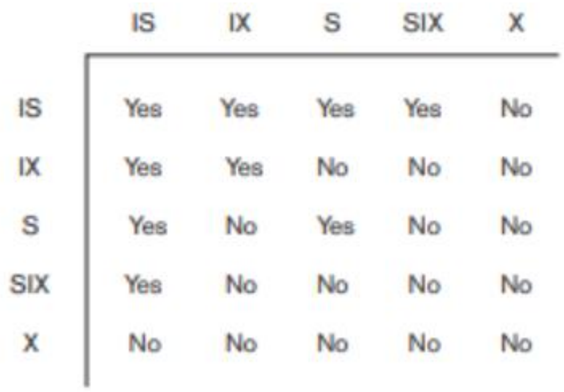
Q1:- The compatibility matrix of figure shows that IS and IX locks are compatible. Explain why this is valid.



	IS	IX	S	SIX	X
IS	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No
S	Yes	No	Yes	No	No
SIX	Yes	No	No	No	No
X	No	No	No	No	No

In database systems, compatibility between locking mechanisms refers to the ability of different types of locks to work together without causing deadlocks or interference with each other's operations. The compatibility matrix determines which locks can coexist peacefully without conflicting.

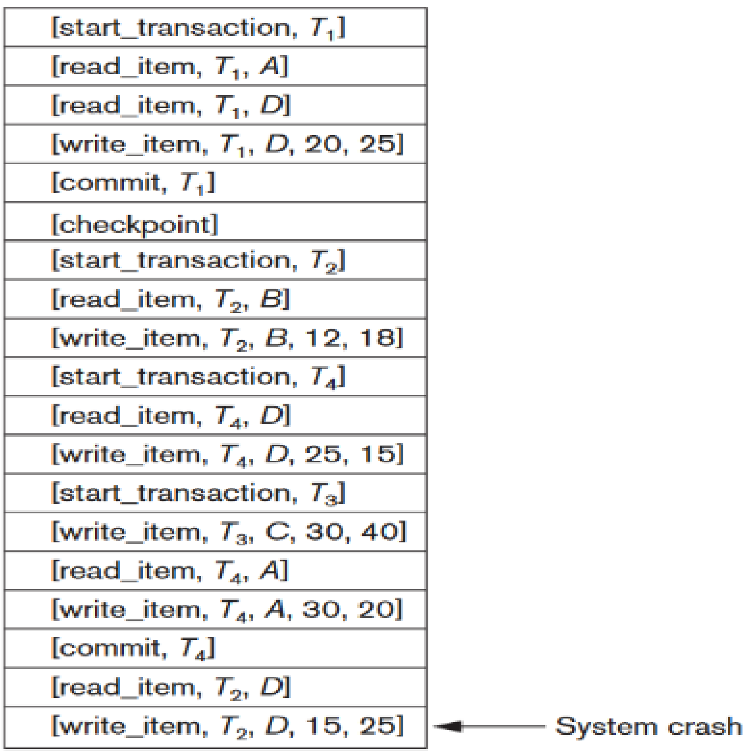
The compatibility between IS (Intent Share) and IX (Intent Exclusive) locks is typically valid due to their nature and the purpose they serve in a database locking system-

1. **Intent Locks**: Both IS and IX locks are intent locks, meaning they are used to signal an intent to acquire locks at a higher level of granularity (e.g., table or page level). They are not used for actually locking data but to indicate the intention to lock certain resources.
2. **Intent Share (IS) Locks**: These locks are used to indicate that a transaction intends to read data at a lower level of granularity, such as a row or page, in a shared mode. Multiple transactions can hold IS locks simultaneously, allowing them to read the same data concurrently without conflicting.
3. **Intent Exclusive (IX) Locks**: These locks signal an intent to modify data at a lower level of granularity. They are compatible with IS locks because while they indicate an intention to modify data, they do not prevent other transactions from reading that data. Multiple transactions can hold IX locks on the same resource concurrently.
4. **Compatibility Criteria**: IS and IX locks are compatible because they serve different purposes - IS for shared read access and IX for intent to modify - and they do not conflict in terms of access rights. An IS lock indicates that a transaction wants to read data, while an IX lock

indicates a transaction's intent to modify data. These intentions can coexist without causing deadlock situations or blocking each other.

Therefore, in the compatibility matrix, IS and IX locks are marked as compatible because their intentions—read and intent to modify—can coexist harmoniously without creating conflicts or blocking each other in a way that would cause deadlocks or concurrency issues.

Q2:- Figure 22.6 shows the log corresponding to a particular schedule at the point of a system crash for four transactions T1 , T2 , T3, and T4 . Suppose that we use the immediate update protocol with checkpointing. Describe the recovery process from the system crash. Specify which transactions are rolled back, which operations in the log are redone and which (if any) are undone, and whether any cascading rollback takes place.



[start_transaction, T_1]	
[read_item, T_1, A]	
[read_item, T_1, D]	
[write_item, T_1, D, 20, 25]	
[commit, T_1]	
[checkpoint]	
[start_transaction, T_2]	
[read_item, T_2, B]	
[write_item, T_2, B, 12, 18]	
[start_transaction, T_4]	
[read_item, T_4, D]	
[write_item, T_4, D, 25, 15]	
[start_transaction, T_3]	
[write_item, T_3, C, 30, 40]	
[read_item, T_4, A]	
[write_item, T_4, A, 30, 20]	
[commit, T_4]	
[read_item, T_2, D]	
[write_item, T_2, D, 15, 25]	$\leftarrow$ System crash

Immediate Update Protocol- In immediate update protocols, changes made by a transaction are immediately written to the database, reducing the risk of lost updates upon a system crash. The recovery process involves a combination of undoing uncommitted changes and redoing committed changes.

Checkpointing- Periodic checkpoints create a stable point in the transaction log and database, aiding recovery. During checkpointing:

- All modified buffers are forced to disk.
- A record of transactions active at the time of checkpointing is written to the log.
- A checkpoint record is written to the log to mark the completion of the checkpoint**.**

The immediate update recovery with checkpoint procedure involves steps to recover a database after a system crash using immediate update protocol and periodic checkpointing. This method ensures that the database remains consistent and durable despite a crash. Here are the steps involved-

1. **Identify Checkpoint**: After a crash, the recovery process begins by identifying the most recent checkpoint in the log before the crash occurred. This provides a stable starting point for recovery.
2. **Analysis Phase:**
   - **Identify Committed Transactions**: Analyse the log to identify transactions committed before the checkpoint. Transactions committed before the checkpoint are considered durable and don't need rollback.
   - **Identify Uncommitted Transactions:** Determine which transactions were active but not committed before the checkpoint. These transactions need to be rolled back.
3. **Rollback Uncommitted Transactions:**
   Undo operations of uncommitted transactions by applying the log records in reverse order from the most recent checkpoint until the crash point.
4. **Redo Phase:**
   - **Reapply Committed Transactions:** Redo the operations of committed transactions that occurred after the last checkpoint up to the point of the crash. This involves applying changes recorded in the log to the database to ensure durability.
5. **Complete Recovery:**
   Once rollback and redo operations are completed, the database should be in a consistent state, reflecting only the changes made by committed transactions after the last checkpoint.
6. **Reset Checkpoint:**

Update the checkpoint record in the log to reflect the completion of the recovery process.

7. A cascading rollback occurs when the undo operations for an aborted (or uncommitted) transaction affect other transactions that have already committed and interacted with the data modified by the aborted transaction.

About the Example-

- In the given example T1 is committed before the checkpoint so it does not need to be rolled back.
- There is no uncommitted active transaction before the checkpoint.
- T2 and T3 are undone starting from crash point up to recent checkpoint since they are not committed.
- T4 is redone since it is committed after the checkpoint.
- No cascading rollback takes place here since T4 operated on D and A data items which are not modified by T2 and T3.

Q3:- Compare inheritance in the EER model to inheritance in the OO model.

Inheritance in the Enhanced Entity-Relationship (EER) model and Object-Oriented (OO) model represents a way to establish relationships between entities or classes, allowing for the reuse of attributes, methods, or properties among related entities or classes. However, there are significant differences in how inheritance is implemented and utilized in these two models-

Enhanced Entity Relationship Model-

1. **Conceptual Modeling:** EER is primarily used in database design for conceptual modeling. Inheritance in the EER model is represented through subtypes and supertypes.
2. **Hierarchy Structure:** In EER, entities are organized into a hierarchy where the subtypes inherit attributes and relationships from their supertypes.
3. **Specialization and Generalization:** EER model uses concepts of specialization and generalization to describe the inheritance relationships. Specialization involves defining specific entities based on the common attributes or relationships of a general entity (generalization).

4. **Subtype Inheritance:** Subtypes inherit attributes and relationships from their supertype. This allows for the reuse of common attributes and relationships within the hierarchy.
5. **Constraints:** EER allows defining constraints such as disjointness (where an entity can belong to only one subtype) and completeness (whether all entities must belong to a subtype).

Object Oriented Model-

1. **Programming Paradigm:** Object-Oriented models are used in software development and programming languages for building software systems using classes and objects.
2. **Class Inheritance**: Inheritance in the OO model is achieved through classes. A class can inherit attributes and methods from a superclass (base class).
3. **Code Reusability:** Inheritance facilitates code reusability by allowing subclasses to inherit common functionality (attributes and methods) from their superclass.
4. **Polymorphism:** OO inheritance often involves concepts like polymorphism, where a subclass can override or extend the behavior of methods inherited from its superclass.
5. **Dynamic Binding:** OO languages typically support dynamic binding, allowing objects to be treated as instances of both their own class and their superclass, enabling runtime flexibility.

Comparison-

- EER inheritance focuses on data modeling and database design, while OO inheritance is geared towards code organization and reuse in software development.
- EER uses entity hierarchies with subtypes and supertypes, while OO uses class hierarchies with subclasses and superclasses.
- EER models allow for the definition of constraints related to subtypes, while OO models emphasize the flexibility of inheritance and polymorphism.
- While both models leverage inheritance for reuse and abstraction, they operate within different contexts and are optimized for distinct purposes: EER for data modeling and OO for software development.

Q4:- Consider the AIRPORT database described in Exercise 4.21. Specify a number of operations/methods that you think should be applicable to that application. Specify the ODL classes and methods for the database.

To define the operations/methods and ODL (Object Definition Language) classes for the AIRPORT database, we can consider the following:

ODL Classes-

1. **Airport**: Attributes might include airport_code, name, location, and capacity.

2. **Flight**: Attributes could include flight_number, departure_time, arrival_time, departure_location, and arrival_location.

3. **Passenger**: Attributes could include passenger_id, name, age, and seat_number.

Relevant methods/operations for these classes could include:

Airport Class Methods-

1. **viewFlights():** Displays all flights associated with the airport.

2. **viewPassengers()**: Displays all passengers at the airport.

3. **updateCapacity():** Updates the capacity of the airport.

Flight Class Methods-

1. **updateTime()**: Modifies the departure or arrival time of the flight.

2. **addPassenger()**: Adds a passenger to the flight manifest.

3. **removePassenger()**: Removes a passenger from the flight manifest.

Passenger Class Methods-

1. **bookFlight()**: Books a flight for the passenger.

2. **cancelFlight()**: Cancels a booked flight for the passenger.

3. **updateDetails()**: Allows the passenger to update their personal information.

These methods and classes would provide a basic framework to operate the AIRPORT database effectively, allowing for functionalities like managing flights, passengers, and airport information efficiently.

Q5:- Consider a distributed database for a bookstore chain called National Books with three sites called EAST, MIDDLE, and WEST. The relation schemas are given above. Consider that BOOKS are fragmented by $price amounts into:

- B1: BOOK1: $price up to $20
- B2: BOOK2: $price from $20.01 to $50
- B3: BOOK3: $price from $50.01 to $100
- B4: BOOK4: $price $100.01 and above

Similarly, BOOK_STORES are divided by zip codes into:

- S1: EAST: Zip up to 35000
- S2: MIDDLE: Zip 35001 to 70000
- S3: WEST: Zip 70001 to 99999

Assume that STOCK is a derived fragment based on BOOKSTORE only.

a. Consider the query:

SELECT Book#, Total_stock

FROM Books

WHERE $price &gt; 15 AND $price &lt; 55;

Assume that fragments of BOOKSTORE are non-replicated and assigned based on region.

Assume further that BOOKS are allocated as:

- **EAST: B1, B4**
- **MIDDLE: B1, B2**
- **WEST: B1, B2, B3, B4**

Assuming the query was submitted in EAST, what remote subqueries does it generate? (Write in SQL.)

b. If the price of Book# = 1234 is updated from $45 to $55 at site MIDDLE, what updates does that generate? Write in English and then in SQL.

c. Give a sample query issued at WEST that will generate a subquery for MIDDLE.

d. Write a query involving selection and projection on the above relations and show two possible query trees that denote different ways of execution.

a. Remote subqueries-
   - At East:
     SELECT B.title, BS.name
     FROM BOOKS B, BOOKSTORE BS
     WHERE B.book_store_id = BS.book_store_id AND B.price < 50
   - At Middle:
     SELECT B.title, BS.name
     FROM BOOKS_Middle B, BOOKSTORE_Middle BS
     WHERE B.book_store_id = BS.book_store_id AND B.price < 50
   - At West:
     SELECT B.title, BS.name
     FROM BOOKS_West B, BOOKSTORE_West BS
     WHERE B.book_store_id = BS.book_store_id AND B.price < 50

b. Updates Generated by Price Change of Book#= 1234 at MIDDLE:
   UPDATE BOOKS_Middle
   SET price = 55
   WHERE book_id = 1234

c. Sample Query Issued at WEST Generating a Subquery for MIDDLE:

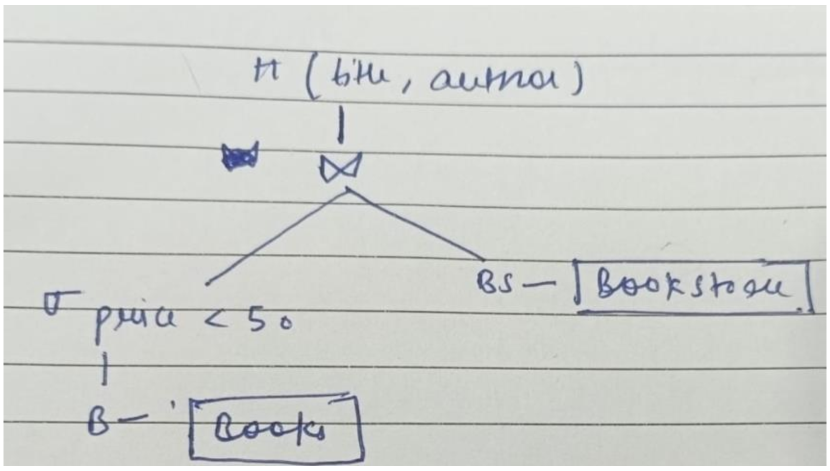
```
SELECT B.title, BS.name
FROM BOOKS_West B, BOOKSTORE_West BS
WHERE B.book_store_id = BS.book_store_id AND BS.zip_code IN
(
   SELECT zip_code
   FROM BOOKSTORE_Middle
);
```

d. Query Involving Selection and Projection:

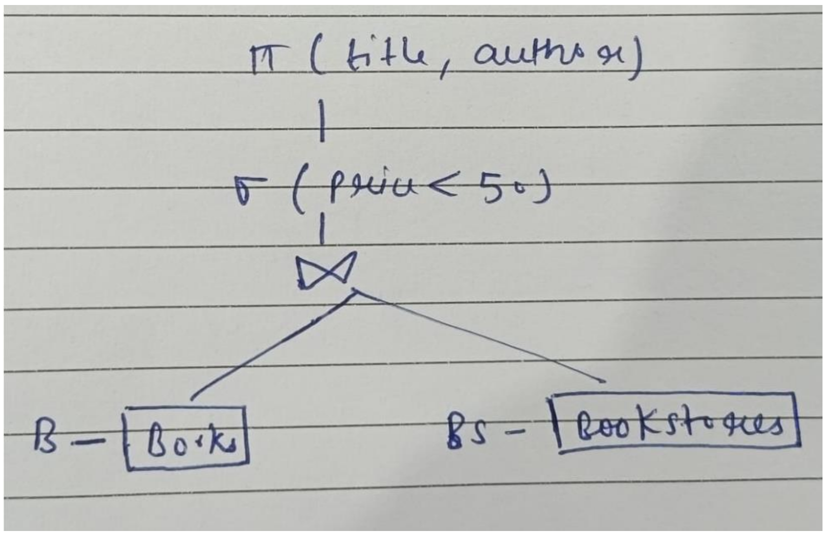
```
SELECT B.title, B.author, BS.name
FROM BOOKS B, BOOKSTORE BS
WHERE B.book_store_id = BS.book_store_id AND B.price < 50;
```

Query Trees: -

1. Selection before Join:
    - Select books with price < $50 from BOOKS.
    - Join the selected books with BOOKSTORE.



2. Join before Selection:
    - Join all books with BOOKSTORE.
    - Select books with price < $50 from the result of the join.


π (title, author)
σ (price < 50)
⋈
B — Books
BS — Bookstores