BALLARI INSTITUTE OF TECHNOLOGY & MANAGEMENT

Jnana Gangotri" Campus,Ballari-Hosapete Road,Near Allipura, Ballari,Karnataka- 583104

Phone: 08392-237167 / 237190 **Mobile:**+91 99024 99388 **E-mail id:**bitmbly@gmail.com

Department of AIML

Academic Year 2022-23

Subject Name : Database Management System Laboratory with Mini Project

Subject Code : 18CSL58

Semester : V

Section : A & B

Prepared on	Prepared by the faculty	Verified by (Stream Coordinator)
10/10/2022	Phani Ram Prasad	**(Dr. Aradhana D)**
	Pratibha Mishra	
	Md Shafiulla	
	Kiran M	
	Sheetal J	
	Syeda Badrunissa Begum	

INDEX

Slno	List of Experiment	Page no
1.	Introduction to SQL	1-18
2.	Library Database	19-28
3.	Order Database	29-35
4.	Movie Database	36-42
5.	College Database	43-52
6.	Company Database	53-60

INTRODUCTION TO SQL

Introduction to SQL

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

SQL Commands

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

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

- **Data Control Language (DCL)** - These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

Data Definition Language (DDL)

CREATE TABLE Statement

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

The Syntax for the CREATE TABLE Statement is:

```
CREATE TABLE table_name
(column_name1 datatype constraint,
column_name2 datatype,...
column_nameNdatatype);
```

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

SQL Data Types:

char(size)	Fixed-length character string. Size is specified in parenthesis. Max 255 bytes.
Varchar2(size)	Variable-length character string. Max size is specified in parenthesis.
number(size) or int	Number value with a max number of column digits specified in parenthesis.
Date	Date value in 'dd-mon-yy'. Eg., '07-jul-2004'
number(size,d) or real	Number value with a maximum number of digits of "size" total, with a maximum number of "d" digits to the right of the decimal.

SQL Integrity Constraints:

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

Constraints can be defined in two ways:

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

1) Primary key:

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

Syntax to define a Primary key at column level:

```
Column_namedatatype [CONSTRAINT constraint_name] PRIMARY KEY
```

Syntax to define a Primary key at table level:

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

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

2) Foreign key or Referential Integrity:

This constraint identifies any column referencing the PRIMARY KEY in another table. It establishes a relationship between two columns in the same table or between

different tables. For a column to be defined as a Foreign Key, it should be a defined as a Primary Key in the table which it is referring. One or more columns can be defined as Foreign key.

Syntax to define a Foreign key at column level:

```
[CONSTRAINT constraint_name] REFERENCES
```

```
referenced_table_name(column_name)
```

Syntax to define a Foreign key at table level:

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

```
referenced_table_name(column_name);
```

3) Not Null Constraint:

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

Syntax to define a Not Null constraint:

```
[CONSTRAINT constraint name] NOT NULL
```

4) Unique Key:

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

Syntax to define a Unique key at column level:

```
[CONSTRAINT constraint_name] UNIQUE
```

Syntax to define a Unique key at table level:

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

5) Check Constraint:

This constraint defines a business rule on a column. All the rows must satisfy this

rule. The constraint can be applied for a single column or a group of columns.

Syntax to define a Check constraint:

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

ALTER TABLE Statement

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

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

Syntax to add a column

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

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

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

Syntax to drop a column

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

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

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

Syntax to modify a column

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

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

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

Syntax to add PRIMARY KEY constraint

```
ALTER TABLE table_nameADD CONSTRAINT constraint_name PRIMARY KEY
column_name;
```

Syntax to drop PRIMARY KEY constraint

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

The DROP TABLE Statement

DROP TABLE statement is used to delete a table.

DROP TABLE table_name;

TRUNCATE TABLE Statement

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

Then, use the TRUNCATE TABLE statement:

TRUNCATE TABLE table_name;

Data Manipulation Language (DML):

The SELECT Statement

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

SELECT Syntax:

SELECT * FROM table_name;

The SELECT DISTINCT Statement

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

SELECT DISTINCT Syntax:

SELECT DISTINCT column_name(s)
FROM table_name;

The WHERE Clause

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

WHERE Syntax:

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

The AND & OR Operators

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

The ORDER BY Clause

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

ORDER BY Syntax:

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

The GROUP BY Clause

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

GROUP BY Syntax;

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

WHERE column_name operator value

GROUP BY column_name(s);

Group functions	Meaning
AVG([DISTINCT|ALL],N])	Returns average value of n
COUNT(*|[DISTINCT|ALL]expr)	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;
```

Transaction Control language(TCL)

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;

Data Control Language (DCL)

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

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

DCL defines two commands,

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

To Allow a User to create Session

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

To Allow a User to create Table

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

To provide User with some Space on Tablespace to store Table

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

To Grant all privilege to a User

```
grant sysdba to username
```

To Grant permission to Create any Table

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

STORED PROCEDURES in SQL:

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

Syntax

Following is the basic syntax of Stored procedure creation.

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

Example

Consider the CUSTOMERS table having the following records.

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.

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

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.

```
Select * from customers;

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

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

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
```

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.

EXPERIMENT 1: LIBRARY DATABASE

OBJECTIVES:

- **To create and insert values into a table**
- **To understand the working of operators (between, and, not etc).**
- **To create a view for a given table.**

1. ***Consider the following schema for a Library Database:***

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

BOOK_AUTHORS**(**Book_id, Author_Name)

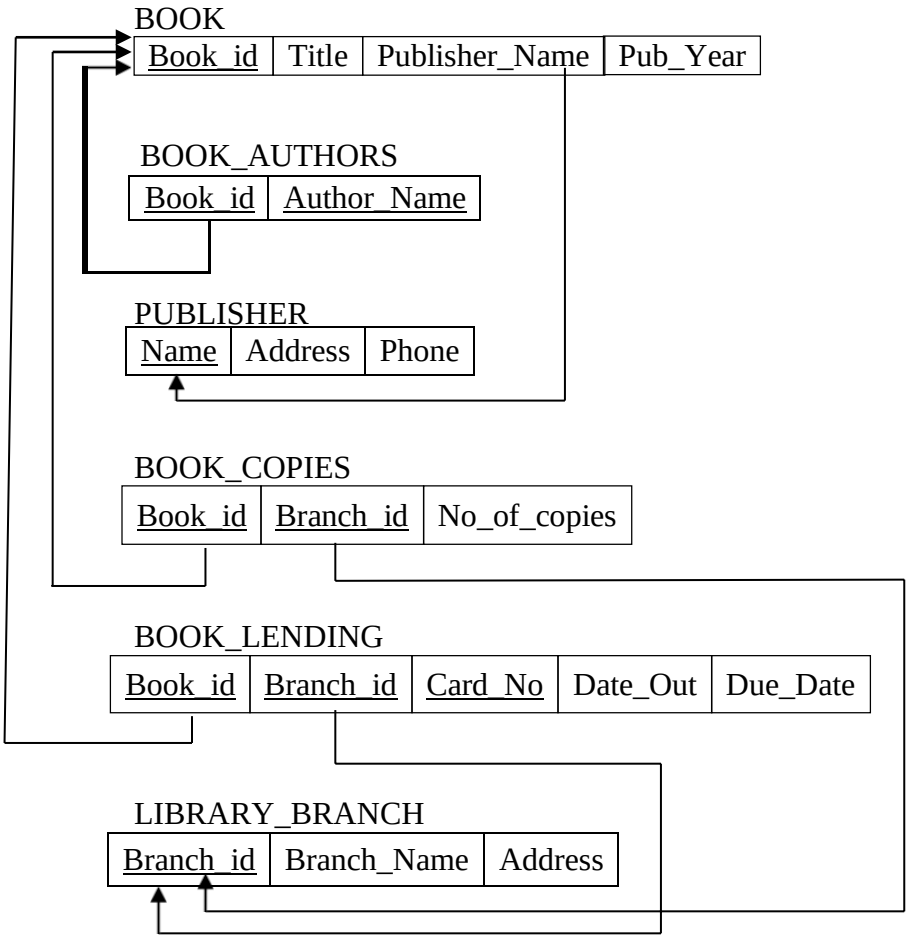
PUBLISHER**(**Name, Address, Phone)

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)

Step 1: SCHEMA DIAGRAM



STEP 2: ER DIAGRAM

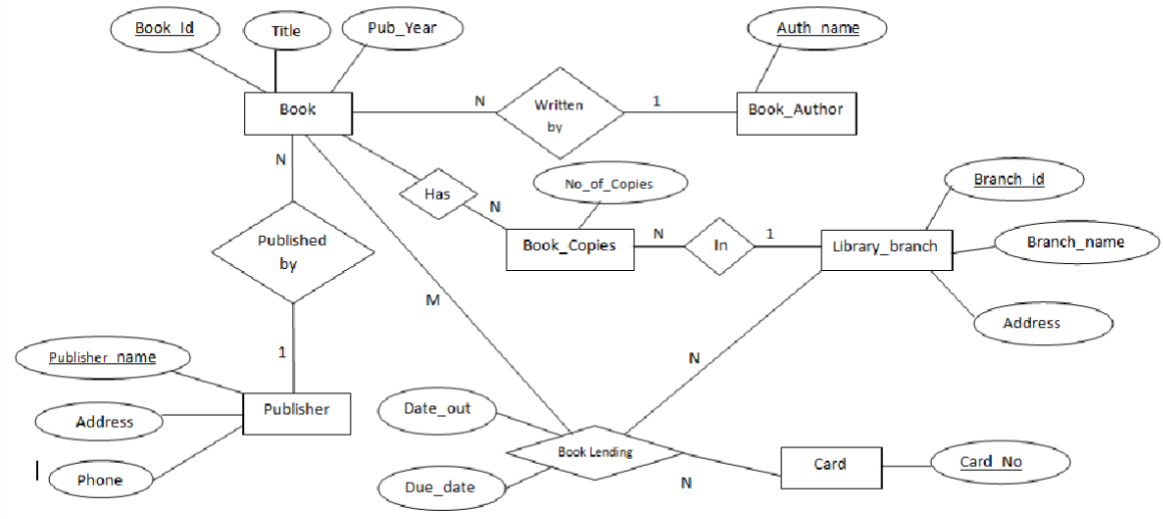


Table Creation

SQL> **CREATE TABLE** Publisher

(Pub_name VARCHAR(20) PRIMARY KEY ,

Address VARCHAR(20),

Phone NUMBER(10)

);

SQL> **INSERT INTO** Publisher **VALUES(**'sunstar' , 'mangalore', '987654302'**);**

SQL> **INSERT INTO** Publisher **VALUES(**'pearson' , 'bangalore', '987612345'**);**

SQL> **INSERT INTO** Publisher **VALUES(**'wiley','new delhi','654802'**);**

SQL> **INSERT INTO** Publisher **VALUES(**'s chand','bangalore','23465765'**);**

SQL> **INSERT INTO** Publisher **VALUES(**'mcgraw hills','mumbai','54884698'**);**

SQL> **INSERT INTO** Publisher **VALUES(**'mit press','cochin','65465463'**);**

SQL> **INSERT INTO** Publisher **VALUES(**'princeton','kolkata','54565436'**);**

SQL> **SELECT** * **FROM** Publisher;

Pub_name	Address	Phone
McGraw Hills	Mumbai	54884698
MIT Press	Cochin	65465463
Pearson	Bangalore	987612345
Princeton	Kolkata	54565436
S Chand	Bangalore	23465765
Sunstar	Mangalore	987654302
Wiley	New Delhi	654802

SQL> **CREATE TABLE** Book

(Book_id NUMBER(10) PRIMARY KEY NOT NULL,

Title VARCHAR(20),

Publisher_name VARCHAR(20),

Pub_year NUMBER(10),

FOREIGN KEY (Publisher_name) REFERENCES publisher(Pub_name));

SQL> **INSERT INTO** Book **VALUES(**911,'The Lost Tribe','Wiley',2009**);**

SQL> **INSERT INTO** Book **VALUES(**912,'Database Systems','Princeton',2013**);**

SQL> **INSERT INTO** Book **VALUES(**913,'Nanotechnology','MIT Press',2001**);**

SQL> **INSERT INTO** Book **VALUES(**914,'Vampire Academy','S Chand',2005**);**

SQL> **INSERT INTO** Book **VALUES(**915,'Operating Systems','Wiley',2017**);**

SQL> **INSERT INTO** Book **VALUES(**916,'Nano Physics','Sunstar',2001**);**

SQL> **SELECT * FROM** Book;

Book_id	**Title**	**Publisher_name**	**Pub_year**
911	The Lost Tribe	Wiley	2009
912	Database Systems	Princeton	2013
913	Nanotechnology	MIT Press	2001
914	Vampire Academy	S Chand	2005
915	Operating Systems	Wiley	2017
916	Nano Physics	Sunstar	2001

SQL> **CREATE TABLE** Book_Authors

(Book_id NUMBER(10),

Author_nameVARCHAR(20),

PRIMARY KEY(Book_id,Author_name),

FOREIGN KEY(Book_id) REFERENCES Book(Book_id) ON DELETE CASCADE);

SQL> **INSERT INTO** Book_Authors **VALUES(**911,'Girish'**);**

SQL> **INSERT INTO** Book_Authors **VALUES(**912,'Prem Juneja'**);**

SQL> **INSERT INTO** Book_Authors **VALUES(**913,'Michelle Read'**);**

SQL> **INSERT INTO** Book_Authors **VALUES(**914,'Stephen King'**);**

SQL> **INSERT INTO** Book_Authors **VALUES(**915,'Dev'**);**

SQL> **INSERT INTO** Book_Authors **VALUES(**916,'Shashank'**);**

SQL> **SELECT** * **FROM** Book_Authors;

Book_id	**Author_name**
911	Girish
912	Prem Juneja
913	Michelle Read
914	Stephen King
915	Dev
916	Shashank

SQL> **CREATE TABLE** Library_Branch

(Branch_id NUMBER(10) PRIMARY KEY,

Branch_name VARCHAR(20) NOT NULL,

Address VARCHAR(20)

);

SQL> **INSERT INTO** Library_Branch **VALUES(**1,'Book Park','Sanjaynagar'**);**

SQL> **INSERT INTO** Library_Branch **VALUES(**2,'Vivekanandha','Gopalapuram'**);**

SQL> **INSERT INTO** Library_Branch **VALUES(**4,'Patel','vivek nagar**);**

SQL> **INSERT INTO** Library_Branch **VALUES(**5,'Sharpstown','5 krishna'**);**

SQL> **INSERT INTO** Library_Branch **VALUES(**9,'Central','Gandhipuram'**);**

SQL> **SELECT** * **FROM** Library_Branch;

Branch_id	**Branch_name**	**Address**
1	Book Park	3 Cross Street
2	Vivekanandha	Gopalapuram
4	Patel	4 Marine Drive
5	Sharpstown	5 section
9	Central	Gandhipuram

SQL> **CREATE TABLE** Book_Copies

(Book_id NUMBER(10),

Branch_id NUMBER(10),

No_of_copies NUMBER(10),

PRIMARY KEY (Book_id,Branch_id),

FOREIGN KEY(Book_id) REFERENCES Book(Book_id) ON DELETE CASCADE,

FOREIGN KEY(Branch_id) REFERENCES Library_Branch(Branch_id) ON DELETE CASCADE);

SQL> **INSERT INTO** Book_Copies **VALUES(**911,1,3**);**

SQL> **INSERT INTO** Book_Copies **VALUES(**912,2,4**);**

SQL> **INSERT INTO** Book_Copies **VALUES(**913,4,2**);**

SQL> **INSERT INTO** Book_Copies **VALUES(**914,5,5**);**

SQL> **INSERT INTO** Book_Copies **VALUES(**915,9,7**);**

SQL> **INSERT INTO** Book_Copies **VALUES(**916,1,4**);**

SQL> **INSERT INTO** Book_Copies **VALUES(**911,9,5**);**

SQL> **SELECT * FROM** Book_Copies;

Book_id	Branch_id	No_of_copies
911	1	3
911	9	5
912	2	4
913	4	2
914	5	5
915	9	7
916	1	4

SQL> **CREATE TABLE** Book_Lending

(Book_id NUMBER(10),

Branch_id NUMBER(10),

Card_no NUMBER(10),

Date_out DATE,

Due_date DATE,

PRIMARY KEY(Book_id, Branch_id, Card_no),

FOREIGN KEY(Book_id, Branch_id) REFERENCES Book_Copies(Book_id, Branch_id) ON DELETE CASCADE);

SQL> **INSERT INTO** Book_Lending **VALUES(**911,1,1001,'1-jan-2017','1-may-2017'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**912,2,1002,'11-apr-2017','19-may-2017'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**913,4,1003,'1-jan-2009','1-apr-2009'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**914,5,1004,'11-feb-2001','9-may-2001'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**915,9,1005,'9-jun-2013','21-nov-2013'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**916,1,1009,'18-jan-2017','1-may-2017'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**911,1,1009,'18-jan-2017','1-may-2017'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**912,2,1009,'18-apr-2017','1-may-2017'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**911,1,1002,'9-jan-2017','1-jun-2017'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**911,9,1005,'18-jan-2017','1-may-2017'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**914,5,1009,'18-jan-2017','1-may-2017'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**914,5,1001,'18-jan-2017','1-may-2017'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**916,1,1001,'18-apr-2017','1-may-2017'**);**

SQL> **INSERT INTO** Book_Lending **VALUES(**913,4,1001,'9-jan-2017','1-jun-2017'**);**

SQL> **SELECT * FROM** Book_Lending;

Book_id	Branch_id	Card_no	Date_out	Due_date
911	1	1001	01.01.2017	01.05.2017
912	2	1002	11.04.2017	19.05.2017
913	4	1003	01.01.2009	01.04.2009
914	5	1004	11.02.2001	09.05.2001

915	9	1005	09.06.2013	21.11.2013
916	1	1009	18.01.2017	01.05.2017

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

```
SQL> SELECT B.Book_id, B.title, B.Publisher_name, A.Author_name, L.Branch_name,
        C.No_of_copies
     FROM   Book B, Book_Authors A, Book_Copies C, Library_Branch L
     WHERE  B.Book_id=A.Book_id AND A.Book_id=C.Book_id AND
        C.Branch_id=L.Branch_id;
```

Book_id	Title	Publisher_name	Author_name	Branch_name	No_of_copies
911	The Lost Tribe	Wiley	Girish	Book Park	3
911	The Lost Tribe	Wiley	Girish	Central	5
912	Database Systems	Princeton	Prem Juneja	Vivekanandha	4
913	Nanotechnology	MIT Press	Michelle Read	Patel	2
914	Vampire Academy	S Chand	Stephen King	Sharpstown	5
915	Operating Systems	Wiley	Dev	Central	7
916	Nano Physics	Sunstar	Shashank	Book Park	4

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

```
SQL> SELECT     Card_no AS CARD_NUM
     FROM       Book_Lending
     WHERE      Date_out BETWEEN '2017-01-01' AND '2017-06-30'
     GROUP BY   Card_no
     HAVING     COUNT(Book_id)>3;
```

CARD_NUM
1001
1009

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

SQL> **DELETE**

FROM Book

WHERE Book_id='916';

SQL> **SELECT** * **FROM** Book;

Book_id	Title	Publisher_name	Pub_year
911	The Lost Tribe	Wiley	2009
912	Database Systems	Princeton	2013
913	Nanotechnology	MIT Press	2001
914	Vampire Academy	S Chand	2005
915	Operating Systems	Wiley	2017

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

SQL> **CREATE VIEW** Ppublication AS
SELECT PUB_YEAR
FROM BOOK;

SQL> **SELECT** * **FROM** Ppublication;

PUB_YEAR
2009
2013
2001
2005
2017

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

SQL> **CREATE VIEW** Book_Copies_View **AS SELECT** B.Title, C.Branch_id,

C.No_of_copies

FROM Book B, Book_Copies C

WHERE B.Book_id=C.Book_id;

SQL> **SELECT** * **FROM** Book_Copies_View;

Title	**Branch_id**	**No_of_copies**
The Lost Tribe	1	3
The Lost Tribe	9	5
Database Systems	2	4
Nanotechnology	4	2
Vampire Academy	5	5
Operating Systems	9	7
Nano Physics	1	4

EXPERIMENT 2: ORDER DATABASE

OBJECTIVES:

- **To create and insert values into a table**
- **To understand the working of aggregate functions (count, avg, sum etc).**
- **To create a view for a given table.**

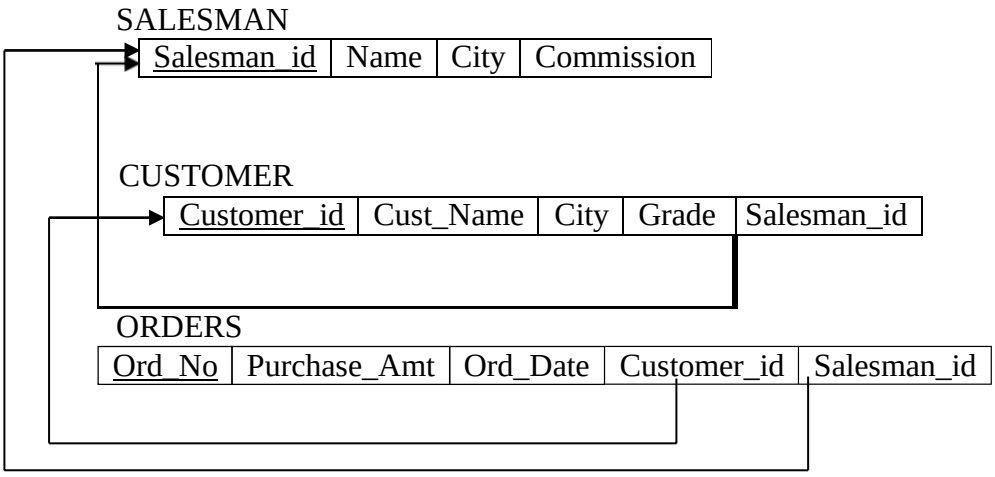
2 Consider the following schema for Order Database:

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)

Step 1: SCHEMA DIAGRAM



Step 2: ER DIAGRAM

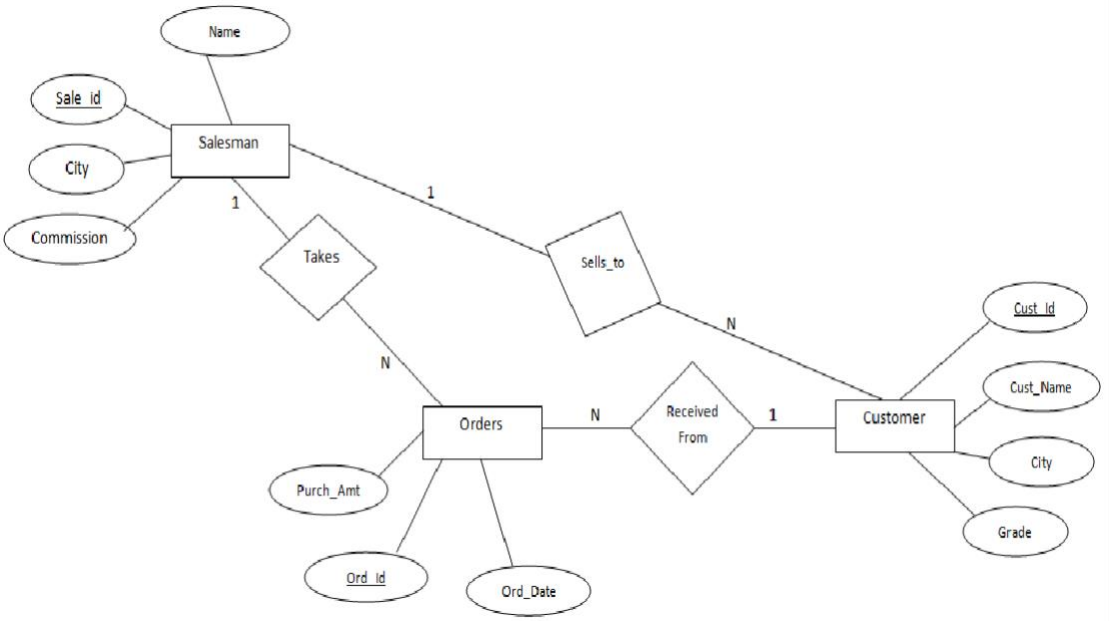


Table Creation

SQL> **CREATE TABLE** salesman

(salesman_id NUMBER(10) PRIMARY KEY,

salesman_name VARCHAR(200),

city VARCHAR(200),

commission NUMBER(10));

SQL> **CREATE TABLE** customer

(customer_id NUMBER(10) PRIMARY KEY,

customer_name VARCHAR(200),

city VARCHAR(200)

grade NUMBER(10),

salesman_id NUMBER(10),

FOREIGN KEY (salesman_id) REFERENCES salesman(salesman_id) ON DELETE CASCADE);

SQL> **CREATE TABLE** order1

(order_num NUMBER(10) PRIMARY KEY,

purchase_amount FLOAT,

order_date DATE,

customer_id NUMBER(10),

salesman_id NUMBER(10),

FOREIGN KEY (salesman_id) REFERENCES salesman(salesman_id) ON DELETE CASCADE,

FOREIGN KEY (customer_id) REFERENCES customer(customer_id));

SQL> **INSERT INTO** salesman **VALUES (**5001, 'Priya',' Bangalore', 15);

SQL> **INSERT INTO** salesman **VALUES (**5002, 'Preethi',' Mysore', 13);

SQL> **INSERT INTO** salesman **VALUES (**5003, 'Poonam',' Mangalore', 11);

SQL> **INSERT INTO** salesman **VALUES (**5004, 'Payal',' Delhi', 12);

SQL> **INSERT INTO** salesman **VALUES (**5005, 'Prerana',' Bombay', 19);

SQL> **SELECT * FROM** salesman;

salesman_id	salesman_name	city	commission
5001	Priya	Bangalore	15
5002	Preethi	Mysore	13
5003	Poonam	Mangalore	11
5004	Payal	Delhi	12
5005	Prerana	Bombay	19

SQL> **INSERT INTO** customer **VALUES (**4001, 'Dimple',' Bombay', 500,5001);

SQL> **INSERT INTO** customer **VALUES (**4002, 'Deepak',' Mangalore', 200,5002);

SQL> **INSERT INTO** customer **VALUES (**4003, 'Dhanush',' Mysore', 300,5003);

SQL> **INSERT INTO** customer **VALUES (**4004, 'Dhyan',' Bangalore', 400,5005);

SQL> **INSERT INTO** customer **VALUES (**4005, 'Dhanya',' Mandya', 100,5001);

SQL> **SELECT** * **FROM** customer;

customer_id	customer_name	city	grade	salesman_id
4001	Dimple	Bombay	100	5001
4002	Deepak	Mangalore	200	5002
4003	Dhanush	Mysore	300	5003
4004	Dhyan	Bangalore	400	5005
4005	Dhanya	Mandya	100	5001
4008	Medha	bangalore	800	5005
4009	Navya	delhi	900	5001

SQL> **INSERT INTO** order1 **VALUES (**90001, '150.5','2009-5-1', 4001,5001);

SQL> **INSERT INTO** order1 **VALUES (**90002, '111.9','2010-11-9', 4004,5002);

SQL> **INSERT INTO** order1 **VALUES (**90003, '1501.4','2005-9-21', 4003,5005);

SQL> **INSERT INTO** order1 **VALUES (**90004, '150.15','2013-12-10', 4002,5003);

SQL> **INSERT INTO** order1 **VALUES (**90005, '199.19','2015-1-11', 4005,5004);

SQL> **SELECT** * **FROM** order1;

order_num	purchase_amount	order_date	customer_id	salesman_id
90001	150.5	01.05.2009	4001	5001
90002	111.9	09.11.2010	4004	5002
90003	1501.4	21.09.2005	4003	5005
90004	150.15	10.12.2013	4002	5003
90005	199.19	11.01.2015	4005	5004

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

```
SQL> SELECT grade,COUNT(DISTINCT customer_id)
     FROM  customer
     GROUP BY grade
     HAVING grade > (SELECT avg(grade)
          FROM customer
          WHERE city='bangalore');
```

grade	COUNT(DISTINCT customer_id)
800	1
900	1

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

```
SQL> SELECT salesman_id, salesman_name
     FROM salesman a
     WHERE 1 <
     (SELECT COUNT(*)
       FROM customer
      WHERE salesman_id=a.salesman_id);
```

salesman_id	salesman_name
5001	Priya
5005	Prerana

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

```
SQL> SELECT salesman.salesman_id, salesman_name, customer_name, commission
     FROM salesman, customer
     WHERE salesman.city = customer.city
     UNION
     SELECT salesman_id, salesman_name, 'NO MATCH', commission
```

FROM salesman

WHERE NOT city = ANY

(SELECT city

FROM customer)

ORDER BY 2 DESC;

salesman_id	salesman_name	customer_name	commission
5001	Priya	Dhyan	15
5001	Priya	medha	15
5005	Prerana	Dimple	19
5002	Preethi	Dhanush	13
5003	Poonam	Deepak	11
5004	Payal	Navya	12

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

SQL> **CREATE VIEW** salesman11 AS

SELECT b.order_date, a.salesman_id, a.salesman_name

FROM salesman a, order1 b

WHERE a.salesman_id = b.salesman_id

AND b.purchase_amount =

(SELECT MAX (purchase_amount)

FROM order1 c

WHERE c.order_date = b.order_date**);**

SQL> **SELECT * FROM** salesman11;

order_date	salesman_id	salesman_name
11.01.2015	5004	Payal
10.12.2013	5003	Poonam

09.11.2010	5002	Preethi
01.05.2009	5001	Priya
21.09.2005	5005	Prerana

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

SQL> **DELETE FROM** salesman WHERE salesman_id=5001;
SQL> **SELECT** * **FROM** salesman;

salesman_id	**salesman_name**	**city**	**commission**
5002	Preethi	Mysore	13
5003	Poonam	Mangalore	11
5004	Payal	Delhi	12
5005	Prerana	Bombay	19

EXPERIMENT 3: MOVIE DATABASE

OBJECTIVES:

- **To create and insert values into a table**
- **To understand the working of JOIN operations (inner, outer).**

3. Consider the schema for Movie Database:

ACTOR**(**Act_id, Act_Name, Act_Gender)

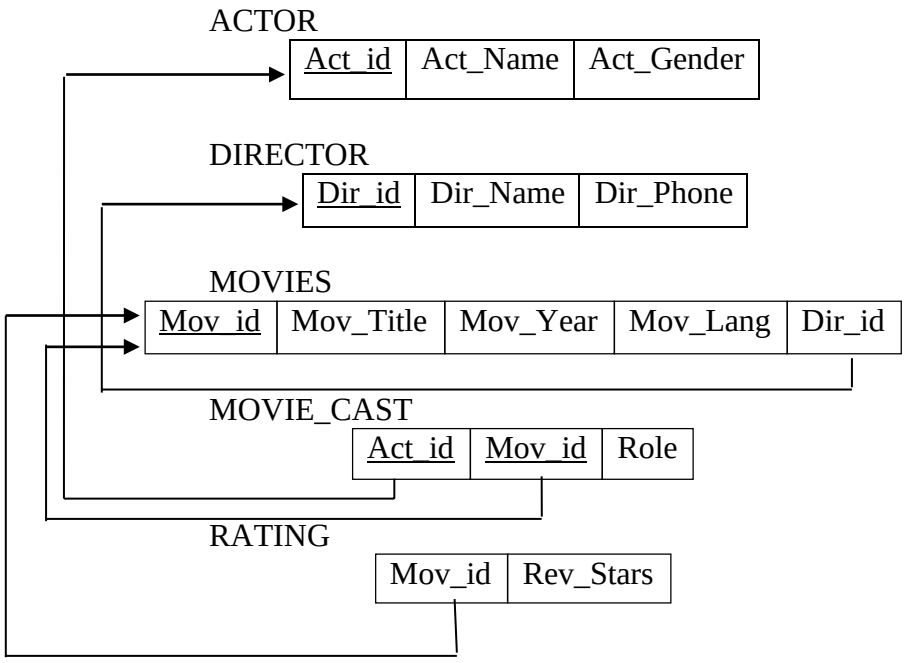
DIRECTOR**(**Dir_id, Dir_Name, Dir_Phone)

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

MOVIE_CAST**(**Act_id, Mov_id, Role)

RATING**(**Mov_id, Rev_Stars)

Step 1: SCHEMA DIAGRAM



Step 2: ER Diagram

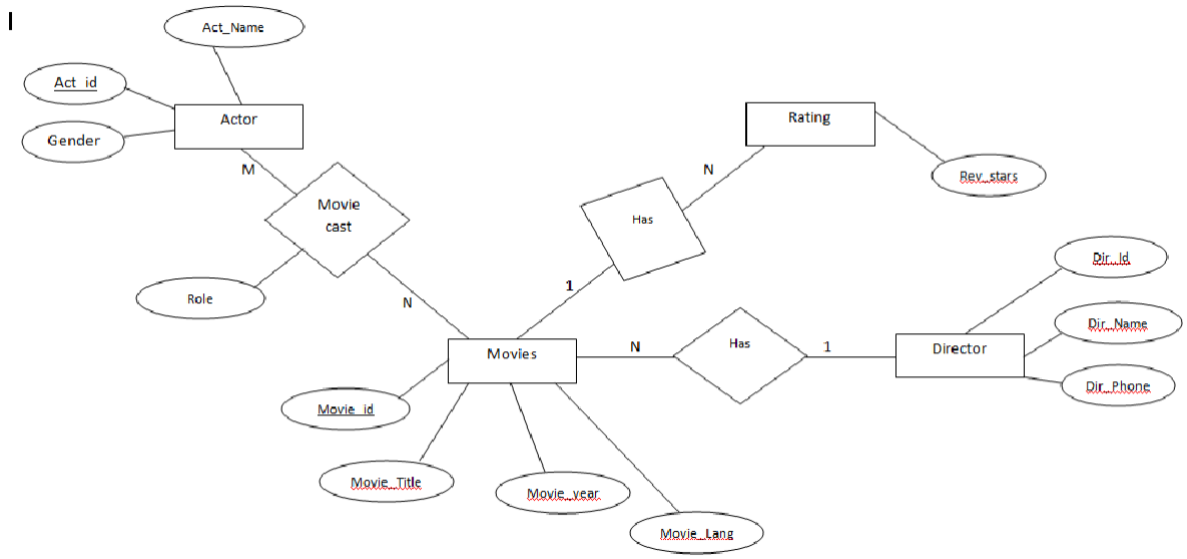


Table Creation

SQL> **CREATE TABLE** actor **(**

act_id NUMBER(10) PRIMARY KEY NOT NULL,

act_name VARCHAR(20) NOT NULL,

act_gender VARCHAR(20) NOT NULL

);

SQL> **CREATE TABLE** director **(**

dir_id NUMBER(10) PRIMARY KEY NOT NULL,

dir_name VARCHAR(20) NOT NULL,

dir_phone NUMBER(10) NOT NULL

);

SQL> **CREATE TABLE** Movie **(**

mov_id NUMBER(10) PRIMARY KEY NOT NULL,

mov_title VARCHAR(20) NOT NULL,

mov_year NUMBER(10) NOT NULL,

mov_lang VARCHAR(20) NOT NULL,

dir_id NUMBER(10) NOT NULL REFERENCES director(dir_id)

);

SQL> **CREATE TABLE** movie_cast **(**

act_id NUMBER(10) NOT NULL REFERENCES actor(act_id),

mov_id NUMBER(10) NOT NULL REFERENCES movie(mov_id),

role VARCHAR(20) NOT NULL,

PRIMARY KEY (act_id1, mov_id1)

);

SQL> **CREATE TABLE** rating **(**

mov_id NUMBER(10) NOT NULL REFERENCES movie(mov_id),

rev_stars NUMBER(10) NOT NULL,

PRIMARY KEY (mov_id11,rev_stars)

);

SQL> **INSERT INTO** actor **VALUES(**1, ' yash', ' M')

SQL> **INSERT INTO** actor **VALUES(**2, ' sudeep ', ' M')

SQL> **INSERT INTO** actor **VALUES(**3, ' kajol ', ' F')

SQL> **INSERT INTO** actor **VALUES(**4, ' madhuri ', ' F')

SQL> **INSERT INTO** actor **VALUES(**5, ' sonam', ' F')

SQL> **SELECT * FROM** actor;

act_id	act_name	act_gender
1	yash	M
2	sudeep	M
3	kajol	F
4	madhuri	F
5	sonam	F

SQL> **INSERT INTO** director **VALUES(**591, ' Hitchcock ', 98515561);

SQL> **INSERT INTO** director **VALUES(**592, ' Steven Spielberg', 97234535);

SQL> **INSERT INTO** director **VALUES(**593, ' David ', 98451246);

SQL> **INSERT INTO** director **VALUES(**594, ' Michael ', 99458933);

SQL> **INSERT INTO** director **VALUES(**595, ' Milos ', 99999999);

SQL> **SELECT** * **FROM** director;

dir_id	dir_name	dir_phone
591	Hitchcock	98515561
592	Steven Spielberg	97234535
593	David	98451246
594	Michael	99458933
595	Milos	99999999

SQL> **INSERT INTO** Movie **VALUES(**901, ' mann ', 2017,' hindi ', 591);

SQL> **INSERT INTO** Movie **VALUES(**902, ' deewana', 2010, ' hindi ', 592);

SQL> **INSERT INTO** Movie **VALUES(**903, ' nuvve ', 1999, ' telugu ', 593);

SQL> **INSERT INTO** Movie **VALUES(**904, ' santosham ', 1997, ' telugu ', 594);

SQL> **INSERT INTO** Movie **VALUES(**905, ' Amadeus ', 2005, ' English ', 595);

SQL> **SELECT** * **FROM** movie;

mov_id	mov_title	mov_year	mov_lang	dir_id1
901	mann	2017	hindi	591
902	deewana	2010	hindi	592
903	nuvve	1999	telugu	593
904	santosham	1997	telugu	594
905	Amadeus	2005	English	595

SQL> **INSERT INTO** movie_cast **VALUES(**1, 901, ' anand');

SQL> **INSERT INTO** movie_cast **VALUES(**2, 902, ' santosh');

SQL> **INSERT INTO** movie_cast **VALUES(**3, 903, ' priya');

SQL> **INSERT INTO** movie_cast **VALUES(**4, 904, ' vidya');

SQL> **INSERT INTO** movie_cast **VALUES(**5, 905, ' veena');

SQL> **INSERT INTO** movie_cast **VALUES**(5, 904, ' aarohi');

SQL> **SELECT** * **FROM** movie_cast;

act_id1	mov_id1	Role
1	901	Anand

2	902	santosh
3	903	Priya
4	904	Vidya
5	905	Veena

SQL> **INSERT INTO** rating **VALUES(**901, '8');

SQL> **INSERT INTO** rating **VALUES(**902, '7');

SQL> **INSERT INTO** rating **VALUES(**903, '1');

SQL> **INSERT INTO** rating **VALUES(**904, '5');

SQL> **INSERT INTO** rating **VALUES(**905, '4');

SQL> **SELECT** * **FROM** rating;

mov_id11	rev_stars
901	8
902	7
903	1
904	5
905	4

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

SQL> **SELECT** mov_title

FROM Movie m,director d

WHERE d.dir_id=m.dir_id and d.dir_name='Hitchcock';

mov_title
mann

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

SQL> **SELECT** mov_title, act_name, role

FROM movie

JOIN movie_cast

ON movie_cast.mov_id=movie.mov_id

JOIN actor

ON movie_cast.act_id=actor.act_id

WHERE actor.act_id IN **(**

SELECT act_id

FROM movie_cast

GROUP BY act_id **HAVING** COUNT**(***)>=2**);**

mov_title	act_name	Role
santosham	sonam	aarohi
Amadeus	sonam	veena

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

SQL> **SELECT** a.act_name, c.mov_title, c.mov_year

FROM actor a, movie_cast b, movie c

WHERE a.act_id=b.act_id

AND b.mov_id=c.mov_id

AND c.mov_year **NOT BETWEEN** 2000 and 2015;

act_name	mov_title	mov_year
yash	mann	2017
kajol	nuvve	1999
madhuri	santosham	1997
sonam	santosham	1997

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.

SQL> **SELECT** mov_title, **MAX(**rev_stars)

FROM movie m, rating r **WHERE** m.mov_id=r.mov_id

GROUP BY mov_title

HAVING MAX(rev_stars)>0

ORDER BY mov_title;

mov_title	**MAX(**rev_stars)
Amadeus	4
deewana	7
mann	8
nuvve	1
santosham	5

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

SQL> **UPDATE** rating **SET** rev_stars=5

WHERE mov_ID **in (SELECT** m.mov_ID

FROM Movie m, rating r

WHERE M.mov_ID=r.mov_ID **and** m.dir_id **in**

(SELECT dir_id **FROM** director

WHERE dir_name='david'));

mov_id11	**rev_stars**
901	8
902	7
903	1
904	5
905	4

EXPERIMENT 4: COLLEGE DATABASE

OBJECTIVES:

- **To create and insert values into a table**
- **To understand the sub queries & JOIN operations.**

4. Consider the schema for College Database:

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

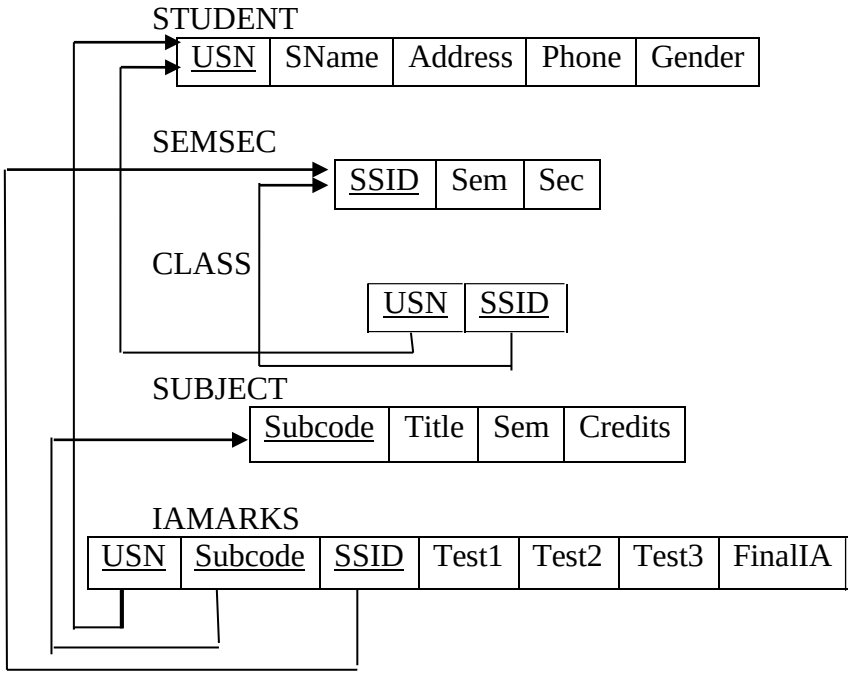
SEMSEC**(**SSID, Sem, Sec)

CLASS**(**USN, SSID)

SUBJECT**(**Subcode, Title, Sem, Credits)

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

Step 1: SCHEMA DIAGRAM



Step 2: ER DIAGRAM

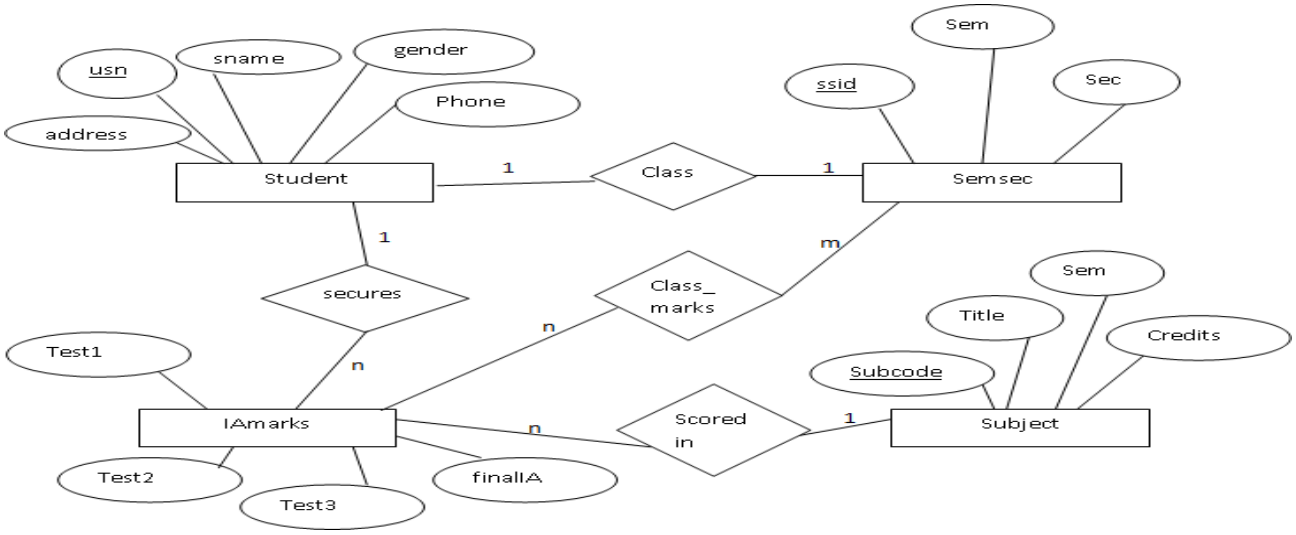


Table Creation

SQL> **CREATE TABLE** Student

(Usn VARCHAR(20) PRIMARY KEY,
Sname VARCHAR(20) NOT NULL,
Address VARCHAR(20),
Phone NUMBER(10),
Gender VARCHAR(2)
);

SQL> **CREATE TABLE** Semsec

(Ssid NUMBER(10) PRIMARY KEY,
Sem NUMBER(10),
Sec VARCHAR(10)
);

SQL> **CREATE TABLE** CLASS

(Usn VARCHAR(20) REFERENCES Student(Usn),
Ssid NUMBER(10) REFERENCES Semsec(Ssid),

```
    PRIMARY KEY(Usn,Ssid)
);
```

SQL> **CREATE TABLE** Subject

```
( Subcode     VARCHAR(20) PRIMARY KEY,
  Sub_title   VARCHAR(20)  NOT  NULL,
  Sem         NUMBER(10),
  Credits     NUMBER(10)
);
```

SQL> **CREATE TABLE** IAmarks

```
(Usn          VARCHAR(20),
Subcode       VARCHAR(20) REFERENCES Subject(Subcode),
Ssid          NUMBER(10),
Test1         NUMBER(10),
Test2         NUMBER(10),
Test3         NUMBER(10),
Finalia       NUMBER(6,2),
PRIMARY KEY(Usn,Subcode,Ssid),
FOREIGN KEY(Usn,Ssid) REFERENCES Class(Usn,Ssid)
 );
```

SQL> **INSERT INTO** Student **VALUES**('3br15cs001','priya','gandhinagar',97689989,'F');

SQL> **INSERT INTO** Student **VALUES** ('3br15cs002', 'poonam', 'sanjaynagar', 977868239, 'F');

SQL> **INSERT INTO** Student **VALUES**('3br15cs003','prem','m v colony',899999989,'M');

SQL> **INSERT INTO** Student **VALUES** ('3br15cs004','preetham','5 krishna', 989864489, 'M');

SQL> **INSERT INTO** Student **VALUES** ('3br15cs005', 'prerana', 'gandhinagar', 987577989, 'F');

SQL> **INSERT INTO** Student **VALUES** ('3br15cs007','neha','11 mh colony',987577989,'F');

SQL> **INSERT INTO** Student **VALUES** ('3br15cs008','aarav','Sansad Marg', 987577989, 'M');

SQL> **INSERT INTO** Student **VALUES** ('3br15cs009','aarush','Lodhi Road', 987577989, 'M');

SQL> **INSERT INTO** Student **VALUES** ('3br15cs011','dhanya','Palam Marg', 987577989, 'F');

SQL> **SELECT * FROM** Student;

Usn	**Sname**	**Address**	**Phone**	**Gender**
3br15cs001	priya	gandhinagar	97689989	F
3br15cs002	poonam	sanjaynagar	977868239	F
3br15cs003	prem	m v colony	899999989	M
3br15cs004	preetham	5 krishna	989864489	M
3br15cs005	prerana	gandhinagar	987577989	F
3br15cs007	neha	11 mh colony	987577989	F
3br15cs008	aarav	Sansad Marg	987577989	M
3br15cs009	aarush	Lodhi Road	987577989	M
3br15cs011	dhanya	Palam Marg	987577989	F
3br15cs019	advik	viveknagar	987987081	M
3br15cs021	aarohi	Palam Marg	98975991	F

SQL> **INSERT INTO** Semsec **VALUES**(11,1,'A');

SQL> **INSERT INTO** Semsec **VALUES**(12,5,'B');

SQL> **INSERT INTO** Semsec **VALUES**(13,7,'A');

SQL> **INSERT INTO** Semsec **VALUES**(14,7,'B');

SQL> **INSERT INTO** Semsec **VALUES**(15,5,'A');

SQL> **INSERT INTO** Semsec **VALUES**(18,8,'C');

SQL> **INSERT INTO** Semsec **VALUES**(17,1,'B');

SQL> **INSERT INTO** Semsec **VALUES**(21,8,'A');

SQL> **INSERT INTO** Semsec **VALUES**(29,5,'B');

SQL> **INSERT INTO** Semsec **VALUES**(31,7,'C');

SQL> **INSERT INTO** Semsec **VALUES**(19,7,'C');

SQL> **SELECT * FROM** Semsec;

Ssid	Sem	Sec
11	1	A
12	5	B
13	7	A
14	7	B
15	5	A
17	1	B
18	8	C
19	7	C
21	8	A
29	5	B
31	7	C
91	4	C
99	4	C

SQL> **INSERT INTO** Class **VALUES**('3br15cs001',11);

SQL> **INSERT INTO** Class **VALUES**('3br15cs002',12);

SQL> **INSERT INTO** Class **VALUES**('3br15cs003',13);

SQL> **INSERT INTO** Class **VALUES**('3br15cs004',14);

SQL> **INSERT INTO** Class **VALUES**('3br15cs005',15);

SQL> **INSERT INTO** Class **VALUES**('3br15cs005',18);

SQL> **INSERT INTO** Class **VALUES**('3br15cs004',18);

SQL> **INSERT INTO** Class **VALUES**('3br15cs008',17);

SQL> **INSERT INTO** Class **VALUES**('3br15cs009',21);

SQL> **INSERT INTO** Class **VALUES**('3br15cs011',29);

SQL> **INSERT INTO** Class **VALUES**('3br15cs011',11);

SQL> **INSERT INTO** Class **VALUES**('3br15cs001',17);

SQL> **INSERT INTO** Class **VALUES**('3br15cs002',29);

SQL> **INSERT INTO** Class **VALUES**('3br15cs003',12);

SQL> **INSERT INTO** Class **VALUES**('3br15cs004',13);

SQL> **INSERT INTO** Class **VALUES**('3br15cs005',14);

SQL> **INSERT INTO** Class **VALUES**('3br15cs008',11);

SQL> **INSERT INTO** Class **VALUES**('3br15cs001',13);

SQL> **INSERT INTO** Class **VALUES**('3br15cs004',15);

SQL> **INSERT INTO** Class **VALUES**('3br15cs011',21);

SQL> **INSERT INTO** Class **VALUES**('3br15cs009',29);

SQL> **SELECT * FROM** Class;

Usn	**Ssid**
3br15cs001	11
3br15cs001	13
3br15cs001	17
3br15cs002	12
3br15cs002	29
3br15cs003	12
3br15cs003	13
3br15cs004	13
3br15cs004	14
3br15cs004	15
3br15cs004	18
3br15cs005	14
3br15cs005	15
3br15cs005	18
3br15cs008	11
3br15cs008	17
3br15cs009	21
3br15cs009	29
3br15cs011	11

3br15cs011	21
3br15cs011	29
3br15cs019	91
3br15cs021	99

SQL> **INSERT INTO** Subject **VALUES**('15cs01','dbms',5,4);

SQL> **INSERT INTO** Subject **VALUES**('15cs02','os',7,4);

SQL> **INSERT INTO** Subject **VALUES**('15cs03','ss',5,4);

SQL> **INSERT INTO** Subject **VALUES**('15cs04','c',1,8);

SQL> **INSERT INTO** Subject **VALUES**('15cs05','cn',7,8);

SQL> **SELECT * FROM** Subject;

Subcode	**Sub_title**	**Sem**	**Credits**
15cs01	dbms	5	4
15cs02	os	7	4
15cs03	ss	5	4
15cs04	C	1	8
15cs05	cn	7	8

SQL> **INSERT INTO** IAmarks **VALUES**('3br15cs001','15cs01',11,11,12,13,NULL);

SQL> **INSERT INTO** IAmarks **VALUES**('3br15cs002','15cs02',12,19,12,13,NULL);

SQL> **INSERT INTO** IAmarks **VALUES**('3br15cs003','15cs03',13,11,19,20,NULL);

SQL> **INSERT INTO** IAmarks **VALUES**('3br15cs004','15cs04',14,19,12,18,NULL);

SQL> **INSERT INTO** IAmarks **VALUES**('3br15cs005','15cs05',15,19,18,19,NULL);

SQL> **INSERT INTO** IAmarks **VALUES**('3br15cs005','15cs05',18,11,13,19,NULL);

SQL> **SELECT * FROM** IAmarks;

Usn	**Subcode**	**Ssid**	**Test1**	**Test2**	**Test3**	**Finalia**
3br15cs001	15cs01	11	11	12	13	*NULL*
3br15cs002	15cs02	12	19	12	13	*NULL*
3br15cs003	15cs03	13	11	19	20	*NULL*
3br15cs004	15cs04	14	19	12	18	*NULL*

3br15cs005	15cs05	15	19	18	19	*NULL*
3br15cs005	15cs05	18	11	13	19	*NULL*

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

SQL> **SELECT** s.usn,s.sname,s.address,s.phone

FROM Student s, Semsec n, Class c

WHERE n.sem=4 and n.sec='C' and s.usn=c.usn and c.ssid=n.ssid;

usn	**sname**	**address**	**phone**
3br15cs019	advik	viveknagar	987987081
3br15cs021	aarohi	Palam Marg	98975991

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

SQL> **SELECT** c.ssid room_no, sec section, sem semester,

SUM(CASE WHEN s.gender='M' **THEN** 1 **ELSE** 0 **END**) **AS** MALE,

SUM(CASE WHEN s.gender='F' **THEN** 1 **ELSE** 0 **END**) **AS** FEMALE

FROM Class c, Student s, Semsec m

WHERE s.usn=c.usn **AND** m.ssid=c.ssid **GROUP BY** c.ssid, sem, sec;

room_no	**section**	**semester**	**male**	**female**
11	A	1	1	2
12	B	5	1	1
13	A	7	2	1
14	B	7	1	1
15	A	5	1	1
17	B	1	1	1
18	C	8	1	1
21	A	8	1	1
29	B	5	1	2

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

SQL> **CREATE VIEW** Test1_Marks **AS**

SELECT test1 **FROM** IAmarks **WHERE** usn='3br15cs005'

test1
19
11

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

SQL> **UPDATE** iamarks **SET** finalia= (**GREATEST**(test1+test2,test2+test3,test3+test1))/2;

SQL> **SELECT** * **FROM** IAmarks;

Usn	Subcode	Ssid	Test1	Test2	Test3	Finalia
3br15cs001	15cs01	11	11	12	13	12.5
3br15cs002	15cs02	12	19	12	13	16
3br15cs003	15cs03	13	11	19	20	19.5
3br15cs004	15cs04	14	19	12	18	18.5
3br15cs005	15cs05	15	19	18	19	19

5. Categorize students based on the following criterion:

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

If FinalIA = 12 to 16 then CAT = 'Average'

If FinalIA< 12 then CAT = 'Weak'

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

SQL> **SELECT** i.usn,i.test1,i.test2,i.test3,i.finalia,

(**CASE WHEN** i.finalia>=17 **AND** i.finalia<=20 **THEN** 'outstanding'

WHEN i.finalia>=12 **AND** i.finalia<=17 **THEN** 'average'

WHEN i.finalia>12 **THEN** 'weak'

ELSE 'invalid'

END) **AS** grade

FROM IAmarks i, Class c, Semsec s

WHERE i.usn=c.usn **AND** c.ssid=s.ssid **AND** s.sem=8 and s.sec **IN** ('A','B','C')

Usn	**test1**	**test2**	**test3**	**finalia**	**grade**
3br15cs004	19	12	18	18.5	Outstanding
3br15cs005	19	18	19	*18.5*	Outstanding
3br15cs005	11	13	19	*16*	Average

EXPERIMENT 5: COMPANY DATABASE

OBJECTIVES:

- **To create and insert values into a table.**
- **To understand the working of JOIN operations (inner, outer).**

5. Consider the schema for Company Database:

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

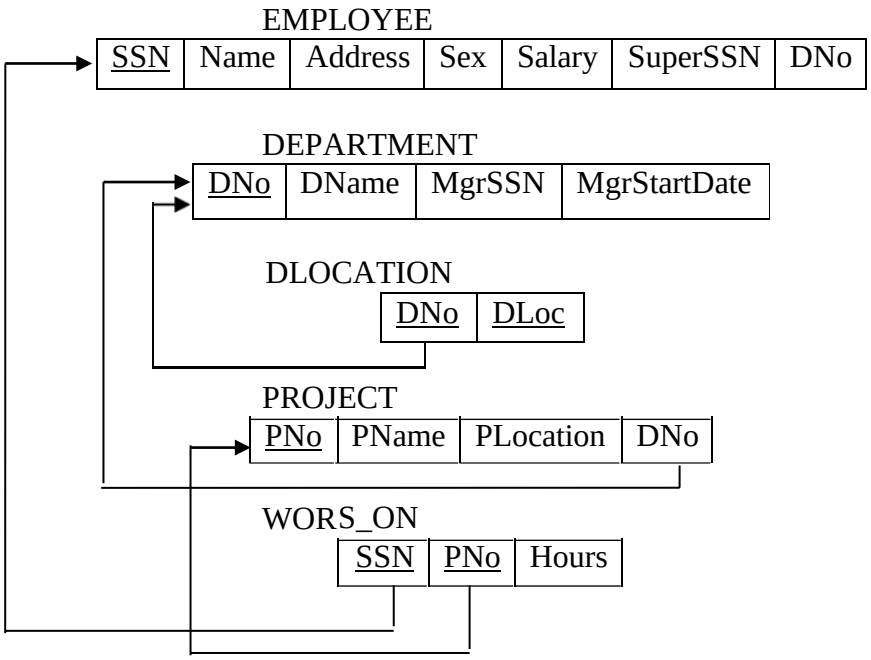
DEPARTMENT**(**DNo, DName, MgrSSN, MgrStartDate)

DLOCATION**(**DNo,DLoc)

PROJECT**(**PNo, PName, PLocation, DNo)

WORKS_ON**(**SSN, PNo, Hours)

Step 1: SCHEMA DIAGRAM



STEP 2: ER DIAGRAM

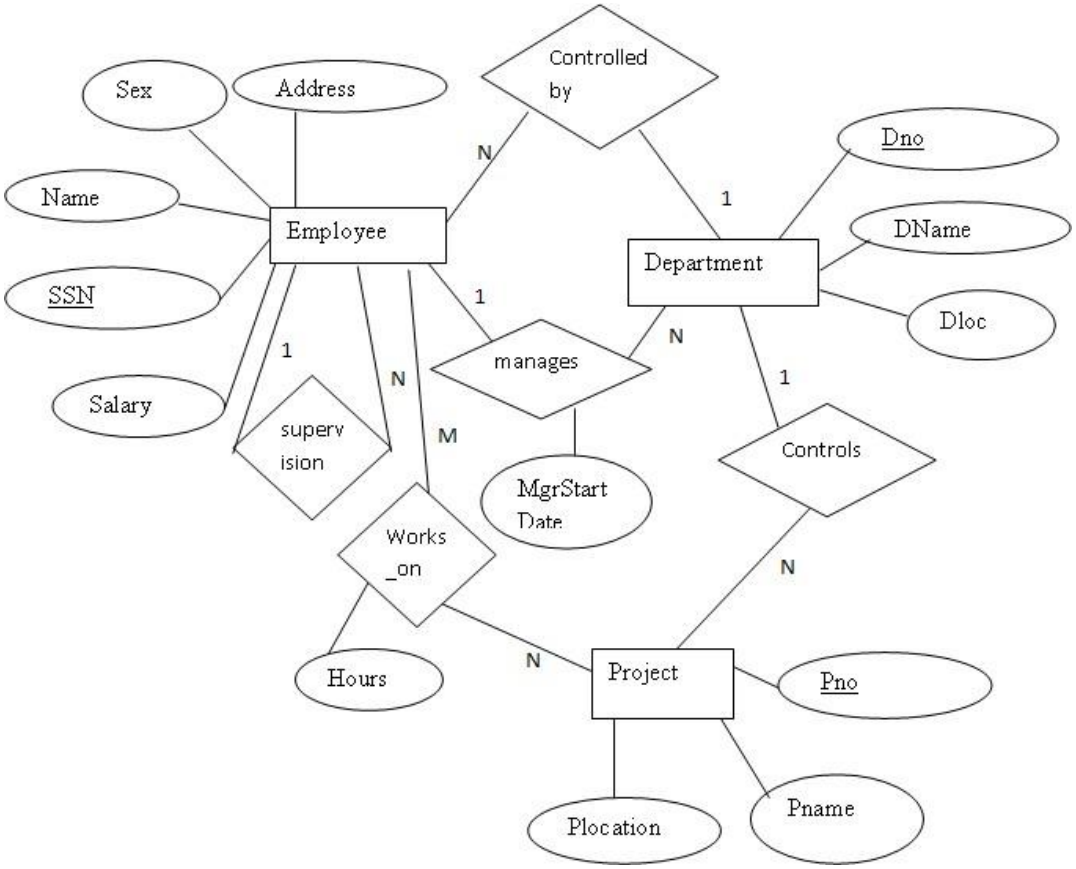


Table Creation

SQL> **CREATE TABLE** Department **(**

DNO NUMBER(10) PRIMARY KEY NOT NULL,

DNAME VARCHAR(20) NOT NULL,

MGRSSN NUMBER(10),

MGRSTARTDATE DATE,

);

SQL> **CREATE TABLE** Employee **(**

SSN NUMBER(10) PRIMARY KEY NOT NULL,

NAME VARCHAR(20) NOT NULL,

ADDRESS VARCHAR(20),

SEX VARCHAR(5),

SALARY NUMBER(10),

```
    SUPERSSN        VARCHAR(20),
    DNO             NUMBER(10)          NOT NULL,
    FOREIGN KEY(DNO) REFERENCES Department(DNO)
    );
```

```
SQL> CREATE TABLE Dept_Locations (
    DNO             NUMBER(10)      NOT NULL,
    DLOCATION           VARCHAR(20)     NOT NULL,
    PRIMARY KEY (DNO, DLOCATION),
    FOREIGN KEY (DNO) REFERENCES Department(DNO)
);
```

```
SQL> CREATE TABLE Project(
    PNO             NUMBER(10)      PRIMARY KEY  NOT NULL,
    PNAME           VARCHAR(20)     NOT NULL,
    PLOCATION           VARCHAR(20),
    DNO             NUMBER(10)      NOT NULL,
    FOREIGN KEY (DNO) REFERENCES Department(DNO)
);
```

```
SQL> CREATE TABLE Works_On (
    ESSN        NUMBER(10)      NOT NULL,
    PNO         NUMBER(10)      NOT NULL,
    HOURS       NUMBER(10),
    PRIMARY KEY (ESSN, PNO),
    FOREIGN KEY (ESSN) REFERENCES Employee (SSN),
    FOREIGN KEY (PNO) REFERENCES Project (PNO)
);
```

SQL> **INSERT INTO** Department **VALUES (**1,'cse',11,'12-aug-12'**);**

SQL> **INSERT INTO** Department **VALUES (**2,'cse',12,'17-may-12'**);**

SQL> **INSERT INTO** Department **VALUES (**3,'accounts',13,'7-jun-17'**);**

SQL> **INSERT INTO** Department **VALUES (**4,'mech',14,'14-jul-11'**);**

SQL> **INSERT INTO** Department **VALUES (**5,'eee',15,'19-nov-07'**);**

SQL> **INSERT INTO** Department **VALUES (**9,'research',19,'11-jan-99'**);**

SQL> **INSERT INTO** Department **VALUES (**8,'accounts',18,'7-jun-17'**);**

SQL> **SELECT * FROM** Department;

DNO	DNAME	MGRSSN	MGRSTARTDATE
1	cse	11	12-aug-2012
2	cse	12	17-may-2012
3	accounts	13	07-jun-2017
4	mech	14	14-jul-2011
5	eee	15	19-nov-2007
8	accounts	18	07-jun-2017
9	research	19	11-jan-1999

SQL> **INSERT INTO** Employee **VALUES (**11,'priya','gandhinagar','F',900000,12,3**);**

SQL> **INSERT INTO** Employee **VALUES (**12,'prem','sanjaynagar','M',500000,11,5**);**

SQL> **INSERT INTO** Employee **VALUES (**13,'rakshith','delhi','M',900000,13,5**);**

SQL> **INSERT INTO** Employee **VALUES (**14,'shreyas','bangalore','F',900000,15,5**);**

SQL> **INSERT INTO** Employee **VALUES (**15,'poonam','mysore','F',900000,14,5**);**

SQL> **INSERT INTO** Employee **VALUES (**18,' Preethi','mangalore','F',900000,15,5**);**

SQL> **INSERT INTO** Employee **VALUES (**19,'scott','madras','M',900000,15,3**);**

SQL> **INSERT INTO** Employee **VALUES (**17,'scott','vizag','M',1500000,11,3**);**

SQL> **INSERT INTO** Employee **VALUES (**21,'medha','hubli','F',1500000,11,5**);**

SQL> **SELECT * FROM** Employee;

SSN	NAME	ADDRESS	SEX	SALARY	SUPERSSN	DNO

11	priya	gandhinagar	F	900000	12	3
12	prem	sanjaynagar	M	500000	11	5
13	rakshith	Delhi	M	900000	13	5
14	shreyas	Bangalore	F	900000	15	5
15	poonam	Mysore	F	900000	14	5
17	scott	Vizag	M	1500000	11	3
18	Preethi	mangalore	F	900000	15	5
19	scott	Madras	M	900000	15	3
21	medha	Hubli	F	1500000	11	5

SQL> **ALTER TABLE** Department **ADD FOREIGN KEY (**MGRSSN)

REFERENCES Employee**(**SSN**);**

SQL> **INSERT INTO** Dept_Locations **VALUES (**1, 'gandhinagar'**);**

SQL> **INSERT INTO** Dept_Locations **VALUES (**2, 'delhi'**);**

SQL> **INSERT INTO** Dept_Locations **VALUES (**3, 'bangalore'**);**

SQL> **INSERT INTO** Dept_Locations **VALUES (**4, 'Mysore'**);**

SQL> **INSERT INTO** Dept_Locations **VALUES (**5, 'madras'**);**

SQL> **INSERT INTO** Dept_Locations **VALUES (**9, 'sanjaynagar'**);**

SQL> **SELECT** * **FROM** Dept_Locations;

DNO	**DLOCATION**
1	gandhinagar
2	delhi
3	bangalore
4	Mysore
5	madras
9	sanjaynagar

SQL> **INSERT INTO** Project **VALUES (**2221,’iot’,’mysore’,1**);**

SQL> **INSERT INTO** Project **VALUES (**2222,'analytics','bangalore',2**);**

SQL> **INSERT INTO** Project **VALUES (**2223,'cc','delhi',3**);**

SQL> **INSERT INTO** Project **VALUES (**2224,'mobility','madras',4**);**

SQL> **INSERT INTO** Project **VALUES (**2225,'iot','mangalore',5**);**

SQL> **INSERT INTO** Project **VALUES (**2229,'net','gandhinagar',9**);**

SQL> **SELECT** * **FROM** Project;

PNO	PNAME	PLOCATION	DNO
2221	iot	mysore	1
2222	analytics	bangalore	2
2223	cc	delhi	3
2224	mobility	madras	4
2225	iot	mangalore	5
2229	net	gandhinagar	9

SQL> **INSERT INTO** Works_On **VALUES (**11,2221,11.5**);**

SQL> **INSERT INTO** Works_On **VALUES (**12,2222,21.5**);**

SQL> **INSERT INTO** Works_On **VALUES (**13,2223,21.9**);**

SQL> **INSERT INTO** Works_On **VALUES (**14,2224,91.1**);**

SQL> **INSERT INTO** Works_On **VALUES (**15,2225,91.9**);**

SQL> **INSERT INTO** Works_On **VALUES (**19,2229,91.5**);**

SQL> **INSERT INTO** Works_On **VALUES (**19,2225,91.5**);**

SQL> **SELECT** * **FROM** Works_On;

ESSN	PNO	HOURS
11	2221	11
12	2222	21
13	2223	21
14	2224	91
15	2225	91
19	2225	91
19	2229	91

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.

```
SQL> SELECT DISTINCT p.PNO FROM Project p, Department d, Employee e
     WHERE p.DNO = d.DNO AND d.MGRSSN = e.SSN AND e.NAME = 'Scott'
     UNION
     (SELECT DISTINCT a.PNO FROM Project a, Works_On t, Employee n
     WHERE a.PNO = t.PNO AND t.ESSN = n.SSN AND n.NAME = 'Scott');
```

PNO
2225
2229

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

```
SQL> SELECT e.NAME, 1.1*SALARY as increased
     FROM Employee e, Works_on a, Project p
     WHERE e.SSN = a.ESSN AND a.PNO = p.PNO AND p.PNAME = 'IoT';
```

NAME	increased
priya	990000,0
poonam	990000,0
scott	990000,0

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

```
SQL> SELECT SUM (SALARY), MAX (SALARY), MIN (SALARY), AVG (SALARY)
     FROM Employee e, Department d
     WHERE e.DNO = d.DNO AND d.DNAME = 'Accounts';
```

SUM (SALARY)	MAX (SALARY)	MIN (SALARY)	AVG (SALARY)
3300000	1500000	900000	1100000

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

```
SQL> SELECT e.NAME
     FROM Employee e
     WHERE NOT EXISTS (( SELECT PNO
                         FROM Project WHERE DNO = 5)
                       minus
                         (SELECT PNO
                         FROM Works_On a
                         WHERE e.SSN = a.ESSN) );
```

NAME
poonam
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.

```
SQL> SELECT d.DNO, COUNT (*)
     FROM Department d, Employee e
     WHERE d.DNO = e.DNO AND SALARY > 600000 AND
                         e.DNO IN (SELECT e.DNO
                                   FROM Employee e
                                   GROUP BY e.DNO
                                   HAVING COUNT (*) > 5)
                         GROUP BY d.DNO;
```

DNO	(No column name)
5	5