SQL AND RDBMS ASSIGNMENTS

Assignment1: Analyse a given business scenario and create an ER diagram that includes entities, relationships, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to the third normal form.

Entities:

1. Patient

2. Doctor

3. Test

Relationships:

1. Patient visits Doctor

2. Doctor treats Patient

3. Doctor works in Department

4. Patient has Appointment

5. Patient undergoes Test

Attributes:

1. Patient: SS#, name, insurance, date_admitted, date_checked_out

2. Doctor: DSS, Name, Specialization

4. Test: test_ID, test_ name, Date, time, result

Cardinality:

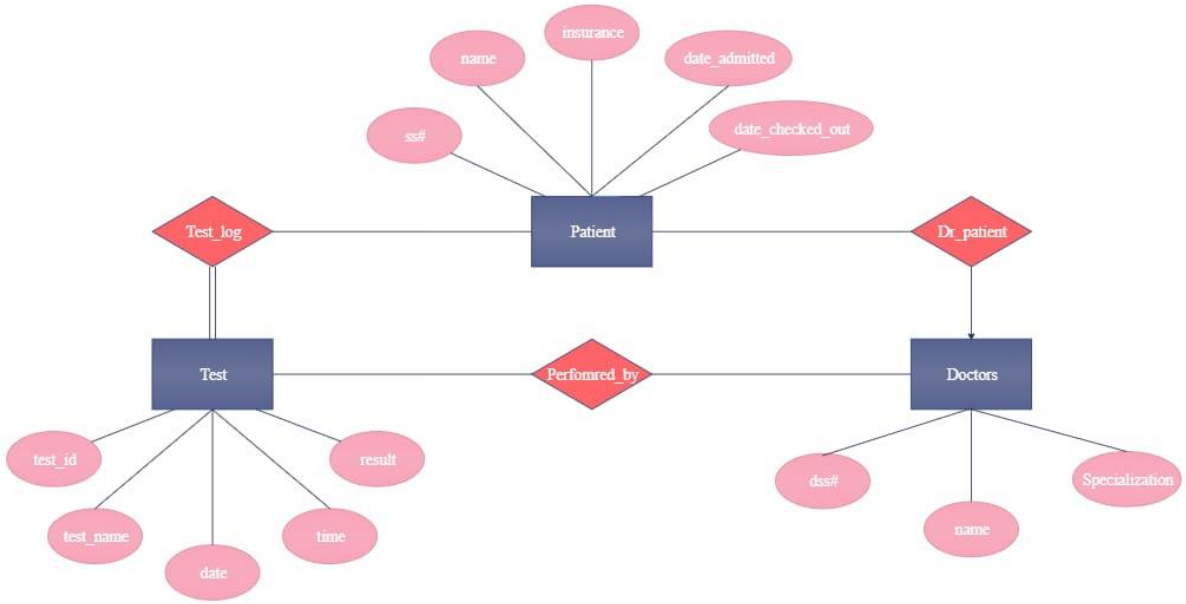
1. One patient can have multiple appointments.

2. One doctor can have multiple appointments, treat multiple patients, and perform multiple tests.

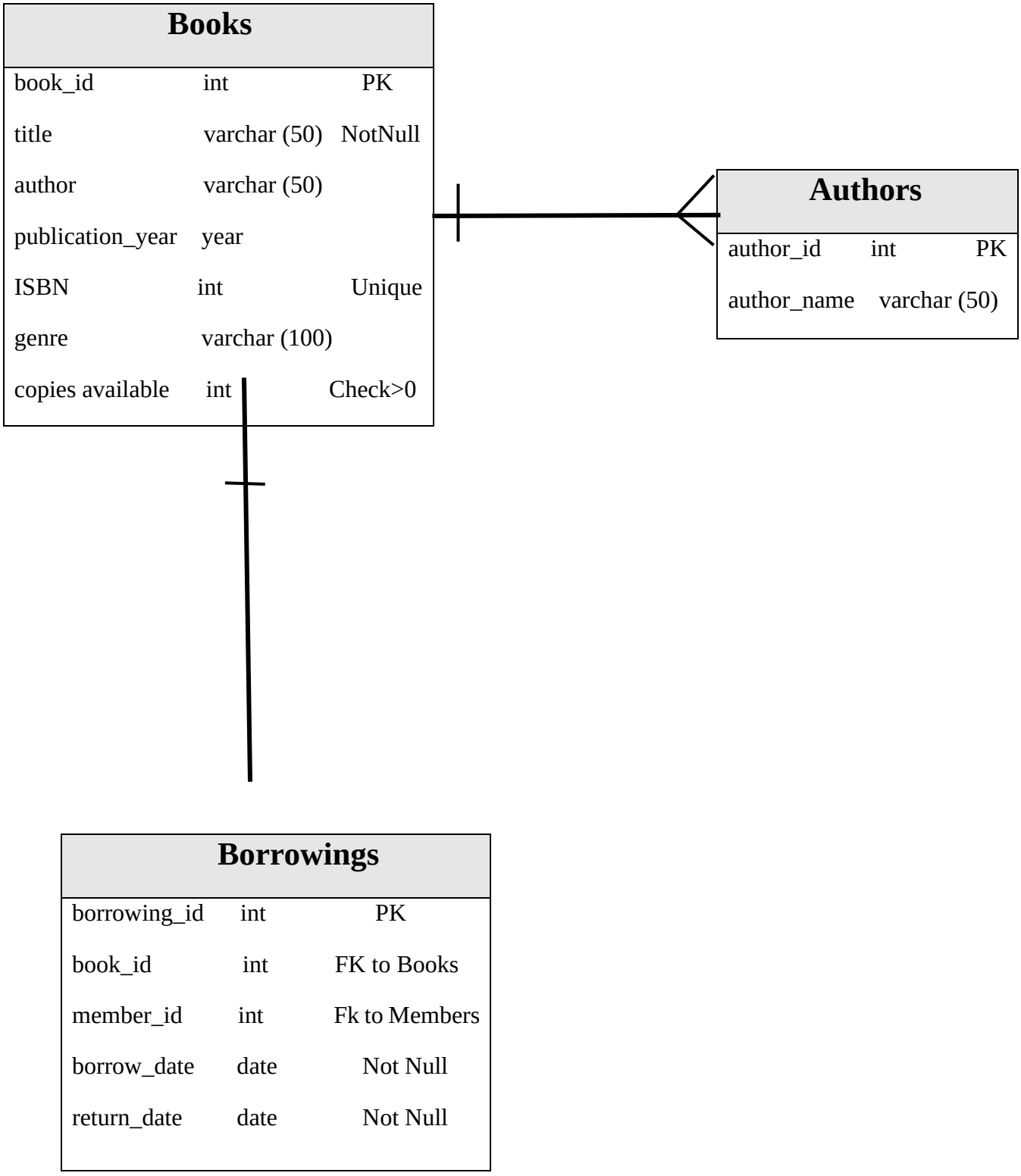
3. One nurse can assist multiple doctors

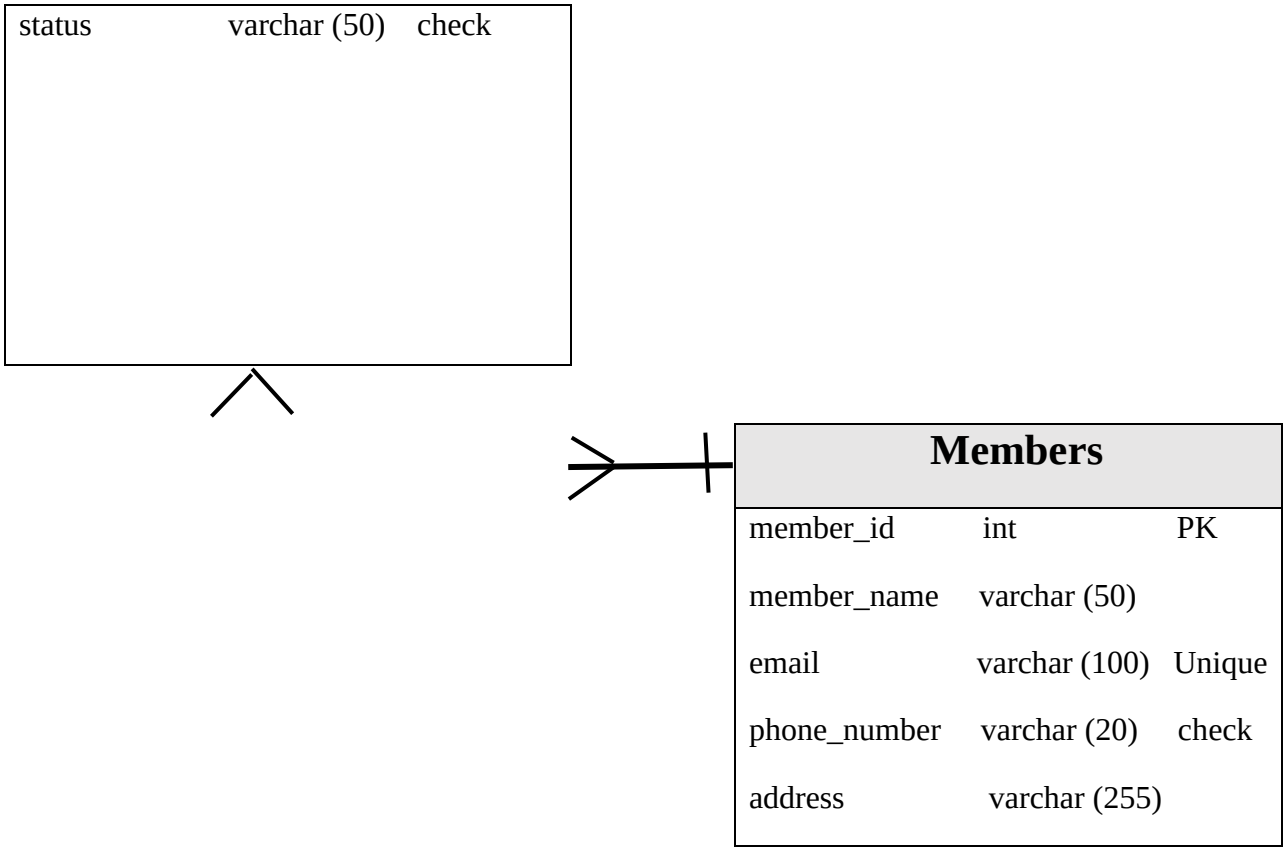
4. One department can have multiple doctors.

ER diagram of Hospital



Assignment2: Design a database schema for a library system, including tables, fields, and constraints like NOT NULL, UNIQUE, and CHECK. Include primary and foreign keys to establish relationships between tables.





Code for above question:

```
CREATE TABLE Authors (
    author_id INT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    nationality VARCHAR(50) NOT NULL
);
CREATE TABLE Books (
    book_id INT PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    author_id INT NOT NULL,
    genre VARCHAR(50),
```

```
    publication_year INT,
    FOREIGN KEY (author_id) REFERENCES Authors(author_id)
);
CREATE TABLE Members (
    member_id INT PRIMARY KEY,
    name VARCHAR (100) NOT NULL,
    email VARCHAR (100) NOT NULL,
    address VARCHAR (200) NOT NULL
);
CREATE TABLE Transactions (
    transaction_id INT PRIMARY KEY,
    book_id INT NOT NULL,
    member_id INT NOT NULL,
    checkout_date DATE NOT NULL,
    return_date DATE NOT NULL,
    FOREIGN KEY (book_id) REFERENCES Books(book_id),
    FOREIGN KEY (member_id) REFERENCES Members(member_id),
    CHECK (checkout_date <= return_date);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
member_id	int	NO	PRI	NULL	
name	varchar(150)	NO		NULL	
email	varchar(200)	NO		NULL	
address	varchar(200)	NO		NULL	

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
transcation_id	int	NO	PRI	NULL	
book_id	int	NO	MUL	NULL	
member_id	int	NO	MUL	NULL	
checkout_date	date	NO		NULL	
return_date	date	NO		NULL	

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
book_id	int	NO	PRI	NULL	
title	varchar(200)	NO		NULL	
author	varchar(200)	NO		NULL	
genre	varchar(150)	YES		NULL	
publication_year	int	YES		NULL	

Assignment3: Explain the ACID properties of a transaction in your own words. Write SQL statements to simulate a transaction that includes locking and demonstrate different isolation levels to show concurrency control.

Acid Properties:

1. Atomicity: This property ensures that either all the operations within a transaction are successfully completed, or none of them are. If any part of the transaction fails, the entire transaction is rolled back to its original state.

2. Consistency: Consistency ensures that the database remains in a valid state before and after the transaction. All integrity constraints, such as foreign key constraints or uniqueness constraints, must be satisfied.

3. Isolation: Isolation ensures that the concurrent execution of transactions results in a state that could be obtained if transactions were executed serially. Isolation levels define the degree to which the operations within one transaction are isolated from the operations of other concurrent transactions.

4. Durability: Durability guarantees that once a transaction has been committed, the changes made by it will persist even in the event of system failure.

```
CREATE TABLE bank_accounts (
    account_id INT PRIMARY KEY,
    balance DECIMAL(10, 2)
);
INSERT INTO bank_accounts (account_id, balance) VALUES
(1, 1000.00),
(2, 2000.00);
BEGIN TRANSACTION;
-- Withdrawal operation
UPDATE bank_accounts
SET balance = balance - 500.00
WHERE account_id = 1;
UPDATE bank_accounts
SET balance = balance + 500.00
WHERE account_id = 2;
Commit;
```

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

SELECT balance FROM bank_accounts WHERE account_id = 1;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

SELECT balance FROM bank_accounts WHERE account_id = 1;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

SELECT balance FROM bank_accounts WHERE account_id = 1;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT balance FROM bank_accounts WHERE account_id = 1;
```

Assignment4: Write SQL statements to CREATE a new database and tables that reflect the library schema you designed earlier. Use ALTER statements to modify the table structures and DROP statements to remove a redundant table.

```
create database library;

use library;

create table books (book_id int primary key, title varchar (50) not null, author varchar (50),

publication_year year, ISBN int unique);

alter table books add column copies_available int;

alter table books add column author_id int;

alter table books drop column author;

desc books;

-- output

/*

Field          Type          Null          Key

book_id        int           NO            PRI

title          varchar (50)          NO

publication_yearyear         YES

ISBN           bigint        YES           UNI

author_id              int   YES

*/
```

```
create table authors (author_id int primary key, author_name varchar (50));
desc authors;
-- output
/*
Field          Type              Null              Key
author_id              int          NO          PRI
author name      varchar (50)    YES
*/
create table borrowings (borrowing_id int primary key, book_id int, member_id int, borrow_date date,
return_date date, status varchar (50));
desc borrowings;
-- output
/*
 Field          Type             Null      Key
borrowing          int           NO          PRI
book_id        int      YES
member_id       int                      YES
borrow_date     date                     YES
return_date           date              YES
status           varchar (50) YES
*/
create table members (member_id int primary key, member_name varchar (50), email varchar (100),
phone_number varchar(20),address varchar(50));
desc members;
```

```
-- output
/*
 Field                Type                Null    Key
member_id             int                         NO      PRI
member_name           varchar(50)         YES
email                 varchar(100)    YES
phone_number  varchar(20)             YES
address               varchar(50)         YES
*/
alter table borrowings modify status varchar(100);
create table books(book_title varchar (20),book_price float);
drop table books;
```

Assignment5: Demonstrate the creation of an index on a table and discuss how it improves query performance. Use a DROP INDEX statement to remove the index and analyse the impact on query execution.

```
use Krishna;
create index index_empname on employee_details (emp_id, emp_name);
explain select * from employee_details where emp_name='Rahul';
-- output
/*
id      select_type         table                       partitions      type
        possible_keys   key         key_len     ref         rows    filtered  Extra
1       SIMPLE              employee_details     NULL        ALL             NULL
          NULL  NULL        NULL    11          10.00       Using where
*/
explain select * from employee_details where emp_id>500;
```

-- output

/*

id select_type table partitions type possible_keys key key_len ref rows filtered Extra

1 SIMPLE employee_details NULL range PRIMARY,index_empname PRIMARY 4 NULL 6 100.00 Using where

*/

drop index index_empname on employee_details;

explain select * from employee_details where emp_id>500;

Assignment6: Create a new database user with specific privileges using the CREATE USER and GRANT commands. Then, write a script to REVOKE certain privileges and DROP the user.

-- creating new database user

create user krishna identified by 'krishna@123';

select user ();

-- output

/* user ()

root localhost

*/

select user from mysql.user;

grant select, update, insert on mydatabase. employee to krishna;

grant all on mydatabase. employee to krishna;

revoke insert on mydatabase. employee from krishna;

revoke update on mydatabase. employee from krishna;

drop user krishna;

Assignment7: Prepare a series of SQL statements to INSERT new records into the library tables, UPDATE existing records with new information, and DELETE records based on specific criteria. Include BULK INSERT operations to load data from an external source.

use library;

desc books;

insert into books values (1241,'Making India Awesome',2005,9781565924796,5748), (1242,'A Bend in the River',2011,9789565923796,5749), (1243,'One indian Girl',2018,9781565924791,5748),

(1244,'A Brush with Life',2006,978156592465,5750);

select * from books;

alter table books modify column ISBN bigint;

desc authors;

insert into authors values (5748,'Chetan Bhagat'), (5749,'V.S. Naipaul'),(5750,'Satish Gujral'),(5751,'Gita Mehta');

select * from authors;

desc borrowings;

insert into borrowings values(1,1241,451,'2024-02-12','2024-02-21','submitted'),(2,1243,452,'2024-04-23','2024-04-29','submitted'),

(3,1244,453,'2024-01-04','2024-01-11','Not submitted');

select * from borrowings;

desc members;

insert into members values (451,'Arun’, ‘arun@623.com',7483327942,'Hyderabad'), (452,'Rajan’, ‘rajan@123.com',783539869,'Sripuram'), (453,'Lohith’, ‘lohit@58.com',7933942983,'Madnpur');

delete from members where member_id=453;

Day 2:

Assignment1: Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer's name and email address for customers in a specific city.

```
use joinsdb;

select * from customers;

-- output

/*
 CustomerId        CustomerName        Contact No        city        email

8241        Ravi        9876543210    Hyderabad    ravi123@gmail.com

8242        Lasaya        9792742853        Chennai
lasaya54@gmail.com

8243        Nani        5333353664        Vizag
nani2002@gmail.com

8244        Bhanu        9454469393        Bengaluru
bhanu56@gmail.com

8245        Sravya        6836489456        Hyderabad
sravyasravs@gmail.com

*/

select customername,email from customers where city='Hyderabad';

/* output

customer name  email

Ravi        ravi123@gmail.com

Sravya        sravyasravs@gmail.com
```

```
*/

select customername, email from customers where city='Chennai';

/* output

customername   email

Laya                    laya54@gmail.com

*/

select customername,email from customers where city='Mumbai';



/* output

 customername, email

 */
```

Assignment2: Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.

```
use joinsdb;

-- inner join

select c. customername, c. city, o. orderitem, o. price from customers c inner join orders o on (c. customerid=o. customerid);

/* output

customername   city          orderitem     price

Ravi            Hyderabad     Mobile        23500

Laya            Chennai       Laptop        64500

Ravi            Hyderabad     Shoe          2000

Nani            Vizag         Watch         4600

*/

select c. customername, o. orderitem, o. price,o.orderdate from customers c inner join orders o on (c.customerid=o.customerid);

select c.customername,c.contactno,c.city,o.orderitem from customers c inner join orders o on (c.customerid=o.customerid) where OrderItem='shoe';

select c. customername, c. contactno, c. city, o.orderitem from customers c inner join orders o on (c. customerid=o.customerid) where city='vizag';
```

-- left join

select c. customerid, c.customername,c.city,o.orderitem,o.price from customers c left join orders o on (c. customerid=o. customerid);

/* output

customerid	customername	city	order item	price
8241	Ravi	Hyderabad	Shoe	2000
8241	Ravi	Hyderabad	Mobile	23500
8242	Laya	Chennai	Laptop	64500
8243	Nani	Vizag	Watch	4600
8244	Bhavya	Bengaluru	NULL	NULL
8245	Sravya	Hyderabad	NULL	NULL

*/

select c. customername, o. orderitem,o.price,o.orderdate from customers c left join orders o on (c. customerid=o.customerid);

select c. customername, c.contactno,c.city,o.orderitem from customers c left join orders o on (c. customerid=o.customerid) where OrderItem='shoe';

select c.customername,c.contactno,c.city,o.orderitem from customers c left join orders o on (c.customerid=o.customerid) where city='vizag';

Assignment3: Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.

```
use joinsdb;

-- sub query a query inside another query

select c. customerid, c. customername, o. orderid, o. OrderItem, orderable from customers c inner join orders o on c. CustomerId=o.CustomerId

where orderable > (select avg (order value) from orders);

/* output

customerid      customername   orderid  OrderItem      order value

8242            Laya                    128            Laptop         17

8241            Ravi                    129            Shoe           25

*/

select c. customername, o. orderid, o. OrderItem, o. price from customers c inner join orders o on c. CustomerId=o.CustomerId

where o.price > (select min(price) from orders);

/* output

customername    orderid                 OrderItem      price
```

Ravi 121 Mobile 23500

Laya 128 Laptop 64500

Nani 130 Watch 4600

*/

-- union will execute two select queries

select c. customername, c. city, o. orderitem, o. price from customers c inner join orders o on (c. customerid=o. customerid)

union

select c. customername, c. contactno, city, o. orderitem from customers c cross join orders o on (c. customerid=o. customerid) where OrderItem='shoe';

/* output

customername city orderitem price

Ravi Hyderabad Mobile 23500

Laya Chennai Laptop 64500

Ravi Hyderabad Shoe 2000

Nani Vizag Watch 4600

*/

Assignment4: Compose SQL statements to BEGIN a transaction, INSERT a new record into the 'orders' table, COMMIT the transaction, then UPDATE the 'products' table, and ROLLBACK the transaction.

set auto commit=0;

start transaction;

insert into orders values(132,8246,'Table','2024-05-16',3000,4);

commit;

select * from orders;

/* output

OrderId CustomerId OrderItem Order Date Price ordervalue

121	8241	Mobile	2024-01-23	23500	5
128	8242	Laptop	2023-12-25	64500	17
129	8241	Shoe	2024-02-07	2000	25
130	8243	Watch	2024-02-16	4600	11
132	8246	Table	2024-05-16	3000	4

```
*/
 start transaction;
 update customers set customerid=8245 where customername='sravya';
select * from customers;
```

```
/* output
```

CustomerId	CustomerName	Contact No	city	email
8241	Ravi	9876543210	Hyderabad	ravi123@gmail.com
8242	Laya	9792742853	Chennai	laya54@gmail.com
8243	Nani	5333353664	Vizag	nani2002@gmail.com
8244	Bhavya	4544693930	Bengaluru	bhavya56@gmail.com
8245	Sravya	6836489456	Hyderabad	sravyasravs@gmail.com

```
*/
 rollback;
 -- after rollback the uncommitted actions will remain same
 select *from customers;
 /* output
```

CustomerId	CustomerName	contact No	city	email
3241	Sravya	6836489456	Hyderabad	sravyasravs@gmail.com
8241	Ravi	9876543210	Hyderabad	ravi123@gmail.com
8242	Laya	9792742853	Chennai	laya54@gmail.com
8243	Nani	5333353664	Vizag	nani2002@gmail.com
8244	Bhavya	4544693930	Bengaluru	bhavya56@gmail.com

*/

Assignment5: Begin a transaction, perform a series of INSERTs into 'orders', setting a SAVEPOINT after each, rollback to the second SAVEPOINT, and COMMIT the overall transaction.

```
set auto commit=0

start transaction;

insert into orders values(134,8246,'sofa','2024-03-15',300000,7);

save point sp1;

insert into orders values (135,8248,'Dressing table','2024-05-23',3000,8);

insert into orders values(140,8249,'TV','2024-05-23',45000,9);

save point sp2;


insert into orders values(138,8243,'fridge','2023-05-16',35000,15);

save point sp3;

select * from orders;
```

/* output

OrderId	CustomerId	OrderItem	Order Date	Price	ordervalue
121	8241	Mobile	2024-01-23	23500	5
128	8242	Laptop	2023-12-25	64500	17
129	8241	Shoe	2024-02-07	2000	25
130	8243	Watch	2024-02-16	4600	11
132	8246	Table	2024-05-16	3000	4
134	8246	sofa	2024-03-15	300000	7
135	8248	Dressing table	2024-05-23	3000	8

138 8243 fridge 2023-05-16 35000 15

139 8247 bag 2023-05-11 3000 2

140 8249 TV 2024-05-23 45000 9

*/

rollback to save point sp2;

select * from orders;

/*output

OrderId CustomerId OrderItem Order Date Price ordervalue

121 8241 Mobile 2024-01-23 23500 5

128 8242 Laptop 2023-12-25 64500 17

129 8241 Shoe 2024-02-07 2000 25

130 8243 Watch 2024-02-16 4600 11

132 8246 Table 2024-05-16 3000 4

134 8246 sofa 2024-03-15 300000 7

135 8248 Dressing table 2024-05-23 3000 8

139 8247 bag 2023-05-11 3000

commit;

Assignment6: Draft a brief report on the use of transaction logs for data recovery and create a hypothetical scenario where a transaction log is instrumental in data recovery after an unexpected shutdown.

Introduction:

Transaction logs are crucial components of database management systems that record all changes made to a database. These logs serve as a reliable source of information for recovering data in the event of system failures or unexpected shutdowns.

Importance of Transaction Logs:

1. Data Integrity: Transaction logs ensure data integrity by recording every transaction before it is committed to the database. This allows for rollbacks or recovery to a specific point in time.

2. Recovery Point: They provide a recovery point in case of system failures, allowing databases to be restored to a consistent state prior to the failure.

3. Performance Monitoring: Transaction logs also aid in performance monitoring and troubleshooting, as they track changes and can identify potential issues.

Hypothetical Scenario:

Imagine a scenario where a large e-commerce company experiences an unexpected server shutdown during a peak shopping period, resulting in potential data loss and customer disruption. However, due to the implementation of transaction logs, the company's database administrator can initiate a successful data recovery process.

Scenario Details:

1. Unexpected Shutdown: The e-commerce platform experiences a sudden server shutdown due to a power outage.

2. Data Loss Concerns: Concerns arise about potential data loss, including ongoing transactions and customer orders that were being processed.

3. Transaction Logs Utilization: The database administrator leverages transaction logs to restore the database to its state just before the shutdown.

4. Recovery Process: By analysing the transaction logs, the administrator identifies the last committed transactions before the shutdown.

5. Database Restoration: Using this information, the administrator restores the database to the point just before the unexpected shutdown, ensuring minimal data loss and maintaining data consistency.

6. Customer Impact Mitigation: The quick recovery minimizes disruption for customers, allowing them to resume their transactions seamlessly.

Conclusion:

Transaction logs play a vital role in data recovery, especially in scenarios of unexpected shutdowns or system failures. By maintaining a record of all database transactions, transaction logs enable organizations to restore data integrity and minimize downtime, ultimately ensuring business continuity and customer satisfaction.