CANARA ENGINEERING COLLEGE

Benjanapadavu, D.K, 574219

LAB MANUAL

DATA BASE MANAGEMENT SYSTEM LABORATORY WITH MINI PROJECT

17CSL58

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

DBMS LABORATORY WITH MINI PROJECT [As per Choice Based Credit System (CBCS) scheme] (Effective from the academic year 2017-2018) SEMESTER – V			
Subject Code	17CSL58	IA Marks	40
Number of Lecture Hours/Week	01I + 02P	Exam Marks	60
Total Number of Lecture Hours	40	Exam Hours	03

PART-A: SQL Programming (Max. Exam Mks. 50)

1. 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.
2. Create Schema and insert at least 5 records for each table. Add appropriate database constraints.

PART-B: Mini Project (Max. Exam Mks. 30)

1. Use Java, C#, PHP, Python, or any other similar front-end tool. All applications must be demonstrated on desktop/laptop as a stand-alone or web based application (Mobile apps on Android/IOS are not permitted.)

EXP	DETAILS	HRS
I	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. 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.	**3**

	5. Create a view of all books and its number of copies that are currently available in the Library.	
II	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 salesman who had more than one customer. 3. List all the salesman 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.	**3**
III	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.	**3**
IV	Consider the schema for College Database: STUDENT(USN, SName, Address, Phone, Gender)	**3**

	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.	
V	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).	**3**

	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.	
	MINI PROJECT	**25**
TOTAL HOURS		**40**

COURSE OUTCOMES (COs):

SL. NO	**DESCRIPTION**	**PO MAPPING**	**PSO MAPPING**
	After completion of the course, the students will be able to:		
CO:1	Apply the fundamentals of databases to construct ER diagram and relational schema.	1,2	1
CO:2	Choose the appropriate SQL statements to create, retrieve and maintain the relational database.	1, 2	1
CO:3	Make use of MariaDB tool to execute SQL statements.	4,5	-
CO:4	Develop the simple database applications by identifying real world problems using any modern tools.	1,2,3,5,6,11	1
CO:5	Summarize and document the design activity, procedure of implementation and Query results.	6,8, 10,11	-

CHAPTER – 1

BASIC CONCEPTS OF SQL

1.1 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.

1.2 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.

1.2.1 Data Definition Language (DDL)

1.2.1.1 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_name2 datatype, ...
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_namedatatype [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 a 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)
```

1.2.1.2 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_nameADD CONSTRAINT constraint_name PRIMARY KEY
column_name;
```

Syntax to drop PRIMARY KEY constraint

```
ALTER TABLE table_nameDROP PRIMARY KEY;
```

1.2.1.3 The DROP TABLE Statement

The DROP TABLE statement is used to delete a table.

DROP TABLE table_name;

1.2.1.4 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;

1.2.2 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)	Returns the number of rows in the query. When you specify expr, this function considers rows where expr is not null. When you specify the asterisk (*), this function Returns all rows, including duplicates and nulls. You can count either all rows, or only distinct

	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_nameVALUES (value1, value2, value3,...);
```

OR

```
INSERT INTO table_nameVALUES(&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:

```
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;
```

1.2.3 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 transaaction into database.

Following is Commit command's syntax,

commit;

Rollback command

This command restores the database to last commited state. It is also use 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;

1.2.4 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
```

1.3 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	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

1.4 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
```

1.5 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.

1. 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)

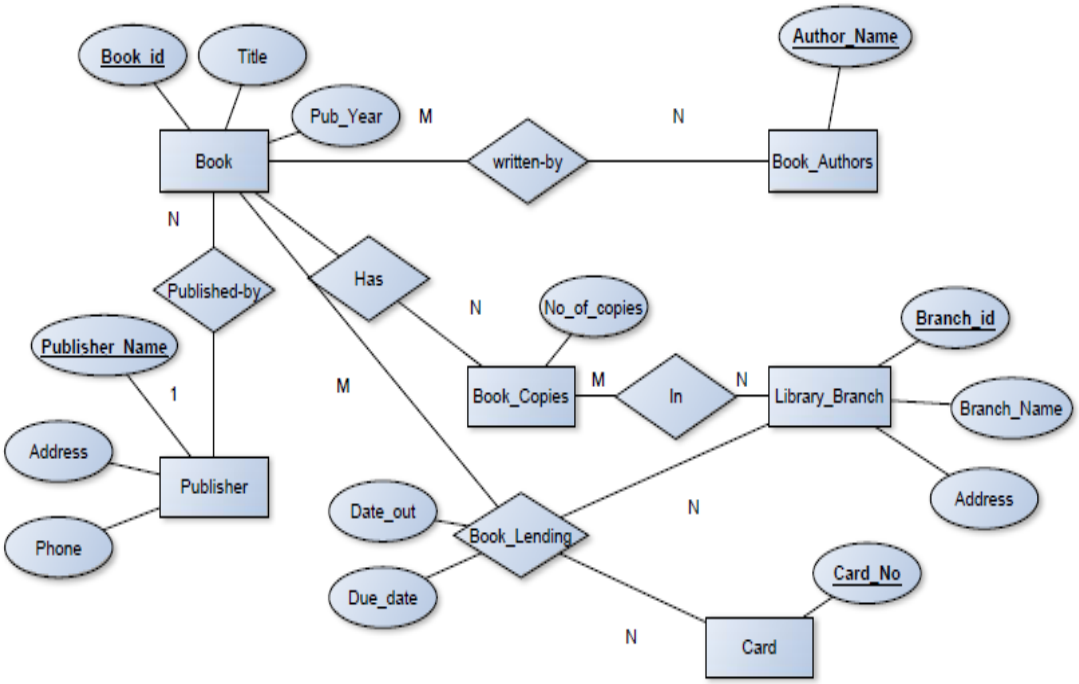
BOOK_LENDING(Book_id, Branch_id, Card_No, Date_Out, Due_Date)

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. **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 Diagram

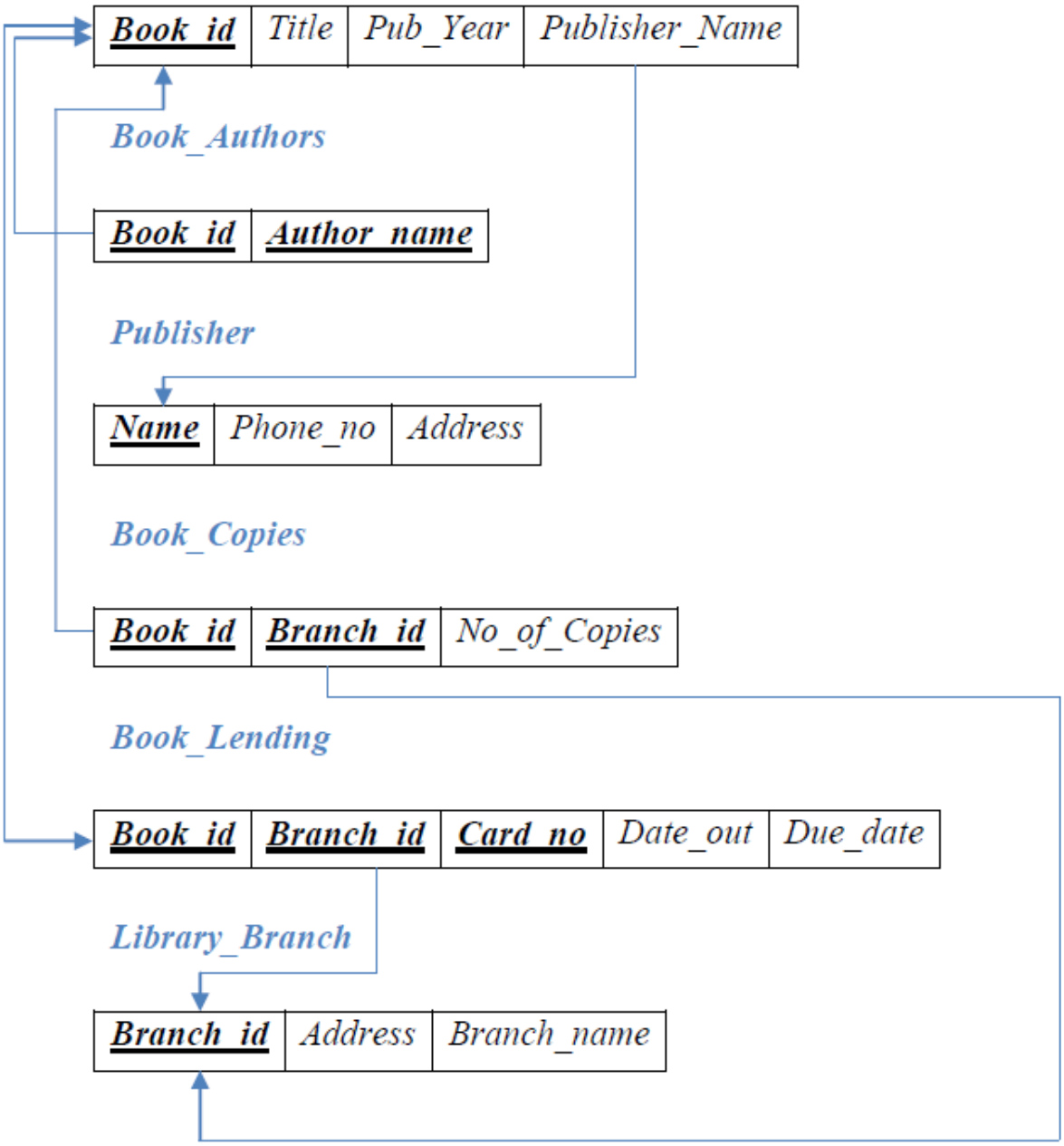


TABLE CREATION:

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

```
);

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

CREATE TABLE BOOK_AUTHORS
(
  AUTHOR_NAME VARCHAR (20),
  BOOK_ID INTEGER,
  FOREIGN KEY(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 VARCHAR(50),
   ADDRESS VARCHAR(50)
);
CREATE TABLE BOOK_COPIES
(
  NO_OF_COPIES INTEGER,
  BOOK_ID INTEGER,
  BRANCH_ID INTEGER,
  FOREIGN KEY(BOOK_ID) REFERENCES BOOK(BOOK_ID) ON DELETE CASCADE,
  FOREIGN KEY(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 INTEGER,
  BRANCH_ID INTEGER,
  CARD_NO INTEGER,
  FOREIGN KEY(BOOK_ID) REFERENCES BOOK (BOOK_ID) ON DELETE CASCADE,
  FOREIGN KEY(BRANCH_ID) REFERENCES LIBRARY_BRANCH(BRANCH_ID) ON
DELETE CASCADE,
  FOREIGN KEY(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 ('2017-01-17','2017-01-01',1,10,101);
INSERT INTO BOOK_LENDING VALUES ('2017-03-15','2017-02-11',3,14,101);
INSERT INTO BOOK_LENDING VALUES ('2017-02-21','2017-03-02',2,13,101);
INSERT INTO BOOK_LENDING VALUES ('2017-04-11','2017-04-22',4,11,101);
INSERT INTO BOOK_LENDING VALUES ('2017-04-12','2017-05-12',1,11,104);
```

QUERIES:

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_NAME
FROM BOOK B, BOOK_AUTHORS A, BOOK_COPIES C, LIBRARY_BRANCH L
WHERE B.BOOK_ID=A.BOOK_ID AND B.BOOK_ID=C.BOOK_ID AND
L.BRANCH_ID=C.BRANCH_ID;
```

BOOK_ID	TITLE	PUBLISHER_NAME	AUTHOR_NAME	NO_OF_COPIES	BRANCH_ID
1	DBMS	MCGRAW-HILL	NAVATHE	10	10
1	DBMS	MCGRAW-HILL	NAVATHE	5	11
2	ADBMS	MCGRAW-HILL	NAVATHE	2	12
2	ADBMS	MCGRAW-HILL	NAVATHE	5	13
3	CN	PEARSON	TANENBAUM	7	14
5	OS	PEARSON	GALVIN	1	10
4	CG	GRUPO PLANETA	EDWARD ANGEL	3	11

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 '2017-01-01'AND'2017-06-30'
GROUP BY CARD_NO
HAVING COUNT(*)>3;
```

CARD_NO
101

3. Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation.

```
DELETE FROM BOOK
WHERE BOOK_ID=3;
```

```
SQL> DELETE FROM BOOK
  2  WHERE BOOK_ID=3;

1 row deleted.

SQL> SELECT * FROM BOOK;

   BOOK_ID TITLE                PUB_YEAR             PUBLISHER_NAME
---------- -------------------- -------------------- --------------------
         1 DBMS                 JAN-2017             MCGRAW-HILL
         2 ADBMS                JUN-2016             MCGRAW-HILL
         4 CG                   SEP-2015             GRUPO PLANETA
         5 OS                   MAY-2016             PEARSON
```

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

```
CREATE TABLE BOOKP
(
  Book_id INT NOT NULL,
  Title VARCHAR(20),
  Publisher_name VARCHAR(20),
  Pub_year INT
)
PARTITION BY RANGE (Pub_year)
( PARTITION q0 VALUES LESS THAN (2005),
PARTITION q1 VALUES LESS THAN (2010),
PARTITION q2 VALUES LESS THAN (2018)
);

INSERT INTO BOOKP VALUES ('802' , 'DATABASE MANAGEMENT
SYSTEM','PEARSON', '2016');
INSERT INTO BOOKP VALUES ('803' , 'COMPUTER NETWORKS','MCGRAW', '2017');
INSERT INTO BOOKP VALUES ('804' , 'COMPUTER GRAPHICS','TATA', '2005');
INSERT INTO BOOKP VALUES ('806' , 'OBJECT ORIENTED
PROGRAMMING','SIDNEY', '2011');
INSERT INTO BOOKP VALUES ('807' , 'Data STRUCTURES','TATA', '2012');
INSERT INTO BOOKP VALUES ('808' , 'SOFTWARE ENGINEERING','PEARSON',
'2000');
```

ALTER TABLE BOOKP DROP PARTITION q0;

5. 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 C.BRANCH_ID=L.BRANCH_ID;
```

BOOK_ID	TITLE	NO_OF_COPIES
1	DBMS	10
1	DBMS	5
2	ADBMS	2
2	ADBMS	5
3	CN	7
5	OS	1
4	CG	3

II. 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.

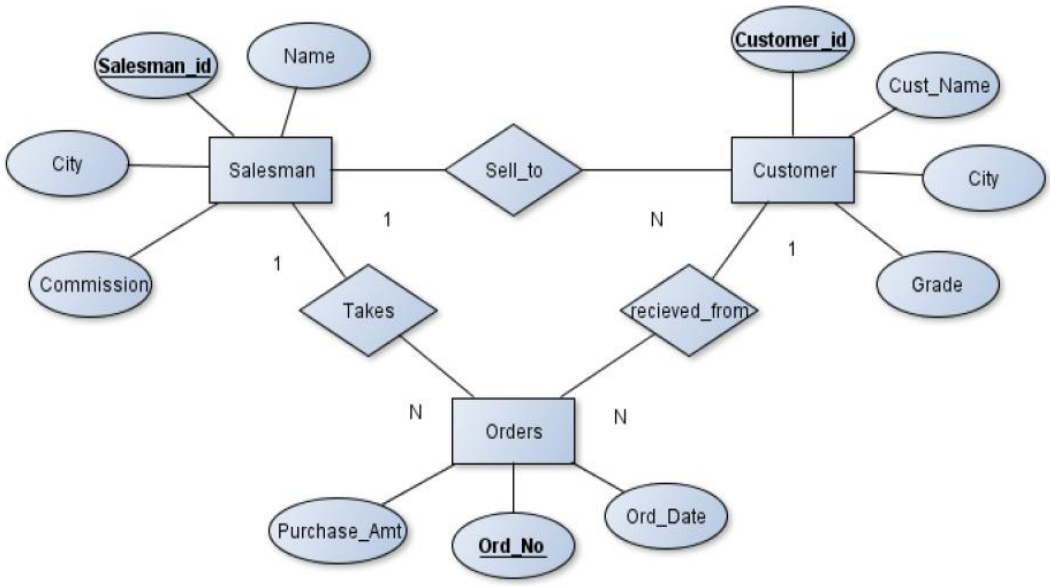
2. Find the name and numbers of all salesman who had more than one customer.

3. List all the salesman 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.

Entity-Relationship Diagram



Schema Diagram

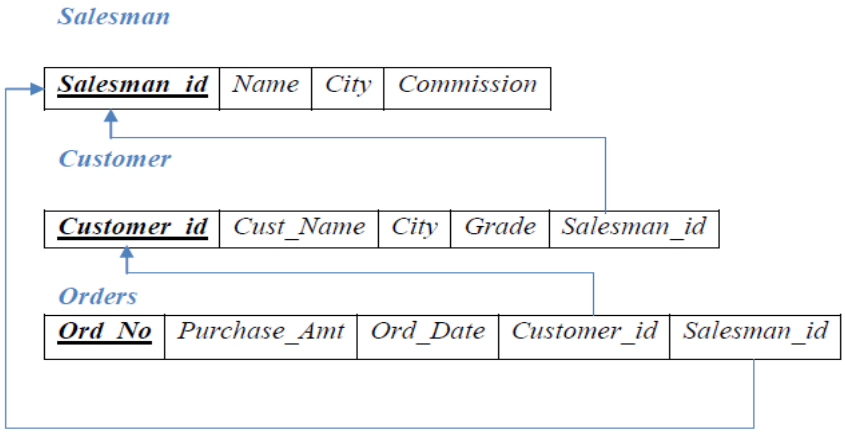


TABLE CREATION

```
CREATE TABLE SALESMAN
(
  SALESMAN_ID INTEGER(4),
  NAME VARCHAR(20),
  CITY VARCHAR(20),
  COMMISSION VARCHAR(20),
  PRIMARY KEY (SALESMAN_ID)
);

CREATE TABLE CUSTOMER
(
  CUSTOMER_ID INTEGER(4),
  CUST_NAME VARCHAR(20),
  CITY VARCHAR(20),
  GRADE INTEGER(3),
  SALESMAN_ID INTEGER(4),
  PRIMARY KEY(CUSTOMER_ID),
  FOREIGN KEY(SALESMAN_ID) REFERENCES SALESMAN (SALESMAN_ID) ON
DELETE SET NULL
);

CREATE TABLE ORDERS
(
```

```
   ORD_NO INTEGER(5),
   PURCHASE_AMT DECIMAL(10,2),
   ORD_DATE DATE,
   CUSTOMER_ID INTEGER(4),
   SALESMAN_ID INTEGER(4),
   PRIMARY KEY (ORD_NO),
   FOREIGN KEY(CUSTOMER_ID) REFERENCES CUSTOMER(CUSTOMER_ID) ON
DELETE CASCADE,
   FOREIGN KEY(SALESMAN_ID) REFERENCES SALESMAN(SALESMAN_ID) ON
DELETE CASCADE
  );
```

INSERTION

```
INSERT INTO SALESMAN VALUES (1000, 'JOHN','BANGALORE','25 %');
INSERT INTO SALESMAN VALUES (2000, 'RAVI','BANGALORE','20 %');
INSERT INTO SALESMAN VALUES (3000, 'KUMAR','MYSORE','15 %');
INSERT INTO SALESMAN VALUES (4000, 'SMITH','DELHI','30 %');
INSERT INTO SALESMAN VALUES (5000, 'HARSHA','HYDRABAD','15 %');

INSERT INTO CUSTOMER VALUES (10,'PREETHI','BANGALORE',100,1000);
INSERT INTO CUSTOMER VALUES (11,'VIVEK','MANGALORE',300,1000);
INSERT INTO CUSTOMER VALUES (12,'BHASKAR','CHENNAI',400,2000);
INSERT INTO CUSTOMER VALUES (13,'CHETHAN','BANGALORE',200,2000);
INSERT INTO CUSTOMER VALUES (14,'MAMATHA','BANGALORE',400,3000);

INSERT INTO ORDERS VALUES (50,5000,'2017-05-04',10,1000);
INSERT INTO ORDERS VALUES (51,450,'2017-01-20',10,2000);
INSERT INTO ORDERS VALUES (52,1000,'2017-01-24',13,2000);
INSERT INTO ORDERS VALUES (53,3500,'2017-04-13',14,3000);
INSERT INTO ORDERS VALUES (54,550,'2017-03-09',12,2000);
```

QUERIES:

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

```
SELECT GRADE, COUNT(DISTINCT CUSTOMER_ID)
```

```
FROM CUSTOMER
GROUP BY GRADE
HAVING GRADE >( SELECT AVG(GRADE)
          FROM CUSTOMER
          WHERE CITY='BANGALORE'
         );
```

```
GRADE COUNT(DISTINCTCUSTOMER_ID)
------ --------------------------
  300                           1
  400                           2
```

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

```
SELECT A.SALESMAN_ID, NAME
FROM SALESMAN A
WHERE 1<(
        SELECT COUNT(*)
        FROM CUSTOMER
        WHERE SALESMAN_ID=A.SALESMAN_ID
     );
```

```
SALESMAN_ID NAME
----------- -----
       1000 JOHN
       2000 RAVI
```

3. List all the salesman and indicate those who have and don't have customers in their cities (Use UNION operation.)

```
SELECT S.SALESMAN_ID, NAME,S.CITY, CUST_NAME
FROM SALESMAN S, CUSTOMER C
WHERE S.CITY = C.CITY
UNION
SELECT S.SALESMAN_ID, S.NAME,S.CITY ,'NO MATCH'
FROM SALESMAN S
WHERE  NOT S.CITY = ANY
              (
                SELECT CITY
```

```
                FROM CUSTOMER CC
                )
 ORDER BY 1;
```

SALESMAN_ID	NAME	CUST_NAME	COMMISSION
4000	SMITH	NO MATCH	30 %
2000	RAVI	CHETHAN	20 %
2000	RAVI	MAMATHA	20 %
2000	RAVI	PREETHI	20 %
3000	KUMAR	NO MATCH	15 %
1000	JOHN	CHETHAN	25 %
1000	JOHN	MAMATHA	25 %
1000	JOHN	PREETHI	25 %
5000	HARSHA	NO MATCH	15 %

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

```
CREATE VIEW HIGHEST_ORDER_SALESMAN AS
SELECT B.ORD_DATE, A.SALESMAN_ID, A.NAME
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
              );
```

ORD_DATE	SALESMAN_ID	NAME
04-MAY-17	1000	JOHN
20-JAN-17	2000	RAVI
24-FEB-17	2000	RAVI
13-APR-17	3000	KUMAR
09-MAR-17	2000	RAVI

5. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.

```
DELETE FROM SALESMAN
WHERE SALESMAN_ID=1000;
```

```
SQL> DELETE FROM SALESMAN
  2  WHERE SALESMAN_ID=1000;

1 row deleted.

SQL> SELECT * FROM SALESMAN;

SALESMAN_ID NAME                 CITY                 COMMISSION
----------- -------------------- -------------------- ----------
       2000 RAVI                 BANGALORE            20 %
       3000 KUMAR                MYSORE               15 %
       4000 SMITH                DELHI                30 %
       5000 HARSHA               HYDRABAD             15 %
```

III 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.

TABLE CREATION:

```
CREATE TABLE ACTOR
(
  ACT_ID INTEGER,
  ACT_NAME VARCHAR(20),
  ACT_GENDER CHAR(1),
  PRIMARY KEY(ACT_ID)
);

CREATE TABLE DIRECTOR
(
  DIR_ID INTEGER,
  DIR_NAME VARCHAR(20),
  DIR_PHONE INTEGER,
  PRIMARY KEY(DIR_ID)
);
```

```
CREATE TABLE MOVIES
(
  MOV_ID INTEGER,
  MOV_TITLE VARCHAR(25),
  MOV_YEAR INTEGER,
  MOV_LANG VARCHAR(12),
  DIR_ID INTEGER,
  PRIMARY KEY(MOV_ID),
  FOREIGN KEY(DIR_ID) REFERENCES DIRECTOR (DIR_ID)
);

CREATE TABLE MOVIE_CAST
(
  ACT_ID INTEGER,
  MOV_ID INTEGER,
  ROLE VARCHAR(10),
  PRIMARY KEY(ACT_ID, MOV_ID),
  FOREIGN KEY(ACT_ID) REFERENCES ACTOR(ACT_ID),
  FOREIGN KEY(MOV_ID) REFERENCES MOVIES(MOV_ID)
);

CREATE TABLE RATING
 (
   MOV_ID INTEGER,
   REV_STARS VARCHAR(25),
   PRIMARY KEY(MOV_ID,REV_STARS),
   FOREIGN KEY(MOV_ID) REFERENCES MOVIES (MOV_ID)
 );
```

INSERTION:

```
INSERT INTO ACTOR VALUES (301,'ANUSHKA','F');
INSERT INTO ACTOR VALUES (302,'PRABHAS','M');
```

```
INSERT INTO ACTOR VALUES (303,'PUNITH','M');
INSERT INTO ACTOR VALUES (304,'JERMY','M');

INSERT INTO DIRECTOR VALUES (60,'RAJAMOULI', 8751611001);
INSERT INTO DIRECTOR VALUES (61,'HITCHCOCK', 7766138911);
INSERT INTO DIRECTOR VALUES (62,'FARAN', 9986776531);
INSERT INTO DIRECTOR VALUES (63,'STEVEN SPIELBERG', 8989776530);

INSERT INTO MOVIES VALUES (1001,'BAHUBALI-2',2017,'TELUGU',60);
INSERT INTO MOVIES VALUES (1002,'BAHUBALI-1',2015,'TELUGU',60);
INSERT INTO MOVIES VALUES (1003,'AKASH',2008,'KANNADA',61);
INSERT INTO MOVIES VALUES (1004,'WAR HORSE',2011,'ENGLISH',63);

INSERT INTO MOVIE_CAST VALUES (301, 1002, 'HEROINE');
INSERT INTO MOVIE_CAST VALUES (301, 1001, 'HEROINE');
INSERT INTO MOVIE_CAST VALUES (303, 1003, 'HERO');
INSERT INTO MOVIE_CAST VALUES (303, 1002, 'GUEST');
INSERT INTO MOVIE_CAST VALUES (304, 1004, 'HERO');

INSERT INTO RATING VALUES (1001, 4);
INSERT INTO RATING VALUES (1002, 2);
INSERT INTO RATING VALUES (1003, 5);
INSERT INTO RATING VALUES (1004, 4);
INSERT INTO RATING VALUES (1001, 3);
INSERT INTO RATING VALUES (1001, 2);
```

QUERIES

1. List the titles of all movies directed by ‘Hitchcock’.

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

```
MOV_TITLE
-------------
AKASH
```

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

```
SELECT DISTINCT(MOV_TITLE)
FROM MOVIES M, MOVIE_CAST MV
WHERE M.MOV_ID=MV.MOV_ID AND ACT_ID IN ( SELECT ACT_ID
                        FROM MOVIE_CAST GROUP BY ACT_ID
                        HAVING COUNT(ACT_ID)>1)
```

```
MOV_TITLE
-------------
BAHUBALI-1
```

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

```
SELECT ACT_NAME, MOV_TITLE, MOV_YEAR
FROM ACTOR A JOIN MOVIE_CAST C JOIN MOVIES M ON A.ACT_ID=C.ACT_ID
AND  C.MOV_ID=M.MOV_ID
WHERE M.MOV_YEAR NOT BETWEEN 2000 AND 2015;
```

```
ACT_NAME             MOV_TITLE                 MOV_YEAR
-------------------- ------------------------- ----------
ANUSHKA              BAHUBALI-2                      2017
```

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.

```
SELECT MOV_TITLE, MAX(REV_STARS)
FROM MOVIES INNER JOIN RATING USING(MOV_ID)
GROUP BY MOV_TITLE
HAVING MAX(REV_STARS)>0
ORDER BY MOV_TITLE;
```

```
MOV_TITLE                 MAX(REV_STARS)
------------------------- --------------
AKASH                     5
BAHUBALI-1                2
BAHUBALI-2                4
WAR HORSE                 4
```

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

```
UPDATE RATING
SET REV_STARS=5
WHERE MOV_ID IN (SELECT MOV_ID FROM MOVIES
WHERE DIR_ID IN (SELECT DIR_ID
FROM DIRECTOR
WHERE DIR_NAME = 'STEVEN SPIELBERG'));
```

```
SQL> SELECT * FROM RATING;

    MOV_ID REV_STARS
---------- -----------
      1001 4
      1002 2
      1003 5
      1004 5
```

IV 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:

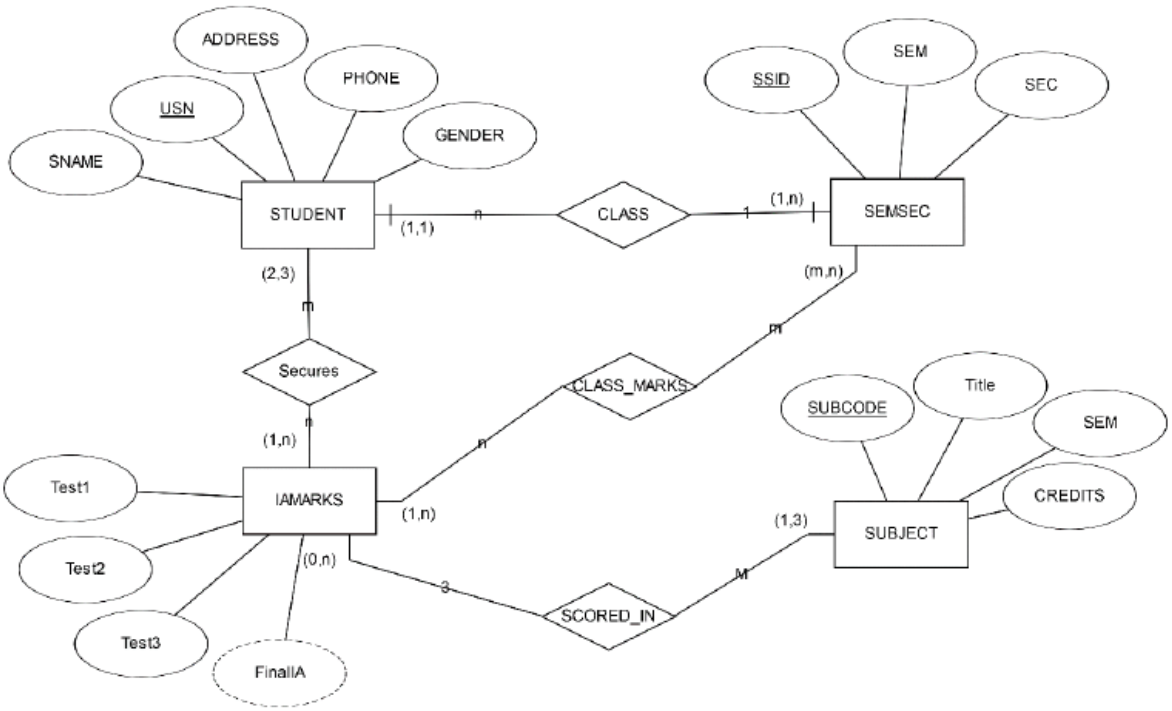
If FinalIA = 17 to 20 then CAT = 'Outstanding'

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.

Entity - Relationship Diagram



Schema Diagram

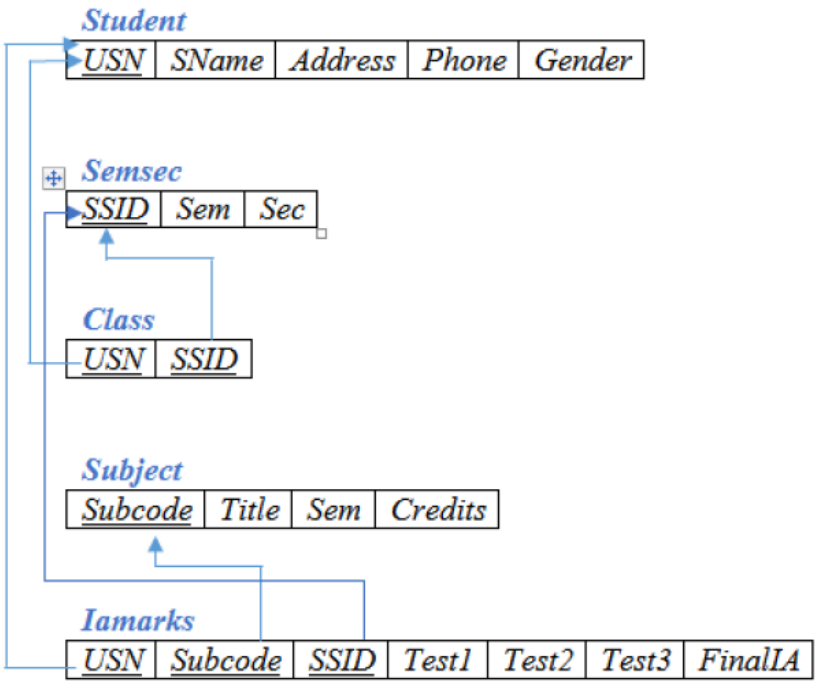


TABLE CREATION

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

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

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

CREATE TABLE SUBJECT
(
 SUBCODE VARCHAR(8),
 TITLE VARCHAR(20),
 SEM INTEGER,
 CREDITS INTEGER,
 PRIMARY KEY (SUBCODE)
);

CREATE TABLE IAMARKS
(
USN VARCHAR(10),
SUBCODE VARCHAR(8),
SSID VARCHAR(5),
TEST1 INTEGER,
TEST2 INTEGER,
TEST3 INTEGER,
FINALIA INTEGER,
PRIMARY KEY (USN, SUBCODE, SSID),
FOREIGN KEY (USN) REFERENCES STUDENT (USN),
FOREIGN KEY (SUBCODE) REFERENCES SUBJECT (SUBCODE),
FOREIGN KEY (SSID) REFERENCES SEMSEC (SSID)
);
```

INSERTION

```
INSERT INTO STUDENT VALUES ('1RN13CS020','AKSHAY','BELAGAVI',
8877881122,'M');
INSERT INTO STUDENT VALUES ('1RN13CS062','SANDHYA','BENGALURU',
7722829912,'F');
INSERT INTO STUDENT VALUES ('1RN13CS091','TEESHA','BENGALURU',
7712312312,'F');
INSERT INTO STUDENT VALUES ('1RN13CS066','SUPRIYA','MANGALURU',
8877881122,'F');
INSERT INTO STUDENT VALUES ('1RN14CS010','ABHAY','BENGALURU',
9900211201,'M');
INSERT INTO STUDENT VALUES ('1RN14CS032','BHASKAR','BENGALURU',
9923211099,'M');
INSERT INTO STUDENT VALUES ('1RN14CS025','ASMI','BENGALURU',
7894737377,'F');
INSERT INTO STUDENT VALUES ('1RN15CS011','AJAY','TUMKUR', 9845091341,'M');

INSERT INTO STUDENT VALUES ('1RN15CS029','CHITRA','DAVANGERE',
7696772121,'F');
INSERT INTO STUDENT VALUES ('1RN15CS045','JEEVA','BELLARY', 9944850121,'M');
INSERT INTO STUDENT VALUES
('1RN15CS091','SANTOSH','MANGALURU',8812332201,'M');
INSERT INTO STUDENT VALUES
('1RN16CS045','ISMAIL','KALBURGI',9900232201,'M');
INSERT INTO STUDENT VALUES
('1RN16CS088','SAMEERA','SHIMOGA',9905542212,'F');
INSERT INTO STUDENT VALUES
('1RN16CS122','VINAYAKA','CHIKAMAGALUR',8800880011,'M');

INSERT INTO SEMSEC VALUES ('CSE8A', 8,'A');
INSERT INTO SEMSEC VALUES ('CSE8B', 8,'B');
INSERT INTO SEMSEC VALUES ('CSE8C', 8,'C');
INSERT INTO SEMSEC VALUES ('CSE7A', 7,'A');
INSERT INTO SEMSEC VALUES ('CSE7B', 7,'B');
INSERT INTO SEMSEC VALUES ('CSE7C', 7,'C');
```

```
INSERT INTO SEMSEC VALUES ('CSE6A', 6,'A');
INSERT INTO SEMSEC VALUES ('CSE6B', 6,'B');
INSERT INTO SEMSEC VALUES ('CSE6C', 6,'C');
INSERT INTO SEMSEC VALUES ('CSE5A', 5,'A');
INSERT INTO SEMSEC VALUES ('CSE5B', 5,'B');
INSERT INTO SEMSEC VALUES ('CSE5C', 5,'C');
INSERT INTO SEMSEC VALUES ('CSE4A', 4,'A');
INSERT INTO SEMSEC VALUES ('CSE4B', 4,'B');
INSERT INTO SEMSEC VALUES ('CSE4C', 4,'C');
INSERT INTO SEMSEC VALUES ('CSE3A', 3,'A');
INSERT INTO SEMSEC VALUES ('CSE3B', 3,'B');
INSERT INTO SEMSEC VALUES ('CSE3C', 3,'C');
INSERT INTO SEMSEC VALUES ('CSE2A', 2,'A');
INSERT INTO SEMSEC VALUES ('CSE2B', 2,'B');
INSERT INTO SEMSEC VALUES ('CSE2C', 2,'C');
INSERT INTO SEMSEC VALUES ('CSE1A', 1,'A');
INSERT INTO SEMSEC VALUES ('CSE1B', 1,'B');
INSERT INTO SEMSEC VALUES ('CSE1C', 1,'C');

INSERT INTO CLASS VALUES ('1RN13CS020','CSE8A');
INSERT INTO CLASS VALUES ('1RN13CS062','CSE8A');
INSERT INTO CLASS VALUES ('1RN13CS066','CSE8B');
INSERT INTO CLASS VALUES ('1RN13CS091','CSE8C');
INSERT INTO CLASS VALUES ('1RN14CS010','CSE7A');
INSERT INTO CLASS VALUES ('1RN14CS025','CSE7A');
INSERT INTO CLASS VALUES ('1RN14CS032','CSE7A');
INSERT INTO CLASS VALUES ('1RN15CS011','CSE4A');
INSERT INTO CLASS VALUES ('1RN15CS029','CSE4A');
INSERT INTO CLASS VALUES ('1RN15CS045','CSE4B');
INSERT INTO CLASS VALUES ('1RN15CS091','CSE4C');
INSERT INTO CLASS VALUES ('1RN16CS045','CSE3A');
INSERT INTO CLASS VALUES ('1RN16CS088','CSE3B');
INSERT INTO CLASS VALUES ('1RN16CS122','CSE3C');
```

```
INSERT INTO SUBJECT VALUES ('10CS81','ACA', 8, 4);
INSERT INTO SUBJECT VALUES ('10CS82','SSM', 8, 4);
INSERT INTO SUBJECT VALUES ('10CS83','NM', 8, 4);
INSERT INTO SUBJECT VALUES ('10CS84','CC', 8, 4);
INSERT INTO SUBJECT VALUES ('10CS85','PW', 8, 4);
INSERT INTO SUBJECT VALUES ('10CS71','OOAD', 7, 4);
INSERT INTO SUBJECT VALUES ('10CS72','ECS', 7, 4);
INSERT INTO SUBJECT VALUES ('10CS73','PTW', 7, 4);
INSERT INTO SUBJECT VALUES ('10CS74','DWDM', 7, 4);
INSERT INTO SUBJECT VALUES ('10CS75','JAVA', 7, 4);
INSERT INTO SUBJECT VALUES ('10CS76','SAN', 7, 4);
INSERT INTO SUBJECT VALUES ('15CS51', 'ME', 5, 4);
INSERT INTO SUBJECT VALUES ('15CS52','CN', 5, 4);
INSERT INTO SUBJECT VALUES ('15CS53','DBMS', 5, 4);
INSERT INTO SUBJECT VALUES ('15CS54','ATC', 5, 4);
INSERT INTO SUBJECT VALUES ('15CS55','JAVA', 5, 3);
INSERT INTO SUBJECT VALUES ('15CS56','AI', 5, 3);
INSERT INTO SUBJECT VALUES ('15CS41','M4', 4, 4);
INSERT INTO SUBJECT VALUES ('15CS42','SE', 4, 4);
INSERT INTO SUBJECT VALUES ('15CS43','DAA', 4, 4);
INSERT INTO SUBJECT VALUES ('15CS44','MPMC', 4, 4);
INSERT INTO SUBJECT VALUES ('15CS45','OOC', 4, 3);
INSERT INTO SUBJECT VALUES ('15CS46','DC', 4, 3);
INSERT INTO SUBJECT VALUES ('15CS31','M3', 3, 4);
INSERT INTO SUBJECT VALUES ('15CS32','ADE', 3, 4);
INSERT INTO SUBJECT VALUES ('15CS33','DSA', 3, 4);
INSERT INTO SUBJECT VALUES ('15CS34','CO', 3, 4);
INSERT INTO SUBJECT VALUES ('15CS35','USP', 3, 3);
INSERT INTO SUBJECT VALUES ('15CS36','DMS', 3, 3);

INSERT INTO IAMARKS (USN, SUBCODE, SSID, TEST1, TEST2, TEST3)
VALUES('1RN13CS091','10CS81','CSE8C', 15, 16, 18);
INSERT INTO IAMARKS (USN, SUBCODE, SSID, TEST1, TEST2, TEST3) VALUES
```

('1RN13CS091','10CS82','CSE8C', 12, 19, 14);

INSERT INTO IAMARKS (USN, SUBCODE, SSID, TEST1, TEST2, TEST3) VALUES ('1RN13CS091','10CS83','CSE8C', 19, 15, 20);

INSERT INTO IAMARKS (USN, SUBCODE, SSID, TEST1, TEST2, TEST3) VALUES ('1RN13CS091','10CS84','CSE8C', 20, 16, 19);

INSERT INTO IAMARKS (USN, SUBCODE, SSID, TEST1, TEST2, TEST3) VALUES ('1RN13CS091','10CS85','CSE8C', 15, 15, 12);

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

```
SELECT S.*, SS.SEM, SS.SEC
FROM STUDENT S, SEMSEC SS, CLASS C
WHERE S.USN = C.USN AND
SS.SSID = C.SSID AND
SS.SEM = 4 AND SS.SEC='C';
```

USN	SNAME	ADDRESS	PHONE	G	SEM	S
1RN15CS091	SANTOSH	MANGALURU	8812332201	M	4	C

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

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

SEM	S	G	COUNT
3	A	M	1
3	B	F	1
3	C	M	1
4	A	F	1
4	A	M	1
4	B	M	1
4	C	M	1
7	A	F	1
7	A	M	2
8	A	F	1
8	A	M	1
8	B	F	1
8	C	F	1

3. Create a view of Test1 marks of student USN '4CB13CS101' in all subjects.

```
CREATE VIEW TEST1_MARKS
AS
SELECT TEST1, SUBCODE
FROM IAMARKS
WHERE USN = '4CB05CS101';
```

TEST1	SUBCODE
15	10CS81
12	10CS82
19	10CS83
20	10CS84
15	10CS85

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

```
UPDATE IAMARKS
SET Finalia = (GREATEST(Test1+Test2,Test2+Test3,Test3+Test1)/2);
```

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

```
SELECT S.USN,S.SNAME,SS.SEM,SS.SEC,IA.TEST1,IA.TEST2,IA.TEST3,IA.FINALIA,
 (CASE
 WHEN IA.FINALIA BETWEEN 17 AND 20 THEN 'OUTSTANDING'
 WHEN IA.FINALIA BETWEEN 12 AND 16 THEN 'AVERAGE'
 ELSE
 'WEAK'
 END) AS GRADE
 FROM STUDENT S, SEMSEC SS, IAMARKS IA
 WHERE S.USN = IA.USN AND
 SS.SSID = IA.SSID AND
 SS.SEM = 8 AND SS.SEC IN('A','B','C');
```

USN	SNAME	ADDRESS	PHONE	G	CAT
1RN13CS091	TEESHA	BENGALURU	7712312312	F	OutStanding
1RN13CS091	TEESHA	BENGALURU	7712312312	F	OutStanding
1RN13CS091	TEESHA	BENGALURU	7712312312	F	OutStanding
1RN13CS091	TEESHA	BENGALURU	7712312312	F	OutStanding
1RN13CS091	TEESHA	BENGALURU	7712312312	F	Average

V 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.

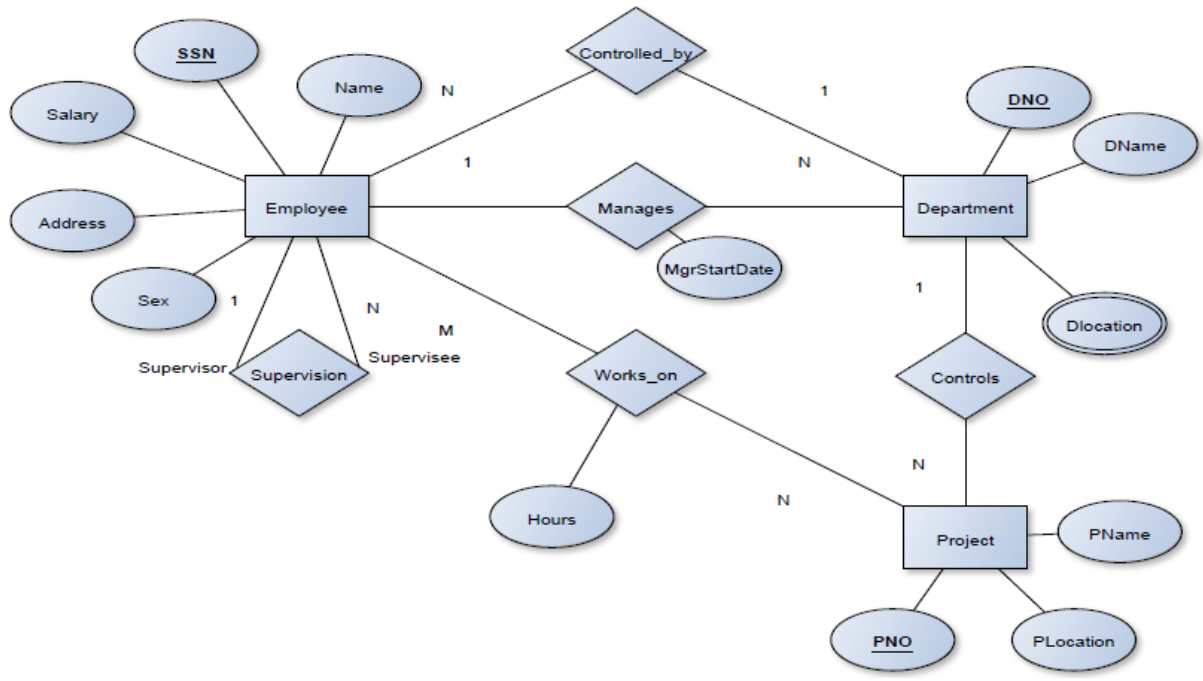
2. Show the resulting salaries if every employee working on the 'IoT' project is given a 10 percent raise.

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.

Entity-Relationship Diagram



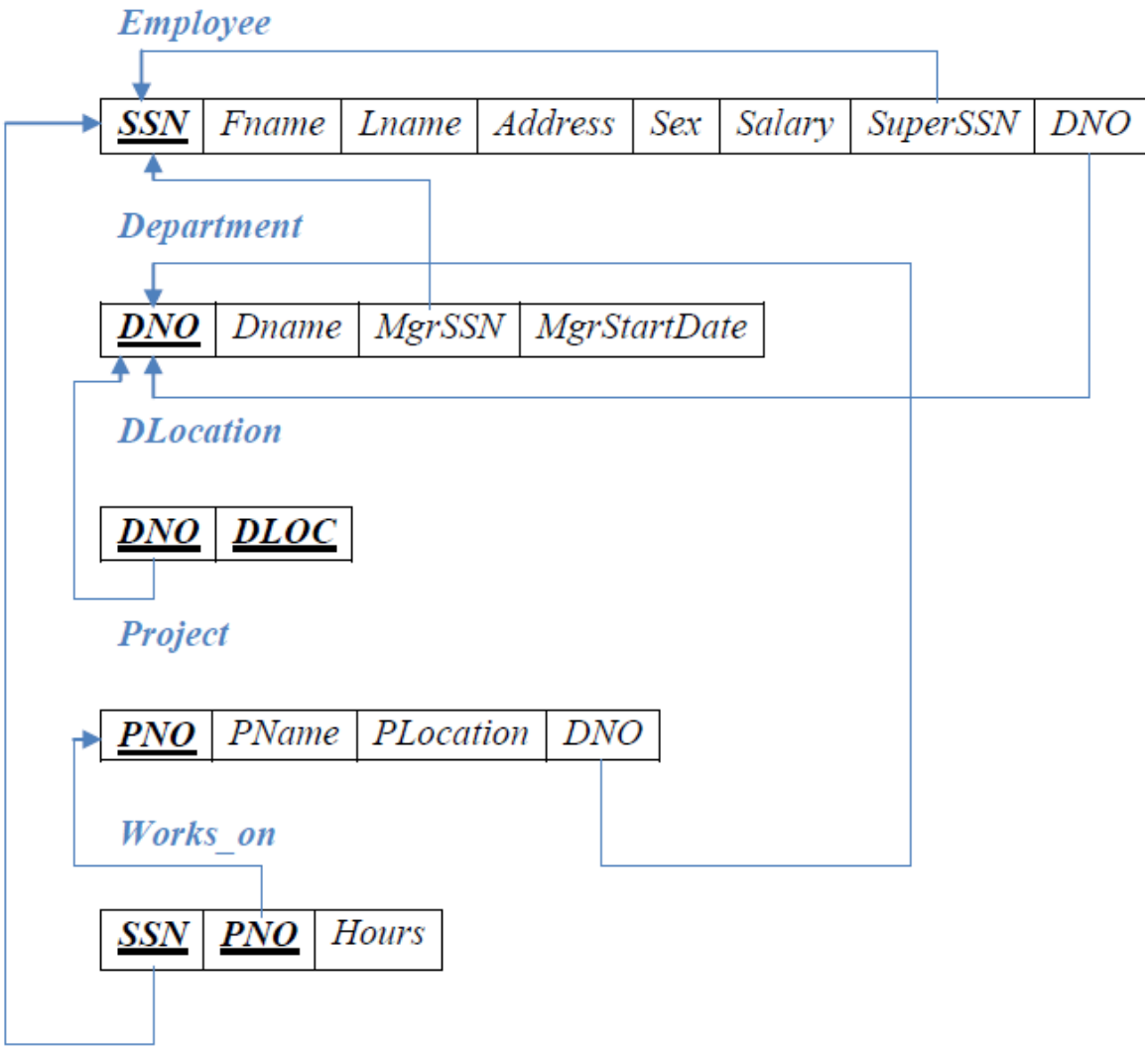


TABLE CREATION

CREATE TABLE DEPARTMENT

(DNO VARCHAR(20) PRIMARY KEY,

DNAME VARCHAR(20),

MGRSTARTDATE DATE);

CREATE TABLE EMPLOYEE

(

SSN VARCHAR(20) PRIMARY KEY,

FNAME VARCHAR(20),

LNAME VARCHAR(20),

```
  ADDRESS VARCHAR(20),
  SEX CHAR(1),
  SALARY INTEGER,
  SUPERSSN VARCHAR(20),
  DNO VARCHAR(20),
  FOREIGN KEY(SUPERSSN) REFERENCES EMPLOYEE(SSN),
  FOREIGN KEY(DNO) REFERENCES DEPARTMENT(DNO)
);
ALTER TABLE DEPARTMENT ADD MGRSSN VARCHAR(20);
ALTER TABLE DEPARTMENT ADD CONSTRAINT FK FOREIGN KEY(MGRSSN)
REFERENCES EMPLOYEE(SSN);

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

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

  CREATE TABLE WORKS_ON
  (
    HOURS INTEGER (2),
    SSN VARCHAR(20),
```

```
   PNO INTEGER ,
   FOREIGN KEY(SSN) REFERENCES EMPLOYEE (SSN),
   FOREIGN KEY(PNO) REFERENCES PROJECT(PNO),
   PRIMARY KEY (SSN, PNO)
 );
```

INSERTION

```
INSERT INTO EMPLOYEE(SSN,FNAME,LNAME,ADDRESS,SEX,SALARY)
VALUES('RNSECE01','JOHN','SCOTT','BANGALORE','M',450000);

INSERT INTO EMPLOYEE (SSN, FNAME, LNAME, ADDRESS, SEX, SALARY)
VALUES('RNSCSE01','JAMES','SMITH','BANGALORE','M', 500000);

INSERT INTO EMPLOYEE (SSN, FNAME, LNAME, ADDRESS, SEX, SALARY)
VALUES('RNSCSE02','HEARN','BAKER','BANGALORE','M', 700000);

INSERT INTO EMPLOYEE (SSN, FNAME, LNAME, ADDRESS, SEX, SALARY)
VALUES('RNSCSE03','EDWARD','SCOTT','MYSORE','M', 500000);

INSERT INTO EMPLOYEE (SSN, FNAME, LNAME, ADDRESS, SEX, SALARY)
VALUES('RNSCSE04','PAVAN','HEGDE','MANGALORE','M', 650000);

INSERT INTO EMPLOYEE (SSN, FNAME, LNAME, ADDRESS, SEX, SALARY)
VALUES('RNSCSE05','GIRISH','MALYA','MYSORE','M', 450000);

INSERT INTO EMPLOYEE (SSN, FNAME, LNAME, ADDRESS, SEX, SALARY)
VALUES('RNSCSE06','NEHA','SN','BANGALORE','F', 800000);

INSERT INTO EMPLOYEE (SSN, FNAME, LNAME, ADDRESS, SEX, SALARY)
VALUES('RNSACC01','AHANA','K','MANGALORE','F', 350000);

INSERT INTO EMPLOYEE (SSN, FNAME, LNAME, ADDRESS, SEX, SALARY)
VALUES('RNSACC02','SANTHOSH','KUMAR','MANGALORE','M', 300000);
```

```
INSERT INTO EMPLOYEE (SSN, FNAME, LNAME, ADDRESS, SEX, SALARY)
VALUES('RNSISE01','VEENA','M','MYSORE','M', 600000);

INSERT INTO EMPLOYEE (SSN, FNAME, LNAME, ADDRESS, SEX, SALARY)
VALUES('RNSIT01','NAGESH','HR','BANGALORE','M', 500000);



INSERT INTO DEPARTMENT VALUES ('1','ACCOUNTS','2001-01-01','RNSACC02');
INSERT INTO DEPARTMENT VALUES ('2','IT','2001-08-01','RNSIT01');
INSERT INTO DEPARTMENT VALUES ('3','ECE','2008-06-01','RNSECE01');
INSERT INTO DEPARTMENT VALUES ('4','ISE','2015-01-02','RNSISE01');
INSERT INTO DEPARTMENT VALUES ('5','CSE','2002-06-01','RNSCSE05');


 UPDATE EMPLOYEE
 SET SUPERSSN=NULL, DNO='3'
 WHERE SSN='RNSECE01';

 UPDATE EMPLOYEE
 SET SUPERSSN='RNSCSE02', DNO='5'
 WHERE SSN='RNSCSE01';

 UPDATE EMPLOYEE
 SET SUPERSSN='RNSCSE03', DNO='5'
 WHERE SSN='RNSCSE02';

 UPDATE EMPLOYEE
 SET SUPERSSN='RNSCSE04', DNO='5'
 WHERE SSN='RNSCSE03';

 UPDATE EMPLOYEE
 SET  DNO='5', SUPERSSN='RNSCSE05'
```

```
WHERE SSN='RNSCSE04';

UPDATE EMPLOYEE
SET DNO='5', SUPERSSN='RNSCSE06'
WHERE SSN='RNSCSE05';

UPDATE EMPLOYEE
SET DNO='5', SUPERSSN=NULL
WHERE SSN='RNSCSE06';

UPDATE EMPLOYEE
SET DNO='1', SUPERSSN='RNSACC02'
WHERE SSN='RNSACC01';

UPDATE EMPLOYEE
SET DNO='1', SUPERSSN=NULL
WHERE SSN='RNSACC02';

UPDATE EMPLOYEE
SET DNO='4', SUPERSSN=NULL
WHERE SSN='RNSISE01';

UPDATE EMPLOYEE
SET DNO='2', SUPERSSN=NULL
WHERE SSN='RNSIT01';
```

```
INSERT INTO DLOCATION VALUES ('BANGALORE', '1');
INSERT INTO DLOCATION VALUES ('BANGALORE', '2');
INSERT INTO DLOCATION VALUES ('BANGALORE', '3');
INSERT INTO DLOCATION VALUES ('MANGALORE', '4');
INSERT INTO DLOCATION VALUES ('MANGALORE', '5');
```

```
INSERT INTO PROJECT VALUES (100,'IOT','BANGALORE','5');
INSERT INTO PROJECT VALUES (101,'CLOUD','BANGALORE','5');
INSERT INTO PROJECT VALUES (102,'BIGDATA','BANGALORE','5');
INSERT INTO PROJECT VALUES (103,'SENSORS','BANGALORE','3');
INSERT INTO PROJECT VALUES (104,'BANK MANAGEMENT','BANGALORE','1');
INSERT INTO PROJECT VALUES (105,'SALARY MANAGEMENT','BANGALORE','1');
INSERT INTO PROJECT VALUES (106,'OPENSTACK','BANGALORE','4');
INSERT INTO PROJECT VALUES (107,'SMART CITY','BANGALORE','2');

INSERT INTO WORKS_ON VALUES (4, 'RNSCSE01', 100);
INSERT INTO WORKS_ON VALUES (6, 'RNSCSE01', 101);
INSERT INTO WORKS_ON VALUES (8, 'RNSCSE01', 102);
INSERT INTO WORKS_ON VALUES (10,'RNSCSE02', 100);
INSERT INTO WORKS_ON VALUES (3, 'RNSCSE04', 100);
INSERT INTO WORKS_ON VALUES (4, 'RNSCSE05', 101);
INSERT INTO WORKS_ON VALUES (5, 'RNSCSE06', 102);
INSERT INTO WORKS_ON VALUES (6, 'RNSCSE03', 102);
INSERT INTO WORKS_ON VALUES (7, 'RNSECE01', 103);
INSERT INTO WORKS_ON VALUES (5, 'RNSACC01', 104);
INSERT INTO WORKS_ON VALUES (6, 'RNSACC02', 105);
INSERT INTO WORKS_ON VALUES (4, 'RNSISE01', 106);
INSERT INTO WORKS_ON VALUES (10,'RNSIT01', 107);
```

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.

```
(SELECT DISTINCT P.PNO
FROM PROJECT P, DEPARTMENT D, EMPLOYEE E
WHERE E.DNO=P.DNO
AND D.MGRSSN=E.SSN
AND E.LNAME='SCOTT')
UNION
(SELECT DISTINCT P1.PNO
```

```
FROM PROJECT P1, WORKS_ON W, EMPLOYEE E1
WHERE P1.PNO=W.PNO
AND E1.SSN=W.SSN
AND E1.LNAME='SCOTT');
```

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

```
SELECT E.FNAME, E.LNAME, 1.1*E.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';
```

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.

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

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

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE E
WHERE NOT EXISTS(SELECT PNO
        FROM PROJECT P
        WHERE DNO='5'
        AND P.PNO NOT IN ( SELECT PNO
            FROM WORKS_ON
            WHERE E.SSN=SSN));
```

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.

```
SELECT D.DNO, COUNT(*)
FROM DEPARTMENT D, EMPLOYEE E
WHERE D.DNO=E.DNO
AND E.SALARY>600000
AND D.DNO IN (SELECT E1.DNO
        FROM EMPLOYEE E1
        GROUP BY E1.DNO
         HAVING COUNT(*)>5)
GROUP BY D.DNO;
```