

CHAPTER - 1

BASIC CONCEPTS OF PostgreSQL

1.1 Introduction to PostgreSQL

PostgreSQL is an object-relational database management system (ORDBMS) based on the INGRES (INteractive Graphics REtrieval System) package. Developed at the University of California, Berkeley. The POSTGRES (Post Ingres) project started in 1985, and version 1 was released to a small number of external users in June of 1989. PostgreSQL has become the most advanced open source database, available all over the world. PostgreSQL has become a popular Database as a Service (DBaaS) among the current clouds. PostgreSQL can be delivered as DBaaS on many clouds, such as Amazon Web Services (AWS), Google Cloud SQL, Microsoft Azure, Heroku, and EnterpriseDB Cloud. PostgreSQL supports a large part of the SQL standard and offers many modern features: complex queries, foreign keys, triggers, updatable views, transactional integrity, multiversion concurrency control. PostgreSQL can be extended by the user in many ways, for example by adding new data types, functions, operators, aggregate functions, index methods and procedural languages. It is fully ACID(Atomicity, Consistency, Isolation, Durability) compliant which makes it an ideal choice for OLTP (Online Transaction Processing). It is also capable of performing database analytics. It can be integrated with mathematical software like Matlab and R.

1.2 PostgreSQL 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_nameN datatype);
```

- *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.

PostgreSQL Data Types:

<u>Character Datatypes</u>	<u>Geometric Data Types</u>
<u>Numeric Datatypes</u>	<u>Enumerated Types</u>
<u>Binary Data Types</u>	<u>Range Type</u>
<u>Network Address Type</u>	<u>UUID type</u>
<u>Text Search Type</u>	<u>XML type</u>
<u>Date/Time Datatypes</u>	<u>JSON Type</u>
<u>Boolean Type</u>	<u>Pseudo-Types</u>



Character Datatypes

PostgreSQL supports character data types for storing text values. PostgreSQL builds character data types off of the same internal structures. PostgreSQL offers three character data types: CHAR(n), VARCHAR(n), and TEXT.

Name Description

varchar(n) Allows you to declare variable-length with a limit

Char(n) Fixed-length, blank padded

Text Use can use this data type to declare a variable with unlimited length

Numeric Datatypes

PostgreSQL supports two distinct types of numbers:

- Integers
- Floating-point numbers

Name	Store size	Range
smallint	2 bytes	-32768 to +32767
integer	4 bytes	-2147483648 to +2147483647
bigint	8 bytes	-9223372036854775808 to 9223372036854775807
decimal variable		If you declared it as decimal datatype ranges from 131072 digits before the decimal point to 16383 digits after the decimal point
numeric variable		If you declare it as the number, you can include number up to 131072 digits before the decimal point to 16383 digits after the decimal point
real	4 bytes	6 decimal digits precision
double	8 bytes	15 decimal digits precision

Binary Data Types

A binary string is a sequence of octets or bytes. Binary Postgres Data Types are divided in two ways.

- Binary strings allow storing odds of value zero
- Non-printable octets



Character strings not allow zero octets and also disallows any other octet values and sequences which are invalid as per the database's character set encoding rules.

Name	Storage size	Description
Byte	1 to 4 bytes plus the size of the binary string	Variable-length binary string

Network Address Type

Many applications store network information like IP addresses of users or sensors. PostgreSQL has three native types which help you to optimize the network data.

Name	Size	Description
cider	7 or 19 bytes	IPv4 and IPv6 networks
Inet	7 or 19 bytes	IPv4 and IPv5 host and networks
macaddr	6 bytes	MAC addresses

PostgreSQL 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_name datatype [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  
referenced table name(c
```

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_name  
datatype;
```

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_name  
datatype;
```

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  
column n
```

Syntax to drop PRIMARY KEY constraint

```
ALTER TABLE table_name DROP 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_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:

Syntax

CHAPTER - 2

Following is the basic syntax of stored procedure creation.

LIBRARY DATABASE

Q1) Consider the following schema for a Library Database:

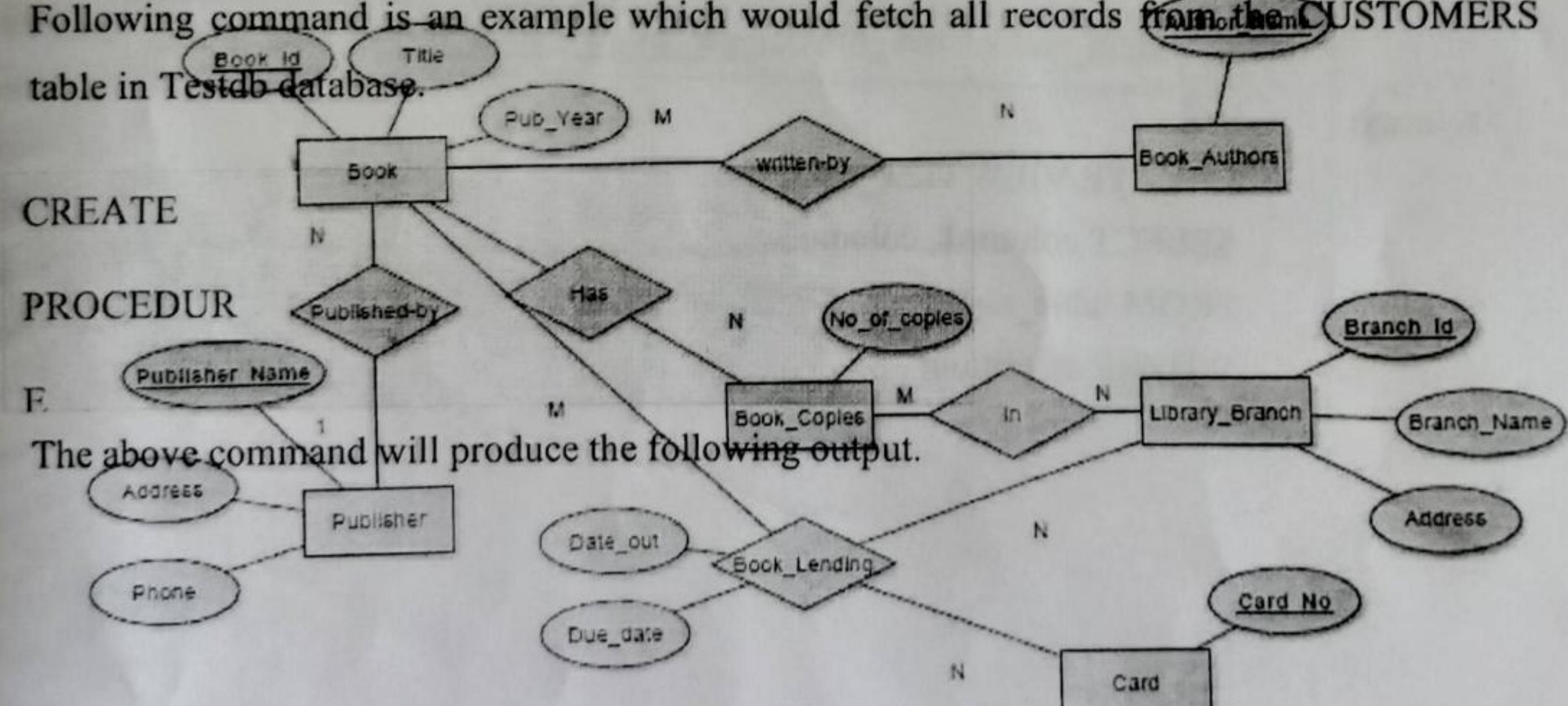
- r BOOK (Book_id, Title, Publisher_Name, Pub_Year)
- e BOOK_AUTHORS (Book_id, Author_Name)
- e PUBLISHER (Name, Address, Phone)
- a BOOK_COPIES (Book_id, Branch_id, No_of_Copies)
- BOOK_LENDING (Book_id, Branch_id, Card_No, Date_Out, Due_Date)

Example LIBRARY_BRANCH (Branch_id, Branch_Name, Address)

Consider the CUSTOMERS table having the following records.

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.
4. Manipulation operation. 1500.00
5. Partition the BOOK table based on year of publication. Demonstrate its working with a simple query.
6. Create a view of all books and its number of copies that are currently available in the Library.
7. Hardik 27 Mumbai 8500.00
8. Muffy 24 Indore 10000.00

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



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 VIEWS IN SQL

- o Views in SQL are virtual table. A view also contains rows and columns.
- o To create the view, we can select the fields from one or more tables present in the database.
- o A view can either have specific rows based on certain condition or all the rows of a table.
The reasons for using Views"
- o Views restrict access to the data because the view can display selective columns and rows from the table.
- o Views provide groups of users with access to data according to their particular permissions.
- o Views can be used to retrieve data from several tables, providing data independence for users.

Syntax:

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

Syntax

CHAPTER - 2

Following is the basic syntax of stored procedure creation.
LIBRARY DATABASE

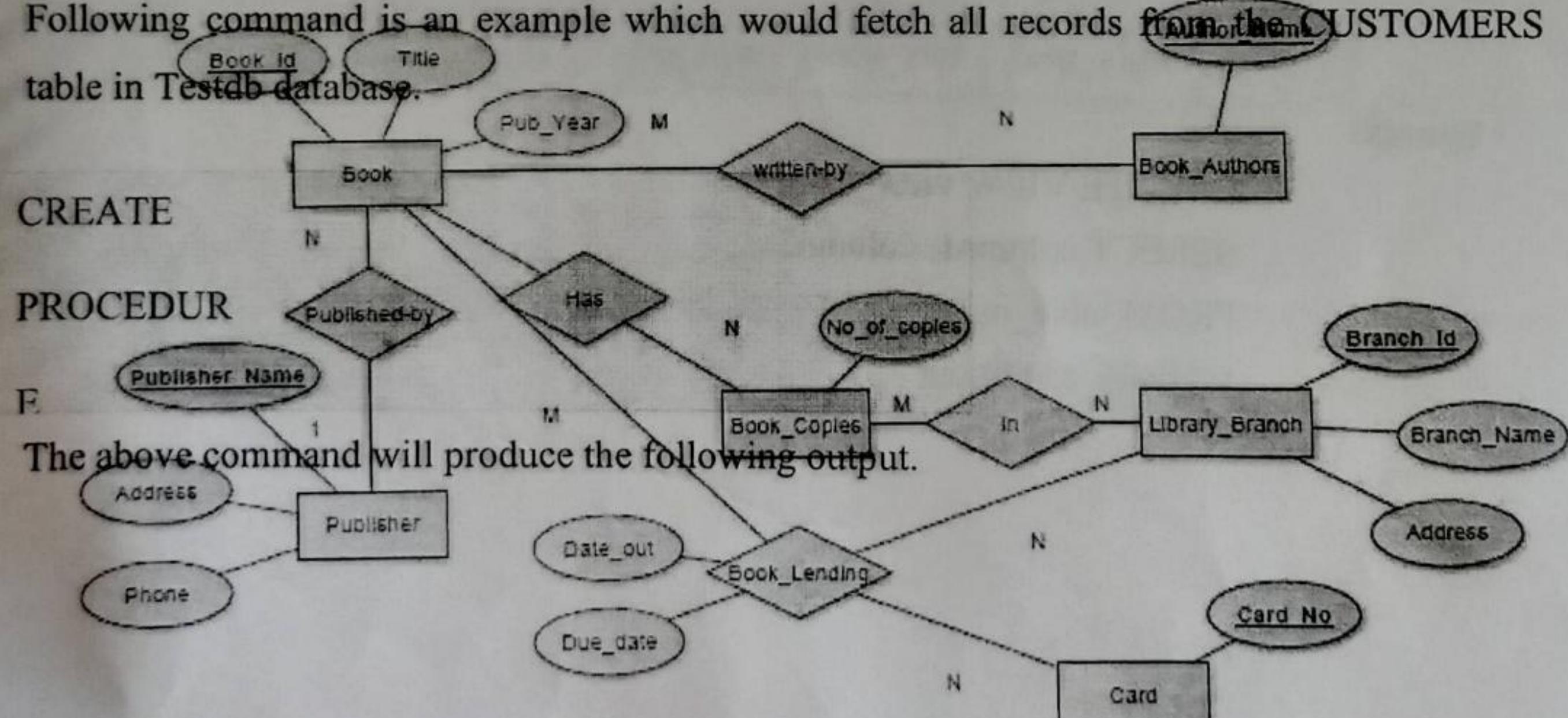
Q1) Consider the following schema for a Library Database:

r BOOK (Book_id, Title, Publisher_Name, Pub_Year)
 e BOOK_AUTHORS (Book_id, Author_Name)
 e PUBLISHER (Name, Address, Phone)
 a BOOK_COPIES (Book_id, Branch_id, No-of_Copies)
 Example BOOK_LENDING (Book_id, Branch_id, Card_No, Date_Out, Due_Date)
 LIBRARY_BRANCH (Branch_id, Branch_Name, Address)

Consider the SQL CUSTOMERS table having the following records.

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
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.
6. ER Diagram
7. Muffy 24 Indore 10000.00

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



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

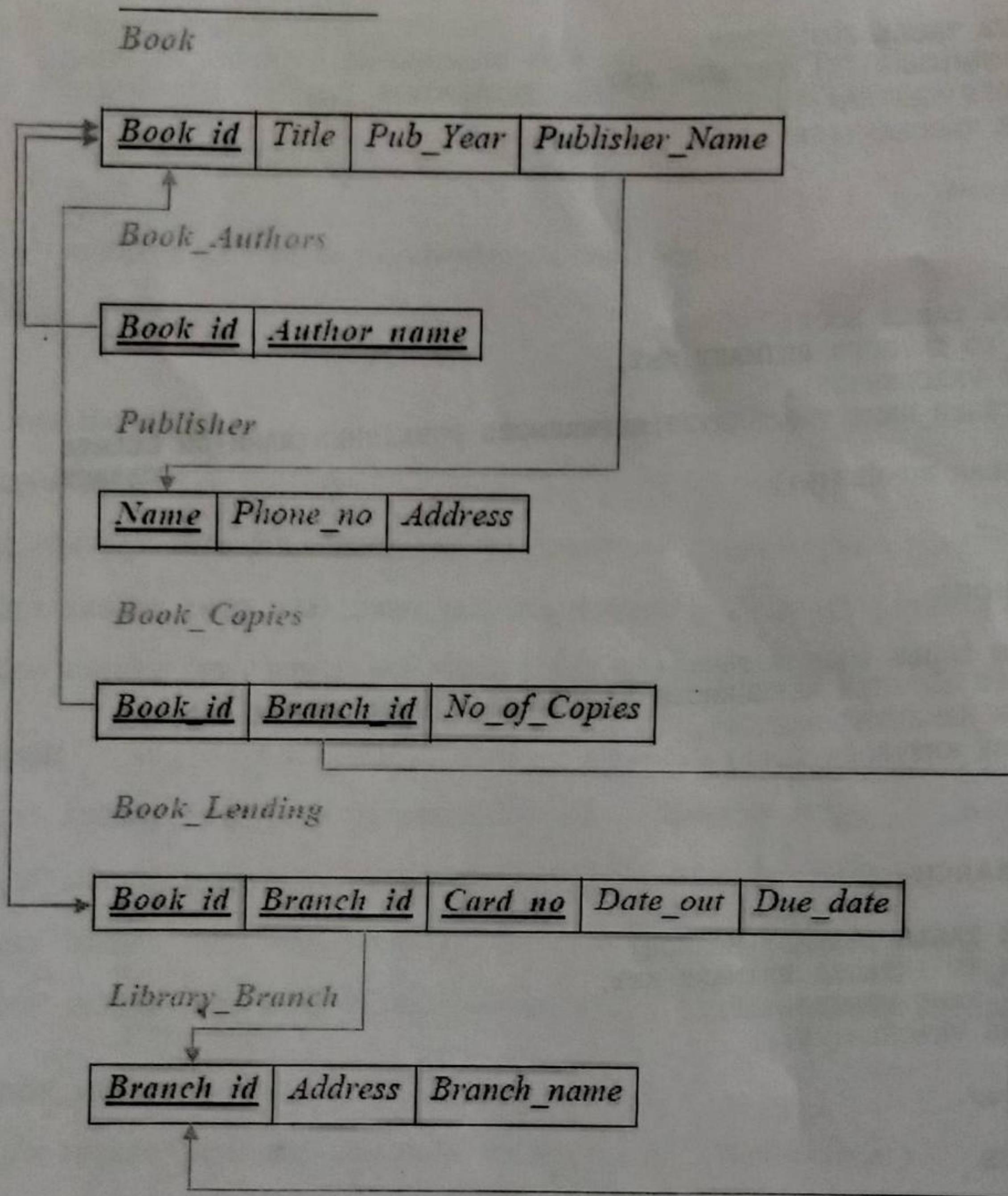
SCHEMA:

Table Creation:**PUBLISHER**

```
SQL> CREATE TABLE PUBLISHER(
    NAME VARCHAR(18) PRIMARY KEY,
    ADDRESS VARCHAR(10),
    PHONE VARCHAR(10));
```

Table created.

BOOK

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

Table created.

BOOK_AUTHORS

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

Table created.

LIBRARY_BRANCH

```
SQL> CREATE TABLE LIBRARY_BRANCH(
    BRANCH_ID INTEGER PRIMARY KEY,
    BRANCH_NAME VARCHAR(18),
    ADDRESS VARCHAR(15));
```

Table created.

BOOK_COPIES

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

Table created.



BOOK_LENDING

```
SQL> CREATE TABLE BOOK_LENDING(
    BOOK_ID INTEGER REFERENCES BOOK(BOOK_ID) ON DELETE CASCADE,
    BRANCH_ID INTEGER REFERENCES LIBRARY_BRANCH(BRANCH_ID) ON DELETE
    CASCADE,
    CARD_NO INTEGER,
    DATE_OUT DATE,
    DUE_DATE DATE,
    PRIMARY KEY(BOOK_ID, BRANCH_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 BOOK VALUES(1111, 'SE', 'PEARSON', 2005);

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

SQL> INSERT INTO BOOK VALUES(3333, 'ANATOMY', 'PEARSON', 2010);

SQL> INSERT INTO BOOK VALUES(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, 'THOMAS');
```



LIBRARY_BRANCH

```
SQL> INSERT INTO LIBRARY_BRANCH VALUES(11, 'CENTRAL TECHNICAL', 'MG ROAD');  
SQL> INSERT INTO LIBRARY_BRANCH VALUES(22, 'MEDICAL', 'BH ROAD');  
SQL> INSERT INTO LIBRARY_BRANCH VALUES(33, 'CHILDREN', 'SS PURAM');  
SQL> INSERT INTO LIBRARY_BRANCH VALUES(44, 'SECRETARIAT', 'SIRAGATE');  
SQL> INSERT INTO LIBRARY_BRANCH 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_LENDING VALUES(2222, 11, 1, '10-JAN-2017', '20-AUG-2017');  
SQL> INSERT INTO BOOK_LENDING VALUES(3333, 22, 2, '09-JUL-2017', '12-AUG-2017');  
SQL> INSERT INTO BOOK_LENDING VALUES(4444, 55, 1, '11-APR-2017', '09-AUG-2017');  
SQL> INSERT INTO BOOK_LENDING VALUES(2222, 11, 5, '09-AUG-2017', '19-AUG-2017');  
SQL> INSERT INTO BOOK_LENDING VALUES(4444, 33, 1, '10-JUN-2017', '15-AUG-2017');  
SQL> INSERT INTO BOOK_LENDING VALUES(1111, 11, 1, '12-MAY-2017', '10-JUN-2017');  
SQL> INSERT INTO BOOK_LENDING VALUES(3333, 22, 1, '10-JUL-2017', '15-JUL-2017');
```

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

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

4 rows selected.

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

BOOK_ID	AUTHOR_NAME
1111	SOMMERRVILLE
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	BRANCH_ID	NO_OF_COPIES
1111	11	5
3333	22	6
4444	33	10
2222	11	12
4444	55	3

5 rows selected.

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

BOOK_ID	BRANCH_ID	CARD_NO	DATE_OUT	DU_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-JUL-17	15-AUG-17
1111	11	1	12-MAY-17	10-JUN-17
3333	22	1	10-JUL-17	15-JUL-17

7 rows selected.

SQL> SELECT * FROM LIBRARY_BRANCH;

BRANCH_ID	BRANCH_NAME	ADDRESS
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.

```
SELECT LB.BRANCH_NAME, B.BOOK_ID, TITLE, PUBLISHER_NAME, AUTHOR_NAME,
       NO_OF_COPIES
  FROM BOOK B, BOOK_AUTHORS BA, BOOK_COPIES BC, LIBRARY_BRANCH LB
 WHERE B.BOOK_ID = BA.BOOK_ID AND
       BA.BOOK_ID = BC.BOOK_ID AND
       BC.BRANCH_ID = LB.BRANCH_ID
 GROUP BY LB.BRANCH_NAME, B.BOOK_ID, TITLE, PUBLISHER_NAME,
          AUTHOR_NAME, NO_OF_COPIES;
```

BRANCH_NAME	BOOK_ID	TITLE	PUBLISHER_NAME	AUTHOR_NAME	NO_OF_COPIES
GENERAL	4444	ENCYCLOPEDIA	SAPNA	THOMAS	3
MEDICAL	3333	ANATOMY	PEARSON	HENRY GRAY	6
CHILDREN	4444	ENCYCLOPEDIA	SAPNA	THOMAS	10
CENTRAL TECHNICAL	1111	SE	PEARSON	SOMMERVILLE	5
CENTRAL TECHNICAL	2222	DBMS	MCGRAW	NAVATHE	12

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

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

CARD_NO
1

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 = '3333';

1 row deleted.
```

```
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	BRANCH_ID	NO_OF_COPIES
1111	11	5
4444	33	10
2222	11	12
4444	55	3

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

BOOK_ID	BRANCH_ID	CARD_NO	DATE_OUT	DU_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.

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

BOOK_ID	TITLE	PUBLISHER_NAME	PUB_YEAR
2222	DBMS	MCGRAW	2004
1111	SE	PEARSON	2005
3333	ANATOMY	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.

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

View created.

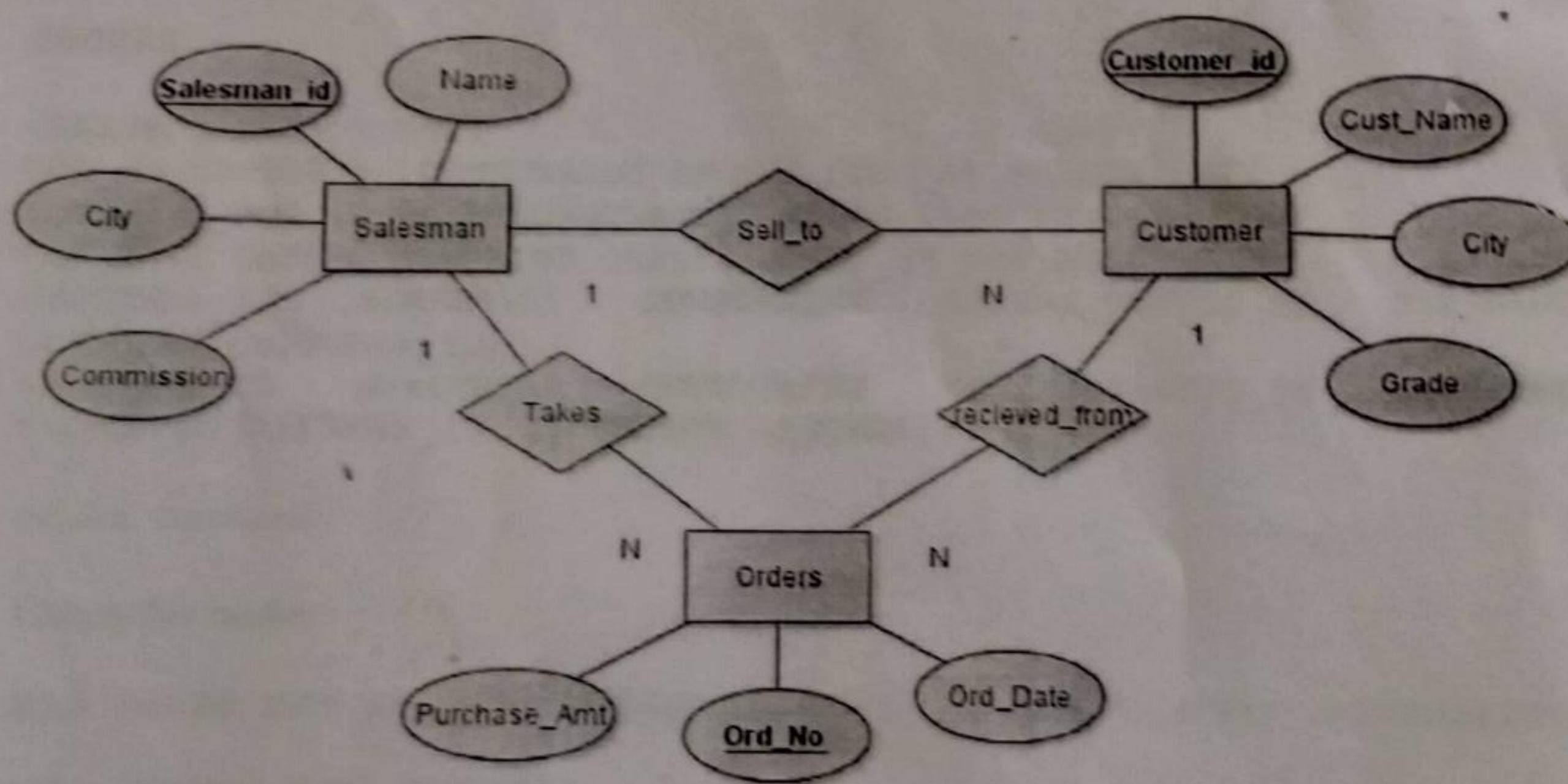
```
SQL> SELECT * FROM BOOKS_AVAILABLE;
```

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

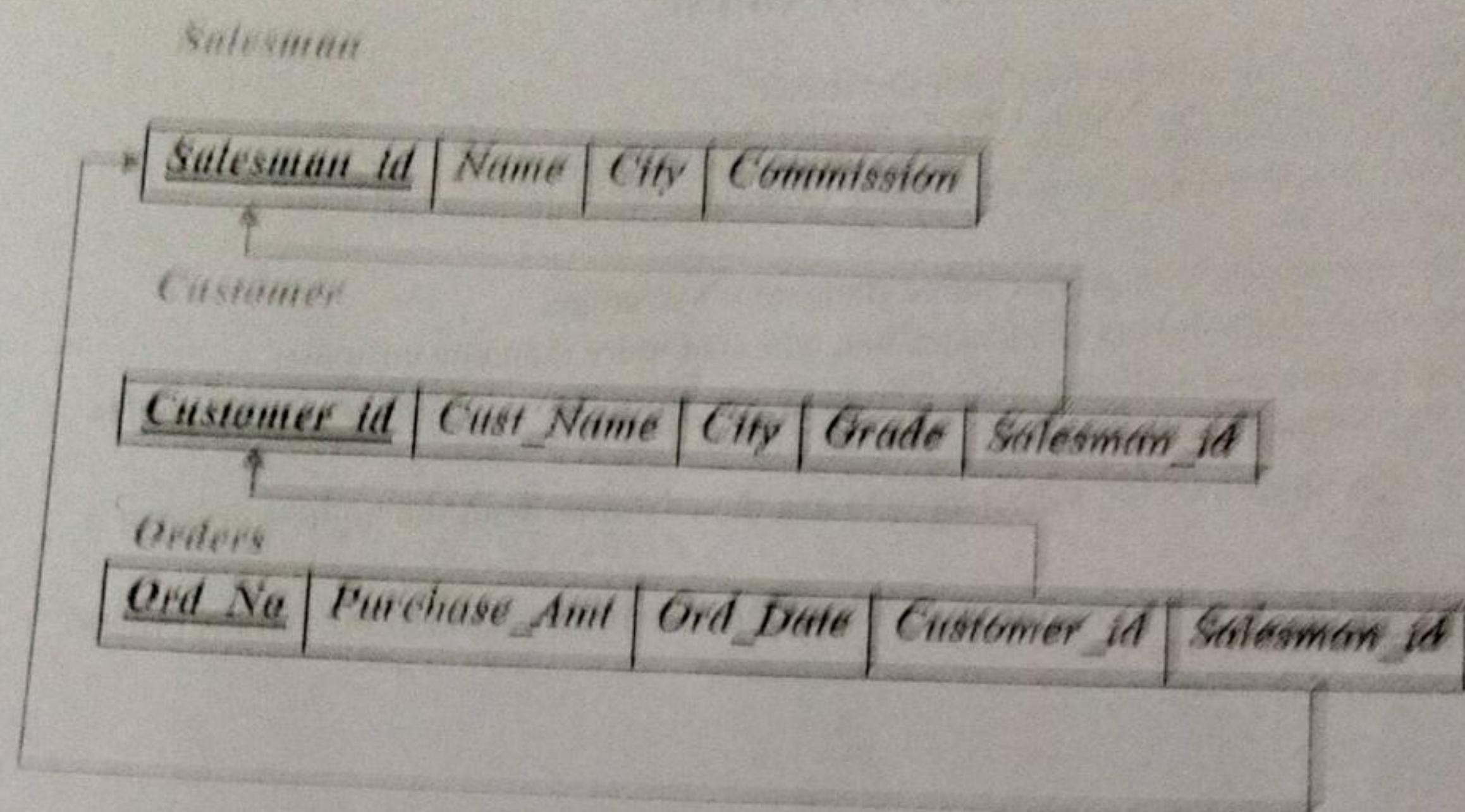
CHAPTER – 3

ORDER DATABASE

- 2) 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:

SCHEMAS



23 | Page



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 VARCHAR(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

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

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

COUNT(CUSTOMER_ID)
```

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

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

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.)

```
(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));
```

NAME
ASHWIN
BINDU
LAVANYA
RAJ
ROHIT

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

```
CREATE VIEW SALES_HIGHERODER 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.

```
SELECT * FROM SALES_HIGHERODER;
SALESMAN_ID PURCHASE_AMT
-----
2000      300000
```

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;
```

1 row deleted.

```
SELECT * FROM SALESMAN;
```

SALESMAN_ID	NAME	CITY	COMMISSION
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	
22	TCS	BENGALURU	4	2000
33	WIPRO	mysore	7	
44	TCS	mysore	6	2000
55	ORACLE	TUMKUR	3	3000

```
SELECT * FROM ORDERS;
```

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

CHAPTER - 4

MOVIE DATABASE

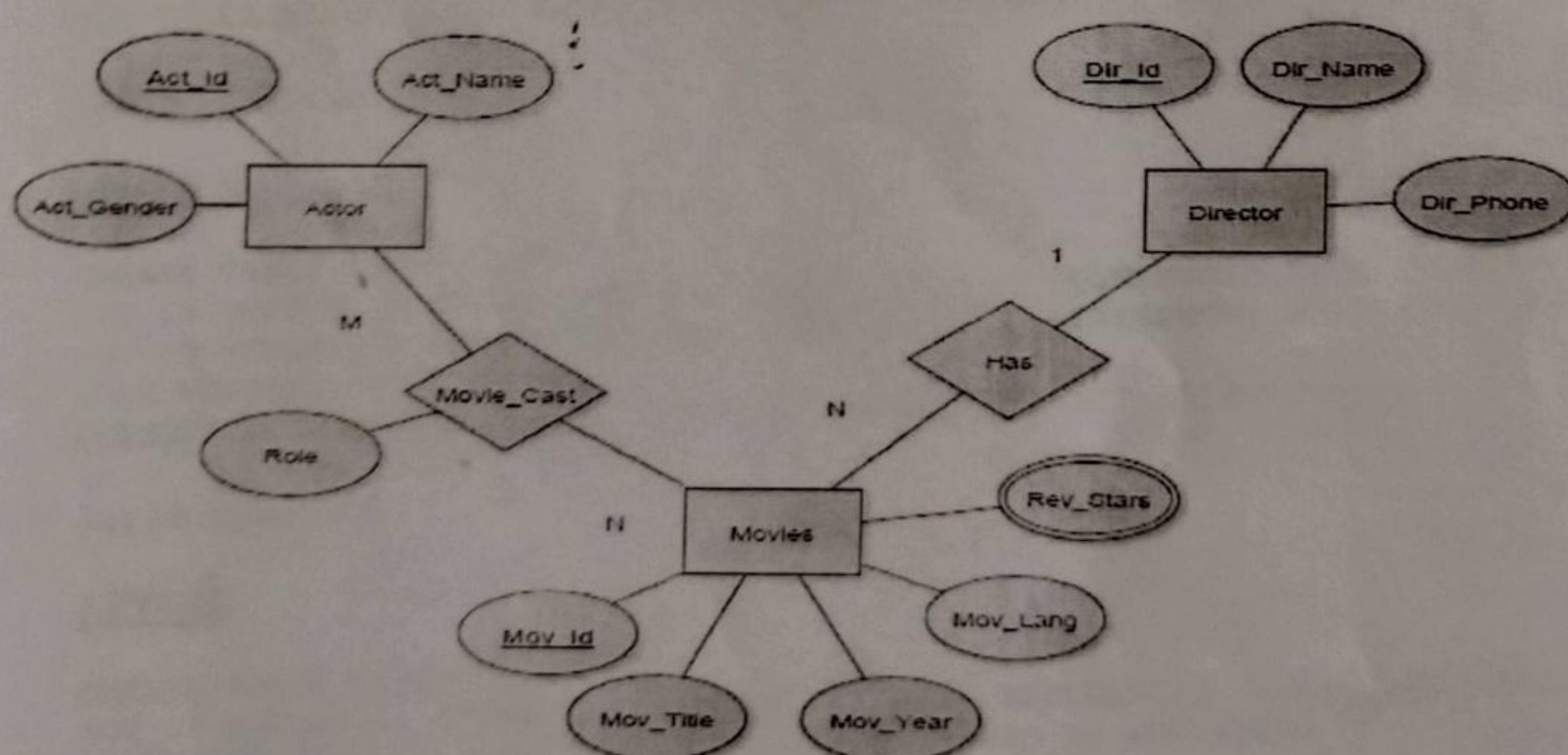
- 3) Consider the schema for Movie Database:

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:



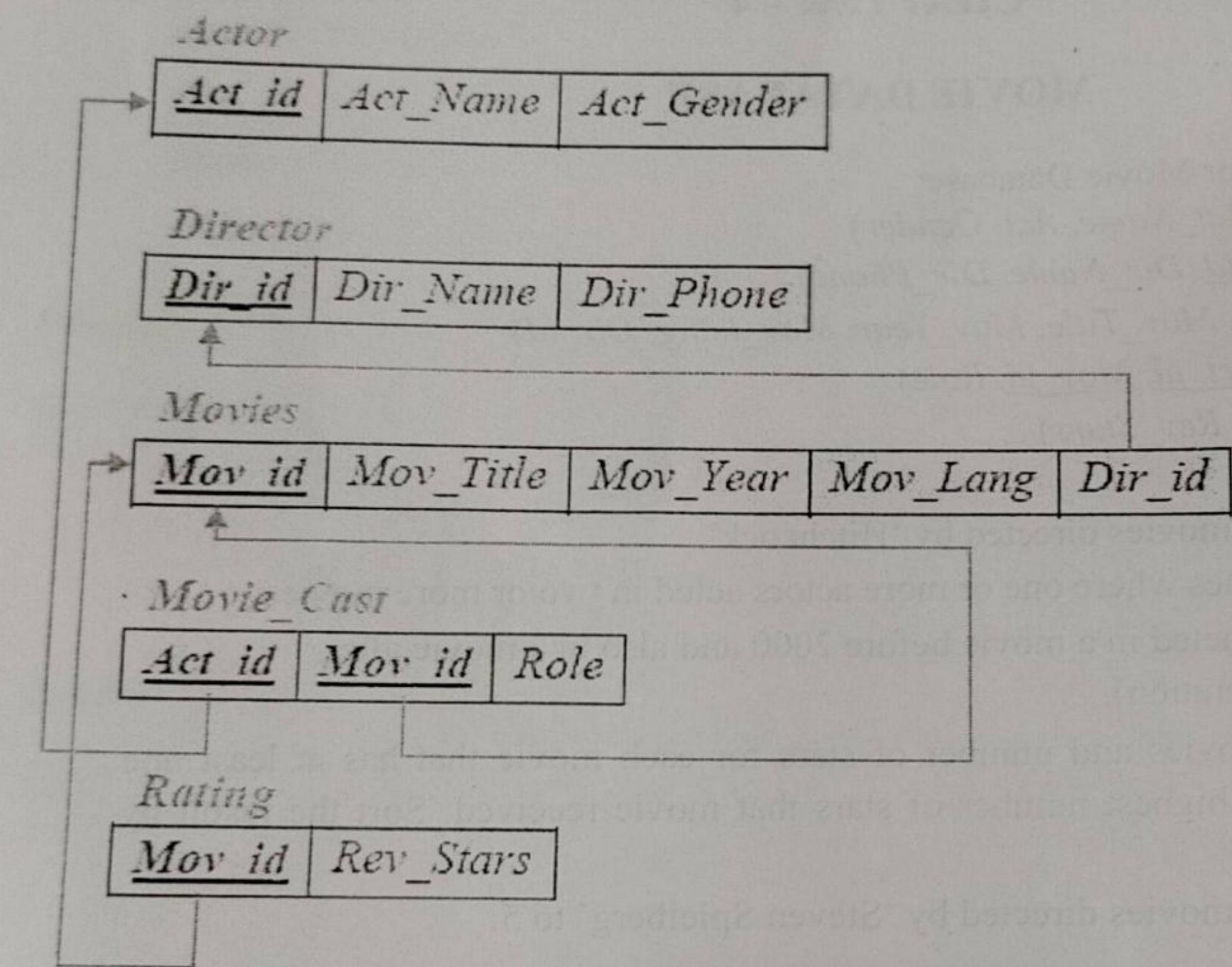
SCHEMA:

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.



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	DHIGANTEH	M
555	ANGELA	F

SQL> SELECT * FROM DIRECTOR;

DIR_ID	DIR_NAME	DIR_PHONE
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	2222	VILAN
333	4444	HERO
111	4444	HEROIN
444	3333	GUEST
444	5555	HERO
555	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'.

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

MOV_TITLE

NOTORIOUS
REAR WINDOW

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

```
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);
```

MOV_TITLE

MALE
MANASARE

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

```
(SELECT ACT_NAME
FROM ACTOR A
JOIN MOVIE_CAST C
ON
A.ACT_ID=C.ACT_ID
4. 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);
```

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.

```
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
```

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.

```
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.

```
SELECT * FROM RATING
```

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



CHAPTER - 5

COLLEGE DATABASE

4). 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:

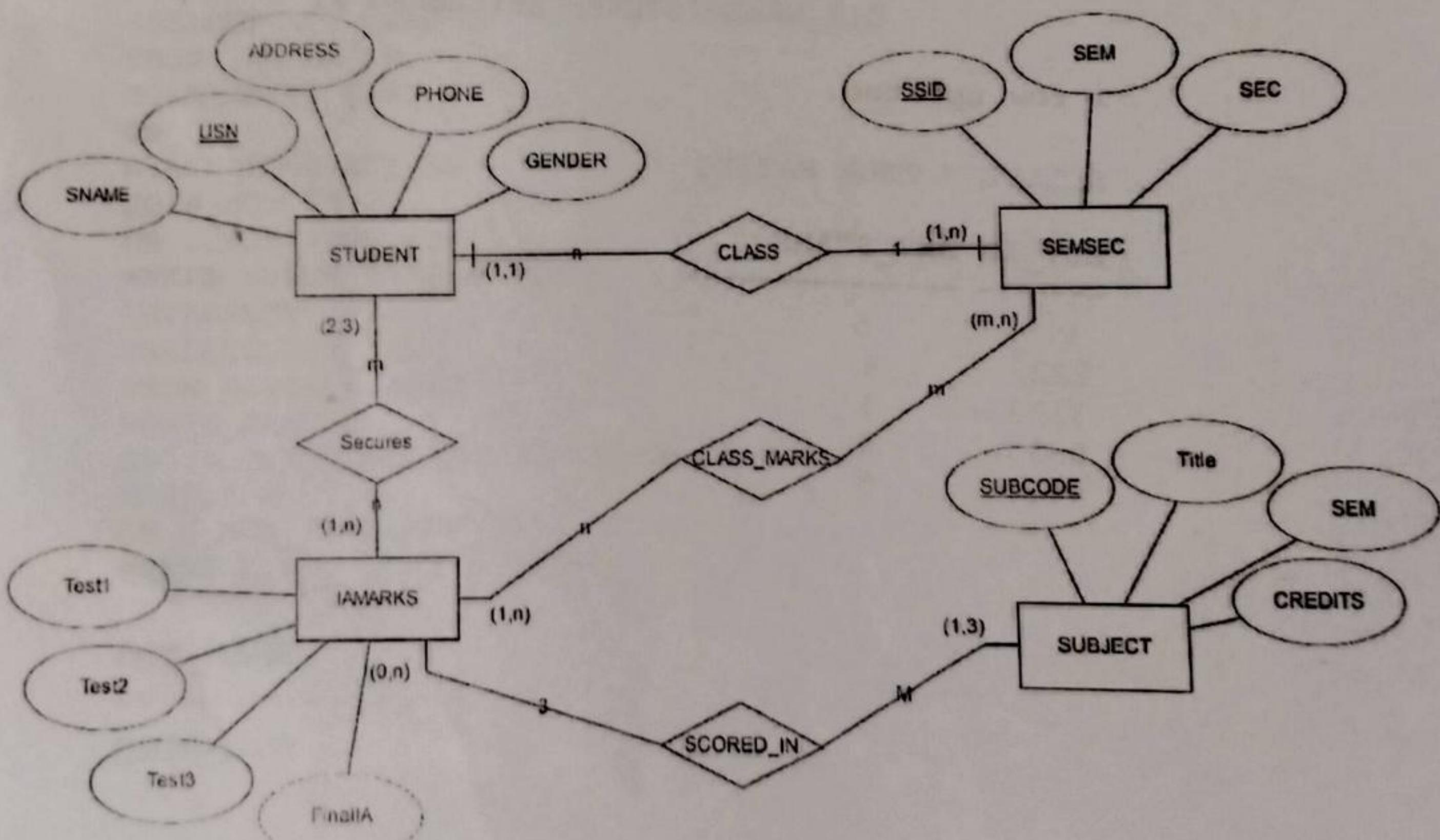


Table Creation:**STUDENT**

```
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.

CLASS

```
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));
```

Table created.

SUBJECT

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

Table created.

IAMARKS

```
CREATE TABLE IAMARKS  
(USN VARCHAR(10),  
SUBCODE VARCHAR(8),  
SSID VARCHAR(5), TEST1  
NUMBER(2), TEST2  
NUMBER(2),
```

```

TEST3 NUMBER(2),
FINALIA NUMBER(3),
PRIMARY KEY(USN, SUBCODE, SSID),
FOREIGN KEY(JSN) 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');

```

```
select * from student;
```

USN	SNAME	ADDRESS	PHONE	G
1cg15cs001	Abhi	tumkur	9875698410	M
1cg15cs002	amulya	gubbi	8896557412	F
1cg16me063	chetnan	nittur	7894759522	M
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	5 A
3B	3 B
7A	7 A
2C	2 C
4B	4 B
4C	4 c

CLASS:

```

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

```

```
select * from class;
```

USN	SSID
1cg15cs001	5A

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.

```
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;
```

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

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

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

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 NUMBER;
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.

```
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
SUBSEM=7
```

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



CHAPTER – 6

COMPANY DATABASE

5). Consider the schema for Company Database:

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

DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate)

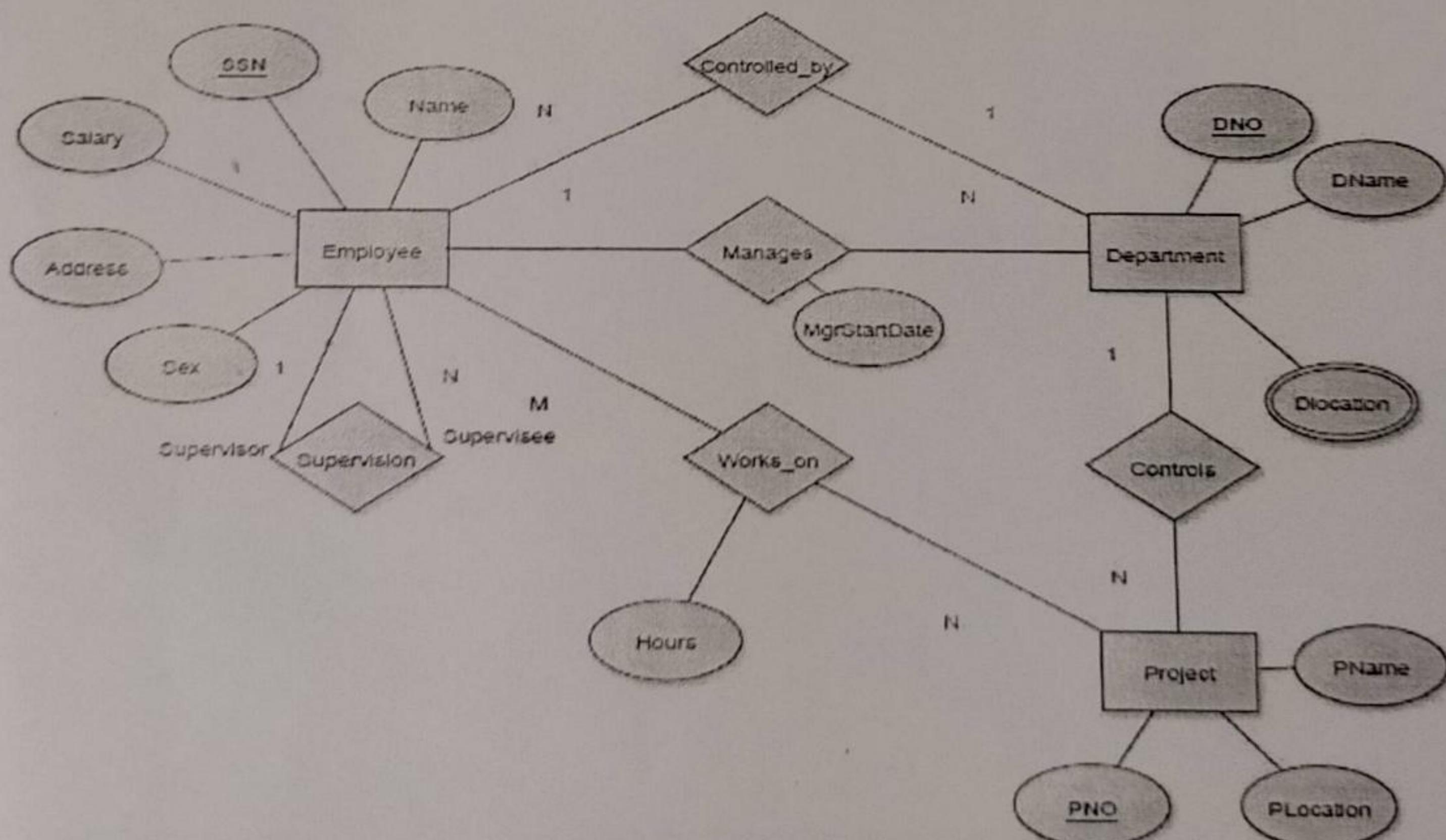
DLOCATION (DNo, DLoc)

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:

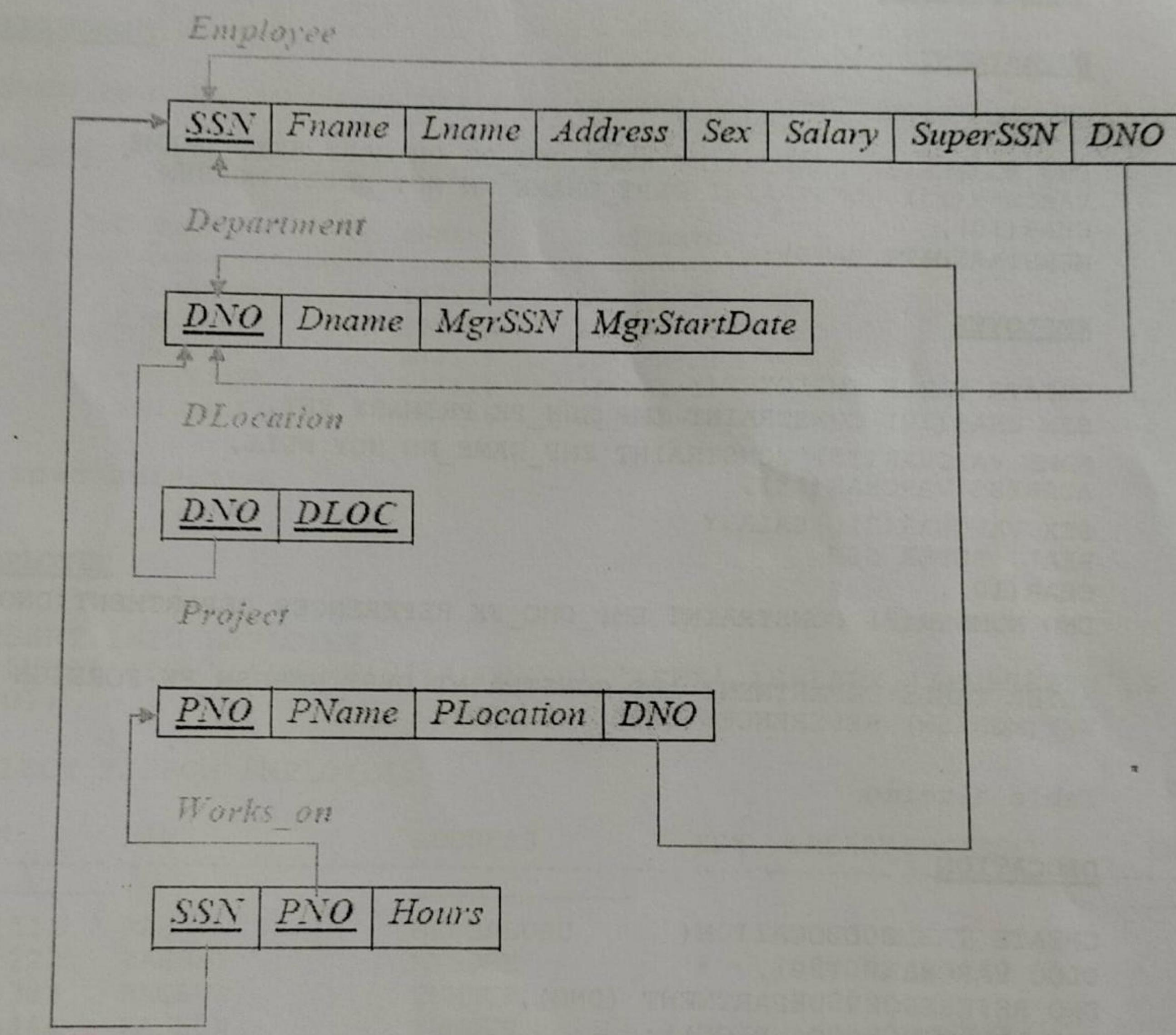
SCHEMA:

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, '&SUPERSSN', &
DNO);
```

```
SELECT * FROM EMPLOYEE;
```

SSN	NAME	ADDRESS	SEX	SALARY	SUPERSSN	DNO
111111	RAJ	BENGALURU	M	700000		1
222222	RASHMI	mysore	F	400000	111111	2
333333	RAGAVI	TUMKUR	F	800000		3
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 last name is 'Scott', either as a worker or as a manager of the department that controls the project.

```
(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');
```

PNO

111

333

444

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

```
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';
```

SSN	NAME	ADDRESS	SEX	SALARY SUPERSSN	DNO
111111	RAJ	BENGALURU	M	700000	1
222222	RASHMI	mysore	F	440000 111111	2
333333	RAGAVI	TUMKUR	F	880000	3
444444	RAJESH	TUMKUR	M	715000 333333	3
555555	RAVINDER	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.

```
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).

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

```
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 ;
```