Testing of Serializability

Serializability graph is used to test the serializability of a schedule.

For testing of Serializability the simple and efficient method is to construct a directed graph called a precedence graph from s

$$G = (V, E)$$

where V consists of set of vertices and E consists of a set of edges. The set of vertices is used to contain all the transactions participating in the schedule.

The set of edges is used to contain all edges $T_i \rightarrow T_j$ for which one of the three conditions holds.

1. Create a node $T_i \rightarrow T_j$ if T_i executes write (Q) before T_j executes read (Q).
2. Create a node $T_i \rightarrow T_j$ if T_i executes read (Q) before T_j executes write (Q).
3. Create a node $T_i \rightarrow T_j$ if T_i executes write (Q) before T_j execute write (Q).

Precedence graph for schedule S

b A precedence graph contains a single edge $T_i \rightarrow T_j$ then all the instruction of T_i are executed before the first instruction of T_j is executed.

↳ A precedence graph for schedule S, contains a cycle, then S is non serializable. If the precedence graph has no cycle, then S is known as serializable.

For ex.

T_1	T_2	T_3
Read (A)		
	Read (B)	
$A := f_1(A)$		
		Read (C)
	$B := f_2(B)$	
	Write (B)	
		$C := f_3(C)$
		Write (C)
Write (A)		
		Read (B)
	Read (A)	
	$A := f_4(A)$	
Read (C)		
	Write (A)	
$C := f_5(C)$		
Write (C)		
		$B := f_6(B)$
		Write (B).

Schedule S1

Explanation :

Read (A)

↳ In T_1, no subsequent writes to A, so no new edges.

Read (B)

↳ In T_2, no subsequent writes to B, so no new edges.

Read (C)
↳ In T_3, no subsequent write to C, so no new edges.

Write (D)
↳ D is subsequently read by T_3, so add edge $T_2 \rightarrow T_3$.

Write (C)
↳ C is subsequently read by T_1, so add edge $T_3 \rightarrow T_1$

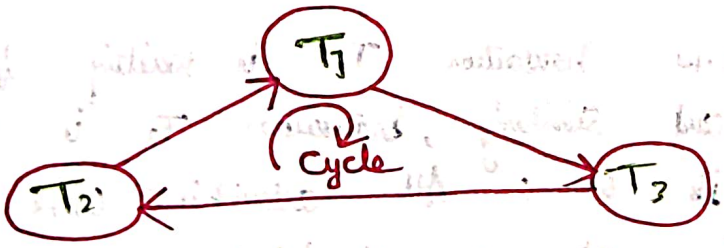
Write (A)
↳ A is subsequently read by T_2, so add edge $T_1 \rightarrow T_2$

Write (A)
↳ In T_2 no subsequent reads to A so no new edges.

Write (C)
↳ In T_1 no subsequent read to B so no new edges.

Write (B)
↳ In T_2 no subsequent read to B so no new edges

Precedence graph for schedule S1 :



The precedence graph for schedule S1 contains a cycle thats why schedule S1 is non Serializable.

When it does not form any cycle then the schedule is Serializable.

Deadlock in DBMS.

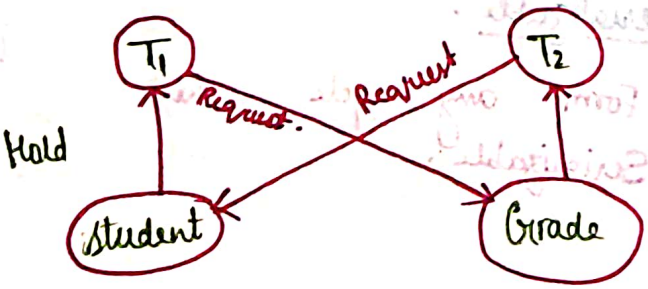
↳ A deadlock is a condition where two or more transaction are waiting indefinitely for one another to give up locks.

↳ A System is said to be in deadlock state when in a set of transaction, Every transaction is waiting for another transaction to finish its operations.

↳ A deadlock is a state of statements that may result when two or more transaction are each waiting for locks held by the other to be released.

For Example:- In the student table transaction T_1 holds a lock on some rows and needs to update some rows in the grade table. Simultaneously, transaction T_2 holds locks on some rows in the grade table and needs to update the rows in the student table held by transaction T_1

Now the main Problem arises Transaction T_1 is waiting for T_2 to release its locks and Similarly, transaction T_2 is waiting for T_1 to release its lock: All activities come to a halt state and remain at a standstill.



"Jai Shree Ram"

(1)

Aspect	1NF	2NF	3NF	BCNF	4 NF	5 NF
Focus.	Eliminates repeating groups	Remove partial dependencies	Remove transitive dependencies	ensures every determinant is a candidate key.	Remove multi-valued dependencies	Remove join dependencies
Key Requirement	Yes	Yes	Yes	Yes	Yes	Yes.
Attribute Requirement	Atomic Attributes	No Partial dependency on any candidate key	No transitive dependency on any candidate key	every non trivial functional dependency is a candidate key.	No multi valued dependency exists.	Each join dependency is implied by candidate key.
Partial Dependency.	Allowed	Not Allowed	Not Allowed	Not Allowed	Not allowed	Not Allowed.
Transitive dependency	Allowed	Allowed	Not Allowed	Not Allowed	Not Allowed	Not Allowed.

Multi-valued dependency	Allowed	Allowed	Allowed	Allowed	Not Allowed	Not Allowed.
Join dependency	Allowed	Allowed	Allowed	Allowed	Allowed	Not Allowed
Example Scenario	Table with repeating groups, atomic columns	Table where each non-key column is dependent on the table	Table with no transitive dependency b/w non-key columns	Every table is in 3NF and each non-trivial FD is a candidate key.	Table with no non-trivial multivalued dependency	Table with no non-trivial join dependency.
Additional Notes	Simplest form focuses on atomicity.	Build on 1NF eliminates Partial dependency.	Build on 2NF, eliminates transitive dependency.	Stronger version of 3NF.	Further builds on BCNF	ensures tables are decomposed without loss of data.

2. In relational model what do you mean by Cardinality ?

Ans. In the Relational model of DB. Cardinality refers to the no. of elements in a set or the no. of rows in a table.

↳ Cardinality refers to two different contents.

1. Cardinality of a Table

↳ This refers to the no. of rows (tuple) in a table.

For eg. If a table Students has 100 rows the Cardinality of the table is 100.

11. Cardinality of Relationships.

↳ This refers to the uniqueness of data values contained in a particular column (attribute) of a table. and more commonly to the relationships b/w tables.

↳ The relationship Cardinality specifies the no. of instances of one entity that can or must be associated with each instance of another entity.

Thy types of Cardinality

(i) one to one (1:1)

(ii) one to Many (1:N)

(iii) Many to one (N:1)

(iv) Many to Many (N:N)

Q3 How can you map a conceptual model to a relational model. ?

Ans. Mapping a conceptual model (often represented as an Entity - Relationship Diagram, or ERD) to a relational model involves converting entities, relationship and attributes into table column and keys.

Step by step mapping ...

1. entities to tables.

↳ Each entity in the conceptual model becomes a table in the relational model.

↳ attributes of the entity become column.

↳ PK of the entity become PK of the table.

2. Attributes to Column

↳ Attribute to column.

↳ include appropriate data types for each column.

↳ If an attribute is a multi valued attribute

create a separate table for it.

3. Relationship to Foreign keys

⇒ One to one Relationship

↳ Add the PK of one entity as foreign key in the related entity table.

↳ One to many

↳ Add the PK of the "one" side entity as a foreign key in the "many" side entity's table.

↳ Many to Many

↳ Create a new table to represent the relationship.

↳ This new table will have foreign keys referencing the PK of the related entities.

↳ PK of this new table can be composite key made up of foreign keys.

4. Weak entities

↳ Convert weak entities into tables.

↳ Include a FK column that references the PK of the owning entity table.

The PK of weak entity table be typically a composite key consisting of its own identifier and the foreign key.

5. Composite Attributes

- Break down composite attributes into individual attributes and map them to separate columns.

6. Multi - Valued Attributes

- Create a new table for multi valued Attribute.
- This new table will include a FK column referencing the PK of the original entity.
- The PK of the new table will usually be a composite key made up of the FK and the multi valued attribute.

Q4. List two reason why we may choose to define a view.

Ans.

1. Data security and Access control

↳ view can be used to restrict access to specific data in a DB, thereby enhancing security. By granting users access to views rather than base tables, you can control what data users see and manipulate.

For instance

↳ Sensitive data Protection

↳ you can create a view that excludes sensitive columns (eg. salary information) so users can access non-sensitive data without compromising privacy.

↳ Simplified Access Control

↳ Instead of setting complex permissions on each table, you can grant access to views which can encapsulate multiple permission in one place.

Q. Simplification and Abstraction of complex Queries.

↳ Views can simplify complex Queries by encapsulating them, making it easier for user to retrieve and manipulate data without having to understand or write complex SQL.
This abstraction can lead to more readable and maintainable code.

Concurrency

Q. What is Phantom Phenomenon?

Ans. ↳ It is a specific type of concurrency anomaly that can occur in DB system which transaction are executed concurrently. It happens when a transaction reads a set of rows that satisfy a certain condition, but a subsequent read within the same transaction returns additional rows that were inserted by another concurrent transaction.

Q6) Which protocol always ensures recoverable schedule?

Ans. To ensure a recoverable schedule in DB system protocols are designed to manage transaction concurrency and maintain consistency.

One such protocol.

1. Strict Two-Phase locking (strict 2PL) Protocol.

↳ strict 2PL is a specific type of the two Phase locking (2PL) protocol that ensures a recoverable schedule by imposing additional constraints on how locks are released.

How it works.

(i) lock Acquisition Phase :-

- During this Phase, a transaction can acquire any no. of locks (read or write) it needs, but it cannot release any locks.

(ii) lock Release Phase

↳ once a transaction releases its first lock, it cannot acquire any new locks.

↳ In the strict version of 2PL, all locks (both read and write) held by a transaction are only released when the transaction is committed or aborted. This ensures that no other transaction can read or write any data modified by the transaction until it is finished, preventing cascading rollbacks and ensuring a recoverable schedule.

* Properties of strict 2PL.

↳ Prevents Cascading Rollbacks

↳ Because locks are only released at the end of a transaction, no other transaction can access the modified data until the transaction is complete. This ensures that if a transaction fails, no other transaction has read the uncommitted data, avoiding cascading rollbacks.

↳ Ensures Recoverable schedules

↳ Since transaction cannot read uncommitted data, any schedule produced under strict 2PL will be recoverable. A transaction can only commit if all the transactions whose change it has read have also committed.

Shared lock — read
Exclusive lock —
pre clearing lock

Serializable lock

Two phase lock

Strict two phase lock..

Q.7 What is the Possible violation if a application program use Isolation level repeatable read

Ans. using the Repeatable Read Isolation level in a DB system provides a higher level of Isolation compared to lower Isolation levels like Read committed. However it is not the highest level of Isolation, and certain type of anomalies can still occur. Specifically the main possible violation that can occur under Repeatable Read is the Phantom Read anomaly.

Q8. Explain with the help of example the concept of insertion anomalies and deletion anomalies.

Ans. Insertion anomaly and deletion anomaly are type of data integrity issue that can occur in relational db, particularly when the db schema is not properly normalized.

These anomalies can lead to redundant, inconsistent or incomplete data, ~~Here are~~

↳ Insertion Anomalies

↳ It occurs when certain attributes cannot be inserted into the db without the presence of other attributes. This often happens when data is stored in a single table that combines multiple entities.

Eg.

Student ID.	StudentName	Course ID	Course Name	Instructor Name
1	Alice	C101	Math	Prof. Smith
2	Bob	C102	Science	Prof. Jones.

If we want to ~~insert~~ insert a new data with course Id, CourseName, InstructorName → Then we don't have to put the data in the table.

we might insert a dummy student, leading to incorrect or misleading data.

↳ Deletion Anomalies

↳ It occurs when the deletion of data representing one fact necessitates the deletion of data representing another fact, which should not be deleted. This often results in loss of valuable information

eg.

Stud Id.	Std. Name	CourseId.	Course Name	InstructorName.
1	Alice	C101	Math	Prof. Smith
2.	Bob	C102	Science	Prof Jones
3	Charlie	C101	Math	Prof Smith.

If we want to delete the row with course Id C101 then it also remove both the rows. but we want to remove only 3 one.

This would result in loss of information.

Q? Solution to Anomalies Normalization

Normalization is the Process of organizing data in a DB to reduce redundancy and improve data integrity. It involves decomposing tables into smaller, related tables and defining relationships b/w them.

We can normalize the Student Course table into two separate tables.

1. Student Table :

Student Id	Student Name
1	Alice
2	Bob
3.	Charlie :

2. Course Table :

Course ID	Course Name	Instructor Nam
C101	Math	Prof. Screm
C102	Science	Prof. Jones.
C103	History	Prof Bro

3. Enrollment Table :

Student Id	Course ID.
1	C101
2	C102.
3	C101.

Benifits

- Insertion Anomaly Prevention.
  ↳ we can now add a new course without needing to enroll a student by simply inserting a row into the course table.

- Deletion Anomaly Prevention :
  ↳ we can delete a student's enrollment without losing information about the course. For instance, deleting Charlie enrollment (student ID = 3, course Id = C101) from the enrollment table does not affect the course table.

Q10. What is the goal of Query optimization? Why is optimization important?

Ans:- The goal of Query optimization is to improve the performance and efficiency of DB queries, ensuring that they execute as quickly and with as few resources as possible while still producing the correct results.

Importance of Query optimization

1. Improved Performance
   ↳ Efficient queries result in faster response time, reducing latency and improving overall system performance.
   This is especially critical in application where users expect near-instantaneous responses, such as web App. or real time analytics system.

2. Resource Utilization
   ↳ Optimized queries consume fewer system resources such as CPU, memory, and disk I/O.
   This leads to better resource utilization, allowing the system to handle more concurrent requests and users without degradation in performance.

3. Scalability.
   ↳ As the volume of data and user requests increase, a well optimized DB can scale more effectively. By reducing the

the computational burden of each query, optimization helps the system maintain performance level even under heavy load.

* Cost Reduction
↳ In cloud computing environments where resources are billed based on usage, query optimization can lead to cost savings by reducing the amount of resources consumed. This is particularly important for organizations managing large scale data operations.

* User Experience (But user experience)
↳ Faster query response time improves the user experience by providing more responsive applications and reducing wait time. This can lead to higher user satisfaction and engagement.

* Complexity Management → Query optimization often involves analyzing and understanding the underlying data model, indexes, and query execution plans. This process can uncover inefficiencies or bottlenecks in the db design, leading to improvement in overall system architecture and complexity management.

* Adaptability:
↳ optimization techniques evolve with advancements in DB technology and hardware capabilities.