**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Concept** | **Page no** |
| **CHAPTER 1** | **BASIC CONCEPTS OF SQL** | **1** |
| 1.1 | Introduction to SQL | 1 |
| 1.2 | SQL Commands | 1 |
|  | 1.2.1 DDL Commands | 2 |
|  | 1.2.2 DML Commands | 6 |
|  | 1.2.3 TCL Commands | 9 |
|  | 1.2.4 DCL Commands | 10 |
| 1.3 | Stored Procedures in SQL | 10 |
| 1.4 | SQL Triggers | 12 |
| 1.5 | Views in SQL | 17 |
| **CHAPTER 2** | **LAB PROGRAM 1 - LIBRARY DATABASE** | **18** |
| 2.1 | Problem Statement | 18 |
| 2.2 | ER Diagram | 18 |
| 2.3 | Schema Diagram | 19 |
| 2.4 | Creating Tables | 20 |
| 2.5 | Inserting Values | 21 |
| 2.6 | Queries and Solutions | 24 |
| **CHAPTER 3** | **LAB PROGRAM 2 - ORDER DATABASE** | **29** |
| 3.1 | Problem Statement | 29 |
| 3.2 | ER Diagram | 29 |
| 3.3 | Schema Diagram | 30 |
| 3.4 | Creating Tables | 31 |
| 3.5 | Inserting Values | 31 |
| 3.6 | Queries and Solutions | 32 |
| **CHAPTER 4** | **LAB PROGRAM 3 - MOVIE DATABASE** | **37** |
| 4.1 | Problem Statement | 37 |
| 4.2 | ER Diagram | 37 |
| 4.3 | Schema Diagram | 38 |
| 4.4 | Creating Tables | 39 |
| 4.5 | Inserting Values | 40 |
| 4.6 | Queries and Solutions | 42 |
| **CHAPTER 5** | **LAB PROGRAM 4 - COLLEGE DATABASE** | **46** |

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

**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\_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.

**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 ‘yyyy-mm-dd’. Eg., ‘2007-10-04’ |
| 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\_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 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:**

*[CON*STRAINT 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

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\_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\_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 | amedabad | 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.

**CHAPTER – 2**

**LIBRARY DATABASE**

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

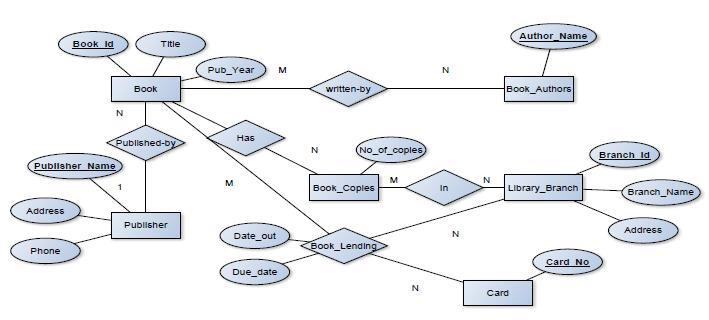
3. Delete a book in BOOK table. Update the contents of other tables to reflect this data manipulation operation.

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

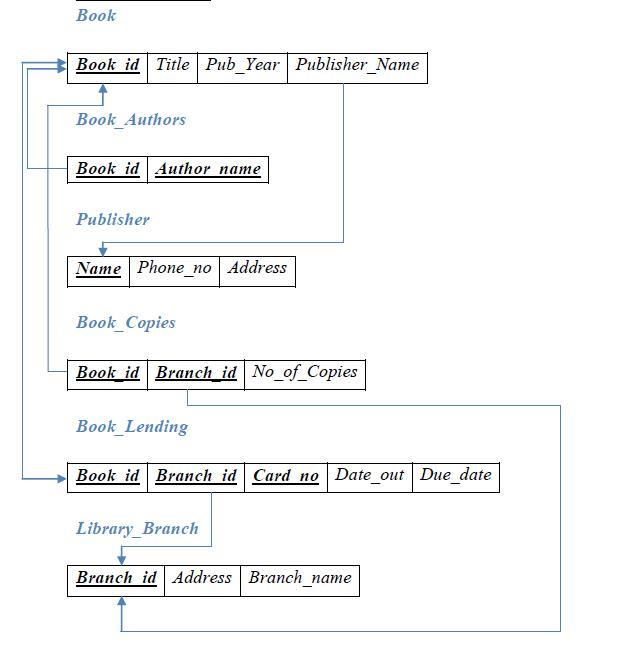
Library.

**ER-Diagram:**



**SCHEMA:**





**Table Creation:**

**PUBLISHER**

create table publisher

(

name varchar(30),

address varchar(30) not null,

phone varchar(12),

primary key(name)

);

**BOOK**

create table book

(

bookid int not null,

title varchar(20),

name varchar(20) ,

pubyear int,

primary key(bookid,name),

foreign key(name) references publisher(name) on delete cascade

);

**BOOK\_AUTHORS**

create table authors

(

bookid int ,

name varchar(30),

primary key(bookid),

foreign key(bookid) references book(bookid) on delete cascade

);

**LIBRARY\_BRANCH**

create table branch

(

branchid int,

branchname varchar(20) ,

address varchar(200),

primary key(branchid)

);

**BOOK\_COPIES**

create table bookcopies

(

bookid int ,

branchid int ,

noofcopies int ,

primary key(bookid,branchid),

foreign key(bookid) references book(bookid) on delete cascade,

foreign key(branchid) references branch(branchid) on delete cascade

);

**Borrower**

create table borrower

(

cardno int,

name varchar(20),

primary key(cardno)

);

**BOOK\_LENDING**

create table lending

(

bookid int ,

branchid int ,

cardno int,

dateout date,

duedate date ,

primary key(bookid,branchid,cardno),

foreign key(bookid) references book(bookid) on delete cascade,

foreign key(branchid) references branch(branchid) on delete cascade,

foreign key(cardno) references borrower(cardno) on delete cascade

);

**Values for tables:**

**PUBLISHER**

insert into publisher values("Jaico","banglore","77002299");

insert into publisher values("Westland","pune","770022549");

insert into publisher values("Penguin ","mumbai","77002199");

insert into publisher values("Roli","sanaa","77002229");

insert into publisher values("Rupa","thamar","77202299");

insert into publisher values("Hachette","aden","77002299");

insert into publisher values("ayaan","us","77002299");

**BOOK**

insert into book values (1,"learn html","Jaico",2012);

insert into book values (2,"learn java","Jaico",2005);

insert into book values (3,"learn python","Jaico",2016);

insert into book values (4,"learn anythin","Jaico",2010);

insert into book values (5,"c#","Jaico",2011);

insert into book values (6,"linux","Jaico",2017);

**BOOK\_AUTHORS**

insert into authors values (1," DEEPAK T");

insert into authors values (2," GIRISH");

insert into authors values (3," GANESH");

insert into authors values (4," KARTIK KUMAR");

insert into authors values (5," KISHORE");

insert into authors values (6," MANISH");

**LIBRARY\_BRANCH**

insert into branch values (1,"cmrit","banglore");

insert into branch values (2,"cmrims","germany");

insert into branch values (3,"cmrpu","us");

insert into branch values (4,"itpl","mysore");

insert into branch values (5,"nps","mumbai");

**Borrower**

insert into borrower values (1,”pranav”);

insert into borrower values (2,"anisha");

insert into borrower values (3,"kiran ");

insert into borrower values (4,"dilip");

insert into borrower values (5,"sammer");

**BOOK\_COPIES**

insert into bookcopies values (1,1,2);

insert into bookcopies values (2,2,2);

insert into bookcopies values (3,3,2);

insert into bookcopies values (4,4,2);

insert into bookcopies values (5,5,2);

insert into bookcopies values (6,1,2);

insert into bookcopies values (1,3,2);

insert into bookcopies values (1,5,2);

insert into bookcopies values (5,1,2);

insert into bookcopies values (4,1,2);

**BOOK\_LENDING**

insert into lending values (1,1,1,"2017-08-01","2017-08-10");

insert into lending values (2,1,1,"2017-08-01","2017-08-10");

insert into lending values (3,1,1,"2017-08-01","2017-02-10");

insert into lending values (4,1,1,"2017-08-01","2017-06-10");

insert into lending values (3,1,2,"2017-01-01","2017-08-10");

insert into lending values (1,1,3,"2017-01-01","2017-08-10");

insert into lending values (1,2,1,"2017-04-01","2017-04-10");

insert into lending values (3,1,4,"2017-08-01","2017-08-10");

insert into lending values (3,2,3,"2017-04-01","2017-03-10");

insert into lending values (4,2,3,"2017-05-01","2017-08-10");

insert into lending values (5,2,3,"2017-01-01","2017-05-10");

insert into lending values (6,1,4,"2017-01-01","2017-07-10");

insert into lending values (4,1,4,"2017-03-01","2017-08-10");

insert into lending values (5,1,4,"2017-04-01","2017-08-10");

insert into lending values (6,2,2,"2017-04-01","2017-08-10");

insert into lending values (4,4,2,"2017-01-01","2017-08-10");

insert into lending values (2,1,2,"2017-02-01","2017-08-10");

insert into lending values (5,1,2,"2017-03-01","2017-08-10");

insert into lending values (6,3,2,"2017-01-01","2017-08-10");

insert into lending values (2,4,4,"2017-04-01","2017-08-10");

insert into lending values (6,3,4,"2017-01-01","2017-08-10");

insert into lending values (2,4,6,"2017-01-01","2017-08-10");

Queries:

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

**SELECT**

**Distinct book.bookid,**

**book.title,**

**book.name,**

**bookcopies.noofcopies,**

**authors.name AS 'author name'**

**FROM**

**book,**

**bookcopies,**

**authors**

**WHERE**

**book.bookid = authors.bookid**

**AND bookcopies.bookid = book.bookid;**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| bookid | title | name | noofcopies | author name |
| 1 | learn html | Jaico | 2 | DEEPAK T |
| 2 | learn java | Jaico | 2 | GIRISH |
| 3 | learn python | Jaico | 2 | GANESH |
| 4 | learn anythin | Jaico | 2 | KARTIK KUMAR |
| 5 | c# | Jaico | 2 | KISHORE |
| 6 | linux | Jaico | 2 | MANISH |

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

SELECT

lending.cardno, name, COUNT(\*)

FROM

lending ,

borrower

WHERE

dateout BETWEEN '2017-01-01' AND '2017-06-30'

AND lending.cardno = borrower.cardno

GROUP BY lending.cardno

HAVING COUNT(\*) > 3;

|  |  |  |
| --- | --- | --- |
| cardno | name | COUNT(\*) |
| 2 | anisha | 6 |
| 3 | kiran | 4 |
| 4 | dilip | 5 |

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

DELETE FROM book

WHERE

bookid = 1;

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

create table book1

(

bookid int not null,

title varchar(20),

name varchar(20) ,

pubyear int,

primary key(bookid,pubyear)

) PARTITION BY RANGE (pubyear) (

PARTITION p0 VALUES LESS THAN (1991),

PARTITION p1 VALUES LESS THAN (1995),

PARTITION p2 VALUES LESS THAN (1999),

PARTITION p3 VALUES LESS THAN (2003),

PARTITION p4 VALUES LESS THAN (2007)

);

select \* from book1 partition(p1)

**SELECT** BOOK\_ID, TITLE, PUBLISHER\_NAME, PUB\_YEAR

**FROM** BOOK

**GROUP BY** PUB\_YEAR, BOOK\_ID, TITLE, PUBLISHER\_NAME;

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

Library.

CREATE VIEW noofcopiesview AS

SELECT

book.title, SUM(bookcopies.noofcopies)

FROM

book,

bookcopies,

authors

WHERE

book.bookid = bookcopies.bookid

GROUP BY book.bookid;

select \* from noofcopiesview ;

|  |  |
| --- | --- |
| title | SUM(bookcopies.noofcopies) |
| learn html | 36 |
| learn java | 12 |
| learn python | 12 |
| learn anythin | 24 |
| c# | 24 |
| linux | 12 |

**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.

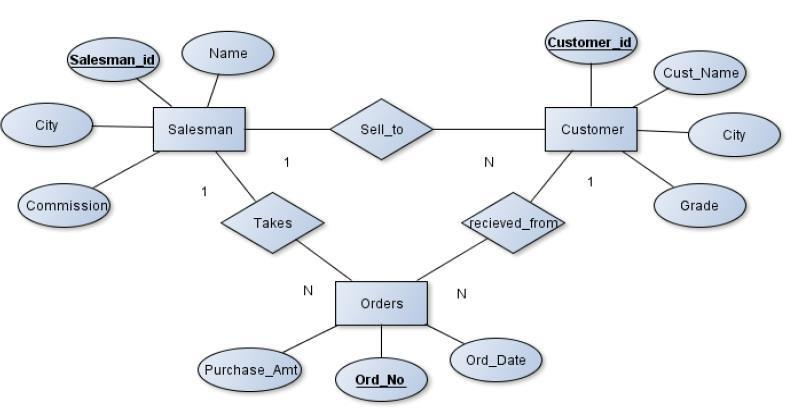
3. List all salesmen and indicate those who have and don’t have customers in their cities

(Use UNION operation.)

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

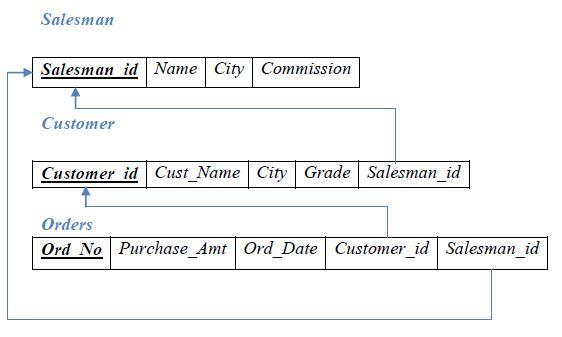
5. Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.

**ER-Diagram:**



**SCHEMA:**





**Table Creation:**

**SALESMAN**

**create table salesman**

**(**

**salesmanid varchar(10) not null,**

**name varchar(20),**

**city varchar(20) not null,**

**commission varchar(5),**

**primary key(salesmanid)**

**);**

**CUSTOMER**

**create table customer**

**(**

**customerid varchar(10) not null,**

**custname varchar(20) not null,**

**city varchar(20) not null,**

**grade int,**

**salesmanid varchar(10),**

**primary key(customerid),**

**foreign key(salesmanid) references salesman(salesmanid) on delete cascade**

**);**

**ORDERS**

create table orders

(

ordno varchar(10) not null,

amount int not null,

orddate date,

customerid varchar(10),

salesmanid varchar(10),

primary key(ordno),

foreign key(customerid) references customer(customerid) on delete cascade,

foreign key(salesmanid) references salesman(salesmanid) on delete cascade

);

**Values for tables**

**SALESMAN**

insert into salesman values('s1','Amir','Bangalore','10%');

insert into salesman values('s2','Daniel','Mysore','20%');

insert into salesman values('s3','Thomas','Mumbai','30%');

insert into salesman values('s4','Raghu','Pune','40%');

insert into salesman values('s5','Ram','Patna','50%');

insert into salesman values('s6','Raja','Goa','15%');

**CUSTOMER**

insert into salesman values('s7','Wadia','Lucknow','5%');

insert into customer values('c1','Athira','Bangalore',100,'s1');

insert into customer values('c2','Barkha','Mysore',200,'s2');

insert into customer values('c3','Tripthi','Mumbai',300,'s3');

insert into customer values('c4','Dia','Pune',400,'s4');

insert into customer values('c5','Deepu','Patna',500,'s5');

insert into customer values('c6','Eliza','Bangalore',100,'s1');

insert into customer values('c7','Clara','Bangalore',200,'s2');

insert into customer values('c8','Deena','Mysore',300,'s2');

insert into customer values('c9','Heena','Mumbai',400,'s3');

insert into customer values('c10','Prabhas','Mumbai',500,'s4');

insert into customer values('c11','Bala','Chennai',200,'s5');

insert into customer values('c12','chandra','Hyderabad',300,'s1');

insert into customer values('c13','Susan','Trivadurm',400,'s2');

insert into customer values('c14','priya','Goa',500,'s3');

insert into customer values('c15','Riya','Udupi',100,'s4');

insert into customer values('c16','Megha','Banaglore',100,'s5');

**ORDERS**

insert into orders values('o1',1000,'2017-08-01','c1','s1');

insert into orders values('o2',2000,'2017-08-02','c2','s2');

insert into orders values('o3',3000,'2017-08-03','c3','s3');

insert into orders values('o4',4000,'2017-08-04','c4','s4');

insert into orders values('o5',5000,'2017-08-05','c5','s5');

insert into orders values('o6',6000,'2017-08-01','c6','s1');

insert into orders values('o7',7000,'2017-08-06','c7','s2');

insert into orders values('o8',8000,'2017-08-06','c8','s3');

insert into orders values('o9',9000,'2017-08-07','c9','s4');

insert into orders values('o10',10000,'2013-08-08','c10','s5');

insert into orders values('o11',1500,'2017-08-01','c11','s1');

insert into orders values('o12',2500,'2017-06-01','c12','s2');

insert into orders values('o13',3500,'2017-03-01','c13','s3');

insert into orders values('o14',4500,'2017-02-01','c14','s4');

insert into orders values('o15',5500,'2017-01-01','c15','s5');

insert into orders values('o16',6500,'2017-05-01','c16','s6');

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

select count(customerid) from customer where grade > ( select avg(grade) from customer where city='Bangalore' );

|  |
| --- |
| count(customerid) |
| 12 |

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

**select s.name , s.salesmanid , count(\*) from customer c , salesman s where s.salesmanid=c.salesmanid group by c.salesmanid , s.name having count(\*) > 1;**

|  |  |  |
| --- | --- | --- |
| name | salesmanid | count(\*) |
| Amir | s1 | 3 |
| Daniel | s2 | 4 |
| Thomas | s3 | 3 |
| Raghu | s4 | 3 |
| Ram | s5 | 3 |

3. List all salesmen and indicate those who have and don’t have customers in their cities

(Use UNION operation.)

(**select distinct s.salesmanid , “same city” from salesman s , customer c where s.city = c.city and c.salesmanid=s.salesmanid)**

**union**

**(select distinct s.salesmanid “different city” from salesman s , customer c where s.city != c.city and c.salesmanid=s.salesmanid);**

|  |  |
| --- | --- |
| salesmanid | same city |
| s1 | same city |
| s2 | same city |
| s3 | same city |
| s4 | same city |
| s5 | same city |
| s1 | different city |
| s2 | different city |
| s3 | different city |
| s4 | different city |
| s5 | different city |

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

**create view highest\_orders as select b.orddate , a.salesmanid , a.name , b.customerid from salesman a , orders b where a.salesmanid = b.salesmanid and b.amount = (select max(amount) from orders c where c.orddate = b.orddate);**

**select \* from highest\_orders;**

|  |  |  |  |
| --- | --- | --- | --- |
| orddate | salesmanid | name | customerid |
| 8/1/2017 | s1 | Amir | c6 |
| 6/1/2017 | s2 | Daniel | c12 |
| 8/2/2017 | s2 | Daniel | c2 |
| 3/1/2017 | s3 | Thomas | c13 |
| 8/3/2017 | s3 | Thomas | c3 |
| 8/6/2017 | s3 | Thomas | c8 |
| 2/1/2017 | s4 | Raghu | c14 |
| 8/4/2017 | s4 | Raghu | c4 |
| 8/7/2017 | s4 | Raghu | c9 |
| 8/8/2013 | s5 | Ram | c10 |
| 1/1/2017 | s5 | Ram | c15 |
| 8/5/2017 | s5 | Ram | c5 |
| 5/1/2017 | s6 | Raja | c16 |

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

**delete from salesman where salesmanid = "s6" ;**

**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.

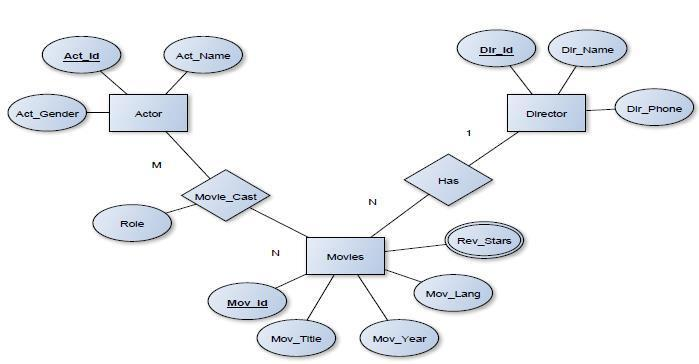
3. List all actors who acted in a movie before 2000 and also in a movie after

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.

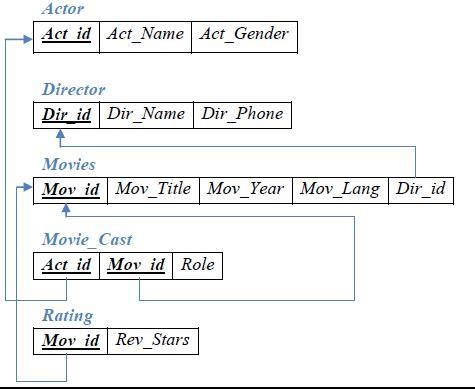
**ER-Diagram:**





**SCHEMA:**





**Table Creation:**

**ACTOR**

**create table actor**

**(**

**act\_id int,**

**act\_name varchar(25) ,**

**act\_gender char(1),**

**primary key(act\_id)**

**);**

**DIRECTOR**

**create table director**

**(**

**dir\_id int ,**

**dir\_name varchar(25) ,**

**dir\_phone int ,**

**primary key(dir\_id)**

**);**

**MOVIES**

**create table movies**

**(**

**mov\_id int,**

**mov\_title varchar(50),**

**mov\_year int not null,**

**mov\_lang varchar(15),**

**dir\_id int ,**

**primary key (mov\_id),**

**foreign key (dir\_id) references director(dir\_id) on delete cascade**

**);**

**MOVIE\_CAST**

**create table movie\_cast**

**(**

**act\_id INT ,**

**mov\_id int,**

**role varchar(45) ,**

**primary key(act\_id,mov\_Id),**

**foreign key (act\_id) references actor(act\_id) on delete cascade,**

**foreign key (mov\_id) references movies(mov\_id) on delete cascade**

**);**

**RATING**

**create table rating**

**(**

**mov\_id int not null,**

**rev\_stars int not null,**

**primary key (mov\_id),**

**foreign key (mov\_id) references movies(mov\_id) on delete cascade**

**);**

**Values for tables:**

**ACTOR**

insert into actor values('101','James','M');

insert into actor values('102','Deborah','F');

insert into actor values('103','Peter','M');

insert into actor values('104','Robert','M');

insert into actor values('105','Murray','M');

insert into actor values('106','Harrison','M');

insert into actor values('107','Nicole','F');

insert into actor values('108','Stephen','M');

insert into actor values('109','Jack','M');

insert into actor values('110','Kate','F');

**DIRECTOR**

insert into director values('201','Alfred','675409');

insert into director values('202','Jack','689543');

insert into director values('203','David','660908');

insert into director values('204','Michael','656432');

insert into director values('205','Milos','600944');

insert into director values('206','Stanley','677543');

insert into director values('207','Roman','660089');

**MOVIES**

insert into movies values('1','Vertigo','1994','English','201');

insert into movies values('2','Innocents','1997','English','201');

insert into movies values('3','Deer Hunter','1972','English','202');

insert into movies values('4','Eyes Wid Shut','2002','English','202');

insert into movies values('5','Wings','2016','English','203');

insert into movies values('6','Usual Suspects','2006','English','204');

insert into movies values('7','Samurai','2017','English','205');

insert into movies values('8','The Prestige','2016','English','206');

insert into movies values('9','American Beauty','2015','English','201');

insert into movies values('10','Walls','2000','English','207');

**MOVIE\_CAST**

insert into movie\_cast values('101','1','James');

insert into movie\_cast values('101','6','Fero');

insert into movie\_cast values('101','2','Eddie');

insert into movie\_cast values('102','5','July');

insert into movie\_cast values('103','3','John');

insert into movie\_cast values('104','7','Adam');

insert into movie\_cast values('105','8','Manus');

insert into movie\_cast values('106','1','Rick');

insert into movie\_cast values('107','8','Rose');

insert into movie\_cast values('107','9','Sam');

insert into movie\_cast values('108','1','Rock');

insert into movie\_cast values('108','5','Bobby');

insert into movie\_cast values('109','10','Ed');

insert into movie\_cast values('110','4','Cathie');

**RATING**

insert into rating values('1','3');

insert into rating values('2','3');

insert into rating values('3','2');

insert into rating values('4','5');

insert into rating values('5','4');

insert into rating values('6','3');

insert into rating values('7','2');

insert into rating values('8','1');

insert into rating values('9','5');

insert into rating values('10','4');

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="Alfred";**

|  |
| --- |
| mov\_title |
| Vertigo |
| Innocents |
| American Beauty |

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

**select mov\_title from movies where mov\_id in(select mov\_id from movie\_cast where act\_id in(select act\_id from movie\_cast group by act\_id having count(\*)>1));**

|  |
| --- |
| mov\_title |
| Vertigo |
| Innocents |
| Wings |
| Usual Suspects |
| The Prestige |
| American Beauty |

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 where act\_id in(select A.act\_id from(select act\_id from movie\_cast join movies on movie\_cast.mov\_id=movies.mov\_id where mov\_year<2000)A,(select act\_id from movie\_cast join movies on movie\_cast.mov\_id=movies.mov\_id where mov\_year>2015)B where A.act\_id=B.act\_id);**

|  |
| --- |
| **act\_name** |
| Stephen |

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 order by mov\_title;**

|  |  |
| --- | --- |
| mov\_title  1 | rev\_stars |
| American Beauty | 5 |
| Deer Hunter | 2 |
| Eyes Wid Shut | 5 |
| Innocents | 3 |
| Samurai | 2 |
| The Prestige | 1 |
| Usual Suspects | 3 |
| Vertigo | 3 |
| Walls | 4 |
| Wings | 4 |

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

update rating set rev\_stars=5 where mov\_id in (select m.mov\_id from movies m,director d where m.dir\_id=d.dir\_id and dir\_name='Michael');

select \* from rating;

|  |  |
| --- | --- |
| mov\_id | rev\_stars |
| 1 | 3 |
| 2 | 3 |
| 3 | 2 |
| 4 | 5 |
| 5 | 4 |
| 6 | 5 |
| 7 | 2 |
| 8 | 1 |
| 9 | 5 |
| 10 | 4 |

**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.

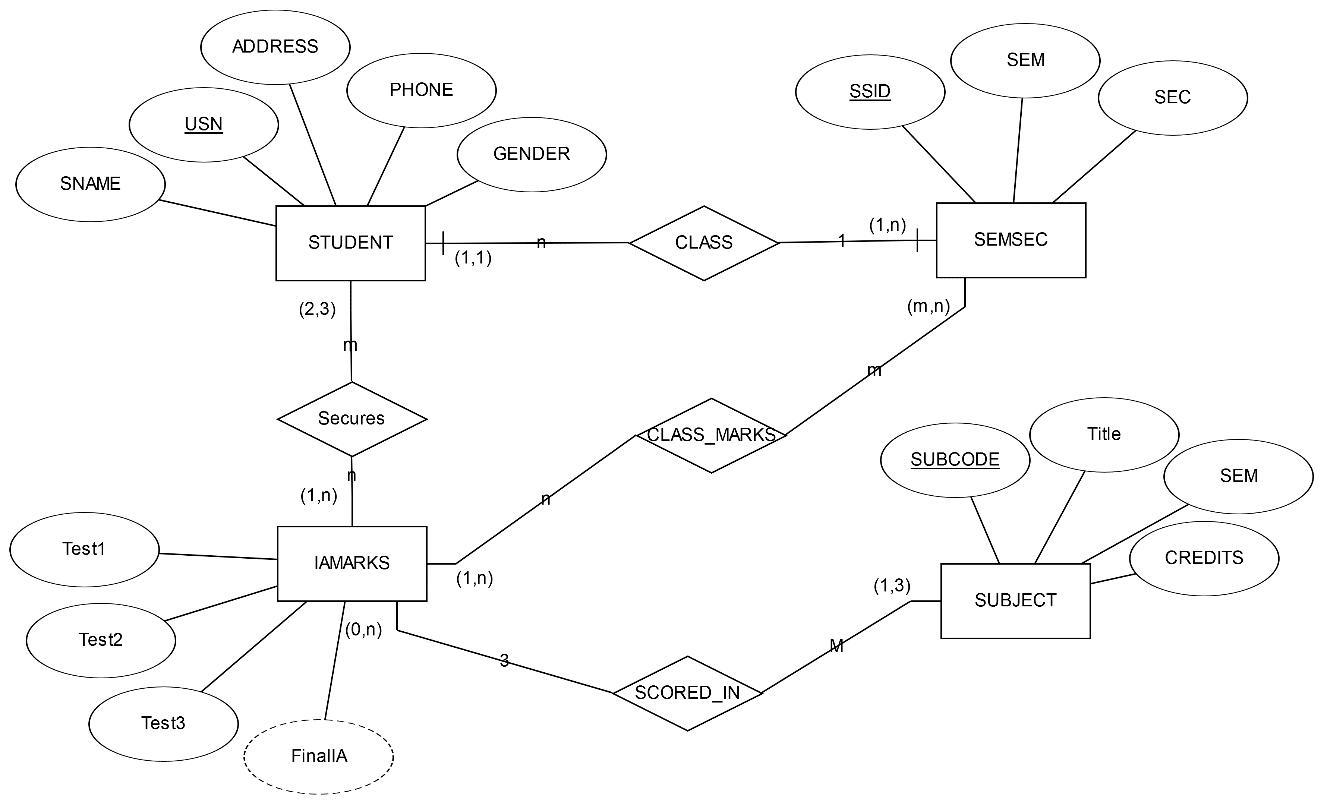
5. Categorize students based on the following criterion:

If FinalIA = 17 to 20 then CAT = ‘Outstanding’ If FinalIA = 12 to 16 then CAT = ‘Average’

If FinalIA< 12 then CAT = ‘Weak’

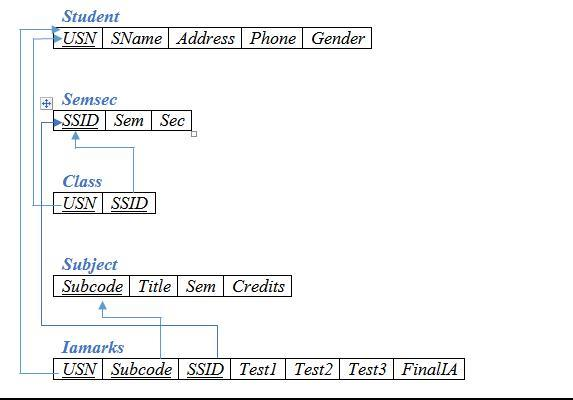
Give these details only for 8th semester A, B, and C section students.

**ER-Diagram:**



**SCHEMA:**





**Table Creation:**

**STUDENT**

CREATE TABLE students (

usn VARCHAR(20) ,

sname VARCHAR(20),

address VARCHAR(20),

phone VARCHAR(10),

gender CHAR(1),

PRIMARY KEY (usn)

);

**SEMSEC**

CREATE TABLE semsec (

ssid INT,

sem INT,

section CHAR(1),

PRIMARY KEY (ssid)

);

**CLASS**

CREATE TABLE class (

usn VARCHAR(20) PRIMARY KEY,

ssid INT,

FOREIGN KEY (ssid)

REFERENCES semsec (ssid)

ON DELETE CASCADE,

FOREIGN KEY (usn)

REFERENCES students (usn)

ON DELETE CASCADE

);

**SUBJECT**

CREATE TABLE sub (

subcode VARCHAR(20) PRIMARY KEY,

title VARCHAR(30),

sem INT,

credits INT

);

**IAMARKS**

CREATE TABLE iamarks (

usn VARCHAR(20),

subcode VARCHAR(20),

ssid INT,

test1 INT,

test2 INT,

test3 INT,

final INT,

PRIMARY KEY (usn , subcode , ssid),

FOREIGN KEY (usn)

REFERENCES students (usn)

ON DELETE CASCADE,

FOREIGN KEY (subcode)

REFERENCES sub (subcode)

ON DELETE CASCADE,

FOREIGN KEY (ssid)

REFERENCES semsec (ssid)

ON DELETE CASCADE

);

**Values for tables:**

**STUDENT:**

**insert into student (usn, sname, address, phone, gender) values**

**("1CR17IS01","Ajith","Bangalore",1010101010,'M'),**

**("1CR17IS02","Barat","Mysore",2020202020,'M'),**

**("1CR17IS03","Clara","Delhi",303030303,'F'),**

**("1CR17IS04","Denis","Pune",4040404040,'M'),**

**("1CR17IS05","Elisa","Patna",505050505,'F'),**

**("1CR17IS06","Franc","Mumbai",6060606060,'M'),**

**("1CR17IS07","Gaury","Bangalore",077777777,'F'),**

**("1CR17IS08","Harry","Kolkatta",88888870,'M'),**

**("1CR17IS09","Isaac","Dehradun",9090909090,'M'),**

**("1CR17IS010","Jay","Sikkim",111110000,'M');**

**insert into student(usn, sname, address, phone, gender) values**

**("1CR16IS41","Aman","Manipal",110110110,'M'),**

**("1CR16IS42","Barka","Hyderabad",1212121212,'F'),**

**("1CR16IS43","Chahana","Chennai",1313131313,'F'),**

**("1CR16IS44","Deepa","Coimbatore",1414141141,'F'),**

**("1CR16IS45","Emad","Mumbai",15151515,'M'),**

**("1CR16IS46","Faroq","Chennai",1616161616,'M'),**

**("1CR16IS47","Ganga","Coimbatore",17171717,'F'),**

**("1CR16IS48","Hemalatha","Hyderabad",1818181818,'F'),**

**("1CR16IS49","Inder","Bangalore",1919191919,'M'),**

**("1CR16IS50","Jamal","Patna",2202020202,'M');**

**insert into student(usn, sname, address, phone, gender) values**

**("1CR15IS61","Arti","Pune",2121212121,'F'),**

**("1CR15IS62","Barti","Delhi",2202202201,'F'),**

**("1CR15IS63","Ceaser","Mysore",212121212,'M'),**

**("1CR15IS64","Dia","Mysore",222222222,'F'),**

**("1CR15IS65","Elmsri","Bangalore",2323232323,'M'),**

**("1CR15IS67","Geetha","Bangalore",252525252,'F'),**

**("1CR15IS68","Harish","Bangalore",565656565,'M'),**

**("1CR15IS69","Iman","Bangalore",2929292929,'M'),**

**("1CR15IS70","Jibin","Mysore",990990909,'M'),**

**("1CR15IS71","Kruthi","Mysore",6767676767,'F'),**

**("1CR15IS72","Lara","Pune",3030303030,'F');**

**insert into student(usn, sname, address, phone, gender) values**

**("1CR14IS81", "Athiya", "Patna", 3131313131, 'F'),**

**("1CR14IS82", "Beema", "New Delhi", 3232323232, 'M'),**

**("1CR14IS83", "Chitra", "Pune",3333333333, 'F'),**

**("1CR14IS84", "Dipika", "Patna", 3434343434, 'F'),**

**("1CR14IS85", "Elizabeth", "Mumbai", 2353535355, 'F'),**

**("1CR14IS86", "Fakruddin", "Mumbai", 363663663, 'M'),**

**("1CR14IS87", "Gary", "Bangalore", 3737373737, 'M'),**

**("1CR14IS88", "Hema", "Kolkatta", 3838383838, 'M'),**

**("1CR14IS89", "Ishana", "Dehradun", 3939393939, 'F'),**

**("1CR14IS90", "Jason", "Manipal", 4040404040, 'M'),**

**("1CR14IS91", "Kirana", "Hyderabad", 4949494949, 'F'),**

**("1CR14IS92", "Lucky", "Chennai", 454545545, 'F');**

**SEMSEC:**

insert into semsec (ssid, sem, section) values (1,2,'A'), (2,2,'B'), (3,2,'C'), (4,4,'A'),

(5,4,'B'), (6,4,'C'), (7,6,'A'), (8,6,'B'), (9,6,'C'), (10,8,'A'), (11,8,'B'), (12,8,'C');

**CLASS:**

insert into class(usn,ssid) values ("1CR17IS01",'1'), ("1CR17IS02",'1'), ("1CR17IS03",'1'),

("1CR17IS04",'2'),("1CR17IS05",'2'),("1CR17IS06",'2'),

("1CR17IS07",'3'),("1CR17IS08",'3'),("1CR17IS09",'3');

insert into class(usn,ssid) values ("1CR16IS41",'4'), ("1CR16IS42",'4'), ("1CR16IS43",'4'),

("1CR16IS44",'4'),("1CR16IS45",'5'),("1CR16IS46",'5'),("1CR16IS47",'6'),("1CR16IS48"

,'6'),("1CR16IS49",'6'),("1CR16IS50",'6');

insert into class(usn,ssid) values ("1CR15IS61",'7'), ("1CR15IS62",'7'), ("1CR15IS63",'7'),

("1CR15IS64",'8'),("1CR15IS65",'8'),("1CR15IS67",9),("1CR15IS68",9),("1CR15IS69",9),("1CR15IS70",9),("1CR15IS71",9),("1CR15IS72",9),("1CR14IS81",10),("1CR14IS82",10),("1CR14IS83",10),("1CR14IS84",10),("1CR14IS85",11),("1CR14IS86",11),("1CR14IS87",11),("1CR14IS88",12),("1CR14IS89",12),("1CR14IS90",12),("1CR14IS91",12),("1CR14IS92",12);7 4

**SUBJECT:**

insert into sub(subcode,title,sem,credits) values ("15PCD23","PCD",2,4),

("15CHE21","CHEM",2,4), ("15ELN22","Basic Electronics",2,4),

("15MAT24","Maths",2,4), ("15CS42","SE",4,4), ("15CS43","DAA",4,4),

("15CS44","MP",4,4), ("15CS46","DC",4,4), ("15CS61","Cryptography",6,4),

("15CS62","CGV",6,4), ("15CS63","SS",6,4), ("15CS64","OS",6,4),

("15CS81","IOT",8,4), ("15CS82","Big Data Analytics",8,4), ("15CS834","SMS",8,4),

("15CS86","Seminar",8,4);

**IAMARKS:**

Insert into iamarks(usn,subcode,ssid,test1,test2,test3)values("1CR17IS01","15PCD23",1,10,12,14), ("1CR17IS01","15CHE21",1,11,12,13),("1CR17IS01","15ELN22",1,13,14,15),("1CR17IS01","15MAT24",1,16,17,18),("1CR16IS41","15CS42",4,19,20,19),("1CR16IS41","15CS43",4,20,20,20),("1CR16IS41","15CS44",4,7,9,10),("1CR16IS41","15CS46",4,10,15,20),("1CR15IS61","15CS61",7,8,12,16),("1CR15IS61","15CS62",7,9,13,17),("1CR15IS61","15CS63",7,10,14,18),("1CR15IS61","15CS64",7,11,15,19),("1CR14IS81","15CS81",10,16,14,20),("1CR14IS81","15CS82",10,20,12,13),("1CR14IS81","15CS834",10,15,16,20),("1CR14IS81","15CS86",10,20,19,18);

insert into iamarks(usn,subcode,ssid,test1,test2,test3)values("1CR14IS82","15CS81",10,12,15,18), ("1CR14IS82","15CS82",10,13,20,12),("1CR14IS82","15CS834",10,15,16,10),("1CR14IS82","15CS86",10,12,9,8);

insert into iamarks(usn,subcode,ssid,test1,test2,test3)values("1CR14IS83","15CS81",10,2,5,8),("1CR14IS83","15CS82",10,3,12,2),("1CR14IS83","15CS834",10,5,6,10),("1CR14IS83","15CS86",10,2,19,18);

insert into iamarks(usn,subcode,ssid,test1,test2,test3)values("1CR14IS84","15CS81",10,12,15,8),("1CR14IS84","15CS82",10,13,0,1),("1CR14IS84","15CS834",10,15,6,1),("1CR14IS84","15CS86",10,2,19,8);

insert into iamarks(usn,subcode,ssid,test1,test2,test3)values("1CR14IS85","15CS81",11,12,5,11),("1CR14IS85","15CS82",11,13,2,13),("1CR14IS85","15CS834",11,14,16,19),("1CR14IS85","15CS86",11,2,14,8),("1CR14IS86","15CS81",11,12,14,13),("1CR14IS86","15CS82",11,3,2,5),("1CR14IS86","15CS834",11,4,6,9),("1CR14IS86","15CS86",11,13,17,18),("1CR14IS87","15CS81",11,13,15,19),("1CR14IS87","15CS82",11,11,2,20),("1CR14IS87","15CS834",11,11,10,6),("1CR14IS87","15CS86",11,12,4,15),("1CR14IS88","15CS81",12,9,13,19),("1CR14IS88","15CS82",12,1,2,20),("1CR14IS88","15CS834",12,10,11,12),("1CR14IS88","15CS86",12,13,14,15),("1CR14IS89","15CS81",12,19,3,9),("1CR14IS89","15CS82",12,13,12,20),("1CR14IS89","15CS834",12,16,17,18),("1CR14IS89","15CS86",12,3,11,15),("1CR14IS90","15CS81",12,16,13,19),("1CR14IS90","15CS82",12,20,2,20),("1CR14IS90","15CS834",12,6,7,20),("1CR14IS90","15CS86",12,6,12,5),("1CR14IS91","15CS81",12,19,10,20),("1CR14IS91","15CS82",12,2,20,20),("1CR14IS91","15CS834",12,20,12,15),("1CR14IS91","15CS86",12,12,5,20),("1CR14IS92","15CS81",12,9,10,20),("1CR14IS92","15CS82",12,20,19,19),("1CR14IS92","15CS834",12,20,15,13),("1CR14IS92","15CS86",12,20,20,20);

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Queries:** |  |  |  |  |  |  |

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

SELECT

students.\*

FROM

students,

semsec,

class

WHERE

semsec.ssid = class.ssid

AND class.usn = students.usn

AND semsec.sem = 4

AND semsec.section = 'c'

GROUP BY students.usn;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Usn | Sname | Address | Phone | gender |
| 1CR16IS47 | Ganga | Coimbatore | 17171717 | F |
| 1CR16IS48 | Hemalatha | Hyderabad | 1818181818 | F |
| 1CR16IS49 | Inder | Bangalore | 1919191919 | M |
| 1CR16IS50 | Jamal | Patna | 2202020202 | M |

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

SELECT

semsec.sem, semsec.section, students.gender, COUNT(\*)

FROM

semsec,

class,

students

WHERE

semsec.ssid = class.ssid

AND student.usn = class.usn

GROUP BY semsec.sem , semsec.section , student.gender;

|  |  |  |  |
| --- | --- | --- | --- |
| Usn | Section | gender | Count(\*) |
| 2 | A | F | 1 |
| 2 | A | M | 2 |
| 2 | B | F | 1 |
| 2 | B | M | 2 |
| 2 | C | F | 1 |
| 2 | C | M | 2 |
| 4 | A | F | 3 |
| 4 | A | M | 1 |
| 4 | B | M | 2 |
| 4 | C | F | 2 |
| 4 | C | M | 2 |
| 6 | A | F | 2 |
| 6 | A | M | 1 |
| 6 | B | F | 1 |
| 6 | B | M | 1 |
| 6 | C | F | 3 |
| 6 | C | M | 3 |
| 8 | A | F | 3 |
| 8 | A | M | 1 |
| 8 | B | F | 1 |
| 8 | B | M | 2 |
| 8 | C | F | 3 |
| 8 | C | M | 2 |

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

CREATE VIEW v2 AS

SELECT

subcode, test1

FROM

iamarks

WHERE

iamarks.usn = '1CR14IS81';

Selcet \* from V2

|  |  |
| --- | --- |
| subcode | test1 |
| 15CS81 | 12 |
| 15CS82 | 13 |
| 15CS834 | 14 |
| 15CS86 | 2 |

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

Trigger should created before inserting the values

CREATE TRIGGER `finaliamarks`

BEFORE INSERT

ON `iamarks`

FOR EACH ROW

set new.final=(((new.test1+new.test2+new.test3) -least(new.test1,new.test2,new.test3))/2)

or

CREATE VIEW finaliamarks AS

(SELECT

usn,ssid,

subcode,

GREATEST(AVG(test1 + test2) / 2,

AVG(test1 + test3) / 2,

AVG(test2 + test3) / 2) AS finalia

FROM

iamarks

GROUP BY usn , ssid,subcode);

UPDATE iamarks i set final=(SELECT finalia from finaliamarks f where i.usn=f.usn and

i.subcode=f.subcode and i.ssid=f.ssid);

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

i.usn,

s.sem,

s.section,

i.subcode,

CASE

WHEN final >= 17 AND final <= 20 THEN 'Outstanding'

WHEN final >= 12 AND final <= 16 THEN 'Average'

ELSE 'Weak'

END AS CAT

FROM

iamarks i,

class c,

semsec s

WHERE

i.usn = c.usn AND c.ssid = s.ssid

AND s.sem = 8

AND s.section IN ('A' , 'B', 'C');

**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.

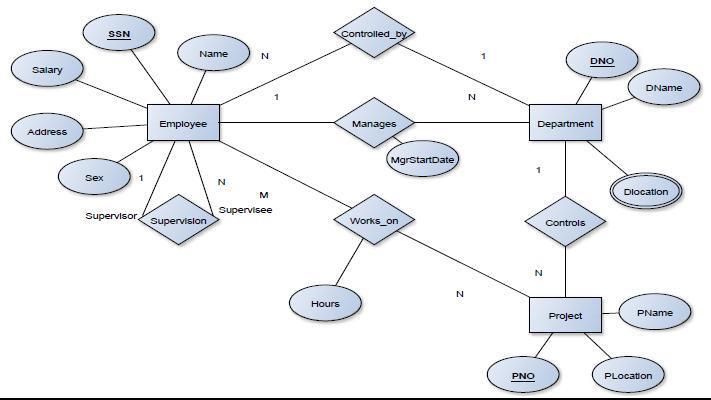
2. Show the resulting salaries if every employee working on the ‘IoT’ project is given a 10 percent raise.

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).

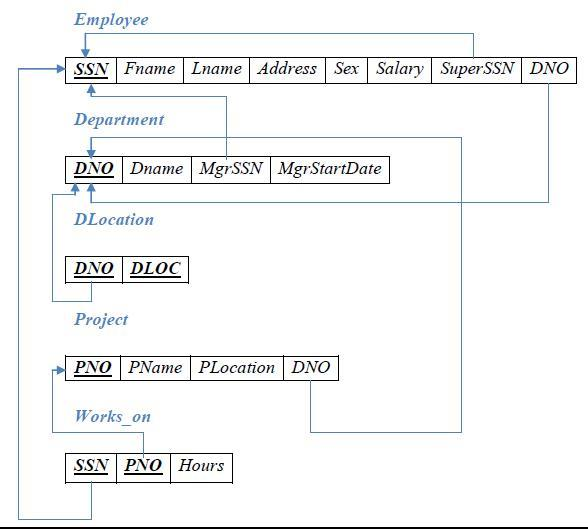
5. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs. 6,00,000.

**ER-Diagram:**



**SCHEMA:**





**Table Creation:**

**EMPLOYEE**

CREATE TABLE Employee (

ssn INT ,

name varchar(30),

address VARCHAR(25),

sex VARCHAR(10),

salary INT,

superssn INT,

dno INT,

PRIMARY KEY (ssn),

FOREIGN KEY (superssn)

REFERENCES Employee (ssn)

);

**DEPARTMENT**

CREATE TABLE department (

dno INT,

dname VARCHAR(25),

mgrssn INT,

mgrstrtdate DATE,

PRIMARY KEY (dno),

FOREIGN KEY (mgrssn)

REFERENCES Employee (ssn) on delete cascade

);

alter table Employee add FOREIGN key (dno) REFERENCES department (dno);

**DLOCATION**

CREATE TABLE dlocation (

dno INT NOT NULL,

dloc VARCHAR(25) NOT NULL,

PRIMARY KEY (dno , dloc),

FOREIGN KEY (dno)

REFERENCES department (dno)

);

**PROJECT**

CREATE TABLE project (

pno INT NOT NULL,

pname VARCHAR(25),

plocation VARCHAR(25),

dno INT,

PRIMARY KEY (pno),

FOREIGN KEY (dno)

REFERENCES department (dno)

);

**WORKS\_ON**

create table works\_on (

SSN int ,

PNO int ,

HOURS int ,

Primary key(SSN,PNO),

foreign key (ssn) references Employee(ssn) on delete cascade ,

foreign key (pno) references project(pno) on delete cascade

);

**Values for tables:**

**DEPARTMENT**

Insert into department(dno,dname) values(1,"Account");

Insert into department(dno,dname) values(2,"Management");

Insert into department(dno,dname) values(3,"Marketing");

Insert into department(dno,dname) values(4,"sales");

Insert into department(dno,dname) values(5,"IT");

Insert into department(dno,dname) values(6,"Research");

**EMPLOYEE**

INSERT INTO `Employee` (`SSN`, `NAME`, `ADDRESS`, `SEX`, `SALARY`, `DNO`) VALUES ('23412356', 'JENNIFER', 'PARIS', 'FEMALE','700000', '4');

INSERT INTO `Employee` (`SSN`, `NAME`, `ADDRESS`, `SEX`, `SALARY`, `DNO`) VALUES ('23456781', 'JAMES', 'NEWYORK', 'MALE','300000', '4');

INSERT INTO `Employee` (`SSN`, `NAME`, `ADDRESS`, `SEX`, `SALARY`, `DNO`) VALUES ('33344555', 'FRANKLIN', 'CALIFORNIA','MALE', '600000', '5');

INSERT INTO `Employee` (`SSN`, `NAME`, `ADDRESS`, `SEX`, `SALARY`, `DNO`) VALUES ('67891234', 'JOYCE', 'WASHINGTON','FEMALE', '400000', '5');

INSERT INTO `Employee` (`SSN`, `NAME`, `ADDRESS`, `SEX`, `SALARY`, `DNO`) VALUES ('123456789', 'JOHN', 'TEXAS', 'MALE','300000', '5');

INSERT INTO `Employee` (`SSN`, `NAME`, `ADDRESS`, `SEX`, `SALARY`, `DNO`) VALUES ('888666555', 'AHMAD', 'CALIFORNIA','MALE', '700000', '4');

INSERT INTO `Employee` (`SSN`, `NAME`, `ADDRESS`, `SEX`, `SALARY`, `DNO`) VALUES ('984600445', 'MARK', 'WASHINGTON','MALE', '800000', '5');

INSERT INTO `Employee` (`SSN`, `NAME`, `ADDRESS`, `SEX`, `SALARY`, `DNO`) VALUES ('56789012', 'ALEENA', 'LONDON', 'FEMALE', '700000', '5');

INSERT INTO `Employee` (`SSN`, `NAME`, `ADDRESS`, `SEX`, `SALARY`, `DNO`) VALUES ('89012345', 'ALICE', 'STAFFORD', 'FEMALE', '1200000', '5');

INSERT INTO `Employee` (`SSN`, `NAME`, `ADDRESS`, `SEX`, `SALARY`, `DNO`) VALUES ('1111111', 'ALI', 'PARIS', 'MALE','700000', '1');

UPDATE `Employee` SET `superssn` = '984600445' WHERE `ssn` = 23412356;

UPDATE `Employee` SET `superssn` = '67891234' WHERE `ssn` = 23456781;

UPDATE `Employee` SET `superssn` = '123456789' WHERE `ssn` = 33344555;

UPDATE `Employee` SET `superssn` = '89012345' WHERE `ssn` = 56789012;

UPDATE `Employee` SET `superssn` = '888666555' WHERE `ssn` = 67891234;

UPDATE `Employee` SET `superssn` = '56789012' WHERE `ssn` = 89012345;

UPDATE `Employee` SET `superssn` = '33344555' WHERE `ssn` = 123456789;

UPDATE `Employee` SET `superssn` = '23456781' WHERE `ssn` = 888666555;

UPDATE `department` SET `mgrssn` = '888666555' WHERE `dno` = 1;

UPDATE `department` SET `mgrssn` = '123456789' WHERE `dno` = 4;

UPDATE `department` SET `mgrssn` = '33344555' WHERE `dno` = 5;

**DLOCATION**

INSERT INTO `dlocation` (`DNO`, `DLOC`) VALUES ('1', 'WASHINGTON');

INSERT INTO `dlocation` (`DNO`, `DLOC`) VALUES ('4', 'CALIFORNIA');

INSERT INTO `dlocation` (`DNO`, `DLOC`) VALUES ('5', 'NEW YORK');

INSERT INTO `dlocation` (`DNO`, `DLOC`) VALUES ('5', 'WASHINGTON');

**PROJECT**

INSERT INTO project (PNAME, PNO, PLOCATION, DNO) VALUES ('PRODUCTA', 1,'HOUSTON', 5);

INSERT INTO project (`PNAME`, `PNO`, `PLOCATION`, `DNO`) VALUES('PRODUCTB', 2, 'WASHINGTON', 5);

INSERT INTO project (`PNAME`, `PNO`, `PLOCATION`, `DNO`) VALUES('PRODUCTC', 3, 'CALIFORNIA', 5);

INSERT INTO project (`PNAME`, `PNO`, `PLOCATION`, `DNO`) VALUES('COMPUTERIZATION',10,'NEW YORK',4);

INSERT INTO project (`PNAME`, `PNO`, `PLOCATION`, `DNO`) VALUES('IOT',20,'PARIS',1);

INSERT INTO project (`PNAME`, `PNO`, `PLOCATION`, `DNO`) VALUES('REORGANIZATION',30,'STAFFORD',4);

**WORKS\_ON**

INSERT INTO `works\_on` (`SSN`, `PNO`, `HOURS`) VALUES ('123456789', '1', '33');

INSERT INTO `works\_on`(`SSN`, `PNO`, `HOURS`) VALUES ('984600445', '1', '50');

INSERT INTO `works\_on` (`SSN`, `PNO`, `HOURS`) VALUES ('888666555', '2', '54');

INSERT INTO `works\_on` (`SSN`, `PNO`, `HOURS`) VALUES ('33344555', '20', '10');

INSERT INTO `works\_on` (`SSN`, `PNO`, `HOURS`) VALUES ('67891234', '10', '20');

INSERT INTO `works\_on` (`SSN`, `PNO`, `HOURS`) VALUES ('23412356', '20', '14');

INSERT INTO `works\_on` (`SSN`, `PNO`, `HOURS`) VALUES ('23456781', '1', '34');

INSERT INTO `works\_on` (`SSN`, `PNO`, `HOURS`) VALUES ('1111111', '1', '33');

INSERT INTO `works\_on` (`SSN`, `PNO`, `HOURS`) VALUES ('1111111', '2', '33');

INSERT INTO `works\_on` (`SSN`, `PNO`, `HOURS`) VALUES ('1111111', '3', '33');

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 p.dno=d.dno and d.mgrssn=e.ssn and name="JENNIFER" union select distinct p.pno from project p, works\_on w,Employee e where w.pno=p.pno and w.ssn=e.ssn and name="JENNIFER";

2. Show the resulting salaries if every employee working on the ‘IoT’ project is given a 10

percent raise.

select name, 0.1\*salary from Employee e, works\_on w, project p where e.ssn=w.ssn and p.pno=w.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(salary) as SUM, max(salary) as MAX, min(salary) as MIN, avg(salary) as AVG from Employee e, department d where e.dno=d.dno and dname="sales";

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

-

SELECT

e.name

FROM

Employee e

WHERE

NOT EXISTS( SELECT DISTINCT

\*

FROM

project p,

works\_on w

WHERE

w.pno = p.pno AND p.dno = 5

AND NOT EXISTS( SELECT

\*

FROM

works\_on w1

WHERE

e.ssn = w1.ssn AND w1.pno = w.pno));

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(\*) as No\_Of\_Employees from department d, Employee e where d.dno=e.dno and e.salary>600000 and e.dno in (select dno from Employee group by dno having count(\*)>5) group by d.dno;

DNO COUNT(SSN)

--------- ----------



3 4