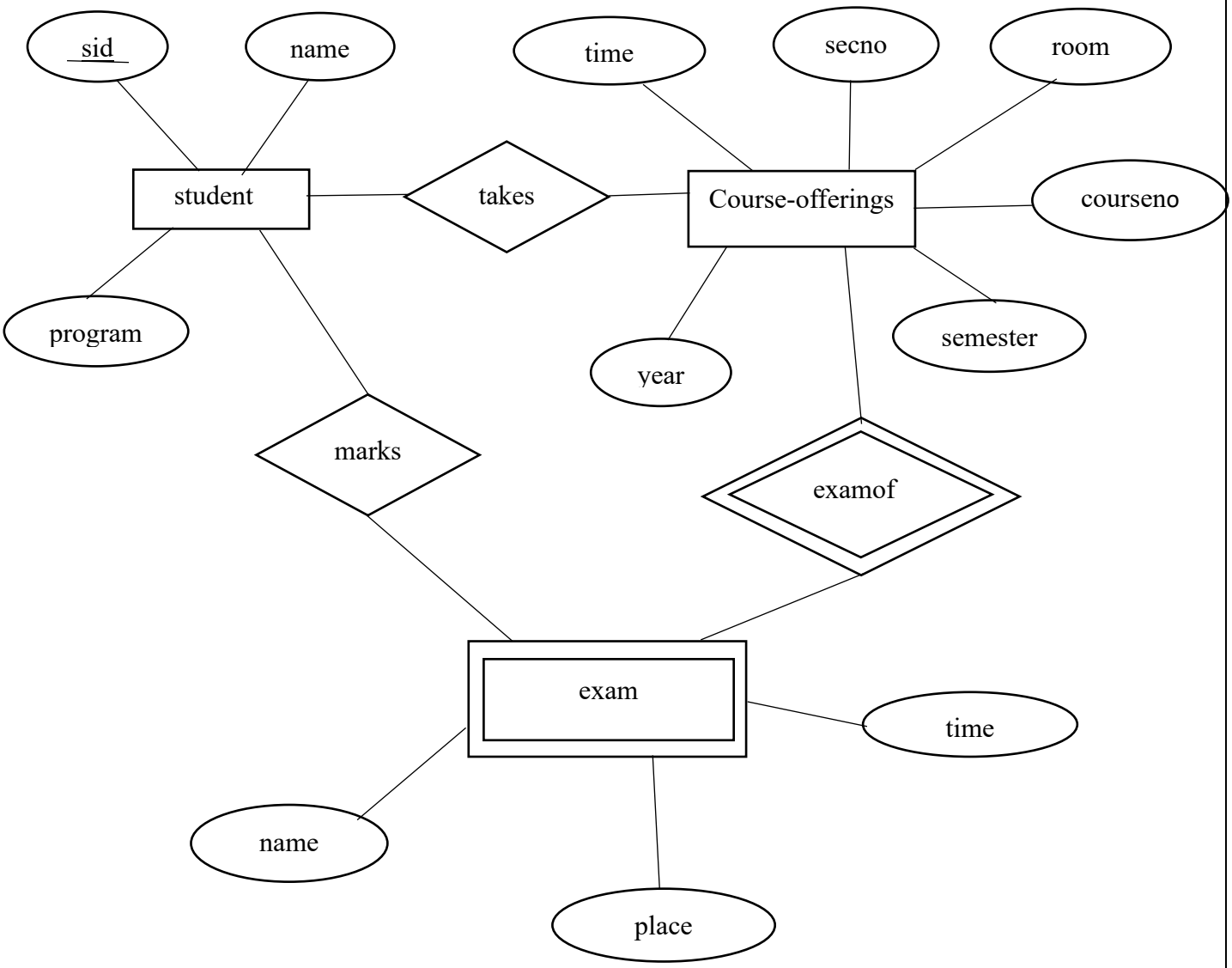
Day 1 Assignments

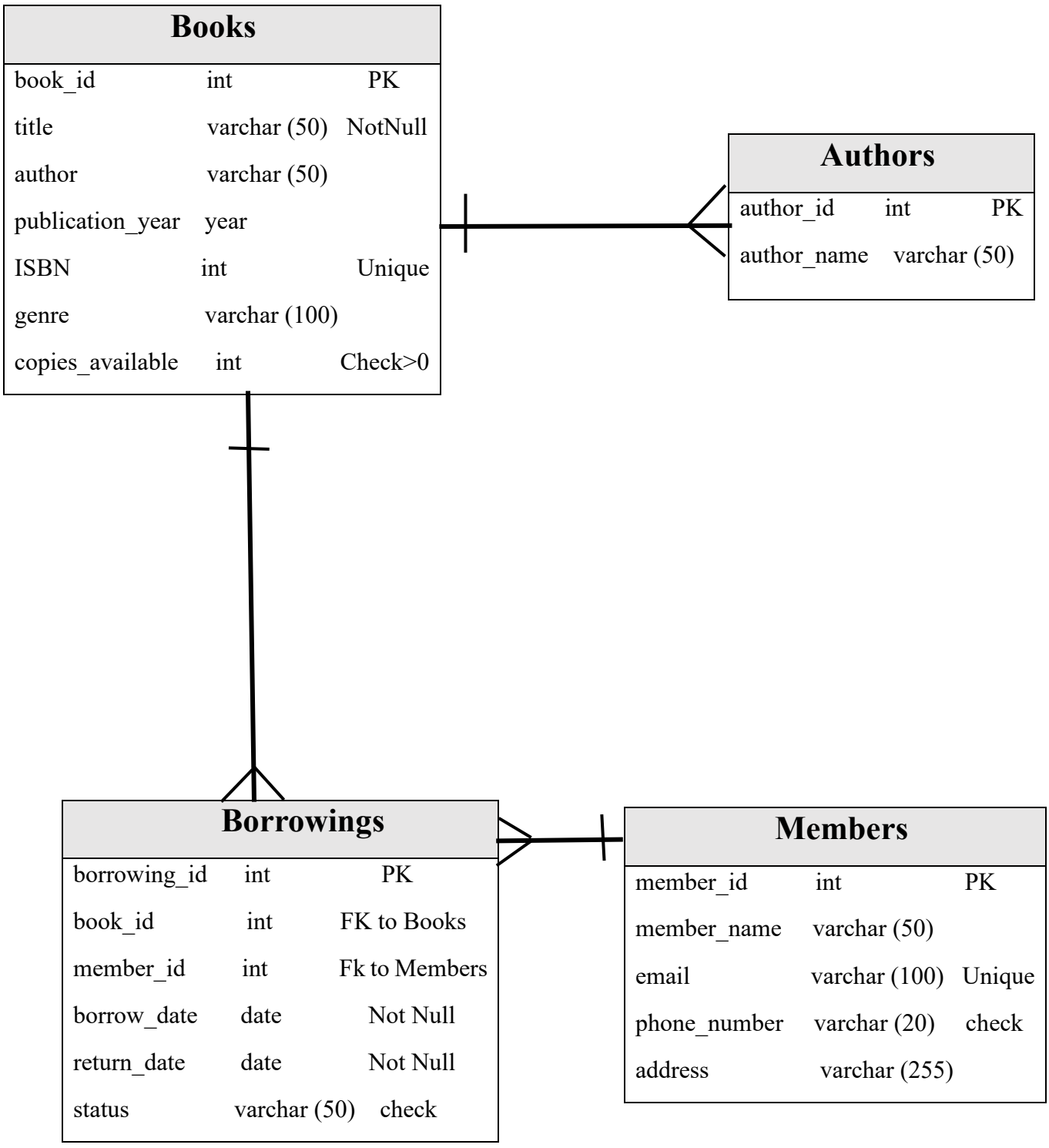
Assignment 1: Analyze a given business scenario and create an ER diagram that includes entities, relationships, attributes, and cardinality. Ensure that the diagram reflects proper normalization up to the third normal form.

E-R diagram that uses only a binary relationship between students and course-offerings. Make sure that only one relationship exists between a particular student and offering pair. Yet you can represent the marks that a student gets in different exams of a course offering.

E-R diagram for marks database



Assignment 2: 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.



Assignment 3: 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.

A transaction is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations. In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called ACID properties.

ACID Properties:

• **Atomicity:** A transaction is either fully executed or not executed at all. It follows the "all or nothing" rule.

• **Consistency:** After a transaction, the database remains in a consistent state. Integrity constraints are maintained.

• **Isolation:** Concurrent transactions do not interfere with each other. Changes made by one transaction are not visible to others until committed.

• **Durability:** Once a transaction is committed, its changes are permanent even in case of system failure.

SQL Statements for Transaction Simulation:

Let's consider a simple example where we transfer money from one account to another.

```
1. -- Assume we have two accounts: Account X and Account Y
2. BEGIN TRANSACTION;
3. -- Deduct 100 from Account X
4. UPDATE Accounts SET Balance = Balance - 100 WHERE AccountNumber = 'X';
5.  -- Add 100 to Account Y
6. UPDATE Accounts SET Balance = Balance + 100 WHERE AccountNumber = 'Y';
7. COMMIT; -- If successful, commit changes; otherwise, rollback
```

Assignment 4: 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_year     year          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_id    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;

Assignment 5: 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 analyze the impact on query execution.

Creating an Index: Improves query performance by allowing the database to quickly locate rows based on the indexed column(s).

Query Execution With Index: Uses index scans, which are generally faster, especially for large tables.

Dropping an Index: Reverts the database to full table scans for queries involving the dropped index column(s), resulting in slower query execution.

Indexes are a powerful tool to optimize query performance, but they should be used judiciously, as they also come with storage overhead and can impact the performance of write operations (INSERT, UPDATE, DELETE) due to the need to maintain the index.

use durgabhavani;

create index index_empname on employee_details(emp_id,emp_name);

explain select * from employee_details where emp_name='Radha';

explain select * from employee_details where emp_id>500;

drop index index_empname on employee_details;

explain select * from employee_details where emp_id>500;

Assignment 6: 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 durgabhavani identified by 'durga@123';

select user();

```
-- output
/* user()
root@localhost
*/

select user from mysql.user;
grant select,update,insert on mydatabase.employee to durgabhavani;
grant all on mydatabase.employee to durgabhavani;
revoke insert on mydatabase.employee from durgabhavani;
revoke update on mydatabase.employee from durgabhavani;
drop user durgabhavani;
```

New user

```
-- New user accesing table of root user
-- Grant all permissions
select user();
use mydatabase;
select * from employee;
insert into employee values(6,'Fhatima',52000,'Hyderabad');
alter table employee add primary key(empno);
update employee set empname='Anjali' where empno=6;
desc employee;

-- after revoke all privileges
insert into employee values(7,'Gowri',52000,'Chennai');
update employee set empname='Fhatima' where empno=6;
```

Assignment 7: 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,'ChetanBhagat'),(5749,'V.S.Naipaul'),(5750,'SatishGujral'),(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,'Anu','anu@123.com',7483328942,'Hyderabad'),(452,'Radha','radha@123.com',783539859,'Secundrabad'),(453,'Laya','laya@54.com',7937942983,'Madhapur');


delete from members where member_id=451;


update borrowings set status="submitted" where member_id=453;
```

Day 2 Assignments

Assignment 1: Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer name and email address for customers in a specific city.

```
use joinsdb;

select * from customers;

-- output
/*
 CustomerId   CustomerName  ContactNo        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         3434544693       Bengaluru        bhavya56@gmail.com
8245         Sravya         6836489456       Hyderabad         sravyasravs@gmail.com
*/


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

/* output
Customername        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
 */
```

Assignment 2: 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	orderitem	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';

Assignment 3: 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,o.ordervalue from customers c inner join orders o on c.CustomerId=o.CustomerId

where o.ordervalue > (select avg(ordervalue) from orders);

/* output

customerid	customername	orderid	OrderItem	ordervalue
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,c.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

*/

Assignment 4: 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 autocommit=0;
start transaction;
insert into orders values(132,8246,'Table','2024-05-16',3000,4);
 commit;
select * from orders;

/* output
```

OrderId	CustomerId	OrderItem	OrderDate	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	ContactNo	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 uncomitted actions will remain same

select *from customers;

/* output
```

CustomerId	CustomerName	ContactNo	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

```
*/
```

Assignment 5: 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 autocommit=0;
start transaction;
insert into orders values(134,8246,'sofa','2024-03-15',300000,7);
savepoint 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);
savepoint sp2;



insert into orders values(138,8243,'fridge','2023-05-16',35000,15);
savepoint sp3;
```

```
select * from orders;
/* output
```

OrderId	CustomerId	OrderItem	OrderDate	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 savepoint sp2;
select * from orders;

/*output
```

OrderId	CustomerId	OrderItem	OrderDate	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	2

```
commit;
```

Assignment 6: 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.

Report on the Use of Transaction Logs for Data Recovery

Introduction

Transaction logs are essential components of database management systems (DBMS). They record all changes made to the database, providing a reliable means to recover data in case of unexpected failures, such as system crashes or power outages. By maintaining a sequential record of all transactions, these logs ensure data integrity and consistency, facilitating both point-in-time recovery and rollback of transactions.

Functionality of Transaction Logs

1.Record Keeping: Transaction logs capture every change to the database, including insertions, updates, deletions, and the state of transactions (committed or uncommitted).

2.Atomicity and Durability: Ensuring the atomicity and durability of transactions, transaction logs guarantee that all operations within a transaction are completed successfully or none at all.

3.Recovery Mechanism: In the event of a failure, the DBMS uses the transaction log to identify the state of the database at the time of the crash and to replay or rollback transactions to restore the database to a consistent state.

Hypothetical Scenario: Data Recovery Using Transaction Logs

Scenario Overview

Imagine a financial services company, FinTech Corp, managing a critical database that stores transaction records for millions of users. This database processes high volumes of financial transactions daily, requiring robust mechanisms for data integrity and availability.

Incident Description

On a busy Monday morning, FinTech Corp experiences an unexpected power outage due to a severe thunderstorm. The outage lasts for several hours, and during this period, the database server shuts down abruptly, interrupting multiple ongoing transactions.

Recovery Using Transaction Logs

1.Initial Analysis:

- After power is restored, the IT team assesses the impact and discovers that several transactions were in progress at the time of the shutdown.
- The database did not get a chance to complete these transactions or ensure they were committed to the disk.

2.Transaction Log Utilization:

- The database management system (DBMS) automatically initiates the recovery process upon startup.
- The DBMS reads the transaction log to determine the last consistent state before the crash.
- The transaction log entries indicate which transactions were committed and which were still in progress when the power outage occurred.

3.Redo and Undo Operations:

- **Redo:** For transactions marked as committed in the transaction log but not yet written to the database files, the DBMS re-applies these changes to ensure all committed transactions are reflected in the database.
- **Undo:** For transactions that were in progress but not committed, the DBMS rolls back these changes to maintain consistency. This involves reverting any partial updates made by these transactions.

4.Verification and Consistency Check:

- After applying the redo and undo operations, the DBMS performs a consistency check to ensure the integrity of the database.
- The database is brought back to a consistent state, with all committed transactions applied and uncommitted transactions rolled back.

Conclusion

Transaction logs are indispensable for database recovery in the event of unexpected failures. By meticulously recording each transaction and maintaining the order of operations, these logs enable DBMSs to restore databases to a consistent state, ensuring data reliability and continuity of operations even under adverse conditions.