UNIT - 1 | Transaction Management

Transaction management ensures data integrity in systems that involve changes to multiple data elements. It's a process that oversees database operations to guarantee data consistency. It ensures all parts of a data change are completed successfully or none are applied.

Transaction: A single unit of work in a database system. Represents a series of database operations (reads, writes, etc.).

Savepoints: Mark a point within a transaction where a rollback can occur.

Transaction States

Transaction States describe the different stages a transaction goes through in its lifecycle within a Database Management System. It ensures data consistency during database operations.

- **Active State:** Executing instructions (reads, writes).
- **Partially Committed State:** Changes temporary (not yet on disk).
- **Committed State:** Successful completion (changes permanent).
- **Failed State (Aborted State):** Error occurred (changes rolled back).

ACID Properties

ACID ensures data reliability in databases. It treats multiple operations as one, guaranteeing a consistent state even in case of failures.

- **Atomicity:** Transactions are all-or-nothing. Everything happens or nothing does.
- **Consistency:** Transactions move the database from one valid state to another.
- **Isolation:** Concurrent transactions don't interfere with each other.
- **Durability:** Committed transactions survive system failures.

Operations in Transactions

Transactions in a DBMS are a series of logically related operations that ensure data integrity.

Data Operations:

- **Read:** Grab data (e.g., checking account balance).
- **Write:** Modify data (e.g., updating balance after transfer).

Transaction Control:

- **Commit:** Finalize changes (e.g., permanent transfer).
- **Rollback:** Undo changes (e.g., insufficient funds during transfer).

Storage Structures

DBMS stores data in specific formats that affect how quickly we can access and use it.

Types of Storage Structures:

1. File Organization: How data is physically stored in a file.

- **Heap Files:** Fast for adding data, slow for searching.
- **Sorted Files:** Fast for searching by key, slow for inserts (requires re-sorting).
- **Hashed Files:** Fast for retrieving specific data by key, less efficient for finding ranges.

2. Indexed Organization: Improves access speed using indexes.

- **B-Trees:** Efficient for searching ranges and specific data.
- **ISAM:** Good for both searching by key and sequential access.

Concurrent Executions

The ability of a DBMS to process multiple transactions simultaneously in a shared database.

Benefits: Increased Throughput, Reduced Wait Time, Improved Resource Utilization.

Concurrent execution can lead to data inconsistencies if not managed properly.

Interleaved Execution: Transactions might access and modify the same data items at the same time, leading to unpredictable results.

Serializability

Serializability ensures that concurrent transactions on a database appear to run one after another, even if they happen at the same time.

1. Conflict Serializability: This is stricter and avoids data inconsistencies by ensuring conflicting operations on the same data item happen in a specific order (read before write, or write before read).

2. View Serializability: This is more relaxed and focuses on the final outcome. As long as the final state of the database is consistent, the exact order of conflicting operations within a transaction doesn't matter. This allows for more concurrent access.

Concurrency Control

Concurrency control is a mechanism in DBMS that manages concurrent access to data by multiple users or applications. It ensures data consistency and integrity in a multi-user environment.

Goals of Concurrency Control: ACID properties, Serializability

Concurrency Problems: When multiple users edit the same data at the same time in a database, things can get messy (data inconsistencies).

Types of Concurrency Problems:

- **Lost Updates:** Two updates collide, and only one wins.
- **Inconsistent Reads:** Reading data twice shows different results due to mid-air updates.
- **Phantom Reads:** New or missing data appears/disappears between reads due to inserts/deletes by others.
- **Dirty Reads:** Reading data that might be later discarded (Seeing Unfinished Work).

Preventing Concurrency Problems:

- **Locking Mechanisms:** Locks prevent conflicts during data updates.
- **Timestamp Ordering:** Timestamps ensure a specific execution order.
- **Optimistic Concurrency Control:** Allows concurrency but checks for conflicts later.

- **Database Isolation Levels:** Define the degree of isolation between transactions.

Concurrency Control Protocols

Mechanisms to manage concurrent access to data by multiple users or processes. Their goal is to ensure data consistency and integrity despite these simultaneous operations.

1. Lock-Based Protocols

- Transactions acquire locks on specific data items (records, tables) before modifying them.
- Prevents other transactions from conflicting modifications.

Modes of Locks:

- **Exclusive Lock (X-lock):** The exclusive access holding transaction can modify the data.
- **Shared Lock (S-lock):** Many users can read, but no one can write.

Granting of Locks: The lock manager determines if a lock request can be granted based on the current locking state and the type of lock requested (X or S).

- **Granting an X-lock:** Only granted if no one else has any lock (read or write) on the data.
- **Granting an S-lock:** Okay if others already have S-locks (read-only access), but blocks any exclusive lock requests.

Two-Phase Locking Protocol (2PL): 2PL ensures a transaction's locking actions have a well-defined order, preventing inconsistencies.

- **Growing Phase (Locking Up):** Grab locks on needed data. No releasing locks yet.
- **Shrinking Phase (Releasing Locks):** Release unused locks. No grabbing new ones now.

Lock Point: The Point at which the growing phase ends.

2. Timestamp-Based Protocols

- Each transaction receives a unique timestamp when it starts.
- Transactions are serialized based on their timestamps, ensuring a specific execution order.

Timestamp Ordering Protocol:

- Assigns a unique timestamp (TS) to each transaction.
- Ensures serializability by ordering transactions based on their timestamps.

Thomas Write Rule (Modification of Timestamp Ordering Protocol):

- A concurrency control mechanism used in database systems to ensure data consistency.
- Improves efficiency over basic timestamp ordering.
- Allows ignoring outdated writes, improving performance compared to strictly rolling back transactions

3. Validation-Based Protocols

- Transactions are validated after their execution, checking for conflicts with previously committed transactions. If a conflict is found, the transaction is rolled back (undone).

Process:

- **Read Phase:** Transaction reads data and stores it in local copies.
- **Validation Phase:** Local copies are compared against actual data for conflicts.
- **Write Phase (if validation succeeds):** Local copies update the actual database.

Deadlock Handling

Deadlocks occur in a DBMS when two or more transactions wait indefinitely for resources (locks) held by each other. This creates a circular dependency, halting all involved transactions.

1. Deadlock Prevention: DBMS analyzes transactions to ensure resource allocation never leads to a deadlock scenario. Prevents deadlocks altogether.

2. Deadlock Avoidance: DBMS maintains information about resource requests and uses algorithms to predict and avoid deadlocks. Prevents potential deadlocks during transactions.

3. Deadlock Detection and Recovery: DBMS monitors transactions and identifies wait-for graphs to detect deadlocks. Allows deadlocks to occur but detects them for resolution.

UNIT - 2 | Database System Architectures

Database architecture is the blueprint for how data is stored, organized, accessed, and managed. It's like the foundation of a house, crucial for smooth data operations.

Types of Database Architectures:

1. Tier Architecture:

- **1-Tier (Single-tier):** Easiest to develop, but least scalable.
- **2-Tier:** Separates user interface from data storage. More scalable.
- **3-Tier:** Most scalable, with a dedicated layer for handling complex business logic

2. Data Model:

- **Relational Model:** Data is stored in tables with rows and columns. Relationships between tables are established through foreign keys.
- **NoSQL Model:** More flexible, suitable for large unstructured datasets.

Centralized Architecture

All data and processing logic reside on a single server. Clients (usually dumb terminals) connect and interact directly with the server.

Pros: Simple to set up and manage. Cost-effective for small databases.

Cons: Limited scalability, Single point of failure.

Client-Server Architecture

Data is stored on a central server, but clients (PCs, workstations) handle user interface and some processing. Clients communicate with the server to access and manipulate data.

Pros: More scalable, Improved performance, More flexible.

Cons: More complex to set up and manage, Requires robust network infrastructure.

Server System Architecture

Refers to the organization of processes and data storage on the server side.

- **Transaction Servers:** Used in relational databases, these handle queries and transactions efficiently.
- **Data Servers:** Used in object-oriented databases, these focus on data access and processing on powerful client machines.

1. Transaction Server Process Structure:

Consists of multiple processes that manage user requests (transactions), data access, and overall system functionality.

Components:

- **Server Processes:** Handle user requests, run them concurrently, and return results.
- **Lock Manager Process:** Ensures data consistency by controlling access.
- **Database Writer Process:** Periodically saves changes from memory to disk.
- **Log Writer Process:** Tracks all database modifications for recovery.
- **Checkpoint Process:** Creates regular backups for faster disaster recovery.
- **Process Monitor Process:** Stores frequently used data for faster access.
- **Shared Memory:** Keeps an eye on everything and restarts processes if needed.

2. Data Servers:

Prioritize data access & processing on client machines. Often used in object oriented databases.

Key Points:

- Clients have more processing power and handle complex queries locally.
- Data servers can ship data (pages or entire objects) to clients for processing.
- Data caching on clients improves performance for frequently accessed data.

Key Differences:

- **Focus:** Transaction servers handle user requests and data consistency, while data servers manage physical storage and access.
- **Location:** Transaction servers are often close to clients, while data servers can be separate for better scalability.

Parallel Systems

Parallel DBMS utilizes multiple processors and disks to improve database performance.

Benefits:

- **Increased Speed:** Queries and operations run concurrently, leading to faster results.
- **Improved Scalability:** allows the system to handle larger datasets & more users efficiently.
- **Enhanced Availability:** If one node fails, others can continue processing.

Architectural Designs:

- **Shared Memory Architecture:** Processors share global memory space for efficient access.

- **Shared Disk Architecture:** Processors have private memories but access shared storage devices for data.
- **Shared-Nothing Architecture:** Each node has its own CPU, memory, and storage.

Speed up parallel systems

- Focuses on **reducing execution time** for a fixed workload.
- Achieved by adding more processing resources (CPUs, machines) to a system.
- Ideally, speedup is linear with the number of resources added.

Scale up parallel systems

- Focuses on handling an **increasing workload** while maintaining performance.
- Achieved by adding resources proportionally to the workload growth.
- Ensures the system can accommodate larger datasets and more users without performance degradation.

Interconnection Networks

Defines how processors or storage devices communicate in a Database Management System with parallel processing capabilities

Bus: Simple shared channel for all devices. Only one device transmits at a time.

Mesh: Processors arranged in a grid, connected to nearest neighbors. Offers multiple paths for data transfer, improving scalability.

Hypercube: Highly scalable network based on n-dimensional cubes. Processors connected only to nodes differing in a single dimension.

Parallel Database Architectures

Parallel databases distribute tasks across multiple processors to improve performance.

1. Shared Memory Architecture: Tightly coupled system with all processors sharing a single memory space. Data and code reside in the shared memory, accessible by all processors.

2. Shared Disk Architecture: Multiple processors connected to a shared storage device (disk). Data resides on the shared disk, while processors have local memory.

3. Shared-Nothing Architecture: Independent nodes (CPU, memory, storage) handle data subsets. Scales well, but complex for multi-location queries.

4. Hierarchical Architecture: Mix of shared-memory and shared-nothing. Upper level acts as a coordinator, distributing tasks to lower-level nodes.

UNIT - 3 | Distributed Databases

A database system that stores data across multiple computers or sites on a network.

Benefits:

- **Scalability:** Easily add more servers to handle growing data volumes or user access.
- **Availability:** If one site fails, others can still function (higher uptime).

- **Performance:** Distribute workload across multiple servers for faster queries.
- **Geographical Distribution:** Data can be located closer to users in different regions.
- **Improved fault tolerance:** Replication mechanisms ensure data consistency even in failure.

Components:

- **Sites:** Individual computers or servers storing database fragments.
- **Nodes:** Processing units within a site that manage data.
- **Distributed Database Management System (DDBMS):** Software that manages the entire distributed database and provides a unified view to users.

Distribution Schemes:

- **Fragmentation:** Dividing the database logically into smaller units for storage across sites.

Challenges:

- **Data Consistency:** Maintaining consistency of data across all sites after updates.
- **Complexity:** Managing data across multiple locations requires more complex software.
- **Network Reliability:** Reliant on a reliable network for communication between sites.

Distributed Data Storage

Distributed data storage is a method of storing information across multiple physical servers, often spread across different locations. It breaks down data into chunks and distributes them for redundancy and improved performance.

Benefits: Scalability, Reliability, Performance, Availability (Similar to Distributed Databases).

Types of Distributed Storage:

- **Distributed File system (DFS):** Makes scattered data appear as a single drive (e.g., HDFS).
- **Distributed Block storage:** Splits data for fast transfers (e.g., SAN).
- **Distributed Object storage:** Stores data in self-contained units (e.g., Amazon S3).

Applications: Cloud storage services, Backup and disaster recovery etc.

Distributed Transactions

A set of database operations executed across multiple interconnected databases (nodes) as a single logical unit. Ensures ACID properties even across distributed data.

Process (often uses Two-Phase Commit):

1. **Global Coordinator:** Initiates the transaction and coordinates participating nodes.
2. **Preparation Phase:** Each node involved prepares to commit changes.
3. **Commit Phase:** Coordinator instructs all nodes to commit.

If a node fails during the process, the transaction is aborted/rolled back.

Challenges:

- **Network failures:** Can disrupt communication and require complex rollback procedures.
- **Data consistency:** Ensuring all nodes reflect the same changes after a transaction.

Commit Protocols

Commit protocols ensure data integrity and consistency in DBMS by coordinating transactions across multiple sites.They guarantee that a transaction either fully succeeds (commits) on all participating sites, or completely fails (aborts) on all sites.

1. One-phase (local): Simple commit for single-site databases.

2. Two-Phase Commit (2PC): Common choice, involves coordinator asking participants if ready, then committing or aborting based on replies.

- **Phase 1:** Preparing to Commit
- **Phase 2:** Committing or Aborting

3. Three-phase (3PC): Adds extra phase for participants to confirm readiness before final commit/abort. (More complex)

- **Phase 1:** Pre-commit (Similar to 2PC)
- **Phase 2:** Waiting for Agreement
- **Phase 3:** Commit or Abort

Concurrency Control in Distributed Databases

- It **Ensures data consistency** when multiple transactions access the same data concurrently across distributed databases.
- Maintains **ACID properties** even in a distributed environment.
- Achieves **serializability**, preventing conflicts.

Single Lock-Manager Approach:

- **Centralized coordinator** manages all locks for the distributed database.
- Transactions requesting a lock send a message to the lock manager.

Distributed Lock Manager

- **Locks are managed locally** at each database site.
- Transactions coordinate locking across sites using protocols.
- **Protocols:** Two-phase commit (2PC) & Timestamp ordering.

Parallel Databases

A database system that leverages multiple processors (CPUs) and disks to improve performance through parallel processing of tasks.

Benefits: Increased Throughput, Faster Queries, Improved Scalability, High Availability.

Architectures:

- **Shared-Memory Architectures:** CPUs share memory, good for small setups.
- **Shared-Disk Architectures:** CPUs have private memory, share disk storage (more complex).
- **Shared-Nothing Architectures:** Each node has its own CPU, memory, disk.

I/O Parallelism

Speeds up data retrieval from disk by dividing relations (tables) across multiple disks.

Process:

- Data is partitioned based on a chosen attribute.
- Each partition is stored on a separate disk.
- Queries can then access data from multiple disks simultaneously.
- Results from each partition are combined after processing.

Implementation:

- Requires a DBMS that supports parallel processing.
- Data partitioning strategy needs to be defined (e.g., by hash function, round-robin).

Inter-Query Parallelism

The ability of a DBMS to execute multiple queries from various applications simultaneously.

- **Focuses on throughput:** Aims to handle a higher volume of transactions.
- **Independent execution:** Each query runs separately but concurrently with others.

Intra-Query Parallelism

A technique to speed up complex queries by breaking them down into smaller, independent tasks that can be executed concurrently on multiple processors.

Implementation:

- **Decomposing the Query:** The DBMS identifies independent operations within the query.
- **Parallelization:** These operations are assigned to different processors for simultaneous execution.
- **Combining Results:** The DBMS gathers and combines the partial results from each processor to generate the final query output.

Intraoperation Parallelism

Intraoperation parallelism focuses on parallelizing individual operations within a single SQL query. This approach significantly improves query performance for large datasets.

1. Parallel Sort: Parallel sort divides the data into smaller chunks and sorts them concurrently on multiple processors.

2. Parallel Join: Parallel join partitions the tables and performs the join operation on each partition concurrently.

Benefits: Significant performance improvement for complex queries involving large datasets.

Interoperation Parallelism

Interoperation parallelism deals with executing different operations within a single query expression in parallel.

1. Pipelined parallelism: Overlaps the execution of subsequent operations in a query.

2. Independent parallelism: Run independent tasks within a query simultaneously, like filtering separate tables.

Benefits of Interoperation Parallelism:

- Significantly reduces overall query execution time by overlapping operations.
- Makes efficient use of multiple processors in a system.

UNIT - 4 | PL/SQL

Procedural Language extensions to SQL. Combines the power of SQL with procedural programming features (control flow, loops, etc.)

Key Features:

- **Procedural constructs:** control flow statements (if-then-else, loops) for complex logic
- **Cursors:** for iterating through result sets of SQL statements
- **Error handling:** exceptions to manage errors during program execution
- **Integration with SQL:** execute SQL statements directly within PL/SQL code

Advantages of PL/SQL

- **Combines SQL and Procedural Code:** Logic and manipulate data within the same program.
- **Improved Performance:** Compiled code reduces network traffic and improves speed.
- **Increased Productivity:** Enables modular and reusable code.
- **Portable and Scalable:** Works across Oracle environments and handles complex tasks.
- **Object-Oriented Features:** Promotes better code organization.
- **Error Handling and Security:** Manages errors and restricts unauthorized access.
- **Web Application Development:** Can be used to create dynamic web content.

PL/SQL Blocks

- Encapsulate a set of PL/SQL statements to perform a specific task.

Structure:

- **DECLARE:** Define variables, cursors, exceptions, etc. (Optional)
- **BEGIN:** Marks the start of executable statements.
- **Executable Statements:** Code to perform operations.
- **END:** Marks the end of the block.

Character Set

- Set of characters a database can understand and store. Determines valid characters in identifiers, literals, and comments.

Literals

- Fixed values directly included in your code.
- **Types:** Numeric, Character, String literals, Boolean.

PL/SQL Data Types

1. Scalar Data Types: Represent single values.

- **NUMBER:** Stores numeric values with precision and scale.
- **BOOLEAN:** Represents true or false
- **CHAR/VARCHAR2:** Stores fixed or variable length character strings/
- **DATE:** Stores date and time information.

Subtypes: Derived from base types (like NUMBER) with additional constraints.

- `POSITIVE`: Restricts a number to positive values (e.g., `age POSITIVE`)
- `NATURAL`: Restricts a number to non-negative values (e.g., `balance NATURAL`)

2. Collections: Group similar data items together.

- **VARRAY:** Ordered collection with a fixed size at declaration.
- **TABLE:** Associative collection with key-value pairs.

3. Large Objects (LOBs): Store large data like text, images, or audio.

- **BLOB:** Binary Large Object.
- **CLOB:** Character Large Object.

4. Record Types: Group related variables of different data types into a single unit.

5. Reference Types: Store references to existing data structures. (e.g., `emp_cursor REF CURSOR;` (reference to a cursor))

Variables

- Variables act as temporary storage locations for data during program execution.
- They allow you to manipulate data within your code.

Declaring Variables: Variables need to be declared using the `DECLARE` keyword. Declaration specifies the variable name, data type, and optionally an initial value.

Constants

Constants are fixed values that cannot be changed during program execution.

Declaring Constants: using the `CONSTANT` keyword.

CONSTANT constant_name data_type := value [NOT NULL];

Attributes in PL/SQL

1. Cursor Attributes: Provide information on a cursor's status after execution.

- **%ISOPEN:** Checks if the cursor is open.
- **%FOUND:** Indicates if a FETCH retrieved a row (TRUE) or not (FALSE).
- **%NOTFOUND:** Opposite of %FOUND (TRUE if no row fetched, FALSE otherwise).
- **%ROWCOUNT:** Returns the number of rows fetched using the cursor.

2. %TYPE Attribute: Helps in declaring variables based on existing data types. Ensures type compatibility between the PL/SQL variable and the referenced source.

Control Structures

1. Conditional Control: Determines which block of code to execute based on a boolean condition

- **IF-THEN-ELSE Statement:** Executes one of two code blocks based on a condition.
- **CASE Statement:** Evaluates an expression against multiple conditions and executes corresponding code blocks.

2. Iterative Control: Executes a block of code repeatedly until a specific condition is met.

- **LOOP Statements:** Repeatedly execute a block of code until a condition is satisfied.
- **FOR Loop:** Iterates a specific number of times based on a counter variable.
- **WHILE Loop:** Continues execution as long as a condition remains TRUE.

3. Sequential Control: Executes statements in the order they appear in your code

Cursors in PL/SQL

- A cursor acts as a pointer to a result set retrieved from a SELECT statement.
- It allows you to process data one row at a time, unlike fetching the entire set upfront.

Types of Cursors:

1. Implicit Cursors: Created automatically by PL/SQL for any SELECT statement.

2. Explicit Cursors: Defined by the programmer for granular control over data retrieval.

Steps to Use Explicit Cursors: Declare the Cursor > Open the Cursor > Fetch Data > Close Cursor.

Benefits of Explicit Cursors: Enhanced control over data processing. Memory efficiency when dealing with large datasets.

Exception Handling

Exceptions: Errors that disrupt normal program flow during execution.

Two Types:

- **System-defined Exceptions:** Predefined errors by PL/SQL.
- **User-defined Exceptions:** Programmer-created exceptions for specific error conditions.

Key Concepts:

- **RAISE Statement:** Explicitly raises an exception to signal an error.
- **Exception Block:** Captures and handles raised exceptions.

Triggers

Triggers are PL/SQL blocks stored in the database and can be invoked repeatedly. They are automatically executed upon specific events.

Event-driven execution: Triggers fire in response to various database events, including:

- **DML (Data Manipulation Language) events:** INSERT, UPDATE, DELETE (DML)
- **DDL (Data Definition Language) events:** CREATE, ALTER, DROP (DDL)
- **Database events:** Errors, logins/logouts, startup/shutdown

Trigger Timing:

- **BEFORE:** Runs code to validate data or set values **before** changes occur.
- **AFTER:** Executes code to audit changes or react to them **after** they happen.
- **INSTEAD OF:** Less common, replaces the standard DML operation with custom logic.

Trigger Benefits:

- **Ensure data quality:** Validate data and prevent invalid entries.
- **Automate tasks:** Reduce manual work by automating.
- **Enhance security:** Restrict data access and implement additional security measures.
- **Maintain data consistency:** Keep data synchronized across multiple tables.

Procedures

Procedures are reusable blocks of PL/SQL code that perform specific tasks. They promote modular design by encapsulating functionality.

Structure: A procedure consists of a header (specification) and a body:

- **Header:** Defines the procedure name, optional parameters (IN for input, OUT for output), and return type (if applicable).
- **Body:** Contains the executable code (declarations, statements, exception handling).

Benefits:

- **Reusability:** Procedures can be called from other PL/SQL code, reducing redundancy.
- **Modularity:** Break down complex tasks into smaller, manageable units.
- **Maintainability:** Easier to modify and debug code within procedures.
- **Performance:** Can improve performance by reducing network traffic.

Packages

- A way to organize and group related PL/SQL code elements. Think of them as reusable modules containing procedures, functions, variables, cursors, and more.

Benefits:

- **Modularity:** Break down large codebases into smaller, manageable units.
- **Reusability:** Share common code across different parts of your application.
- **Maintainability:** Easier to understand, modify, and debug code.
- **Encapsulation:** Control access to code elements (public vs. private).

Package Specification:

- Declares the public interface (available elements).
- Includes declarations for procedures, functions, variables, cursors, etc.

Package Body (Optional):

- Contains the implementation (code) for public elements.
- Also includes private declarations for helper functions or variables.

Using Packages:

- Reference public elements using the `package_name.element_name` syntax.