KEY,  
  ACT_NAME VARCHAR(18) CONSTRAINT ACTOR_ACTNAME_NN NOT NULL,  
  ACT_GENDER VARCHAR(2) CONSTRAINT ACTOR_ACTGENDER_NN NOT NULL);
```

Table created.

DIRECTOR

```
CREATE TABLE DIRECTOR(  
  DIR_ID NUMBER(5) CONSTRAINT DIRECTOR_DIRID_PK PRIMARY KEY,  
  DIR_NAME VARCHAR(18) CONSTRAINT DIRECTOR_DIRNAME_NN NOT NULL,  
  DIR_PHONE VARCHAR(10) CONSTRAINT DIRECTOR_DIRPHONE_NN NOT NULL);
```

Table created.

MOVIES

```
CREATE TABLE MOVIES(  
  MOV_ID NUMBER(5) CONSTRAINT MOVIES_MOVID_PK PRIMARY KEY,  
  MOV_TITLE VARCHAR(10) CONSTRAINT MOVIES_MOVTITLE_NN NOT NULL,  
  MOV_YEAR NUMBER(5) CONSTRAINT MOVIES_MOVYEAR_NN NOT NULL,  
  MOV_LANG VARCHAR(10) CONSTRAINT MOVIES_MOVLANG_NN NOT  
  NULL,  
  DIR_ID NUMBER(5) CONSTRAINT MOVIES_DIRID_FK REFERENCES DIRECTOR(DIR_ID));
```

Table created.

MOVIE_CAST

```
CREATE TABLE MOVIE_CAST(  
  ACT_ID NUMBER(5) CONSTRAINT MOVIECAST_ACTID_FK REFERENCES ACTOR(ACT_ID),  
  MOV_ID NUMBER(5) CONSTRAINT MOVIECAST_MOVID_FK REFERENCES  
  MOVIES(MOV_ID),ROLE VARCHAR(10),  
  CONSTRAINT MOVIECAST_ACTID_MOVID_PK PRIMARY KEY(ACT_ID,MOV_ID));
```

Table created.

RATING

```
CREATE TABLE RATING(  
MOV_ID NUMBER(5) CONSTRAINT RATING_MOVID_FK REFERENCES MOVIES(MOV_ID),  
REV_STARS NUMBER(1) CONSTRAINT RATING_REVSTARS_NN NOT NULL, CONSTRAINT  
RATING_MOVID_PK PRIMARY KEY(MOV_ID))
```

Table created.

PREETI M H

Description of Schema:

SQL> DESC ACTOR

Name	Null?	Type
ACT_ID	NOT NULL	NUMBER(5)
ACT_NAME	NOT NULL	VARCHAR2(18)
ACT_GENDER	NOT NULL	VARCHAR2(2)

SQL> DESC DIRECTOR

Name	Null?	Type
DIR_ID	NOT NULL	NUMBER(5)
DIR_NAME	NOT NULL	VARCHAR2(18)
DIR_PHONE	NOT NULL	VARCHAR(10)

SQL> DESC MOVIES

Name	Null?	Type
MOV_ID	NOT NULL	NUMBER(5)
MOV_TITLE	NOT NULL	VARCHAR2(10)
MOV_YEAR	NOT NULL	NUMBER(5)
MOV_LANG	NOT NULL	VARCHAR2(10)
DIR_ID		NUMBER(5)

SQL> DESC RATING

Name	Null?	Type
MOV_ID	NOT NULL	NUMBER(5)
REV_STARS	NOT NULL	NUMBER(1)

Values for tables:

SQL> INSERT INTO ACTOR VALUES(&ACT_ID,&ACT_NAME,&ACT_GENDER);

SQL> INSERT INTO DIRECTOR VALUES(&DIR_ID,&DIR_NAME,&DIR_PHONE);

SQL> INSERT INTO MOVIES VALUES (&MOV_ID,'&MOV_TITLE', '&MOV_YEAR',
'&MOV_LANG', &DIR_ID);

SQL> INSERT INTO MOVIE_CAST VALUES(&ACT_ID,&MOV_ID,&ROLE);

SQL> INSERT INTO RATING VALUES (&MOV_ID, &REV_STARS);

SQL> SELECT * FROM ACTOR;

ACT_ID	ACT_NAME	AC
--------	----------	----

111	DEEPA SANNIDHI	F
222	SUDEEP	M
333	PUNEETH	M
444	DHIGANTH	M
555	ANGELA	F

V Sem CSE-AI

SQL> SELECT * FROM DIRECTOR;

DIR_ID	DIR_NAME	DIR_PHON E

101	HITCHCOCK	112267809
102	RAJ MOULI	152358709
103	YOGARAJ	272337808
104	STEVEN SPIELBERG	363445678
105	PAVAN KUMAR	385456809

SQL> SELECT * FROM MOVIES;

MOV_ID	MOV_TITLE	MOV_YEAR	MOV_LANG	DIR_ID

1111	LASTWORLD	2009	ENGLISH	104
2222	EEGA	2010	TELUGU	102
4444	PARAMATHMA	2012	KANNADA	103
3333	MALE	2006	KANNADA	103
5555	MANASARE	2010	KANNADA	103
6666	REAR WINDOW	1954	ENGLISH	101
7777	NOTORIOUS	1946	ENGLISH	101

SQL> SELECT * FROM MOVIE_CAST;

ACT_ID	MOV_ID	ROLE

222	333	2222 VILAN
111		4444 HERO
444		4444 HEROIN
		3333 GUEST
444555		5555 HERO
		7777 MOTHER

```
SQL> SELECT * FROM RATING;  
      MOV_ID REV_STARS
```

1111	3
2222	4
3333	3
5555	4
4444	5

1. List the titles of all movies directed by 'Hitchcock'.

Variant-1

```
SELECT MOV_TITLE  
FROM MOVIES M, DIRECTOR D  
WHERE D.DIR_ID=M.DIR_ID AND  
      DIR_NAME='HITCHCOCK';
```

Variant-2

```
SELECT MOV_TITLE  
FROM MOVIES M NATURAL JOIN DIRECTOR D  
WHERE DIR_NAME='HITCHCOCK';
```

Variant-3

```
SELECT MOV_TITLE  
FROM MOVIES M INNER JOIN DIRECTOR D ON D.DIR_ID=M.DIR_ID  
WHERE DIR_NAME='HITCHCOCK';
```

OUTPUT

MOV_TITLE

NOTORIOUS
REAR WINDOW

2. Find the movie names where one or more actors acted in two or more movies.

VARIANT-1

```
SELECT MOV_TITLE
FROM MOVIES M, MOVIE_CAST MC
WHERE M.MOV_ID=MC.MOV_ID AND
      MC.ACT_ID IN (SELECT ACT_ID
                    FROM MOVIE_CAST
                    GROUP BY ACT_ID
                    HAVING COUNT(MOV_ID) >=2) ;
```


VARIANT-2

```

SELECT MOV_TITLE
FROM MOVIES M NATURAL JOIN MOVIE_CAST MC
WHERE MC.ACT_ID IN (SELECT ACT_ID
                    FROM MOVIE_CAST
                    GROUP BY ACT_ID
                    HAVING COUNT(MOV_ID) >= 2);

```

VARIANT-3

```

SELECT MOV_TITLE
FROM MOVIES M NATURAL JOIN MOVIE_CAST MC
WHERE MC.ACT_ID NOT IN (SELECT ACT_ID
                       FROM MOVIE_CAST
                       GROUP BY ACT_ID
                       HAVING COUNT(MOV_ID) < 2);

```

MOV_TITLE

MALE

MANASARE

1. List all actors who acted in a movie before 2000 and also in a movie after 2015 (use JOIN operation).

VARIANT-1

```

(SELECT ACT_NAME
FROM ACTOR A JOIN MOVIE_CAST C ON A.ACT_ID=C.ACT_ID
JOIN MOVIES M ON C.MOV_ID=M.MOV_ID
WHERE M.MOV_YEAR < 2000)

```

INTERSECT

```

(SELECT ACT_NAME
FROM ACTOR A JOIN MOVIE_CAST C ON A.ACT_ID=C.ACT_ID JOIN
MOVIES M ON C.MOV_ID=M.MOV_ID
WHERE M.MOV_YEAR > 2015);

```

VARIANT-2

```

(SELECT ACT_NAME

```

```

FROM ACTOR A NATURAL JOIN MOVIE_CAST C JOIN MOVIES M ON
C.MOV_ID=M.MOV_ID
WHERE M.MOV_YEAR < 2000)
INTERSECT
(SELECT ACT_NAME
FROM ACTOR A JOIN MOVIE_CAST C ON A.ACT_ID=C.ACT_ID JOIN
MOVIES M ON C.MOV_ID=M.MOV_ID
WHERE M.MOV_YEAR > 2015); VARIANT-
3

SELECT ACT_NAME
FROM ACTOR A NATURAL JOIN MOVIE_CAST C NATURAL JOIN MOVIES M
WHERE M.MOV_YEAR < 2000)
INTERSECT
(SELECT ACT_NAME
FROM ACTOR A JOIN MOVIE_CAST C ON A.ACT_ID=C.ACT_ID JOIN
MOVIES M ON C.MOV_ID=M.MOV_ID
WHERE M.MOV_YEAR > 2015);

```

OUTPUT:

ACT_NAME

DHIGANTH

4. Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.

VARIANT-1

```

SELECT MOV_TITLE, REV_STARS
FROM MOVIES M, RATING R
WHERE M.MOV_ID=R.MOV_ID AND REV_STARS>=1
ORDER BY MOV_TITLE

```

VARIANT-2

```

SELECT MOV_TITLE, REV_STARS
FROM MOVIES AS M, RATING AS R
WHERE M.MOV_ID=R.MOV_ID AND REV_STARS>=1
ORDER BY MOV_TITLE

```

VARIANT-3

```

SELECT MOV_TITLE, REV_STARS
FROM MOVIES M, RATING R
WHERE M.MOV_ID=R.MOV_ID AND REV_STARS>=1
ORDER BY MOV_TITLE ASC

```

```

MOV_TITLE REV_STARS
-----

```

```

EEGA          4
LASTWORLD     3
MALE          3
MANASARE      4
PARAMATHMA    5

```

5. Update rating of all movies directed by 'Steven Spielberg' to 5.

VARIANT-1

```

UPDATE RATING
SET REV_STARS=5
WHERE MOV_ID IN (SELECT MOV_ID
                  FROM MOVIES M, DIRECTOR D
                  WHERE M.DIR_ID=D.DIR_ID AND
                  DIR_NAME='STEVEN SPIELBERG');

```

1 row updated.

VARIANT-2

```

UPDATE RATING
SET REV_STARS=5
WHERE MOV_ID IN (SELECT MOV_ID
                  FROM MOVIES M, DIRECTOR D
                  WHERE M.DIR_ID=D.DIR_ID AND
                  DIR_NAME LIKE 'STEVEN
                  SPIELBERG');

```

VARIANT-3

```
UPDATE RATING
SET REV_STARS>4
WHERE MOV_ID IN (SELECT MOV_ID
                  FROM MOVIES M NATURAL JOIN
                  DIRECTOR D WHERE DIR_NAME
                  LIKE 'STEVEN SPIELBERG') ;
```

OUTPUT :

```
SELECT * FROM RATING
```

```
MOV_ID REV_STARS
```

MOV_ID	REV_STARS
1111	5
2222	4
3333	3
5555	4
4444	5

EXPIREMENT - 4

COLLEGE DATABASE

4) Aim: Introduce concepts of PLSQL and usage on the table.

Program: Consider the schema for College Database:

Consider the schema for College Database:

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

SEMSEC (SSID, Sem, Sec)

CLASS (USN, SSID)

SUBJECT (Subcode, Title, Sem, Credits)

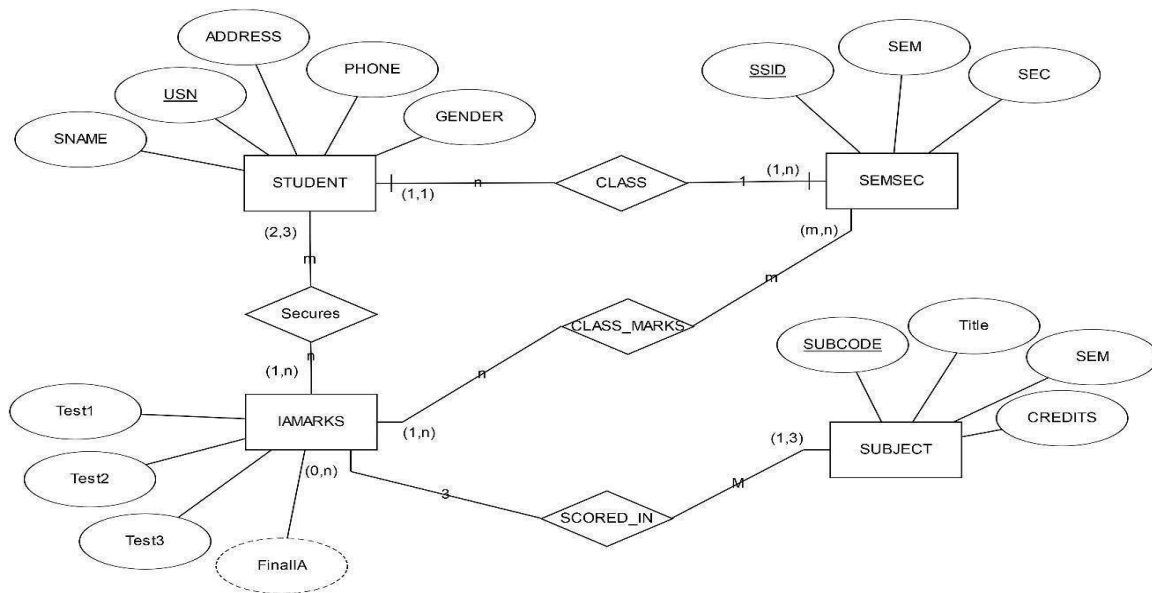
IAMARKS (USN, Subcode, SSID, Test1, Test2, Test3, FinalIA)

Write SQL queries to

1. List all the student details studying in fourth semester 'C' section.
2. Compute the total number of male and female students in each semester and in each section.
3. Create a view of Test1 marks of student USN '1BI15CS101' in all subjects.
4. Calculate the FinalIA (average of best two test marks) and update the corresponding table for all students.
5. Categorize students based on the following criterion: If FinalIA = 17 to 20 then CAT = 'Outstanding'
If FinalIA = 12 to 16 then CAT = 'Average'
If FinalIA < 12 then CAT = 'Weak'

Give these details only for 8th semester A, B, and C section students.

ER-Diagram:



SCHEMA:



18CSL58 DBMS Laboratory with Mini Project
Table Creation:

STUDENT

```
CREATE TABLE STUDENT (USN  
VARCHAR(10) PRIMARY KEY,  
SNAME VARCHAR(25),  
ADDRESS VARCHAR(25),  
PHONE VARCHAR(10),  
GENDER CHAR(1));
```

Table created.

SEMSEC

```
CREATE TABLE SEMSEC  
SSID VARCHAR(5) PRIMARY KEY,  
SEM NUMBER(2),  
SEC CHAR(1));
```

Table created.

V Sem CSE-AI

CLASS

```
CREATE TABLE CLASS  
(USN VARCHAR(10),  
SSID VARCHAR(5),  
PRIMARYKEY(USN,SSID),  
FOREIGN KEY(USN) REFERENCES STUDENT(USN),  
FOREIGN KEY(SSID) REFERENCES SEMSEC(SSID));
```

Table created.

SUBJECT

```
CREATE TABLE SUBJECT  
(SUBCODE VARCHAR2 (8) PRIMARY  
KEY,TITLE VARCHAR2 (20),  
SEM NUMBER (2),  
CREDITS NUMBER (2));
```

Table created.

IAMARKS

```
CREATE TABLE IAMARKS (
USN VARCHAR(10),
SUBCODE VARCHAR(8),
SSID VARCHAR(5),
TEST1NUMBER(2),
TEST2 NUMBER(2),
TEST3 NUMBER (2),
FINALIA NUMBER (3),
PRIMARY KEY (USN, SUBCODE, SSID),
FOREIGN KEY(USN) REFERENCES STUDENT(USN),
FOREIGN KEY(SUBCODE) REFERENCES SUBJECT(SUBCODE),
FOREIGN KEY(SSID) REFERENCES SEMSEC(SSID));
```

Table created.

Values for tables:**STUDENT:**

```
INSERT INTO STUDENT VALUES ('&USN','&sname','&address','&phone','&gender');
```

```
SQL> select * from student;
```

USN	SNAME	ADDRESS	PHONE	G
1cg15cs001	Abhi tumkur	9875698410 M	1cg15cs002 amulya gubbi	8896557412 F
1cg16me063	chethan nittur	7894759522 M		

CSE-AISVCE

1cg14ec055 raghavi	sspuram	9485675521	F
1cg15ee065 sanjay	bangalore	9538444404	M

SEMSEC:

INSERT INTO SEMSEC VALUES ('&SSID', '&sem','&sec');

select * from semsec;

SSID	SEM S
5A	5A 3B 3B 7A
7A	
2C 2 C 4B 4 B	
4c	4c

CLASS:

INSERT INTO CLASS VALUES ('&USN','&SSID');

select * from class;

USN	SSID
1cg15cs001	5A
1cg15cs002	5A
1cg16me063	3B
1cg14ec055	7A
1cg15ee065	3B
1cg15ee065	4c
1cg15cs002	4c

SUBJECT:

INSERT INTO SUBJECT VALUES ('10CS81','ACA', 8, 4);

select * from subject;

SUBCODE	TITLE	SEM	CREDITS
15cs53	dbms	5	4
15cs33	ds	3	4
15cs34	co	3	4

15cs158	dba	52	
10cs71	oomd	7	4

IAMARKS:

INSERT INTO IAMARKS VALUES

('&USN', '&SUBCODE', '&SSID', '&TEST1', '&TEST2', '&TEST3');

select * from iamarks;

USN	SUBCODE	SSID	TEST1	TEST2	TEST3	FINALIA
1cg15cs001	15cs53	5A	18	19	15	19
1cg15cs002	15cs53	5A	15	16	14	16
1cg16me063	15cs33	3B	10	15	16	16
1cg14ec055	10cs71	7A	18	20	21	21
1cg15ee065	15cs33	3B	16	20	17	19
1cg15ee065	15cs53	4c	19	20	18	20

Queries:

1. List all the student details studying in fourth semester 'C' section.

VARIANT-1:

```
select s.usn, sname, address, phone, gender from student
s, class c, semsec ss where sem=4 and sec='c' and
s.ssid=c.ssid and c.usn=s.usn;
```

VARIANT-2 select s.usn, sname, address, phone,
gender from student s, class c, semsec ss
where sem=4 and sec LIKE '%c' and ss.ssid=c.ssid and
c.usn=s.usn;

VARIANT-3

```
select s.usn, sname, address, phone, gender from student as
s, class as c, semsec ss where sem =4 and sec LIKE
'%c' and ss.ssid=c.ssid and c.usn=s.usn;
```

output

USN	SNAME	ADDRESS	PHONE	G
1cg15ee065	Sanjay	bangalore	9538444404	M
1cg15cs002	Amulya	gubbi	8896557412	F

2. Compute the total number of male and female students in each semester and in each section.

Variant-1

```
SELECT SEM, SEC, GENDER, COUNT(*)
FROM STUDENT S, SEMSEC SS, CLASS C
WHERE S.USN=C.USN AND C.SSID=SS.SSID
GROUP BY SEM, SEC, GENDER
ORDER BY SEM;
```

Variant-2

```
SELECT SEM, SEC, GENDER, COUNT(*)
FROM STUDENT S, SEMSEC SS, CLASS C
WHERE S.USN=C.USN AND C.SSID=SS.SSID
GROUP BY SEM, SEC, GENDER
ORDER BY SEM ASC;
```

Variant-3

```
SELECT SEM, SEC, GENDER, COUNT(*)
FROM STUDENT as S, SEMSEC as SS, CLASS C
WHERE S.USN=C.USN AND C.SSID=SS.SSID
GROUP BY SEM, SEC, GENDER
ORDER BY SEM ASC;
```

OUTPUT:

SEM	S	G	COUNT (*)
3	B	M	2
4	c	F	1
			1
4	c	M	1
			1
5	A	F	1
			1
5	A	M	
7	A	F	

3. Create a view of Test1 marks of student USN '1BI15CS101' in all subjects.

Variant-1

```
CREATE VIEW TEST1 AS
SELECT SUBCODE, TEST1
FROM IAMARKS
WHERE USN='1cg15ee065';
```

View created.

Variant-2

```
CREATE VIEW TEST1 AS
SELECT SUBCODE, TEST1
FROM IAMARKS
WHERE USN LIKE '1cg15ee065'; View
created.
```

Variant-3

```
CREATE VIEW TEST1 AS
SELECT SUBCODE, TEST1
FROM IAMARKS
WHERE USN = '1cg15ee065';
Order by TEST1; View
created.
```

SQL> select * from test1;

SUBCODE	TEST1
15cs33	16
15cs53	19

4. Calculate the FinalIA (average of best two test marks) and update the corresponding table for all students.

```
CREATE OR REPLACE PROCEDURE AVG IS
CURSOR C_IAMARKS IS
SELECT GREATEST(TEST1,TEST2) AS A, GREATEST(TEST1,TEST3) AS B,
GREATEST(TEST3,TEST2) AS C FROM IAMARKS
WHERE FINALIA IS NULL

FOR UPDATE;
```

```

C_A NUMBER;
C_B NUMBER;
C_C NUMBER;
C_SM UMBER;C_AV NUMBER;
BEGIN
OPEN C_IAMARKS; LOOP
FETCH C_IAMARKS INTO C_A, C_B, C_C; EXIT WHEN C_IAMARKS%NOTFOUND;
DBMS_OUTPUT.PUT_LINE(C_A||' '||C_B||' '||C_C); IF(C_A!=C_B) THEN
C_SM:=C_A+C_B; ELSE
C_SM:=C_A+C_C; END IF; C_AV:=C_SM/2;
DBMS_OUTPUT.PUT_LINE('SUM='||C_SM);
DBMS_OUTPUT.PUT_LINE('AVERAGE='||C_AV); UPDATE IAMARKS
SET FINALIA=C_AV
WHERE CURRENT OF C_IAMARKS; END LOOP;

```

CLOSE C_IAMARKS; END AVG;

Procedure created.

SQL> BEGIN

2 AVG;

3 END;

PL/SQL procedure successfully completed.

SQL> SELECT * FROM IAMARKS;

USN	SUBCODE	SSID	TEST1	TEST2	TEST3	FINALIA
1cg15cs001	15cs53	5A	18	19	15	19
1cg15cs002	15cs53	5A	15	16	14	16
1cg16me063	15cs33	3B	10	15	16	16
1cg14ec055	10cs71	7A	18	20	21	21
1cg15ee065	15cs33	3B	16	20	17	19
1cg15ee065	15cs53	4c	19	20	18	20

6 rows selected.

5. Categorize students based on the following criterion:

If FinalIA = 17 to 20 then CAT = 'Outstanding' If

FinalIA = 12 to 16 then CAT = 'Average'

If FinalIA < 12 then CAT = 'Weak'

Give these details only for 8th semester A, B, and C section students.

```
SQL> SELECT S.USN, S.SNAME, S.ADDRESS, S.PHONE, S.GENDER,
CASE
    WHEN IA.FINALIA BETWEEN 17 AND 20 THEN 'OUTSTANDING'
    WHEN IA.FINALIA BETWEEN 12 AND 16 THEN 'AVERAGE'
    ELSE 'WEAK'
END AS CAT
FROM STUDENT S, SEMSEC SS, IAMARKS IA, SUBJECT SUB
WHERE S.USN=IA.USN AND SS.SSID=IA.SSID AND
SUB.SUBCODE=IA.SUBCODE AND SUB.SEM=7
```

USN	SNAME	ADDRESS	PHONE	G CAT
1cg14ec055	raghavi	sspuram	9485675521	F WEAK

EXPIREMENT – 5

COMPANY DATABASE

5). Aim: Demonstrate the core concepts on table like nested and correlated nesting queries and also EXISTS and NOT EXISTS keywords.

Program: Consider the schema for Company Database:

Consider the schema for Company Database:

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

DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate)

DLOCATION (DNo,DLoc)

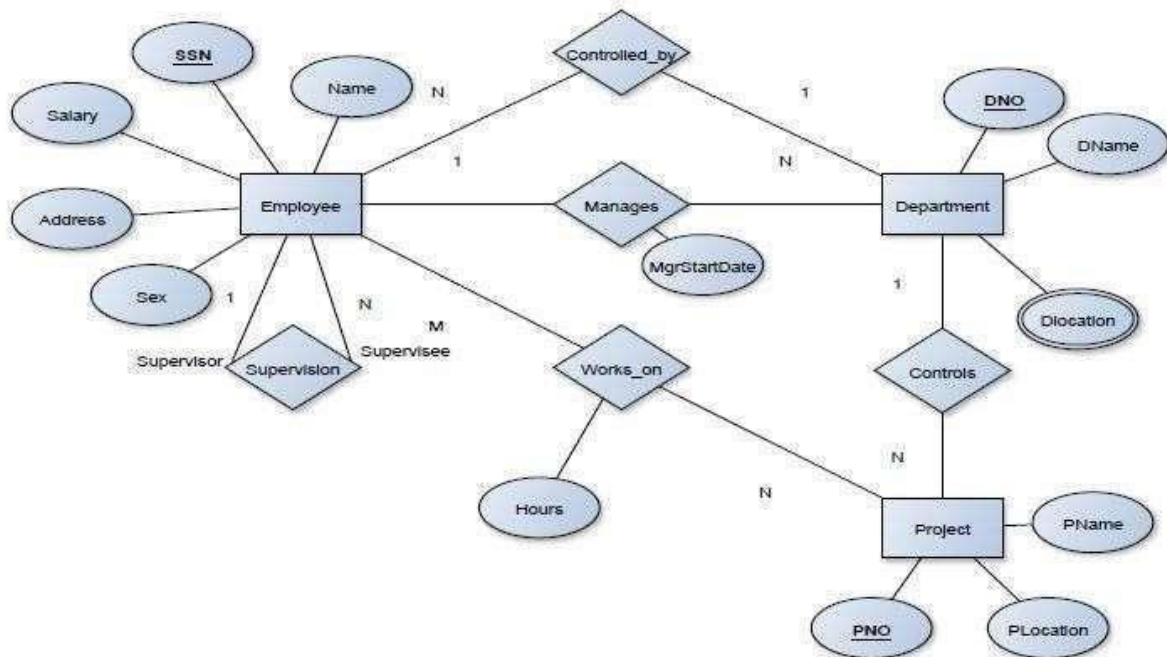
PROJECT (PNo, PName, PLocation,

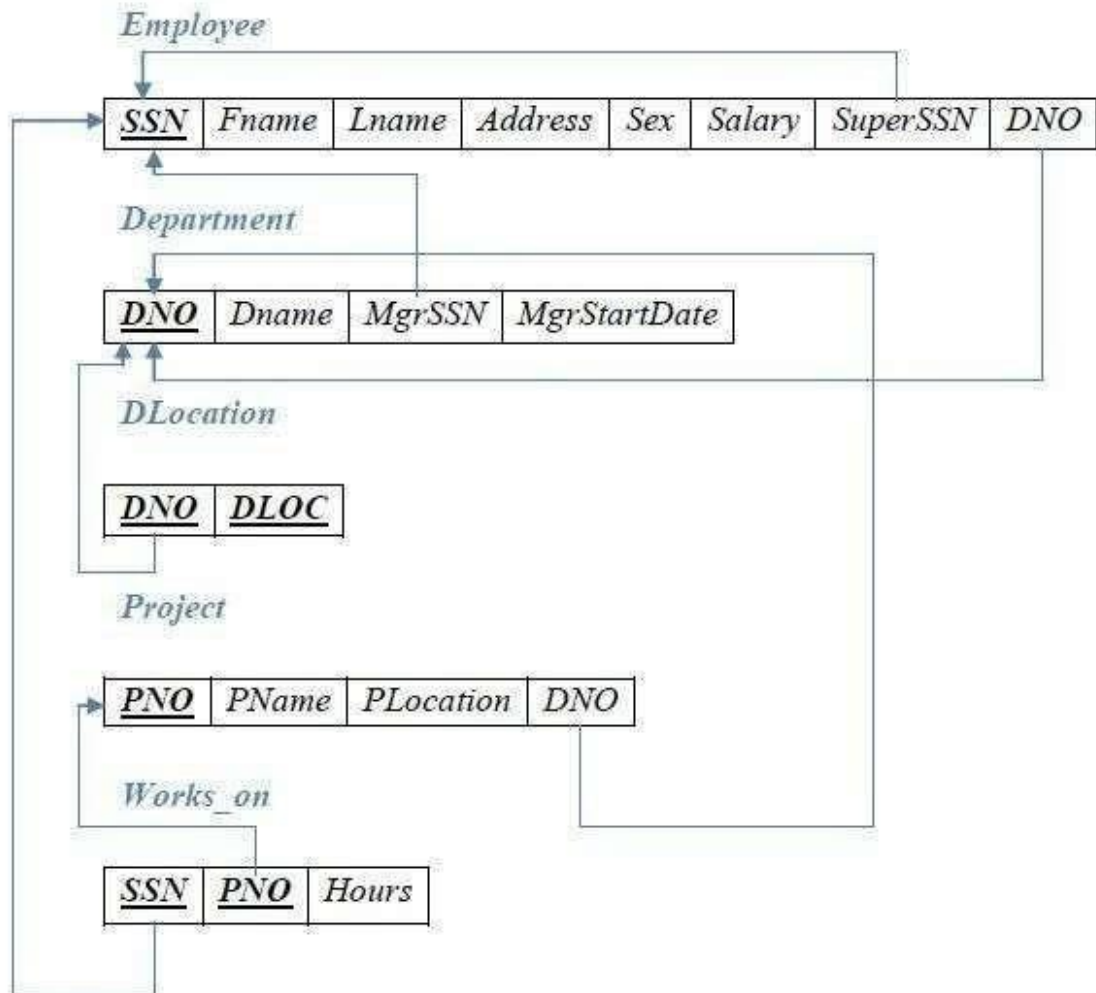
DNo) WORKS_ON (SSN, PNo, Hours) Write

SQL queries to

1. Make a list of all project numbers for projects that involve an employee whose last name is 'Scott', either as a worker or as a manager of the department that controls the project.
2. Show the resulting salaries if every employee working on the 'IoT' project is given a 10 percent raise.
3. Find the sum of the salaries of all employees of the 'Accounts' department, as well as the maximum salary, the minimum salary, and the average salary in this department
4. Retrieve the name of each employee who works on all the projects controlled by department number 5 (use NOT EXISTS operator).
5. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs. 6,00,000.

ER-Diagram:

**SCHEMA:**

**Table Creation:****DEPARTMENT**

```
CREATE TABLE DEPARTMENT(
DNO NUMBER(3) CONSTRAINT DEPT_DNO_PK PRIMARY KEY, DNAME
VARCHAR(15) CONSTRAINT DEPT_DNAME_NN NOT NULL, MGRSSN
CHAR(10),
MGRSTARTDATE DATE);
```

EMPLOYEE

```
CREATE TABLE EMPLOYEE(
SSN CHAR(10) CONSTRAINT EMP_SSN_PK PRIMARY KEY, NAME
VARCHAR(18) CONSTRAINT EMP_NAME_NN NOT NULL,
ADDRESS VARCHAR(18),
SEX VARCHAR(3), SALARY
```

REAL, SUPER_SSN

CHAR(10),

DNO NUMBER(3) CONSTRAINT EMP_DNO_FK REFERENCES DEPARTMENT(DNO));

ALTER TABLE DEPARTMENT ADD CONSTRAINT DEPT_MGRSSN_FK FOREIGN
KEY(MGRSSN) REFERENCES EMPLOYEE(SSN);

Table altered.

DLOCATION

CREATE TABLE DLOCATION(
DLOC VARCHAR2 (20),
DNO REFERENCES DEPARTMENT (DNO),
PRIMARY KEY (DNO, DLOC));

PROJECT

CREATE TABLE PROJECT(
PNO INTEGER PRIMARY KEY,
PNAME VARCHAR2 (20),
PLOCATION VARCHAR2 (20),
DNO REFERENCES DEPARTMENT (DNO));

WORKS_ON

CREATE TABLE
WORKS_ON(HOURS
NUMBER (2),
SSN REFERENCES EMPLOYEE (SSN),
PNO REFERENCES PROJECT(PNO),
PRIMARY KEY (SSN, PNO));

Values for tables:

DEPARTMENT

INSERT INTO DEPARTMENT VALUES(&DNO,&DNAME,&MGRSSN,&MGRSTARTDATE);

SELECT * FROM DEPARTMENT;

DNO	DNAME	MGRSSN	MGRSTARTD
1	RESEARCH	111111	10-AUG-12
2	ACCOUNTS	222222	10-AUG-10
3	AI	333333	15-APR-12

4 NETWORKS

111111

18-MAY-14

5 BIGDATA

666666

21-JAN-10

5 rows selected.

EMPLOYEE

INSERT INTO EMPLOYEE

VALUES('&SSN','&NAME','&ADDRESS','&SEX','&SALARY','&SUPERSRN','&DNO);

SELECT * FROM EMPLOYEE;

SSN	NAME	ADDRESS	SE	SALARY	SUPERSRN	DNO
111111	RAJ	BENGALURU	M	700000		1
222222	RASHMI	MYSORE	F	400000	111111	2
333333	RAGAVI	TUMKUR	F	800000	3	4
444444	RAJESH	TUMKUR	M	650000	333333	3
555555	RAVEESH	BENGALURU	M	500000	333333	3
666666	SCOTT	ENGLAND	M	700000	444444	5
777777	NIGANTH	GUBBI	M	200000	222222	2
888888	RAMYA	GUBBI	F	400000	222222	3
999999	VIDYA	TUMKUR	F	650000	333333	3
100000	GEETHA	TUMKUR	F	800000		3

10 rows selected.

DLOCATION

INSERT INTO DLOCATION VALUES(&DNO,'&DLOC');

SELECT * FROM DLOCATION;

DNO DLOC

1	MYSORE
1	TUMKUR
2	BENGALURU

3 GUBBI
4 DELHI
5 BENGALURU

6 rows selected.

PROJECT

INSERT INTO PROJECT VALUES(&PNO,'&PNAME','&PLOCATION','&DNO');

SELECT * FROM PROJECT;

PNO	PNAME	PLOCATION	DNO
111	IOT	GUBBI	3
222	TEXTSPEECH	GUBBI	3
333	IPSECURITY	DELHI	4
444	TRAFICANAL	BENGALURU	5
555	CLOUDSEC	DELHI	1

5 rows selected.

WORKS_ON

INSERT INTO WORKS_ON VALUES('&SSN',&PNO,&HOURS);

SELECT * FROM WORKS_ON
;

SSN	PNO	HOURS
666666	333	4
666666	111	2
111111	222	3
555555	222	2
333333	111	4
444444	111	6
222222	111	2

8 rows selected.

1. Make a list of all project numbers for projects that involve an employee whose lastname is 'Scott', either as a worker or as a manager of the department that controls the project.

Variant-1

```
(SELECT DISTINCT PNO
FROM PROJECT P, DEPARTMENT, D.EMPLOYEE E
WHERE P.DNO=D.DNO AND
      SSN=MGRSSN AND NAME='SCOTT')
UNION
(SELECT DISTINCT P.PNO
FROM PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE P.PNO=W.PNO AND W.SSN=E.SSN AND NAME='SCOTT');
```

Variant-2

```
(SELECT DISTINCT PNO
FROM PROJECT P, DEPARTMENT D, EMPLOYEE E
WHERE P.DNO=D.DNO AND SSN=MGRSSN AND NAME='SCOTT') EXIST
(SELECT DISTINCT P.PNO
FROM PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE P.PNO=W.PNO AND W.SSN=E.SSN AND NAME='SCOTT');
```

Variant-3

```
SELECT DISTINCT P.PNO
FROM PROJECT P, DEPARTMENT D, EMPLOYEE E
WHERE E.DNO=D.DNO AND D.DNO=P.DNO AND
      (E.LNAME='SCOTT' OR
       D.MGR_SS IN
         (SELECT SSN FROM EMPLOYEE WHERE
          LNAME='SCOTT'));
```

PN

O11

1 333

444

2. Show the resulting salaries if every employee working on the 'IoT' project is given a 10 percent raise.

Variant-1

```
SELECT FNAME, LNAME, 1.1*SALARY AS INCR_SAL
FROM EMPLOYEE E, WORKS_ON W, PROJECT P
WHERE E.SSN=W.SSN AND W.PNO=P.PNO
AND P.PNAME='IOT';
```

Variant-2

```
SELECT E.SSN, E.FNAME, SUM(E.SALARY
+(E.SALARY*0.1) AS HIKE_10_PER
FROM EMPLOYEE E, DEPARTMENT D WHERE
E.DNO=D.DNO
AND D.DNAME='IOT' GROUP
BY SSN, FNAME;
```

Variant-3

```
SELECT FNAME, LNAME, 1.1*SALARY AS INCR_SAL
FROM EMPLOYEE E WHERE
E.SSN
IN (SELECT SSN
FROM WORKS_ON W, PROJECT P
WHERE W.SSN AND W.PNO=P.PNO
AND P.PNAME='IOT');
```

SSN	NAME	ADDRESS	SEX	SALARY	SUPERSSN	DNO	111111	RAJ
BENGALURU	M	700000	1 222222	RASHMI	MYSORE	F		
440000111111		2 333333	RAGAVI	TUMKUR	F	880000		
3 444444	RAJESH	TUMKUR	M	715000	333333	3		
555555	RAVEESH	BENGALURU	M	500000	333333	3		
666666	SCOTT	ENGLAND	M	770000	444444	5 777777		
NIGANTH	GUBBI	M	200000	222222	2 888888	RAMYA	GUBBI	F
400000	222222	3						
999999	VIDYA	TUMKUR	F	650000	333333	3 100000	GEETHA	
	TUMKUR	F	800000		3			

10 rows selected.

3. Find the sum of the salaries of all employees of the 'Accounts' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

Variant-1

```
SELECT SUM(SALARY),MAX(SALARY),MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE E,DEPARTMENT D
WHERE DNAME='ACCOUNTS' AND D.DNO=E.DNO;
```

Variant-2

```
SELECT SUM(SALARY) AS SUM_SALARY,
MAX(SALARY) AS MAX_SALARY,
MIN(SALARY) AS MIN_SALARY,
AVG(SALARY) AS AVG_SALARY
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.DN=D.DNO AND D.DNAME='ACCOUNTS';
```

Variant-3

```
SELECT SUM(SALARY),MAX(SALARY),MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE E,DEPARTMENT D
WHERE DNAME='ACCOUNTS' AND D.DNO=E.DNO;

SUM (SALARY)MAX(SALARY)MIN(SALARY)AVG(SALARY)
```

-	-	-	-
-	440000	200000	320000

4. Retrieve the name of each employee who works on all the projects controlled by department number 5 (use NOT EXISTS operator).

Variant-1

```
SELECT NAME FROM EMPLOYEE
WHERE NOT EXISTS((SELECT PNO
                  FROM PROJECT
                  WHERE DNO=5) MINUS (SELECT
                  PNO
                  FROM WORKS_ON
                  WHERE E.SSN=W.SSN));
```

Variant-2

```
SELECT NAME FROM EMPLOYEE
WHERE NOT EXISTS((SELECT PNO
                  FROM PROJECT
                  WHERE DNO=5)
                  NOT EXISTS (SELECT PNO FROM
                  WORKS_ON
                  WHERE E.SSN=W.SSN));
```

Variant-3

```
SELECT NAME FROM EMPLOYEE
WHERE NOT EXISTS((SELECT PNO
                  FROM PROJECT WHERE DNO=5)
                  MINUS (SELECT PNO
                  FROM WORKS_ON
                  WHERE E.SSN=W.SSN)
                  )
```

NAME

SCOTT

5. For each department that has more than five employees, retrieve the department number
And the number of its employees who are making more than Rs.6,00,000.

Variant-1

```
SELECT DNO, COUNT(SSN)
FROM EMPLOYEE
WHERE SALARY>600000 AND DNO IN (SELECT DNO
FROM EMPLOYEE
GROUP BY DNO
HAVING COUNT(SSN)>5) GROUP BY
DNO;
```

Variant-2

```
SELECT DNO, COUNT(*) AS NO-OF-EMP FROM
EMPLOYEE E, DEPARTMENT D
WHERE E.DNO= D.DNO AND E.SALARY>600000 AND
DNO IN (SELECT E1.DNO FROM
EMPLOYEE E1
GROUP BY E1.DNO
HAVING COUNT(*)>5)
GROUP BY DNO;
```

Variant-3

```
SELECT DNO,COUNT(SSN) FROM
EMPLOYEE E
WHERE SALARY>600000 AND DNO EXISTS (SELECT
DNO
FROM EMPLOYEE E
GROUP BY DNO
HAVING COUNT(SSN)>5)
GROUP BY DNO;
```

DNO COUNT(SSN)

3

4

PREETIMH

BIBLIOGRAPHY

1. Elmasri and Navathe: Fundamentals of Database Systems, 5th Edition, Addison-Wesley, 2007
2. Raghu Ramakrishnan and Johannes Gehrke: Database Management Systems, 3rd Edition, McGraw-Hill, 2003.
3. Silberschatz, Korth and Sudharshan: Data base System Concepts, 5th Edition, McGrawHill, 2006.
4. C.J. Date, A. Kannan, S. Swamynatham: A Introduction to Database Systems, 8th Edition, Pearson education, 2006.

VIVA QUESTIONS

1. Define Data.
2. Define Information.
3. Define Database.
4. Define DBMS.
5. What do you mean by processed data?
6. What do you mean by data management?
7. Which are the actions that are performed on the database?
8. Mention the different types of DBMS.
9. Define Data model.
10. Mention the different types of Data models.
11. Why database approach is advantageous than the file system approach?
12. Who is called as the father of RDBMS?
13. What do you mean by redundant data?
14. What do you mean by Data duplication?
15. Mention the different relational algebra operations.
16. Mention the different User interfaces provided by the database system.
17. Mention the different languages provided by the database system
18. What is the difference between select operation in relational algebra and in SQL?
19. What is the difference between JOIN and Cartesian product?
20. Mention the different types of Join operations.
21. What is the difference between EQUIJOIN and NATURAL JOIN?
22. What is the difference between OUTER JOIN and JOIN.?
23. What is the difference between OUTER UNION and UNION?
24. What do you mean by Union Compatibility.?
25. What do you mean by Type Compatibility?
26. Mention the different types of relational constraints. 27. Mention the different types of structural constraints
28. What do you mean by cardinality?
29. What do you mean by cardinality ratio?
30. What do you mean by degree of a relation?
31. What do you mean by entity integrity constraint?
32. What do you mean by referential integrity constraint?
33. What do you mean by NULL constraint?
34. What do you mean by unique constraint?
35. What do you mean by Check constraint?
36. Define functional dependency.
37. Define normalization.
38. Define normal form
39. Mention the different types of normal forms
40. What is the difference between 3NF and BCNF?
41. What do you mean by JOIN dependencies?

42. What do you mean by Inclusion dependencies?
43. What do you mean by Template dependencies?
44. What do you mean by Multivalued dependencies?
45. Define Project Join Normal form.

PREETIMH

46. Define Domain Key Normal form.
47. Mention the informal guidelines for database design.
48. Define super key.
49. Define primary key.
50. Define foreign key.
51. Define unique key.
52. Define prime attribute.
53. Define trivial functional dependency.
54. When a FD is said to be fully FD?
55. Mention the different Armstrong's inference rules.
56. Why Armstrong's inference rules are said to be sound and complete?
57. Define denormalisation.
58. Define Transaction.
59. Mention the ACID properties.
60. Define schedule.
61. Is DBMS usage always advisable or some times we may depend on file base systems? Comment on the statement by describing the situation where DBMS is not a better option & file base systems is better.
62. Describe 3-level architecture of DBMS with details of languages associated at different levels plus the level of data independence.
63. How logical architecture of DBMS differs from physical architecture?
64. Create an E R diagram and relational schema to hold information about the situation in many institutions affiliated to some University, many teachers of different disciplines are teaching to many students enrolled in many courses offered by the university to the students through the institutions. Use concept of keys, aggregation, generalisation, cardinality etc. in a proper way.
65. What is the utility of relational algebra & relational calculus? Name some software's based on these concepts?
66. Comment on the statement "Set theory has contributed a lot to RDBMS" support it with the help of suitable examples.
67. "Redundancy of data is many times beneficial" Justify the statement, also describe the situation when redundancy will mess up the current data base status, at that instance of time what actions you will prefer to take.
68. In Oracle we are having variety of versions Oracle 8, Oracle 9, etc, what does the associated number mean. Again we are having Oracle 8i, Oracle 9i etc, what does this "i" mean.
69. Describe the various file organization techniques? How a binary tree is different from Btree and B+ tree? Under which situation we need to use B+ tree or B tree. Prove "Any relation which is in BCNF is in 3NF, but converse is not true"

70. Which functional dependencies are to be removed to achieve respective normal form? Discuss all the normal forms up to 4NF?
71. What is the mathematical basis of SQL? The SQL statement: select * from student will perform like projection or selection? Give details in support of your answer.

Dept. of CSE-AI, SVCE

PREETIMH