CSE-4845 || **Distributed Database** || **Part-A** & B || Final Term Note *(Hon. **MMR Sir** Supremacy)*

By- **Sorowar Mahabub**, *C201032*

Segment-0

Imp. 01. Discuss about the Normalization of Query Decomposition. What do you know about the Query Optimization?

Normalization of Query Decomposition:
Query decomposition involves mapping a calculus query (such as SQL) to algebraic operations like select, project, join, and rename. The goal is to produce a semantically correct and efficient query that avoids redundant work. The process of query decomposition comprises several steps, and one of the crucial steps is normalization.

- **Normalization Steps:**
- **Lexical and Syntactic Analysis:**
  - Check the validity of the query, similar to compiler checks.
  - Verify attributes and relations.
  - Perform type checking on the qualification.

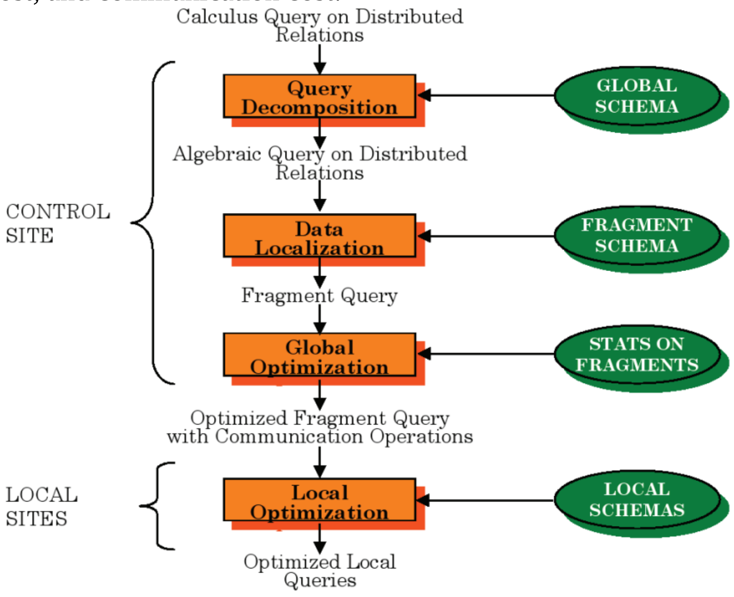
- **Put into Normal Form:**
  - Normalize the query to facilitate further processing.

- In SQL, the WHERE clause is often the most complex part, with arbitrary predicates preceded by quantifiers (e.g., EXISTS, FOR ALL).
- Normalize to Conjunctive Normal Form (CNF) or Disjunctive Normal Form (DNF).
  - CNF: $(p_{11} \wedge p_{12} \wedge \ldots \wedge p_{1n}) \wedge \ldots \wedge (p_{m1} \wedge p_{m2} \wedge \ldots \wedge p_{mn})$
  - DNF: $(p_{11} \wedge p_{12} \wedge \ldots \wedge p_{1n}) \vee \ldots \vee (p_{m1} \wedge p_{m2} \wedge \ldots \wedge p_{mn})$
-
- The DNF allows processing the query as independent conjunctive subqueries linked by unions.

Query Optimization: Query optimization is a crucial and challenging aspect of overall query processing. The objective is to minimize the cost function, which includes I/O cost, CPU cost, and communication cost.

1. **Cost Function:** Minimize I/O cost+ CPU cost+ communication cost

2. **Types of Query Optimizers:**
   - **Exhaustive Search:** Examines all possible execution plans.
   - **Heuristics:** Uses rules of thumb to explore likely efficient plans.
   - **Dynamic:** Adjusts plans dynamically based on runtime feedback.
   - **Hybrid:** Combines multiple optimization techniques.



2. What do you know about the Anomalies? Explain different types of anomalies with example.

Anomalies in database systems are inconsistencies or errors that can arise in the data stored in the database. These anomalies can occur due to improper database design or due to the way that data is manipulated by transactions. Anomalies can lead to data integrity issues and can make it difficult to retrieve accurate information from the database.

There are three main types of anomalies in database systems:

1. Update Anomaly:

An update anomaly occurs when a single operation can change multiple records in the database in a way that violates the database's integrity constraints. This can lead to data inconsistencies and make it difficult to maintain the accuracy of the database.

Example: Consider a database that stores information about employees and their departments. The database has two tables:

- EMPLOYEES: This table stores employee information, including employee ID, name, and department ID.
- DEPARTMENTS: This table stores department information, including department ID, department name, and manager ID.

If the database is not properly designed, an update anomaly could occur when an employee's department is changed.

For example, if an employee is moved from department 1 to department 2, the following update statement could be executed:

```
UPDATE EMPLOYEES
SET DEPT_ID = 2
WHERE EMP_ID = 123;
```

This update statement would change the department ID for employee 123 to 2. However, if employee 123 is also the manager of department 1, then this update would also change the manager ID for department 1 to 2. This is an update anomaly because it has changed multiple records in the database (the employee record and the department record) in a way that violates the database's integrity constraints.

2. Insertion Anomaly

An insertion anomaly occurs when a new record cannot be inserted into the database because it violates the database's integrity constraints. This can lead to data loss and make it difficult to add new information to the database.

Example: Consider the same database as the update anomaly example. If the database is not properly designed, an insertion anomaly could occur when a new employee is hired into a department that does not have a manager. For example, if the following insert statement is executed:

```
INSERT INTO EMPLOYEES (EMP_ID, NAME, DEPT_ID)
VALUES (456, 'John Smith', 3);
```

This inserts statement would fail because department 3 does not have a manager. This is an insertion anomaly because it is not possible to insert a new employee record into the database while maintaining the database's integrity constraints.

3. Deletion Anomaly

A deletion anomaly occurs when a deletion operation can delete multiple records from the database in a way that violates the database's integrity constraints. This can lead to data loss and make it difficult to remove information from the database.

Example: Consider the same database as the update anomaly example. If the database is not properly designed, a deletion anomaly could occur when the manager of a department is deleted. For example, if the following delete statement is executed:

```
DELETE FROM EMPLOYEES
WHERE EMP_ID = 123;
```

This deletes statement would delete the employee record for employee 123. However, if employee 123 is also the manager of department 1, then this delete would also delete the department record for department 1. This is a deletion anomaly because it has deleted multiple records from the database (the employee record and the department record) in a way that violates the database's integrity constraints.

Preventing Anomalies

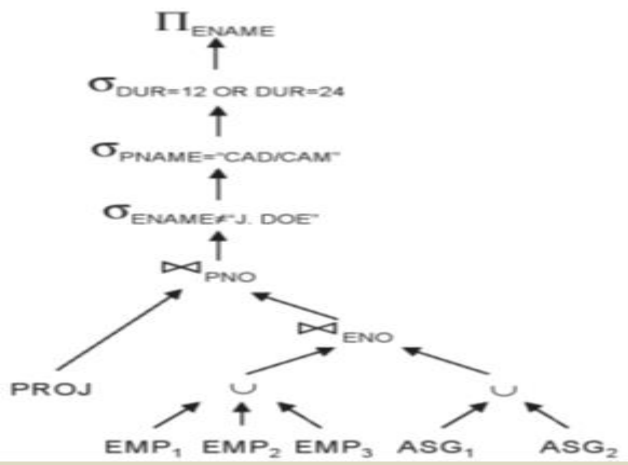
Anomalies can be prevented by using proper database design techniques, such as normalization. Normalization is a process of organizing data in a database to reduce redundancy and ensure data integrity. There are three levels of normalization: first normal form (1NF), second normal form (2NF), and third normal form (3NF). Databases that are normalized to 3NF are generally free from anomalies.

Imp. 3. What do you know about the Data Localization? Discuss about the Normalization of Query Decomposition.

Data localization is a technique used in distributed database systems to optimize query processing by taking into account the distribution of data across different fragments. The goal of data localization is to reduce the amount of data that needs to be transferred between fragments during query execution.

Benefits of data localization:

- Reduced network traffic: By minimizing data transfer between fragments, data localization can significantly reduce network traffic, which can improve the overall performance of distributed database systems.
- Improved query performance: By reducing the amount of data that needs to be processed, data localization can improve the performance of individual queries.
- Reduced contention: Data localization can help to reduce contention for shared resources, such as network bandwidth and CPU time.


Π_{ENAME}
$\sigma_{DUR=12 \text{ OR } DUR=24}$
$\sigma_{PNAME=\text{"CAD/CAM"}}$
$\sigma_{ENAME\neq\text{"J. DOE"}}$
$\bowtie_{PNO}$
$\bowtie_{ENO}$
PROJ
$\cup$
$\cup$
EMP_1 EMP_2 EMP_3 ASG_1 ASG_2

Data localization techniques: There are a number of different data localization techniques that can be used, including:

Fragment replication: Replicating data fragments to multiple nodes can reduce the amount of data that needs to be transferred during query execution.
Data partitioning: Partitioning data fragments based on specific attributes can improve the efficiency of certain types of queries.
Data caching: Caching frequently accessed data can reduce the need to retrieve data from remote fragments.

4. How many steps you need for Distributed Query Processing? Discuss it.

Distributed Query Processing involves transforming a high-level query on a distributed database into an equivalent and efficient lower-level query on relation fragments. It is a more complex process compared to centralized query processing due to the fragmentation/replication of relations and the additional communication costs associated with distributed environments. The distributed query processing can be broken down into several steps:

1. Parsing and Translation:

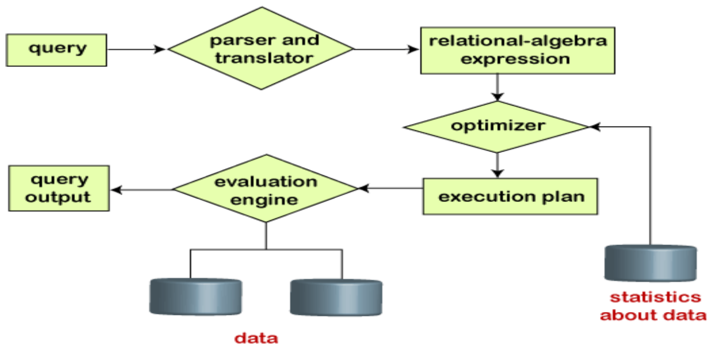
- *Syntax Check:* Ensure that the query follows the correct syntax and verify the relations involved.
- *Translation:* Translate the high-level query (relational calculus/SQL) into an equivalent relational algebra expression that considers the distributed nature of the database and the fragmentation/replication of relations.

2. Optimization:

- *Generate Evaluation Plan:* Generate an optimal evaluation plan for the query, considering the distributed nature of the database and minimizing costs associated with communication, I/O, and CPU.
- *Consider Fragmentation and Replication:* Optimize the query plan by taking into account the distribution of data across fragments and potential replication strategies.
- *Cost Estimation:* Estimate the costs associated with different execution plans and choose the plan with the lowest cost.

3. Evaluation:

- *Execution Plan Execution:* The query-execution engine takes the optimal evaluation plan generated in the optimization step and executes that plan.
- *Answer Retrieval:* Retrieve and return the answers to the query based on the execution of the plan.
- *Handling Communication Costs:* In a distributed environment, communication costs become crucial. The execution plan must consider the location of data fragments, potentially involving data transmission between different nodes.



Steps in query processing

Segment-0

Exercise 5. Indicate whether the given following schedules can produce anomalies; the symbols ci and ai indicate the result (commit or abort) of the transaction: (All examples added)-

Ques. Anomaly: r1(x); w1(x); r2(x); w2(y); a1; c2
Dirty read: r1(x); w1(x); r2(x); w2(y); a1; c2

Ques. Anomaly: r1(x); r2(x); w2(x); w1(x); c1; c2
Update loss: r1(x); r2(x); w2(x); w1(x); c1; c2

Ques. Anomaly: r1(x); w1(x); r2(y); w2(y); a1; c2
No anomaly

Ques. Anomaly: r1(x); r2(x); w2(x); r1(y); c1; c2
No anomaly

Ques. Anomaly: r1(x); r2(x); r2(y); w2(y); r1(z); a1; c2
No anomaly

Ques. Anomaly: r1(x); w1(x); r2(x); w2(x); c1; c2
No anomaly

6. Write down the processing issues of Transactions. What do you know about the Formalization of Transactions?

Processing Issues of Transactions

Transactions are fundamental units of work in database systems. They ensure that data remains consistent in the face of concurrent access and failures. However, there are a number of processing issues that can arise when handling transactions. These issues can be categorized into the following areas:

- **Concurrency control:** Transactions need to be coordinated to ensure that they do not interfere with each other and that the database remains in a consistent state. This can be a challenge, especially when there are a large number of concurrent transactions.
- **Isolation:** Transactions should be isolated from each other, so that they cannot see the uncommitted changes of other transactions. This is important to prevent data anomalies and ensure that transactions are atomic.
- **Durability:** Transactions should be durable, meaning that their effects should be permanent even in the event of system failures. This is typically achieved by logging the changes made by transactions and using the log to recover in case of a failure.

- **Performance:** Transactions should be processed efficiently, so that they do not impact the performance of other transactions or the overall performance of the database system.

Formalization of Transactions

The formalization of transactions provides a rigorous framework for understanding and reasoning about transaction processing. This framework is based on the concept of a transaction model, which is a mathematical model that captures the behavior of transactions. Transaction models can be used to prove properties of transactions, such as consistency, isolation, and durability.

Benefits of Formalizing Transactions

The formalization of transactions has a number of benefits, including:

- **Improved understanding:** Transaction models provide a precise and unambiguous way of understanding the behavior of transactions.
- **Formal reasoning:** Transaction models can be used to prove properties of transactions, such as consistency, isolation, and durability.
- **Design and analysis:** Transaction models can be used to design and analyze transaction processing systems.
- **Correctness proofs:** Transaction models can be used to prove the correctness of transaction processing algorithms.

Example: Formalizing the Two-Phase Locking (2PL) Protocol

The Two-Phase Locking (2PL) protocol is a widely used concurrency control mechanism for transactions. It ensures that transactions do not interfere with each other and that the database remains in a consistent state. The 2PL protocol can be formalized using a transaction model.

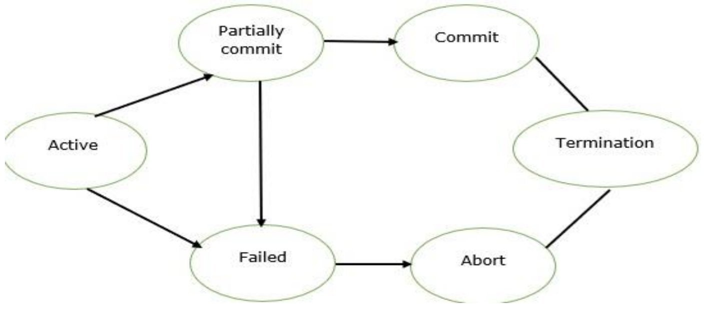
The 2PL protocol consists of two phases:

- Growing phase: In the growing phase, transactions acquire locks on the data items they need.
- Shrinking phase: In the shrinking phase, transactions release the locks they hold.

The 2PL protocol ensures that transactions do not interfere with each other by requiring that transactions acquire a lock on a data item before they can read or write it. If a transaction tries to acquire a lock on a data item that is already locked by another transaction, the transaction is blocked until the lock is released.

7. Briefly describe the states of the Transaction.

The states of a transaction in a database management system (DBMS) represent the different stages in its lifecycle. These states define the progression of a transaction from its initiation to its completion or termination.



1. **Active:** The transaction is currently executing and making changes to the database. It is processing operations and manipulating data.
2. **Partially Committed:** The transaction has executed all of its operations and has reached a consistent state. However, it has not yet been fully committed, and its changes are still visible only to itself. It awaits the final commit decision.
3. **Committed:** The transaction has successfully completed its execution and its changes have been permanently applied to the database. The transaction is now considered part of the database's consistent state.
4. **Aborted:** The transaction has been terminated due to an error or failure. Its changes have been rolled back, and the database is restored to the state it was in before the transaction began.
5. **Failed:** The transaction has encountered an unrecoverable error and cannot proceed. It is considered a failure, and its state is often transitioned to Aborted.

These states represent the fundamental phases of a transaction's lifecycle, ensuring data integrity and consistency in the face of concurrent access and potential failures.

8\1. Write down the steps of process that transforms a high-level query (of relational calculus/SQL) into an equivalent and more efficient lower-level query (of relational algebra).

Transformation of High-Level Queries to Relational Algebra

The transformation of high-level queries (of relational calculus/SQL) into an equivalent and more efficient lower-level query (of relational algebra) is a crucial step in query processing. This process involves translating the user's intent expressed in a high-level language into a sequence of relational algebra operations that can be directly executed by the database engine. The goal of this transformation is to optimize the query for efficient execution while preserving its semantic meaning.

Steps in Query Transformation

The process of transforming a high-level query to relational algebra typically involves the following steps: (**Example:** Transformation of SQL Query to Relational Algebra)

Consider the following SQL query:

```
SELECT ENAME
FROM EMP, ASG
WHERE EMP.ENO = ASG.ENO AND DUR > 37;
```

This query retrieves the names of employees who are managing a project with a duration greater than 37 weeks. The transformation of this query into relational algebra involves the following steps:

1. **Parsing and Semantic Analysis:** The syntax of the query is checked, and the existence of the EMP and ASG relations is verified.

2. **Expression Translation:** The SQL query is translated into an equivalent relational algebra expression:

$$\text{ENAME}\ (\text{EMP} \times \text{ASG}) \bowtie \sigma\text{DUR}>37(\text{EMP} \times \text{ASG}))$$

This expression represents the projection of the ENAME attribute from the result of the join between the EMP and ASG relations followed by a selection based on the DUR attribute.

3. **Optimization:** An alternative expression can be considered:

$$\text{ENAME}\ (\text{EMP} \rtimes\ltimes \text{ENO}\ (\sigma\text{DUR}>37(\text{ASG})))$$

This expression represents a semijoin operation followed by a projection. It is typically more efficient than the first expression because it avoids the expensive and large intermediate Cartesian product.

4. **Physical Query Plan Generation:** The optimized relational algebra expression is translated into a physical query plan, which specifies the actual execution steps, including access methods for EMP and ASG relations, the appropriate join algorithm, and the utilization of any indexes available on the DUR attribute.

8/2. Show that "A transaction partially committed is not necessarily be committed", with necessary example.

Consider a bank transfer transaction where funds are transferred from one account to another. The transaction can be divided into two steps:

1. Deduct the amount from the sender's account.
2. Add the amount to the recipient's account.

If the transaction is partially committed, then the first step has been completed, but the second step has not yet been completed. In this state, the transaction is not fully committed, and the funds have not been permanently transferred from the sender's account to the recipient's account.

If the transaction is not completed successfully, then the changes made in the first step will be rolled back, and the sender's account will not be debited. In other words, the transaction will not be committed.

Therefore, a transaction that is partially committed is not necessarily be committed. It is possible for the transaction to be rolled back and the changes made in the first step to be reversed.

Segment-0

9. Write down the properties of Transactions? Briefly discuss it.

ACID Properties: Transactions are fundamental units of work in database systems. They ensure that data remains consistent in the face of concurrent access and failures. ACID properties are a set of four critical properties that transactions must satisfy to ensure data integrity and consistency. These properties are:

1. **Atomicity:** A transaction is either fully completed or fails entirely. There is no intermediate state where only part of the transaction is executed.

2. **Consistency:** A transaction must transform the database from one consistent state to another consistent state. It cannot leave the database in an inconsistent state.

3. **Isolation:** Transactions must be isolated from each other, so that they cannot see the uncommitted changes of other transactions. This is important to prevent data anomalies and ensure that transactions are atomic.

4. **Durability:** Transactions should be durable, meaning that their effects should be permanent even in the event of system failures. This is typically achieved by logging the changes made by transactions and using the log to recover in case of a failure.

Brief Discussion: The ACID properties are essential for ensuring the reliability and integrity of database systems. They provide a framework for designing and implementing transaction processing systems that can handle concurrent access and failures.

- **Atomicity:** Atomicity ensures that transactions are all-or-nothing propositions. Either the entire transaction is executed successfully, or none of it is executed. This prevents data from being inconsistently updated due to partial transaction executions.
- **Consistency:** Consistency ensures that transactions maintain the overall integrity of the database. They cannot leave the database in a state that violates any data integrity constraints or relationships. This maintains the overall correctness of the data stored in the database.
- **Isolation:** Isolation ensures that transactions are executed independently and do not interfere with each other's operations. This prevents data anomalies or conflicts that can arise if multiple transactions attempt to modify the same data simultaneously.
- **Durability:** Durability ensures that the effects of committed transactions are permanent, even in the event of system failures. This is achieved by logging transaction changes to stable storage, ensuring that these changes are not lost even if the system crashes or loses power.

The ACID properties are fundamental principles for designing and managing transactions in database systems. They provide a solid foundation for ensuring data integrity, consistency, and reliability in the face of concurrent access and failures.

Exercise 10. Write the possible Transformation of an Given SQL-query into a Relational Algebra-query (Added from Monna).

Example: Transformation of an SQL-query into an RA-query. Relations: EMP (ENO, ENAME, TITLE), ASG (ENO, PNO, RESP, DUR)

Query: *Find the names of employees who are managing a project*?

– High level query

```
SELECT ENAME
FROM EMP, ASG
WHERE EMP.ENO = ASG.ENO AND DUR > 37
```

– Two possible transformations of the query are:

_ Expression 1: _ENAME(_DUR>37∧EMP.ENO=ASG.ENO (EMP × ASG))

_ Expression 2: _ENAME (EMP ⋊⋉ENO (_DUR>37(ASG)))

– Expression 2 avoids the expensive and large intermediate Cartesian product, and therefore typically is better.

11. Write down the properties of Strict 2PL (two phase locking) protocol? When two schedules are said to be Equivalent?

Properties of Strict 2PL (two phase locking) protocol

Strict 2PL, or Rigorous 2PL, is a variation of the Two-Phase Locking (2PL) protocol for concurrency control in database systems. It is a stricter version of 2PL that ensures both serializability and conflict-serializability, which are stronger consistency properties than those guaranteed by standard 2PL.

The main properties of Strict 2PL are:

- **Serializability:** Strict 2PL ensures that the execution of a set of transactions is equivalent to some serial execution, meaning that the final state of the database is the same as if the transactions had been executed one at a time.
- **Conflict-Serializability:** Strict 2PL also ensures conflict-serializability, which means that if two transactions conflict, then their execution is equivalent to some serial execution where one transaction executes entirely before the other.
- **Deadlock Freedom:** Strict 2PL is deadlock-free, meaning that it will not allow a situation to occur where two or more transactions are waiting for each other to release locks, preventing either of them from making progress.
- **Recoverability:** Strict 2PL ensures that transactions are recoverable, meaning that the effects of a committed transaction are not lost even if the system fails.

Equivalence of Schedules

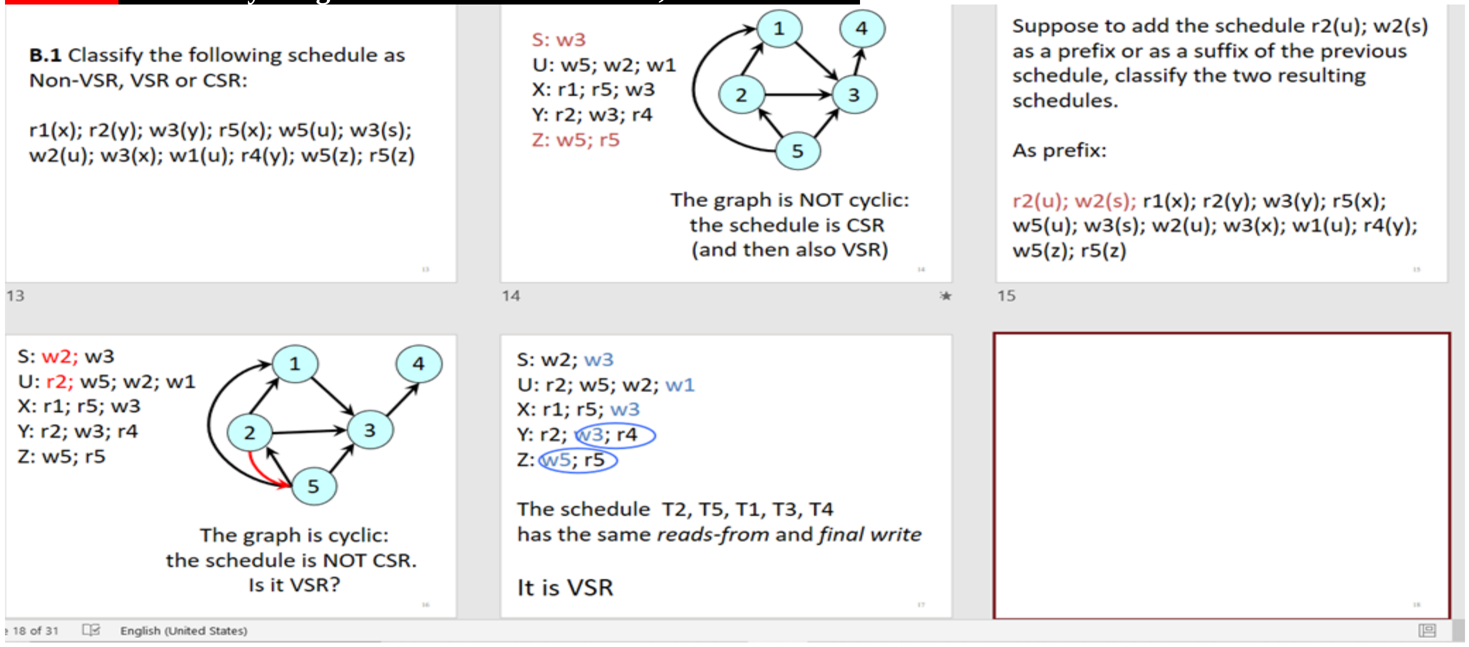
In the context of transaction processing, two schedules are said to be equivalent if they produce the same final state of the database. This means that the order in which the transactions are executed does not affect the overall outcome.

There are two main types of equivalence:

- **State equivalence:** Two schedules are state equivalent if they produce the same final state of the database, regardless of the initial state.
- **Conflict equivalence:** Two schedules are conflict equivalent if they produce the same final state of the database for all possible initial state.

Strict 2PL ensures that all schedules produced by the protocol are equivalent, meaning that the order in which transactions are executed does not affect the overall outcome. This is important for ensuring data consistency and avoiding anomalies.

Exercise 12. Classify the given schedule as Non-VSR, VSR or CSR: -



B.1 Classify the following schedule as Non-VSR, VSR or CSR:

r1(x); r2(y); w3(y); r5(x); w5(u); w3(s); w2(u); w3(x); w1(u); r4(y); w5(z); r5(z)

13

S: w3
U: w5; w2; w1
X: r1; r5; w3
Y: r2; w3; r4
Z: w5; r5

The graph is NOT cyclic: the schedule is CSR (and then also VSR)

14

Suppose to add the schedule r2(u); w2(s) as a prefix or as a suffix of the previous schedule, classify the two resulting schedules.

As prefix:

r2(u); w2(s); r1(x); r2(y); w3(y); r5(x); w5(u); w3(s); w2(u); w3(x); w1(u); r4(y); w5(z); r5(z)

15

S: w2; w3
U: r2; w5; w2; w1
X: r1; r5; w3
Y: r2; w3; r4
Z: w5; r5

The graph is cyclic: the schedule is NOT CSR. Is it VSR?

S: w2; w3
U: r2; w5; w2; w1
X: r1; r5; w3
Y: r2; w3; r4
Z: w5; r5

The schedule T2, T5, T1, T3, T4 has the same *reads-from* and *final write*

It is VSR

18 of 31 English (United States)

Segment-0

Imp. 13. What is the site failure in 2PC protocol? Write down the phases of three phase commit protocol(3PC).

The 2PC protocol is a distributed commit protocol used to ensure atomicity (either all participants commit or all participants abort) in distributed transactions. The protocol involves two phases:

1. **Prepare Phase:**
   - The coordinator sends a prepare message to all participants, asking them if they are ready to commit.
   - Each participant replies with either "ready" or "not ready."
2. **Commit or Abort Phase:**
   - If all participants reply "ready," the coordinator sends a commit message to all participants.
   - If any participant replies "not ready" or if a timeout occurs, the coordinator sends an abort message to all participants.

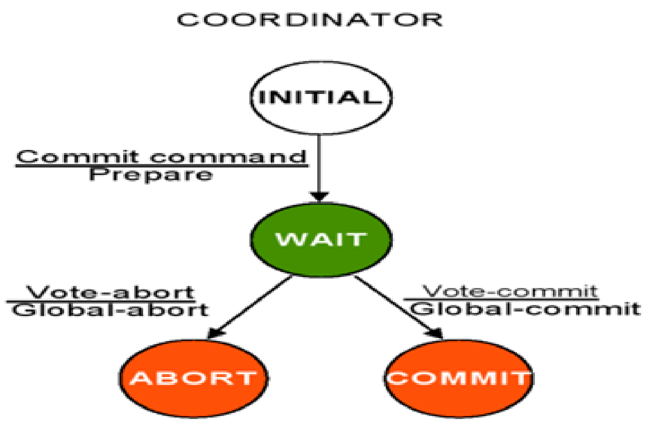
Site failure in the context of the Two-Phase Commit (2PC) protocol refers to the situation where a participant site in the distributed transaction becomes unavailable or fails to respond during the protocol execution. This can occur due to various reasons, such as hardware or software failures, network disruptions, or power outages.

Now, let's discuss the implications of a site failure in the 2PC protocol:

- **Participant Failure:**
  - If a participant fails during the 2PC protocol, and the coordinator receives no response from that participant, the coordinator has to make a decision based on the responses it did receive.
  - If all other participants voted "ready," the coordinator might decide to commit the transaction, assuming the missing participant was ready.
  - If the coordinator receives any "not ready" votes or if it has any doubts, it might decide to abort the transaction.

- **Coordinator Failure:**
  - If the coordinator fails after participants have already agreed to commit but before sending the commit message, a new coordinator must be chosen to complete the protocol.
  - The new coordinator can use information from the previous coordinator's logs to determine the state of the transaction and proceed accordingly.

- **Network Partition or Site Failure:**
  - If there is a network partition or a failure at a participant site after it has voted "ready," the coordinator might not receive the acknowledgment.
  - If the coordinator is unable to determine the outcome of the participant, it may decide to abort the transaction to ensure consistency.



14. Write down the steps of Two-Phase Locking (2PL) Protocol.

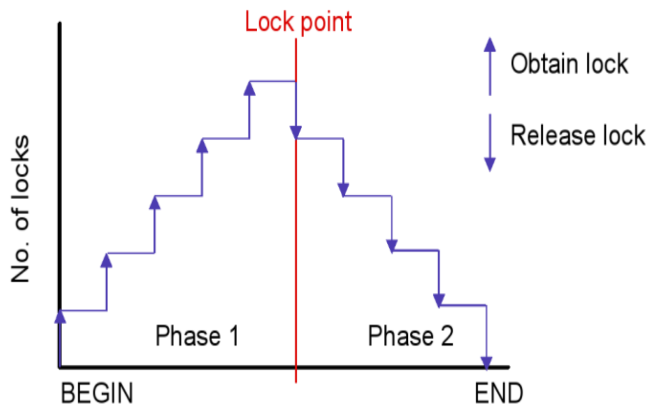
The Two-Phase Locking (2PL) Protocol is a concurrency control method used in database management systems to ensure the consistency and isolation of transactions. The protocol consists of two phases: the growing phase and the shrinking phase. **The steps of the Two-Phase Locking (2PL) protocol are:**

Growing Phase:

1. A transaction requests a lock on a data item by sending a lock request message to the lock manager.
2. The lock manager checks if the lock can be granted based on the locking rules. If the lock can be granted, the lock manager sends a lock grant message to the transaction. Otherwise, the lock manager adds the lock request to a wait queue.
3. If the transaction receives a lock grant message, it proceeds to access the data item. If the transaction receives a lock wait message, it waits until the lock is granted.

Shrinking Phase:

1. A transaction releases a lock on a data item by sending an unlock request message to the lock manager.
2. The lock manager removes the lock from the lock table and sends an unlock grant message to the transaction.
3. The transaction can now commit or abort. If it commits, it notifies the lock manager of the committed data items. The lock manager releases any locks held by the committed transaction. If it aborts, it notifies the lock manager of the aborted data items. The lock manager releases any locks held by the aborted transaction and adds any waiting transactions to the ready queue.



The 2PL protocol ensures that transactions do not interfere with each other and that all transactions see a consistent view of the data.

Imp. 15. What do you know about the site failures in 2PC protocol? What is the Distributed Reliability Protocol and Commit protocol?

Distributed Reliability Protocols: Distributed reliability protocols manage the coordination and recovery of distributed transactions, ensuring that all participating nodes agree on the outcome of a transaction, whether it's a successful completion or an abort. These protocols address the challenges arising from communication delays, node failures, and network disruptions that can occur in distributed environments.

Common distributed reliability protocols include:

- **Two-Phase Commit (2PC):** A widely used protocol that involves two phases – prepare and commit – to ensure that all participants agree on the outcome of a transaction.
- **Three-Phase Commit (3PC):** An extension of 2PC that adds an additional pre-commit phase to handle coordinator failures.
- **Voting Commit:** A protocol that uses voting among participants to determine the outcome of a transaction.

Commit Protocols: Commit protocols are specifically designed to manage the final phase of a distributed transaction, ensuring that all participating nodes agree on whether to commit or abort the transaction. They provide atomicity and durability guarantees for distributed transactions, ensuring that all participants either commit the entire transaction or abort the entire transaction.

Common commit protocols include:

- **Two-Phase Commit (2PC):** The commit phase of 2PC involves sending a commit message to all participants and waiting for commit-ok messages from all participants before declaring the transaction committed.
- **Voting Commit:** The commit phase of voting commit involves collecting votes from all participants and deciding based on the majority vote whether to commit or abort the transaction.
- **Global Lock Manager (GLM):** A centralized approach that uses a global lock manager to coordinate the commit process and ensure data consistency across all participants.

16. What is Reliability and Out-of-place Update? Describe the steps of two-phase commit protocol (2PC).

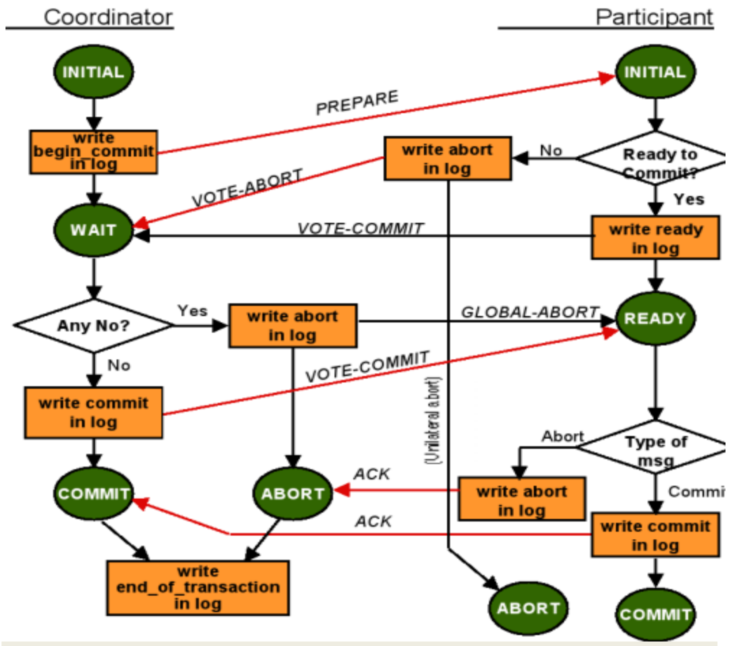
Reliability in the context of data management refers to the ability of a system to maintain data integrity and availability in the face of errors, failures, or disruptions. It ensures that data remains consistent and accessible even when unexpected events occur, such as hardware failures, software glitches, or network outages.

Out-of-place update is a technique used in data management to modify data without overwriting its existing location. Instead, the updated data is placed in a new location, and the original location is marked as obsolete. This approach helps maintain data integrity and consistency by preserving the original data in case of rollback or recovery.

Two-Phase Commit Protocol (2PC)

The Two-Phase Commit protocol (2PC) is a distributed transaction protocol that ensures data consistency across multiple nodes or sites in a distributed system. It coordinates the actions of participating nodes to ensure that all transactions are either successfully completed or aborted consistently.

The 2PC protocol consists of two phases:



Phase 1: Prepare

1. The coordinator initiates the prepare phase by sending a prepare message to all participating nodes.
2. Each participating node receives the prepare message and performs the following steps:
    - Prepares to commit the transaction by writing all transaction changes to its local durable storage.
    - Sends a prepare-ok message to the coordinator if the preparation was successful.
    - Sends a prepare-abort message to the coordinator if the preparation failed.
3. The coordinator waits for prepare-ok messages from all participating nodes.
    - If all nodes respond with prepare-ok, the coordinator proceeds to the commit phase.
    - If any node responds with prepare-abort, the coordinator proceeds to the abort phase.

Phase 2: Commit or Abort

1. The coordinator sends a commit or abort message to all participating nodes based on the outcome of the prepare phase.
2. Each participating node receives the commit or abort message and performs the following steps:
    - If the coordinator sent a commit message, the node commits the transaction and sends a commit-ok message to the coordinator.
    - If the coordinator sent an abort message, the node aborts the transaction and sends an abort-ok message to the coordinator.
3. The coordinator waits for commit-ok or abort-ok messages from all participating nodes.
    - If all nodes respond with commit-ok, the transaction is successfully committed.
    - If any node responds with abort-ok, the transaction is aborted.

The 2PC protocol ensures that all participating nodes agree on the outcome of the transaction, preventing data inconsistencies. It is a widely used protocol in various distributed systems, including databases, messaging systems, and workflow management systems.

17. What is Concurrency Control? What do you know about the Locking based concurrency algorithms?

Concurrency control is a set of techniques that ensure data consistency and transaction integrity in the presence of multiple concurrent users accessing and modifying a shared database. It manages the simultaneous access to shared data by multiple users, preventing conflicts and ensuring that all users see a consistent view of the data.

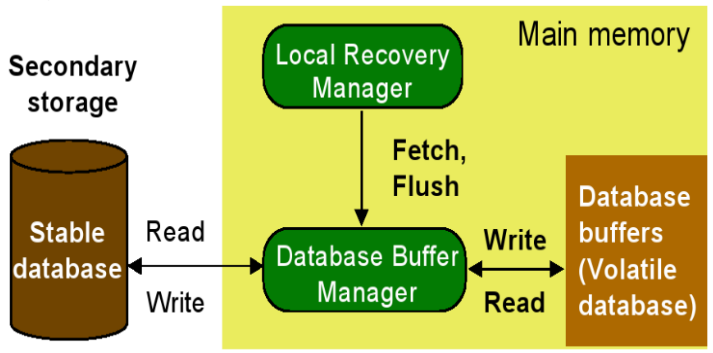
Locking-based concurrency algorithms are a type of concurrency control mechanism that uses locks to control access to shared data. A lock is a mechanism that prevents other users from modifying a data item while it is being accessed by a user. Locks can be applied at different levels of granularity, from individual data items to entire tables or even the entire database. **Common locking-based concurrency algorithms include:**

- **Two-Phase Locking (2PL):** A simple and widely used algorithm that involves two phases: growing and shrinking. In the growing phase, transactions acquire locks on the data items they need, and in the shrinking phase, they release the locks they hold.
- **Optimistic Locking:** An algorithm that defers lock acquisition until a transaction is about to commit. This reduces lock contention and improves performance, but it also increases the risk of conflicts.
- **Timestamp-Based Locking:** An algorithm that assigns timestamps to transactions and uses timestamps to determine the order in which locks are granted. This ensures that transactions are serialized in a consistent order.

Locking-based concurrency algorithms are effective at preventing conflicts and ensuring data consistency, but they can also introduce overhead and reduce performance. The choice of locking algorithm depends on the specific requirements of the application, such as the level of concurrency, the type of data being accessed, and the desired performance characteristics.

.

18. Discuss the architecture of Local Recovery Management System.



The architecture of the Local Recovery Manager in a database system typically involves several components working together to ensure data consistency and durability. The main components include:

1. **Database Buffer Manager:**
   - The Database Buffer Manager is responsible for managing the database buffers in the main memory (RAM).
   - It facilitates the transfer of data between the stable database (on stable storage) and the volatile database buffers (in main memory).
   - Handles read and write operations, ensuring efficient loading of data into and flushing of data from the main memory.

2. **Local Recovery Manager:**
   - The Local Recovery Manager is a critical component that coordinates recovery mechanisms to ensure data consistency and durability.
   - It works in conjunction with the Database Buffer Manager and overseas operations such as Fetch and Flush.
   - Coordinates with the Stable Storage component to handle stable-write, stable-read, and clean-up operations.
     - **Fetch:**
       Involves retrieving data from stable storage and loading it into the volatile database buffers. This ensures that the most recent committed data is available in the main memory.
     - **Flush:**
       Involves writing modified data from the volatile database buffers back to stable storage. This operation ensures that changes made to the database are durable and survive failures.
     - **Read/Write Operations:**
       The Local Recovery Manager coordinates with the Database Buffer Manager to handle read and write operations. It may initiate the writing of changes to the log before committing them to the stable database.

3. **Stable Storage:**
   - Stable Storage represents the non-volatile storage that is resilient to failures and retains its contents even in the event of power outages or system crashes.
   - In the context of the Local Recovery Manager, stable storage is involved in stable-write and stable-read operations.
   - Stable storage mechanisms are implemented through a combination of hardware and software to ensure the durability of the stored data.

Segment-0

Imp. 19. What is the Deadlock Management? Describe the steps of three phase commit protocol (3PC).
Imp. 21. What is the Deadlock Management? What we need to do for the Deadlock prevention?

Deadlock Management: Deadlock management is a technique used in database systems to prevent deadlocks, situations where two or more processes are waiting for each other to release resources, causing a standstill. Deadlocks can occur when multiple processes hold exclusive locks on resources needed by others, creating a circular dependency.

There are two main approaches to manage deadlocks:

1. Deadlock prevention: This approach ensures that a deadlock can never occur by analyzing resource allocation requests and only granting them if they cannot lead to a circular dependency. Techniques like resource ordering, lock-down, and preemption are used for prevention.

2. Deadlock detection and recovery: This approach allows deadlocks to occur but identifies them and then takes corrective action to release resources and allow processes to continue. Resource allocation graphs and wait-for graphs are used for detection, and rollback of one or more transactions is used for recovery.

Preventing Deadlocks: Deadlock prevention is a technique used to prevent deadlocks from occurring in the first place. This can be done by carefully designing the system to avoid situations where two or more processes can wait for each other indefinitely. There are a few different approaches to deadlock prevention, including:

1. **Resource Ordering:** Establish a strict ordering of resources that processes must acquire. This prevents circular waiting patterns that lead to deadlocks.
2. **Resource Preemption:** Allow the system to temporarily revoke a resource held by a process if it's required by another process in a critical situation.
3. **Careful Resource Allocation:** Avoid allocating all resources to a single process at the start, as this increases the likelihood of deadlocks.

4. **Hold and Wait:** Ensure that processes only hold the resources they need for the current task and release them promptly when finished.
5. **Deadlock Detection Algorithms:** Implement algorithms that can detect deadlocks when they occur, allowing for appropriate recovery measures.

Three Phase Commit Protocol (3PC)

The Three Phase Commit Protocol (3PC) is a distributed algorithm used to ensure consistency in a distributed database system when committing a transaction across multiple nodes. It improves upon the Two-Phase Commit (2PC) protocol by addressing its vulnerability to coordinator failure.

Here are the steps of 3PC, as highlighted in the image you sent:

Phase 1: Voting/Prepare Phase:

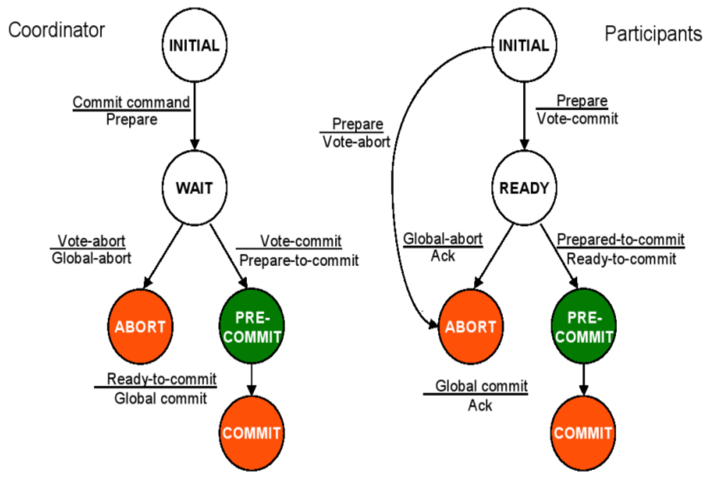
1. Coordinator sends a Prepare message to all participants.
2. Participants vote based on their ability to commit:
   - Vote-commit: If prepared to commit, participant logs Ready and replies with a Vote-commit message.
   - Vote-abort: If unable to commit, participant logs Abort and replies with a Vote-abort message.

Phase 2: Pre-Commit Phase:

1. If all participants vote commit, coordinator broadcasts a Prepare-to-commit message.
2. Participants log Prepared-to-commit and reply with a Ready-to-commit message.
3. If any participant votes abort or no response is received, coordinator broadcasts a Global-abort message.
4. Participants log Abort and send an acknowledgment.

Phase 3: Commit/Deciding Phase:

1. If all participants respond with Ready-to-commit, coordinator broadcasts a Global-commit message.
2. Participants log Commit, commit the transaction, and send an acknowledgment.



3PC Advantages:

- Non-blocking: Doesn't block indefinitely if coordinator fails.
- Fault-tolerant: Handles single site failures.
- Increased consistency: Guarantees all nodes commit or abort a transaction.

3PC Disadvantages:

- Increased complexity compared to 2PC.
- Communication overhead due to additional roundtrip.
- Increased latency for committing transactions.

20. What is the concept of Conceptual design and the Logical design? What is the difference between star schema and snow flake schema design?

Conceptual Design

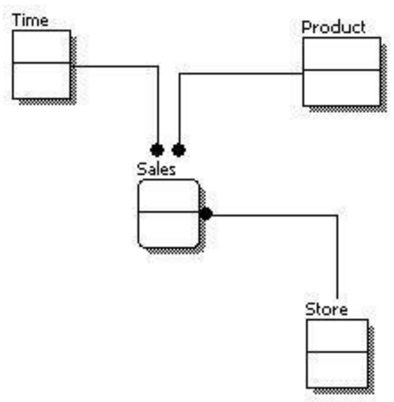
Conceptual design is the initial stage of database design, where the focus is on understanding the problem domain and identifying the entities, attributes, and relationships between them. It involves creating high-level models that represent the real-world entities and their relationships. These models are typically represented using Entity-Relationship Diagrams (ERDs) or Unified Modeling Language (UML) diagrams.

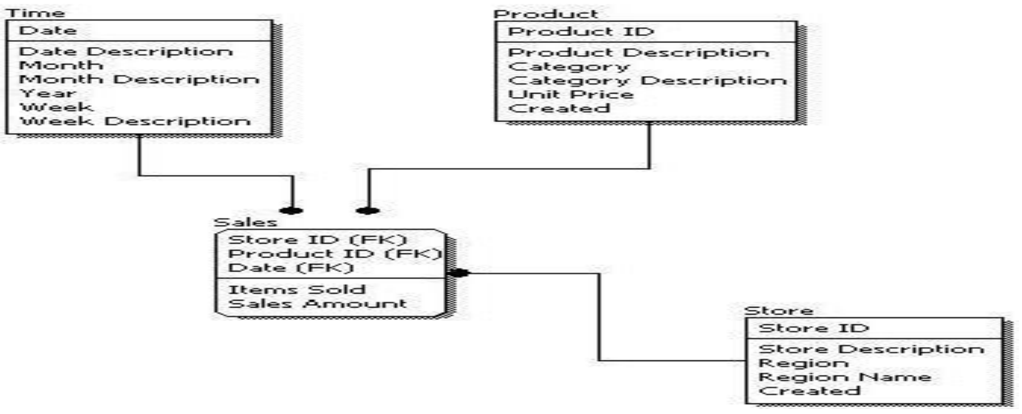
The goal of conceptual design is to establish a common understanding of the data requirements among stakeholders and to identify the core concepts of the system. It helps to ensure that the database is designed to meet the specific needs of the organization.

Logical Design

Logical design takes the conceptual design one step further and translates the high-level models into a more detailed data model. It involves defining the data structures, data types, and relationships within the database. The logical model is typically represented using a specific data modeling language, such as the Data Definition Language (DDL) of a particular database management system (DBMS).

The goal of logical design is to create a detailed representation of the database structure that can be implemented using a specific DBMS. It helps to ensure that the database is efficient and scalable, and that it can meet the performance and security requirements of the organization.





Star Schema vs. Snowflake Schema: Both star schema and snowflake schema are data modeling approaches used for designing data warehouses. They differ in their structure and how they handle data redundancy.

Feature	Star Schema	Snowflake Schema
Definition	A simplified data model with flat tables	A normalized data model with hierarchical tables
Structure	Simpler, with fact tables and dimension tables	More complex, with fact tables, dimension tables, and sub-dimension tables
Data redundancy	Higher	Lower
Data integrity	Lower	Higher
Complexity	Lower	Higher
Suitability	Data analysis, reporting	Complex analysis, reporting
Query performance	Faster	Slower
Design	Top-down	Bottom-up
Normalization	Denormalized	Normalized
Foreign keys	Fewer	More
Advantages	Simple, easy to understand, fast query performance, suitable for large volumes of data	Normalized, reduces redundancy, increases data integrity, allows for complex relationships
Disadvantages	Limited ability to depict complex relationships, data redundancy, may not be suitable for smaller volumes of data	Complex, slower query performance, requires more storage and processing resources

....

Segment-0 (Must 10 Marks)

22. What is data warehouse? Differentiate between standard database and data warehouse.

Data Warehouse

A data warehouse is a specialized database designed to store, manage, and analyze large amounts of historical and current data from disparate sources. It serves as a central repository for business information, enabling organizations to gain insights from their data and make informed decisions.

Key Characteristics of Data Warehouses:

Subject-oriented: Data is organized around business subjects rather than operational applications.
Integrated: Data from multiple sources is consolidated and transformed into a single, consistent format.
Time-variant: Data includes historical and current information, allowing for trend analysis and comparison.
Non-volatile: Once data is loaded into the warehouse, it remains unchanged, ensuring the integrity of historical information.

Standard Database (OLTP)

A standard database, also known as an online transactional processing (OLTP) database, is optimized for storing and managing transactional data. It handles day-to-day operational activities, facilitating data entry, updates, and retrieval.

Key Characteristics of Standard Databases:

Focused on current data: Stores current data required for ongoing operations
Denormalized structure: Data is duplicated to improve performance for frequent access.
Real-time or near real-time data updates: Data is updated frequently to reflect the latest changes.
High-volume data access: Supports high-frequency data access for transactions and data entry.
SQL-based tools: SQL (Structured Query Language) is used for data manipulation and retrieval.

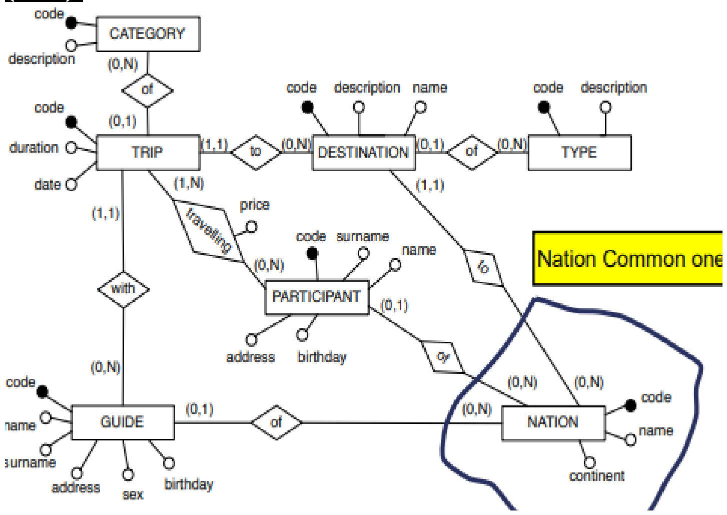
Comparison of Data Warehouses and Standard Databases:

Feature	Data Warehouse	Standard Database (OLTP)
Purpose	Data analysis, reporting, and business intelligence	Transactional processing and real-time data access
Data	Historical and current data	Current data
Structure	Normalized and standardized for consistency	Denormalized for performance
Access	Low-frequency access for analysis	High-frequency access for transactions
Tools	Business intelligence (BI) tools	SQL-based tools
Data volume	Larger volume of data, often terabytes or petabytes	Smaller volume of data, typically megabytes or gigabytes
Update frequency	Lower frequency, typically hourly, daily, or weekly	High frequency, often real-time or near real-time
Data usage	Supports decision-making, trend analysis, and historical insights	Supports daily business operations, transactions, and data entry

...

23. Data Warehouse (DW) related Exercise || Questions:

1. Design a conceptual schema for the Data Warehouse (DW).

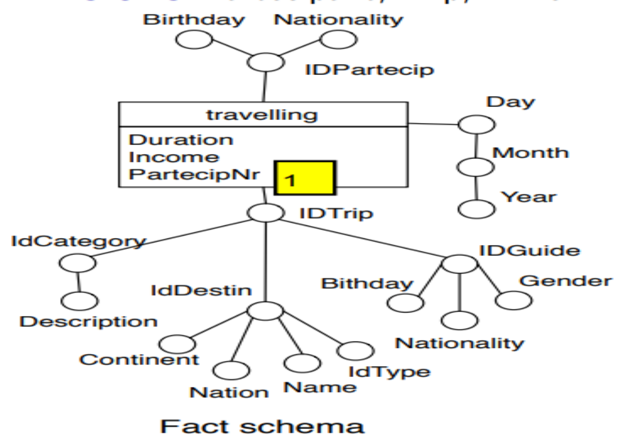


2. What facts and dimensions do you consider?

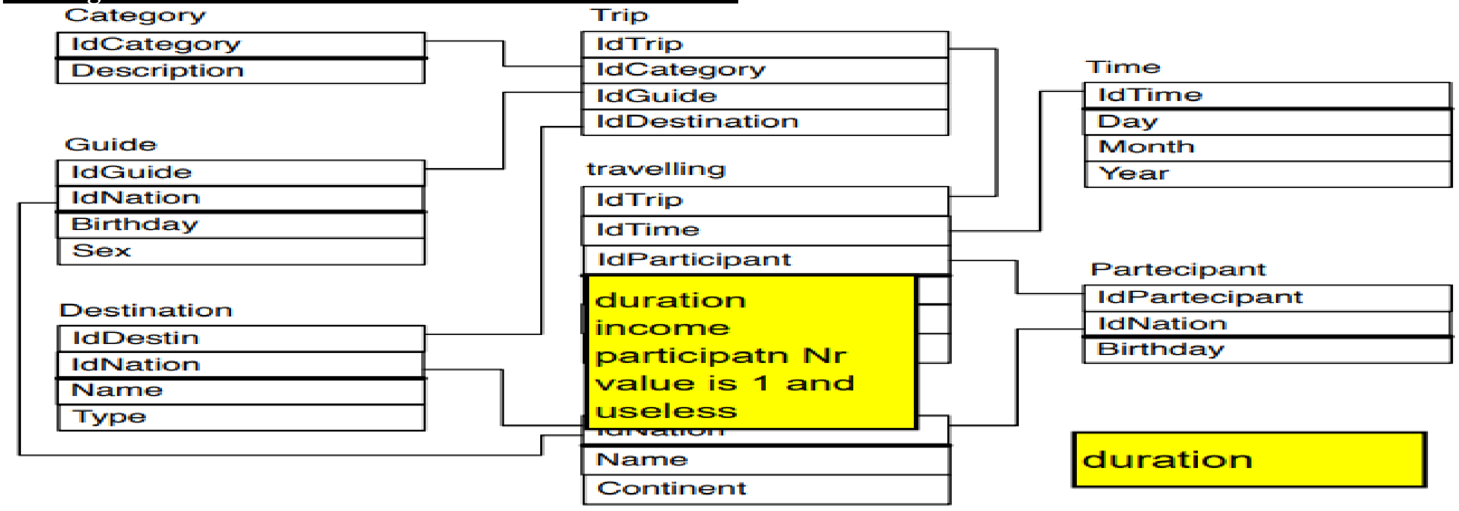
FACT travelling

MEASURES ParticipantNr, Duration, Income

DIMENSIONS Partecipant, Trip, Time



3. Design a Star Schema or Snowflake Schema for the DW.



Data Warehouse (DW) related Exercise (*If possible, go through the PDF*): https://t.me/c/1653334055/127