Arba Minch University

Arba Minch Institute of Technology

Faculty of Computing & Software Engineering

Exit Exam Module for Advanced Database Management System(ADBMS)

Prepared by Instructor Seada Y.

Table of Contents

Chapter 1: - Query Processing and Optimization

Chapter objective

- Explain database query processing and optimization

1.1. Overview of Query Processing

A database query is the **vehicle** for **instructing a** DBMS to update or **retrieve** specific data to/from the physically stored medium. **Query processing** refers to the range of activities involved in **extracting** data from a database.

The activities include:-

- Translation of queries in **high -level database languages** into expressions that can be used at the **physical level of the file system**,
- A variety of query-optimizing transformations, and
- Actual evaluation of queries.

The aims of query processing are to **transform** a query written in a **high-level language**, typically SQL, into a correct and efficient execution strategy expressed in a **low-level language** (implementing the relational algebra), and **to execute the strategy to retrieve the required data.**

1.2. Query Processing steps

There are three phases that a query passes through during the DBMS' processing of that query:

1. Decomposition (Parsing and translation)
2. Optimization
3. Evaluation

1.2.1. Query decomposition

Query decomposition is the first phase of query processing. The **aims** of query decomposition are to **transform a high-level query into a relational algebra query** and to check whether the query is **syntactically and semantically correct**. A query expressed in a high-level query language such as SQL must first be **scanned, parsed**, and **validated.**

Scanner: Identifies the language tokens —such as SQL keywords, attribute names, and relation names—that appear in the text of the query.

Parser: Checks the query syntax to determine whether it is formulated according to the syntax rules (rules of grammar) of the query language.

Validate: Validated by checking that all attribute and relation names are valid and semantically meaningful names in the schema of the particular database being queried.

1.2.2. Query Optimization

In second stage, the query processor **applies rules** to the internal data structures of the query to **transform** these structures into **equivalent**, **but more efficient representations. Selecting** the

proper rules to apply, **when** to apply them and **how** they are applied is the function of the **query optimization engine**. A query typically has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as **query optimization**.

1.2.3. Query Evaluation

The final step in processing a query is the evaluation phase. The best **evaluation plan** candidates generated by the optimization engine is **selected** and then **executed**. **Code generator generates the code** to execute that plan either in compiled or interpreted mode. The **runtime database processor** has the task of **running (executing) the query code**, whether in compiled or interpreted mode, to **produce the query result.**

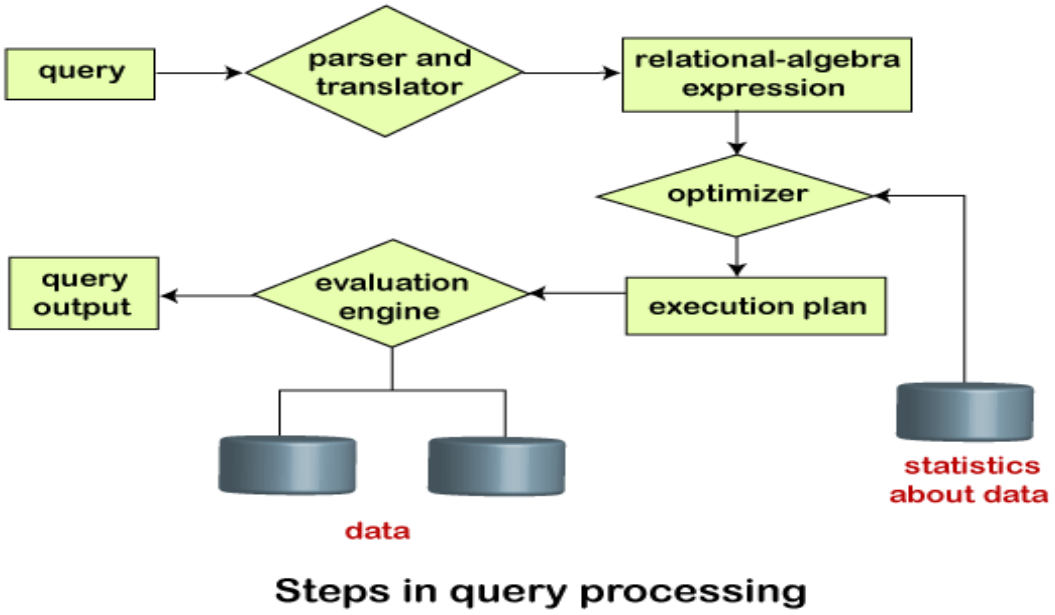


Fig 1: Steps in query processing

Chapter Summery

- **Query processing** refers to the range of activities involved in **extracting** data from a database.
- There are three phases/steps that a query passes through during the DBMS' processing of that query: those are
    - Decomposition (Parsing and translation): - The **aims** of query decomposition are to **transform a high-level query into a relational algebra query** and to check whether the query is **syntactically and semantically correct**.
    - Optimization: - In this stage, the query processor **applies rules** to the internal data structures of the query to **transform** these structures into **equivalent**, **but more efficient representations.**
    - Evaluation: - In this stage, the best **evaluation plan** candidates generated by the optimization engine is **selected** and then **executed**.

Chapter Review Question

1. _____refers to the range of activities involved in **extracting** data from a database.
    A. Query processing
    B. Storage management
    C. DBMS
    D. Evaluation plan
2. In_____ the query processor applies rules to the internal data structures of the query to transform these structures into equivalent, but more efficient representations.
    A. Decomposition
    B. Optimization
    C. Evaluation
    D. All
3. The aims of _____ are to transform a high-level query into a relational algebra query
    A. Decomposition
    B. Optimization
    C. Evaluation
    D. All
4. _____identifies the language tokens that appear in the text of the query.
    A. Scanner
    B. Parser
    C. Validate
    D. None
5. _____checks the query syntax to determine whether it is formulated according to the syntax rules of the query language.
    A. Scanner
    B. Parser
    C. Validate
    D. None

Chapter 2: - Transaction Management and Concurrency Control

Chapter objective

- Explain the basics of transaction management and concurrency control

2.1. Transaction Management

A **transaction** is a *unit* of program under execution that **accesses** and possibly **updates** various data items. It is either **completed** in its entirety or **not done** at all. This logical unit of database processing **includes one or more access operations** (read -retrieval, write - insert or update, delete).

E.g. transaction to transfer $50 from account A to account B:

read(A)

A := A – 50

write(A)

read(B)

B := B + 50

write(B)

A successful transaction changes the database from one consistent state to another. A consistent database state is one in which all data **integrity** constraints are satisfied. To ensure consistency of the database, every transaction must begin with the database in a known consistent state and end in a consistent state.

A transaction may consist of **a single SQL statement** or **a collection of related SQL statements.** If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a **read-only transaction**; otherwise it is known as a **read-write transaction**

A transaction can have one of two outcomes:

- **Committed**: If a transaction completed successfully and the database reaches a new consistent state.
- **Aborted**: If the transaction does not execute successfully. The database must be restored to the consistent state it was before the transaction started. Such a transaction is called **rolled back** or **undone**.

According to the number of users who can use the system concurrently a DBMS is classified into two.

1. **Single-User:** A DBMS is **single-user** if at most one user at a time can use the system

2. **Multiuser:** if many users can use the system and accesses the database *concurrently*. Database systems used in banks, insurance agencies, stock exchanges, supermarkets, and many other applications are multiuser systems.

2.2. Concurrency Control

When two or more users are accessing the database simultaneously and at least one is updating data, there may be **interference** that can result in **inconsistencies**. **Concurrency control** is the process of managing simultaneous operations on the database without having them interfere with one another.

2.2.1. Why Concurrency Control is needed?

A major objective in developing a database is **to enable many users to access shared data concurrently.** The objective of concurrency control is **to ensure the serializability of transactions** in a multiuser database environment.

2.2.2. Problems of Concurrent Sharing

The simultaneous execution of transactions over a shared database can create several **data integrity** and **consistency** problems. The three main problems are **lost updates, uncommitted data,** and **inconsistent retrievals.**

A. *Lost Update*

The **lost update** problem occurs when two concurrent transactions, T1 and T2, are updating the same data element and **one of the updates is lost** (overwritten by the other transaction.)

Assume that you have a product whose **current quantity value is 35**. Also assume that two concurrent transactions, T1 and T2, occur that update the quantity value for some item in the database. The transactions are as below:

Transaction	**Computation**
T1: Buy 100 units	quantity =quantity +100
T2 : Sell 30 units	quantity=qunatitiy-30

Following table shows the **serial execution of those transactions under normal circumstances**, yielding the correct answer quantity = 105.

Time	**Transaction**	**Step**	**Stored Value**
1	T1	Read quantity	35
2	T1	quantity =35 +100	
3	T1	Write quantity	135

4	**T2**	**Read quantity**	**135**
5	**T2**	**quantity =135-30**	
6	**T2**	**Write quantity**	**105**

The sequence depicted below shows how the **lost update problem can arise**. Note that the first transaction (T1) has not yet been **committed** when the second transaction (T2) is executed.

Time	Transaction	Step	Stored Value
1	T1	Read quantity	35
2	**T2**	**Read quantity**	**35**
3	T1	quantity =35 +100	
4	**T2**	**quantity =35-30**	
5	T1	Write quantity	135
6	**T2**	**Write quantity**	**5 (Lost update)**

B. *Uncommitted Data (The Temporary Update (or Dirty Read) Problem)*

Uncommitted data occurs when two transactions, T1 and T2, are executed concurrently and the **first transaction (T1) is rolled back after the second transaction (T2) has already accessed the uncommitted data**.

Example: let's use the same transactions described during the lost updates discussion. T1 is forced to roll back due to an error. T1 transaction is rolled back to eliminate the addition of the 100 units. Because T2 subtracts 30 from the original 35 units, the correct answer should be 5.

Following table shows the **serial execution of those transactions under normal circumstances**, yielding the correct answer quantity = 105.

Time	Transaction	Step	Stored Value
1	T1	Read quantity	35
2	T1	quantity =35 +100	
3	T1	Write quantity	135

4	T1	**Rollback**	**35**
5	**T2**	**Read quantity**	**35**
6	**T2**	**quantity =35-30**	
7	**T2**	**Write quantity**	**5**

Following table shows **how the uncommitted data problem** can arise when the ROLLBACK is completed after T2 has begun its execution.

Time	Transaction	Step	Stored Value
1	T1	Read quantity	35
2	T1	quantity =35 +100	
3	T1	Write quantity	135
4	**T2**	**Read quantity** **(Reading uncommitted data)**	**135**
5	**T2**	**quantity =135-30**	
6	**T1**	**ROLLBACK**	**35**
7	**T2**	**Write quantity**	**105**

C. *Inconsistent Retrievals (The Incorrect Summary Problem)*

Inconsistent retrievals occur when a transaction accesses data ***before*** **and** ***after*** **another transaction(s) finish working with such data. For example**, an inconsistent retrieval would occur if transaction T1 calculated some summary (aggregate) function over a set of data while another transaction (T2) was updating the same data. The problem is that the transaction might **read** some data **before they are changed** and other data **after they are changed**, thereby yielding inconsistent results.

To illustrate that problem, assume the following conditions:

1. T1 calculates the **total quantity** of products stored in the PRODUCT table.
2. At the same time, T2 **updates the quantity** for two of the items in the PRODUCT table.

The two transactions are shown as:

Transaction 1	Transaction 2
Select sum(quantity) from product;	Update product set quantity=quantity+10 where pid = 1003
	Update product set quantity =quantity-10 where pid=1004
	Commit

The following table shows the serial execution of those transactions under normal circumstances.

Product_id(pid)	Before update	After Update
1001	8	8
1002	32	32
1003	15	25(15+10)
1004	23	13(23-10)
1005	8	8
Total	86	86

Below Table demonstrates that **inconsistent retrievals** are possible during the transaction execution, making the result of T1's execution incorrect.

Time	Transaction	Action	Value	Total
1	T1	Read quantity for pid=1001	8	8
2	T1	Read quantity for pid=1002	32	40
3	**T2**	**Read quantity for pid=1003**	**15**	
4	**T2**	**Quantity=qunatity+10**		
5	**T2**	**Write quantity for pid=1003**	**25**	
6	**T1**	**Read quantity for pid=1003**	**25**	**65**
7	**T1**	**Read quantity for pid=1004**	**23**	**88**
8	**T2**	**Read quantity for pid=1004**	**23**	

9	**T2**	**Quantity=qunatity-10**		
10	**T2**	**Write quantity for pid=1004**	**13**	
11	**T2**	**Commit**		
12	T1	Read quantity for pid=1005	8	**96**

D. *The Unrepeatable Read Problem*

A transaction T1 reads the same item twice and **the item is changed by another transaction T2 between the two reads**. Hence, T1 receives different values for its two reads of the same item. **For example**, during an airline reservation transaction, a customer inquiry about seat availability on several flights.

When the customer decides on a particular flight, the transaction then reads the number of seats on that flight a second time before completing the reservation, and it may end up **reading a different value** for the item.

2.3. Characterizing Schedules Based on Serializability

2.3.1. Serial Schedule

*A schedule S is **serial** if*, for every transaction T participating in the schedule, **all the operations of T are executed consecutively** in the schedule. **No interleaving** occurs in serial schedule

In serial execution **there is no interference between transactions**, because only one transaction is executing at any given time. So**, it prevents the occurrence of concurrency problems.** Serial schedule **never leaves the database in an inconsistent sate**, therefore every serial schedule is considered **correct**.

2.3.2. Problems of Serial Schedules

They limit **concurrency or interleaving** of operations. If a transaction waits for an I/O operation to complete, we cannot switch the CPU Processor to another transaction. If some transaction T is long, the other transactions must wait for T **to complete all its operations**.

2.4. Non-Serial Schedule (Concurrent Schedule):

The schedule that is not serial is called non-serial schedule. Here **each sequence interleaves operations from the two transactions.**

2.5. When are two schedules considered equivalent?

There are several ways to define schedule equivalence.

A. Result Equivalency

B. Conflict Equivalency

C. View Equivalence

2.5.1. Result Equivalent

Two schedules are called ***result equivalent*** if they **produce the same final state of the database**. However two different schedules may **accidently** produce the same final state. Hence result equivalence **is not used to define equivalence of schedules**.

2.5.2. Conflict Equivalent

Two schedules are said to be ***conflict equivalent*** if the **order of any two conflicting operations is the same in both schedules.** If a schedule ***S*** can be transformed into a schedule ***S´*** by a series of swaps of **non-conflicting instructions**, we say that *S* and *S´* are **conflict equivalent**.

Any given concurrent schedule is said to be conflict serializable schedule **if it is conflict equivalent to one of the possible serial schedule.** Schedule S is ***conflict serializable*** if it is conflict equivalent to some serial schedule S'.

- **Note:**
  - Being ***Serializable*** is distinct from being ***Serial***.
  - ***A Serial Schedule*** leads to **inefficient utilization of CPU** because of no interleaving of operations from different transactions.
  - ***A Serializable Schedule*** gives the **benefits of concurrent execution without giving up any correctness.**

2.5.3. View Equivalent

Two schedules is said to be view equivalent if the following three conditions hold:

1. **Initial Reads:** If T1 reads the initial data X in S1, then T1 also reads the initial data X in S2.
2. **W-R Conflict:** If T1 reads the value written by T2 in S1, then T1 also reads the value written by T2 in S2.
3. **Final Write:** If T1 performs the final write on the data value in S1, then it also performs the final write on the data value in S2.

Chapter Summery

- A **transaction** is a *unit* of program under execution that **accesses** and possibly **updates** various data items.
- A successful transaction changes the database from one consistent state to another.
- The objective of concurrency control is **to ensure the serializability of transactions** in a multiuser database environment.
- The simultaneous execution of transactions over a shared database can create several **data integrity** and **consistency** problems.

- The four main problems are **lost updates, uncommitted data, inconsistent retrievals,** and **Unrepeatable Read Problem.**
- The **lost update** problem occurs when two concurrent transactions, T1 and T2, are updating the same data element and **one of the updates is lost** (overwritten by the other transaction.)
- Uncommitted data occurs when two transactions, T1 and T2, are executed concurrently and the **first transaction (T1) is rolled back after the second transaction (T2) has already accessed the uncommitted data**.
- **Inconsistent retrievals** occur when a transaction accesses data ***before* and *after*** another transaction(s) finish working with such data.
- Unrepeatable Read Problem occurs when transaction T1 reads the same item twice and **the item is changed by another transaction T2 between the two reads**. Hence, T1 receives different values for its two reads of the same item.
- *A schedule S is **serial** if,* for every transaction T participating in the schedule, **all the operations of T are executed consecutively** in the schedule. **No interleaving** occurs in serial schedule
- The schedule that is not serial is called non-serial schedule. Here **each sequence interleaves operations from the two transactions.**
- There are several ways to define schedule equivalence. Those are Result equivalence, conflict equivalence and view equivalence.

Chapter Review Question

1. which one of the following is **not** true about transaction?
   A. A transaction is a *unit* of program under execution.
   B. It is either completed in its entirety or not done at all.
   C. It includes one or more database access operations.
   D. None of the above
2. _____is the process of managing simultaneous operations on the database without having them interfere with one another.
   A. Database recovery
   B. Concurrency control
   C. Serializability
   D. Non serial schedule
3. The _____problem occurs when two concurrent transactions, T1 and T2, are updating the same data element and one of the updates is overwritten by the other transaction.
   A. Lost updates
   B. Uncommitted data
   C. Inconsistent retrievals
   D. Unrepeatable Read Problem

4. _____occur when a transaction accesses data ***before*** **and** ***after*** another transaction(s) finish working with such data.
    A. Lost updates
    B. Uncommitted data
    C. Inconsistent retrievals
    D. Unrepeatable Read Problem
5. Which one of the following is used to define schedule are equivalence?
    A. Initial Reads
    B. W-R Conflict
    C. Final Write
    D. All

Chapter 3: - Database Recovery Techniques

Chapter Objective

- Use different recovery methods when there is a database failure

Recovery from transaction failures means that the **database is restored to the most recent consistent state just before the failure**. It is a **service provided by the DBMS** to ensure that the database is **reliable** and **remains in a consistent state** in the presence of failure.

3.1. Database Recovery Techniques

Recovery algorithms are techniques to ensure transaction atomicity and durability despite failures. The recovery subsystem, using recovery algorithm, ensures atomicity by undoing the actions of transactions that do not commit and Durability by making sure that all actions of committed transactions survive even if failures occur.

- Two main approaches in recovery process
  - Log-based recovery using WAL protocol.
  - Shadow-paging.

3.2. Write-Ahead Logging (WAL)

Write-Ahead Logging (WAL) protocol ensures that a record entry- of every change to the DB is available while attempting to recover from a crash. WAL ensures that the BFIM of the data item is recorded in the appropriate log entry and that the log entry is flushed to disk before the BFIM is overwritten with the AFIM in the database on disk.

When a log record is written, it is stored in the current log in the DBMS cache and after it written to disk as soon as is feasible. With Write-Ahead Logging, when the a particular data block update, the log blocks must be first written to disk before the data block itself can be written back to disk. IBM DB2, Informix, Microsoft SQL Server, Oracle 8, and Sybase ASE all use a WAL scheme for recovery.

To facilitate the recovery process, the DBMS recovery subsystem may need to maintain a number of lists.

- List of active transactions: transactions started but not committed yet.
- List of committed transactions since last checkpoint.
- List of aborted transactions since last checkpoint.

3.3. Main Recovery Techniques

3.3.1. Deferred Update Techniques.

A transaction **cannot change** the database on disk **until it reaches its commit point.** A transaction does not reach its commit point **until all its update operations are recorded in the log** and **the**

log is force-written to disk –i.e. **WAL**. During commit, the updates are **first recorded in the log** and **then written to the DB.**

If a transaction fails before reaching its commit point, no UNDO is needed because the transaction has not affected the database on disk in any way. If there is a crash, it may be necessary to REDO the effects of committed transactions from the Log because their effect may not have been recorded in the database. Deferred update also known as NO-UNDO/REDO algorithm.

3.3.1.1.Advantages of Deferred Update Techniques

A transaction does not record any changes in the DB on disk until it commits & so never rollback because of transaction failure during transaction execution. A transaction will never read the value of an item that is written by an *uncommitted transaction*; hence no cascading rollback will occur.

3.3.1.2. Drawbacks of Deferred Update Techniques

Limits the concurrent execution of transactions because all items **remain locked until the transaction reaches its commit** point –due to strict 2PL. **Require excessive buffer space to hold all updated items** until the transactions commit.

3.3.2. Immediate Update Techniques

In these techniques, the DB on disk can be updated immediately without any need to wait for the transaction to reach its commit point. However, the update operation must still be recorded in the log (on disk) *before it is applied to the database.* Using WAL protocol- so that we can recover in case of failure. If a transaction fails before reaching commit point, it must be rolled back by undoing the effect of its operations on the DB.

3.3.3. Shadow Paging

The DB is made up of 'n' fixed-size disk pages –blocks. A directory with 'n' entries where the i^{th} entry points to the i^{th} DB page on disk.

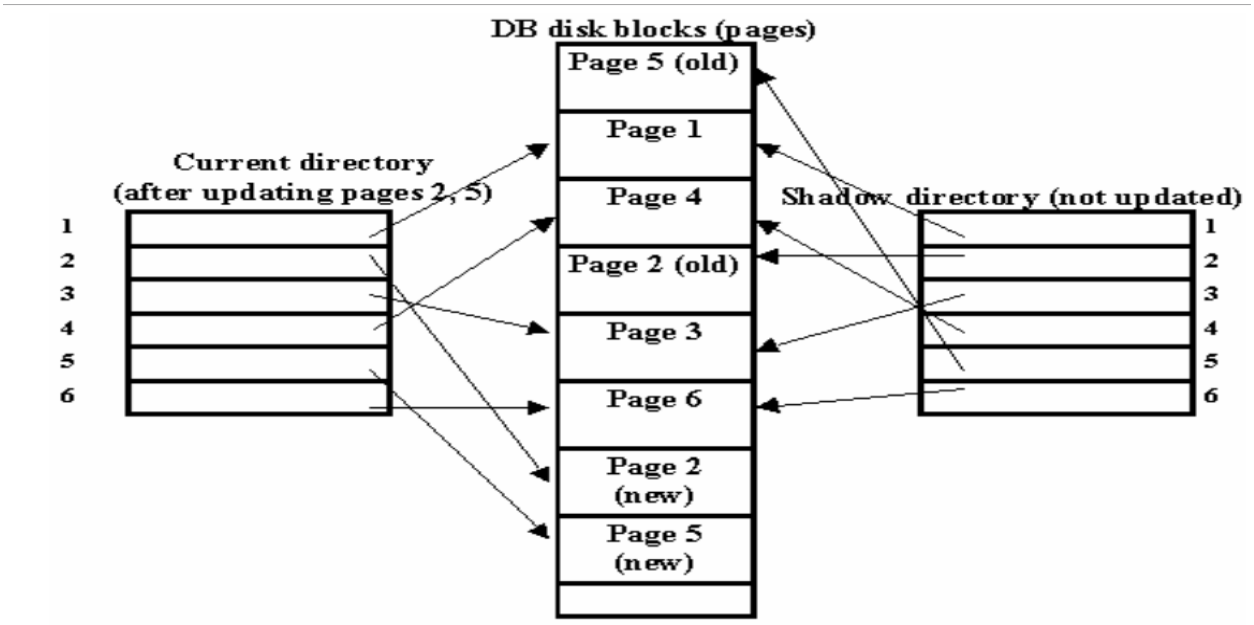
- All references –reads or writes- to the DB pages on disk go through the directory.
- The directory is kept in main memory if not too large.
- The current directory entries point to the most recent or current DB pages on disk.

When a transaction begins executing, the current directory is copied into a shadow directory and the shadow directory is saved on disk. During transaction execution, all updates are performed using the current directory and the shadow directory is never modified.

When a write_item operation is performed

- A new copy of the modified DB page is created and the old copy is not overwritten. Two versions of the page updated by the transaction are kept.
- The new page is written elsewhere on some unused disk block
- The current directory entry is modified to point to the new disk block.

- The shadow directory is not modified.



- **To recover from a failure:**
  - Delete the modified database pages & discard the current directory.
  - The state of the database before transaction execution is available through the shadow directory and is recovered by reinstating the shadow directory.
  - NO-UNDO/NO-REDO technique since neither undoing or redoing of data items.
- **Drawbacks**
  - The updated pages change location on disk. Difficult to keep related DB pages close together on disk without complex storage management strategies.
  - The overhead of writing shadow directories to disk as transactions start is very high.
  - A complicated garbage collection when a transaction commits e.t.c.

Chapter Summery

- Recovery from transaction failures means that the **database is restored to the most recent consistent state just before the failure**.
- Recovery algorithms are techniques to ensure transaction atomicity and durability despite failures.
- There are two main approaches in recovery process, those are Log-based recovery using WAL protocol and Shadow-paging.
- Write-Ahead Logging (WAL) protocol ensures that a record entry- of every change to the DB is available while attempting to recover from a crash.
- There are three main database recovery techniques those are Deferred update, Immediate Update, and Shadow Paging.
- In Deferred update a transaction **cannot change** the database on disk **until it reaches its commit point.** A transaction does not reach its commit point **until all its update operations are recorded in the log** and **the log is force-written to disk.**
- Deferred update also known as NO-UNDO/REDO algorithm
- In Immediate Update techniques, the DB on disk can be updated immediately without any need to wait for the transaction to reach its commit point.
- In shadow paging when a transaction begins executing, the current directory is copied into a shadow directory and the shadow directory is saved on disk.
- During transaction execution, all updates are performed using the current directory and the shadow directory is never modified.

Chapter Review Question

1. _____protocol ensures that a record entry- of every change to the DB is available while attempting to recover from a crash.
   A. Shadow Paging
   B. Write-Ahead Logging (WAL)
   C. Immediate Update Techniques
   D. None of the above
2. Which one of the following is not database recovery technique?
   A. Deferred update
   B. Immediate Update
   C. Strict Schedule
   D. Shadow Paging
3. _____ also known as NO-UNDO/REDO algorithm
   A. Deferred update
   B. Immediate Update
   C. Strict Schedule

D. Shadow Paging

4. _____techniques, the DB on disk can be updated immediately without any need to wait for the transaction to reach its commit point.
   A. Deferred update
   B. Immediate Update
   C. Strict Schedule
   D. Shadow Paging
5. Which one of the following technique is not used Write-Ahead Logging (WAL) protocol?
   A. Deferred update
   B. Immediate Update
   C. Shadow Paging
   D. None

Chapter 4: - Database Security

Chapter Objective

- Describe database security

4.1. Database security

Database security is the mechanisms that **protect** the database against **intentional** or **accidental** threats. A **threat** means **any situation or event** *whether intentional or accidental* that **may adversely affect the organization**.

4.2. Threats to Databases

4.2.1. Loss of integrity

Database integrity refers to the requirement that **information be protected from improper modification**. Integrity is lost if **unauthorized changes are made to the data** by either intentional or accidental acts. If the loss of system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in **inaccuracy**, **fraud**, or **erroneous decisions**.

4.2.2. Loss of availability:

Database availability refers to **making objects available** to a human user or a program to which they have a **legitimate right**. Loss of availability is the database objects is **not available** to legitimate/authorized users.

4.2.3. Loss of confidentiality:

Database confidentiality refers to **the protection of data from unauthorized disclosure.** Unauthorized or unintentional disclosure could result in *loss of public confidence, embarrassment, or legal action against the organization.*

4.3. Control Measures

To protect databases against the different types of threats, it is common to implement ***four*** *kinds of control measures*:

4.3.1. Access control

Access control is a mechanism used to **preventing unauthorized persons from accessing the system itself,** either to obtain information or to make malicious changes in a portion of the database. The security mechanism of a DBMS **must include provisions for restricting access to the database system as a whole.** It is handled by creating **user accounts** and **passwords** to control the login process by the DBMS.

4.3.2. Inference control

It is the security problem associated with **controlling the access to a statistical database.** Statistical databases are used mainly to **produce statistics about various populations.** (*A **population** is a set of tuples of a relation (table) that satisfy some selection condition.).* Statistical users are **not allowed to retrieve individual data**, such as the income of a specific person.

Statistical users are **permitted to retrieve statistical information** about the populations, such as averages, sums, counts, maximums, minimums, and standard deviations.

4.3.3. Flow control

It prevents information from ***flowing in such a way that it reaches unauthorized users.*** A **flow policy *specifies the channels*** along which information is allowed to move. The simplest flow policy specifies just two classes of information: confidential (C) and non-confidential (N) and allows all flows **except** those from class C to class N.

Covert channels are pathways for information to flow implicitly in ways that ***violate the security policy of an organization.***

4.3.4. Encryption

It is **used to protect sensitive data** (such as credit card numbers) that is being transmitted via some type communication network. The data is **encoded** using some **encoding algorithm**. An **unauthorized** user who access encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting algorithms (or keys) to decipher data.

4.4. Database Security and the database administrator (DBA)

The database administrator (DBA) is the **central authority** for managing a database system. The DBA is responsible for the **overall security of the database system.** The DBA has a **DBA account** in the DBMS, sometimes called a **system** or **super user account**, which **provides powerful capabilities that are not made available to regular database accounts and users.**

DBA-privileged commands include commands for performing the following types of actions:

- *Action 1:* Account creation. This action creates a new account and password for a user or a group of users to enable access to the DBMS.
- *Action 2:* Privilege granting. This action permits the DBA to grant certain privileges to certain accounts.
- *Action 3:* Privilege revocation. This action permits the DBA to revoke (cancel) certain privileges that were previously given to certain accounts.
- *Action 4:* Security level assignment. This action consists of assigning user accounts to the appropriate security clearance level.

4.5. Database Audits

DBA Reviewing the system log to **examine all accesses and operations** applied to the database during a certain time period. When an **illegal or unauthorized operation is found**, the DBA can **determine the account number used to perform the operation**.

Database audits are particularly important for **sensitive databases that are updated by many transactions and users**, such as a banking database which is updated by many bank tellers. A database log that is used mainly for security purposes is sometimes called an **audit trail**.

4.6. Authorization Subsystem

There are 3 different authorization subsystems.

1. **Discretionary Access Control**
2. **Mandatory Access Control**
3. **Role Based Access Control**

4.6.1. Discretionary Access Control

These are used to **grant privileges** to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update). The main idea is to **include statements in the query language** that allow the DBA and selected users to **grant and revoke privileges.**

Operations may also be controlled; thus, having an account does not necessarily entitle the account holder to all the functionality provided by the DBMS. Informally, there are **two levels for assigning privileges** to use the database system:

- **The account level: -** At this level, the DBA specifies the **particular privileges** that each account holds **independently** of the relations in the database.
- **The relation (or table) level: -** At this level, the DBA can **control the privilege to access each individual relation** or **view** in the database.

The owner of a relation is given all privileges on that relation. The owner account holder **can pass privileges** on any of the owned relations to other users by **granting** privileges to their accounts. In SQL a **GRANT** command is included for the purpose of granting privileges.

In some cases, it is desirable to grant a privilege to a user temporarily. *For example,* the owner of a relation may want to grant the SELECT privilege to a user for a specific task and then revoke that privilege once the task is completed. In SQL a REVOKE command is included for the purpose of cancelling privileges.

Whenever the owner *A* of a relation *R* grants a privilege on *R* to another account *B*, the privilege can be given to *B with* or *without* the GRANT OPTION. If the GRANT OPTION is given, this means that *B can also grant that privilege on R to other accounts.*

In this way, privileges on *R* can propagate to other accounts without the knowledge of the owner of *R*. If the owner account *A* now revokes the privilege granted to *B*, all the privileges that *B* propagated based on that privilege *should automatically be revoked* by the system.

4.6.2. Mandatory Access Control

Mandatory access control is a security mechanism used to **classifies data and users based on security classes.** This approach would typically be *combined* with the discretionary access control

mechanisms. Typical **security classes** are **top secret** (TS), **secret** (S), **confidential** (C), and **unclassified** (U), where **TS is the highest** level and **U the lowest**. For simplicity, **TS ≥ S ≥ C ≥ U**

Bell-LaPadula model is a model for **multilevel** security, it classifies each **subject** (user, account, program) and **object** (relation, tuple, column, view, operation) into one of the security classifications **TS, S, C,** or **U. Classification** of a subject *S* is referred to as **class(*S*)** and the **classification** of an object *O* as **class(*O*)**. Based on the subject/object classifications ***two restrictions are enforced on data access:***

A subject ***S*** is **not allowed read access** to an object ***O*** unless **class(*S*) ≥ class(*O*)**. This is known as the **simple security property**. A subject ***S*** is **not allowed to write** an object ***O*** unless **class(*S*) ≤ class(*O*).** This is known as the **star property (or * property).**

4.6.3. Role-Based Access Control

Role: is a named **group of related privileges** that can be granted to the user. This method makes it **easier** to revoke and maintain privileges. A user can have access to several roles, and several users can be assigned the same role. Roles are typically created for a database application. Its basic notion is that **privileges and other permissions** are associated with **organizational roles**, rather than individual users. Individual users are then assigned to appropriate roles. Roles can be created using the **CREATE ROLE** and removed by **DROP ROLE** commands. The **GRANT** and **REVOKE** commands can then be used to assign and revoke privileges **from roles**, as well as for individual users when needed.

Chapter Summery

- Database security is the mechanisms that **protect** the database against **intentional** or **accidental** threats.
- There are three main threat to the database those are loss of integrity, loss of confidentiality, and loss of availability.
- Integrity is lost if **unauthorized changes are made to the data** by either intentional or accidental acts.
- Loss of availability is the database objects is **not available** to legitimate/authorized users.
- Loss of confidentiality is Unauthorized or unintentional disclosure of the data.
- To protect databases against the different types of threats, it is common to implement ***four*** *kinds of control measures*: those are Access control, inference control, flow control, and Inscription.
- Access control is a mechanism used to **preventing unauthorized persons from accessing the system itself,** either to obtain information or to make malicious changes in a portion of the database.
- Inference control is the security problem associated with **controlling the access to a statistical database.**

- Flow control prevents information from ***flowing in such a way that it reaches unauthorized users.***
- Encryption is **used to protect sensitive data** (such as credit card numbers) that is being transmitted via some type communication network. The data is **encoded** using some **encoding algorithm**.
- The database administrator (DBA) is the **central authority** for managing a database system. The DBA is responsible for the **overall security of the database system.**
- DBA Reviewing the system log to **examine all accesses and operations** applied to the database during a certain time period.
- A database log that is used mainly for security purposes is sometimes called an **audit trail**.
- There are 3 different authorization subsystems. Those are Discretionary Access Control, Mandatory Access Control, and Role Based Access Control.
- Discretionary Access Control is used to **grant privileges** to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).
- Mandatory access control is a security mechanism used to **classifies data and users based on security classes.**
- **Role:** is a named **group of related privileges** that can be granted to the user. Roles are typically created for a database application.
- Its basic notion is that **privileges and other permissions** are associated with **organizational roles**, rather than individual users. Individual users are then assigned to appropriate roles.

Chapter Review Question

1) _____ is lost if **unauthorized changes are made to the data** by either intentional or accidental acts.

A. Availability
B. Integrity
C. Confidentiality
D. All

2) _____ is unauthorized or unintentional disclosure of the data.

A. Loss of Availability
B. Loss of integrity
C. Loss of Confidentiality
D. All

3) _____is the security problem associated with controlling the access to a statistical database.

A. Access control
B. Inference control

C. Flow control
D. Encryption

4) _____is used to protect sensitive data that is being transmitted via some type communication network.
   A. Access control
   B. Inference control
   C. Flow control
   D. Encryption
5) Which one of the following tasks **not** performed by database administrator (DBA)?
   A. Security level assignment
   B. Privilege granting
   C. Account creation
   D. None of the above
6) The main idea of _____ is to include statements in the query language that allow the DBA and selected users to **grant** and **revoke** privileges**.**
   A. Discretionary Access Control
   B. Mandatory Access Control
   C. Role Based Access Control
   D. All

Chapter 5: - Distributed Database Systems

Chapter Objective

- Design a distributed database system in homogenous and heterogeneous environments

5.1. Distributed Database Systems

The use of a centralized database required that **corporate data be stored in a single central site**, usually a mainframe computer. Data access was provided through **dumb terminals**. The centralized approach worked well to fill the structured information needs of corporations, but it fell short when **quickly moving events required faster response times.**

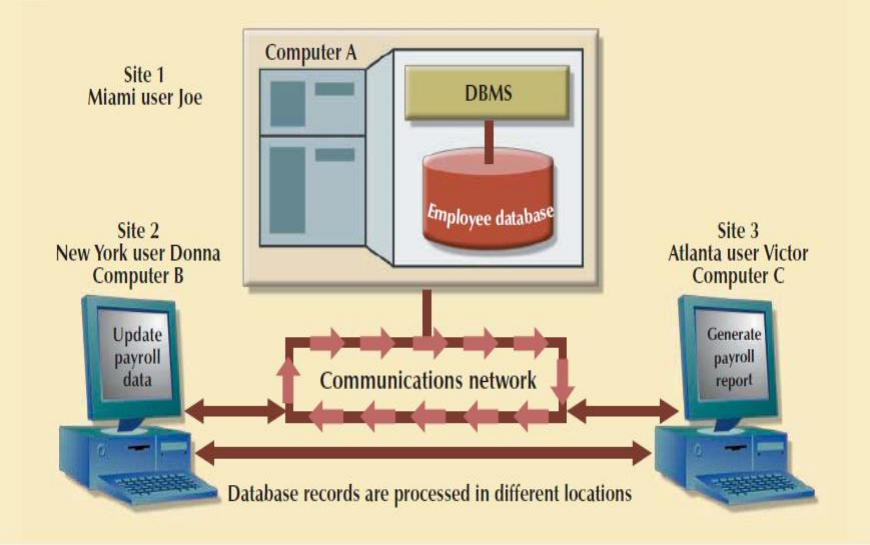
A **distributed database management system** (**DDBMS**) governs the **storage** and **processing** of logically related data over **interconnected computer systems** in which both **data** and **processing** are **distributed among several sites.**

Problems Seen in Centralized Database Management	
Performance degradation	Because of a **growing number of remote locations** over greater distances
High costs	Associated with **maintaining** and **operating** large central (mainframe) database systems.
Reliability problems	Created by **dependence on a central site** (single point of failure syndrome) and the need for data replication.
Scalability problems	Associated with the **physical limits imposed by a single location** (temperature conditioning, and power consumption.)

5.2. Distributed Processing and Distributed Databases

5.2.1. Distributed Processing

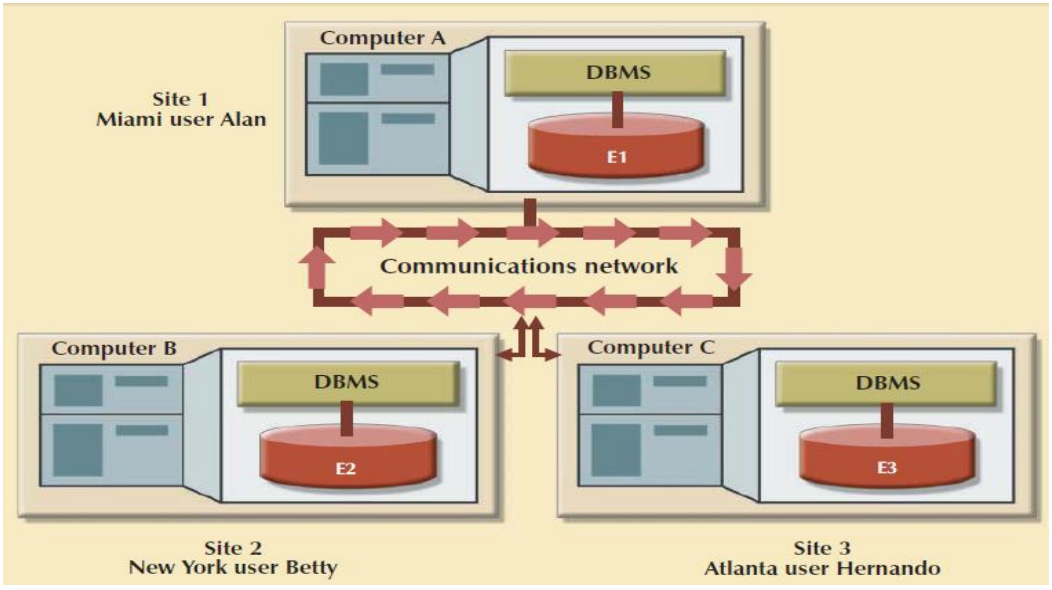
In **distributed processing**, a database's **logical processing is shared** among two or more physically independent sites that are connected through a network. **For example**, the data input/output (I/O), data selection, and data validation might be performed on one computer, and a **report** based on that data might be created on another computer.



The above figure shows that a distributed processing system shares the database processing chores among three sites connected through a communications network. Although the **database resides at only one site (Miami),** each site can access the data and update the database. The database is located on Computer A, a network computer known as the ***database server*.**

5.2.2. Distributed Databases

A **distributed database**, on the other hand, stores **a logically related database** over two or more physically independent sites. The sites are connected via a computer network. *In contrast, the distributed processing system* ***uses only a single-site database*** *but* ***shares the processing chores among several sites.*** In a distributed database system, a database is composed of several parts known as **database fragments**. The database fragments are **located at different sites** and **can be replicated a**mong various sites. Each database fragment is, in turn, managed by its **local database process.**



The database in Figure is divided into three database fragments (E1, E2, and E3) located at different sites. The computers are connected through a network system.

In a **fully distributed database**, the users Alan, Betty, and Hernando **do not need to know the name or location of each database fragment in order to access the database.** Also, the users might be located at sites other than Miami, New York, or Atlanta and still be able to access the database as a **single logical unit.**

Note:

Distributed processing does not require a distributed database, but a **distributed database requires distributed processing** (each database fragment is managed by its own local database process). Both distributed processing and distributed databases require a **network** to connect all components.

5.3. DDBMS Components

The DDBMS must include at least the following components:

1. ***Computer workstations or remote devices*** (sites or nodes) that form the network system. *The distributed database system must be* ***independent*** *of the* ***computer system hardware.***
2. ***Network hardware and software*** **components** that reside in each workstation or device. The network components allow all sites **to interact and exchange data**.
3. ***Communications media*** that carry the data from one node to another. The DDBMS must be **communications media-independent**; that is, it must be able to support several types of communications media.
4. The **transaction processor** (**TP**), which is the ***software component*** found in each computer or device **that requests data**. The transaction processor **receives and processes the application's data requests** (remote and local). The TP is also known as the **application processor** (**AP**) or the **transaction manager** (**TM**).
5. The **data processor (DP),** which is the ***software component*** residing on each computer or device that **stores and retrieves data located at the site.** The **DP** is also known as the **data manager (DM**). A data processor may even be a centralized DBMS.

5.4. Distributed Database Transparency Features

DDBMS transparency features have the common property of allowing the end user to feel like **the database's only user**. In other words, the user believes that (s)he is working with a **centralized DBMS**; all complexities of a distributed database are **hidden** to the user. The DDBMS transparency features are:

5.4.1. Distribution transparency

Which allows a distributed database to be treated as **a single logical database**. If a DDBMS exhibits distribution transparency, the user **does not need to know:**

- That the data are **partitioned**—meaning the table's rows and columns are split vertically or horizontally and stored among multiple sites.
- That the data can be **replicated** at several sites.
- The **data location.**

5.4.2. Transaction transparency

Which allows a transaction to **update** data at **more than one network site.** Transaction transparency ensures that the transaction will be either entirely completed or aborted, thus maintaining database integrity.

5.4.3. Failure transparency

Which ensures that the system will **continue to operate** in the event of a node failure. Functions that were lost because of the failure will be picked up by another network node.

5.4.4. Performance transparency

Which allows the system to **perform** as if it were a centralized DBMS. The system will **not suffer any performance degradation** due to its use on a network or due to the network's platform differences. Performance transparency also ensures that the system will find the most **cost-effective path to access remote data.**

5.5. Distribution Transparency

Distribution transparency allows a physically dispersed database to be managed as though it were a **centralized database.** The level of transparency supported by the DDBMS **varies** from system to system. Three levels of distribution transparency are recognized:

5.5.1. Fragmentation transparency

It is the highest level of transparency. The end user or programmer **does not need to know that a database is partitioned.** Therefore, **neither fragment names nor fragment locations** are specified prior to data access.

5.5.2. Location transparency

It exists when the end user or programmer must specify the database fragment names but **does not** need to specify **where those fragments are located**.

5.5.3. Local mapping transparency

It exists when the end user or programmer must specify **both the fragment names and their locations.**

5.6. Distributed Database Design

- The design of a distributed database introduces three new issues:

Issue	Managed by
1. How to partition the database into fragments. 2. Which fragments to replicate.	Data Fragmentation & Data Replication
3. Where to locate those fragments and replicas.	Data Allocation

5.6.1. Data fragmentation

It allows you to break a single object into two or more segments, or fragments. The object might be a user's database, a system database, or a table. Each fragment can be stored at any site over a computer network. Information about data fragmentation is stored in the **distributed data catalog (DDC),** from which it is accessed by the TP to process user requests.

Three types of data fragmentation at table level are **horizontal, vertical, and mixed.** (A fragmented table can always be re-created from its fragmented parts by a combination of unions and joins.)

A. **Horizontal fragmentation** refers to the division of a relation into subsets (fragments) of tuples (rows). Each fragment is **stored at a different node**, and each fragment has **unique rows.** However, the unique rows all have **the same attributes** (columns). In short, each fragment represents the equivalent of a SELECT statement, with the WHERE clause on a single attribute.
B. **Vertical fragmentation: -** It refers to the division of a relation into attribute (column) subsets. Each subset (fragment) is ***stored at a different node***, and each fragment has **unique columns**—with the exception of the key column, which is common to all fragments. This is the equivalent of the **PROJECT** statement in SQL.
C. **Mixed fragmentation: -** It refers to a combination of horizontal and vertical strategies. In other words, a table may be divided into several horizontal subsets (rows), each one having a subset of the attributes (columns).

5.6.2. Data Replication

Data replication refers to the storage of data copies at multiple sites served by a computer network. Fragment copies can be stored at several sites to serve specific information requirements. Suppose database A is divided into two fragments, A1 and A2. Within a replicated distributed database, fragment A1 is stored at sites S1 and S2, while fragment A2 is stored at sites S2 and S3.

Replicated data are subject to the mutual consistency rule. The **mutual consistency rule** requires that **all copies of data fragments be identical.** Therefore, to maintain data consistency among the replicas, the DDBMS **must ensure that a database update is performed at all sites where replicas exist.**

Three replication scenarios exist: a database can be fully replicated, partially replicated, or un replicated.

1. **Fully Replicated Database**

A **fully replicated database** stores multiple copies of ***each*** database fragment at multiple sites. In this case, all database fragments are replicated. A fully replicated database can be **impractical** due to the amount of **overhead** it imposes on the system.

2. **Partially Replicated Database**

A **partially replicated database** stores multiple copies of *some* database fragments at multiple sites. Most DDBMSs are able to handle the partially replicated database well.

3. **Un Replicated Database**

An **un replicated database** stores each database fragment at a single site. Therefore, there are no duplicated database fragments.

5.6.3. Data Allocation

Data allocation describes the process of deciding **where to locate data.** Data allocation strategies are as follows:

- With **centralized data allocation**, the entire database is stored at **one site.**
- With **partitioned data allocation**, the database is divided into two or more disjointed parts (fragments) and stored at **two or more sites.**
- With **replicated data allocation**, copies of one or more database fragments are stored at several sites.
- Most data allocation studies focus on one issue: ***which* data to locate *where***.

Chapter Summery

- A **distributed database management system** (**DDBMS**) governs the **storage** and **processing** of logically related data over **interconnected computer systems** in which both **data** and **processing** are **distributed among several sites.**
- In **distributed processing**, a database's **logical processing is shared** among two or more physically independent sites that are connected through a network.
- A **distributed database**, on the other hand, stores **a logically related database** over two or more physically independent sites. The sites are connected via a computer network.
- The DDBMS must include at least the following components:
    - *Computer workstations or remote devices*
    - *Network hardware and software* components
    - *Communications media*
    - Transaction processor (TP),
    - Data processor (DP)
- DDBMS transparency features have the common property of allowing the end user to feel like **the database's only user**.
- The DDBMS transparency features are: Distribution transparency, Fragmentation transparency, Location transparency, and Local mapping transparency.
- Distribution transparency allows a physically dispersed database to be managed as though it were a **centralized database.**
- Three levels of distribution transparency are recognized: those are Fragmentation transparency, Location transparency, and Local mapping transparency.
- The design of a distributed database introduces three new issues those are Data Fragmentation, Data Replication, and Data Allocation.
- **Data Fragmentation** allows you to break a single object into two or more segments, or fragments.
- **Data replication** refers to the storage of data copies at multiple sites served by a computer network.
- **Data allocation** describes the process of deciding where to locate data.

Chapter Review Question

1. Which one of the following is not problems of Centralized Database Management?
    A. Performance degradation
    B. Low costs
    C. Reliability problems
    D. Scalability problems

2. Which one of the following is components of DDBMS?
    A. *Network hardware and software* components
    B. Communications media
    C. Transaction processor (TP),
    D. Data processor (DP)
    E. All
3. _____ensures that the system will **continue to operate** in the event of a node failure.
    A. Distribution transparency
    B. Fragmentation transparency
    C. Location transparency
    D. Local mapping transparency
4. _____ exists when the end user or programmer must specify the database fragment names but **does not** need to specify **where those fragments are located.**
    A. Fragmentation transparency
    B. Location transparency
    C. Local mapping transparency
    D. All of the above
5. _____ exists when the end user or programmer must specify **both the fragment names and their locations.**
    A. Fragmentation transparency
    B. Location transparency
    C. Local mapping transparency
    D. All of the above
6. _____refers to the storage of data copies at multiple sites served by a computer network.
    A. Data Fragmentation
    B. Data Replication
    C. Data Allocation
    D. None of the above
7. _____ allows you to break a single object into two or more segments.
    A. Data Fragmentation
    B. Data Replication
    C. Data Allocation
    D. None of the above