Question#01 [CLO 2] **Marks: 8**

A. Highlight the differences between basic, conservative, strict, and rigorous two-phase locking protocols.

Basic 2PL

Expanding (growing) phase

New locks can be acquired but none can be released

Lock conversion upgrades must be done during this phase

Shrinking phase

Existing locks can be released but none can be acquired

Downgrades must be done during this phase

Conservative 2PL

Lock all items before the start of the transaction.

Strict 2PL

Do not unlock before write commit

Rigorous 2PL

Do not unlock before bothe read and write commit

B. Write down a schedule that must comprise three transactions i.e., T1, T2, and T3 with different read and write operations. Convert the schedule into a rigorous two-phase locking schedule.

The answer varies depending on the choice of T1,T2, and T3.

The rigorous 2PL definition is defined in question 1 solution. The schedule should follow the rules.

Question #02 [CLO 2] **Marks: 6**

Explain the levels of isolation in a transaction and specify which isolation level resolves each concurrency problem, such as lost update, temporary update, and incorrect summary problems.

Solution

	Type of Violation		
Isolation Level	Dirty Read	Nonrepeatable Read	Phantom
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

Question #03 [CLO 2] **Marks: 8**

Write down an algorithm for testing the serializability of a schedule. Also, give an example of a conflict serializable schedule.

Solution

1. For each transaction T_i participating in schedule S, create a node labeled T_i in the precedence graph.
2. For each case in S where T_j executes a read_item(X) after T_i executes a write_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
3. For each case in S where T_j executes a write_item(X) after T_i executes a read_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
4. For each case in S where T_j executes a write_item(X) after T_i executes a write_item(X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.
5. The schedule S is serializable if and only if the precedence graph has no cycles.

The example can be different. The example must follow the algorithm.

Question #04 [CLO 3] **Marks: 7.5**

Pakwheels car company manages the production and sales of various car models. The company keeps track of information such as car models, production quantities, sales records, and customer details. Here are some details of the scenario:

- The company has a table named "CarModels" that stores information about different car models, including their unique model IDs, names, and prices.
- The company maintains a table named "Production" to track the production details of each car model. The table contains the model ID, production date, and the quantity of cars produced on that date.
- The company has a table called "Sales" to record the sales transactions. It includes information such as the sales ID, customer ID, car model ID, date of sale, and the quantity of cars sold.
- The company keeps a table name "Customers" to store customer information. It contains details like customer ID, name, address, and contact information.

Write down the relational algebraic expressions for the following:

A. Retrieve the name and prices of all car models.
   π (name, price) (CarModels)
B. Find the total quantity of cars produced fo a specific model.
   γ (SUM(quantity)) (σ (model_id = "specific_model_id") (Production))
C. Determine the total revenue generated from car sales.
   γ (SUM(price * quantity)) (Sales ⋈ CarModels ⋈ Production)
D. Find the car models that have not been sold yet.

π (model_id, name, price) (CarModels - π (model_id, name, price) (Sales ⋈ CarModels))

E. Determine the average price of the sold cars.

γ (AVG(price)) (Sales ⋈ CarModels)

Question #05 [CLO 3] **Marks: 7.5**

Consider the scenario of Pakwheels (mentioned in Question #05). Write down the SQL statements for the following:

A. Retrieve the details of all the customers who have made a purchase.

```
SELECT *
FROM Customers
WHERE customer_id IN (SELECT DISTINCT customer_id FROM Sales);
```

B. Find the car models that have been sold more than 20 times

```
SELECT cm.*
FROM CarModels cm
INNER JOIN Sales s ON cm.model_id = s.model_id
GROUP BY cm.model_id
HAVING COUNT(*) > 20;
```

C. Retrieve the customer details who have made purchases above the average purchase amount

```
SELECT c.*
FROM Customers c
INNER JOIN (
   SELECT customer_id, AVG(quantity * price) AS avg_purchase_amount
   FROM Sales
   GROUP BY customer_id
) s ON c.customer_id = s.customer_id
WHERE (quantity * price) > avg_purchase_amount;
```

D. Find the car models with the highest and lowest sales quantity

```
SELECT cm.*, s.total_quantity
FROM CarModels cm
INNER JOIN (
   SELECT model_id, SUM(quantity) AS total_quantity
   FROM Sales
   GROUP BY model_id
) s ON cm.model_id = s.model_id
ORDER BY s.total_quantity ASC
LIMIT 1;
```

E. Retrieve the top 5 customers with the highest total purchase amount.

```
SELECT c.customer_id, c.name, c.address, c.contact_info, SUM(s.quantity * s.price) AS
total_purchase_amount
FROM Customers c
INNER JOIN Sales s ON c.customer_id = s.customer_id
GROUP BY c.customer_id, c.name, c.address, c.contact_info
ORDER BY total_purchase_amount DESC
LIMIT 5;
```

Question #06 [CLO 3] **Marks: 6.5**

Computer Zone manages their customer's orders. Their system has a single table called "OrderDetails" to store information about each order as shown in Table 01. They are facing data redundancy and anomalies in the current table design. As a database administrator, you have been tasked with normalizing the "OrderDetails" table up to the third normal form (3NF).

Table 01: Order Details of Computer Zone Store

Order ID	Customer ID	Customer Name	Customer Address	Product ID	Product Name	Product Price	Quan -tity	Order Date	Shipping Address
1	101	Ali	Karachi	201	Laptop	125K	2	18-5-20203	Karachi
2	102	Ahmed	Lahore	202	RAM	15K	1	19-5-2023	Lahore
3	103	Alyan	Quetta	203	Head-phones	10K	3	20-5-2023	Quetta

Solution

Table: Customers

CustomerID **CustomerName** **CustomerAddress**

Table: Products

ProductID **ProductName** **ProductPrice**

Table: Orders

OrderID	CustomerID	OrderDate	ShippingAddress

Table: OrderItems

OrderID	ProductID	Quantity

Question #07: [CLO 1] **Marks: 6.5**

Suppose you are designing a database system for a video game company. The company develops and publishes various video games across different platforms. Players can create accounts, purchase games, join multiplayer sessions, and communicate with other players. The company also keeps track of game reviews and ratings. Design an Entity-Relationship Diagram (ERD) to represent the following requirement specifications of the given scenario:

The system is comprised of a Game with its attributes as game_id, title, genre, platform, and release date. The Player information consists of player_id, username, email, password, and registration date. The Purchase details have purchase_id, player_id, and gae_id. The Session information includes session_id, game_id, player_id, start_time, and end_time. The Reviews are represented as review_id, game_id, player_id, rating, and comments.

The relationship among the above entities can be defined as follows:

- A game can have multiple purchases and multiple reviews
- A player can make multiple purchases, write multiple reviews, and participate in multiple sessions.

- A purchase is made by a single player and can be associated with a single game.
- A session is associated with a single game and a single player.
- A review is written by a single player for a single game.

Solution

- Game
  - GameID (primary key)
  - Title
  - Genre
  - Platform
  - ReleaseDate
- Player
  - PlayerID (primary key)
  - Username
  - Email
  - Password
  - RegistrationDate
- Purchase
  - PurchaseID (primary key)
  - PlayerID (foreign key referencing Player)
  - GameID (foreign key referencing Game)
  - PurchaseDate
- Session
  - SessionID (primary key)
  - GameID (foreign key referencing Game)
  - PlayerID (foreign key referencing Player)
  - StartTime
  - EndTime
- Review
  - ReviewID (primary key)
  - GameID (foreign key referencing Game)
  - PlayerID (foreign key referencing Player)
  - Rating
  - Comment

Relationships:

- Game (1) to (N) Purchase
- Game (1) to (N) Review
- Player (1) to (N) Purchase
- Player (1) to (N) Review
- Player (1) to (N) Session

- Game (1) to (N) Session

Good Luck!