**MODULE-5**

**Recovery and Security**

**Database recovery concepts**

Database recovery concepts are fundamental to ensuring data integrity and consistency in database management systems (DBMS). Here's an overview:

1. **Transaction**: A transaction is a logical unit of work that is performed on a database. It's a sequence of operations that either succeeds completely or fails completely.
2. **ACID Properties**: Transactions adhere to ACID properties:
   * Atomicity: Transactions are atomic, meaning they are all or nothing. Either all operations within a transaction are completed, or none are.
   * Consistency: Transactions bring the database from one consistent state to another consistent state. Consistency constraints are maintained.
   * Isolation: Transactions are isolated from each other until they are committed. This ensures that the intermediate state of one transaction is invisible to others.
   * Durability: Once a transaction is committed, its changes are permanent and survive system failures.
3. **Transaction Log**: The transaction log is a record of all changes made to the database. It ensures durability and aids in recovery. It typically contains a record of each transaction's start and end, as well as the changes made by each transaction.
4. **Checkpoint**: Periodically, the DBMS writes all modified data pages from memory to disk. This process is known as a checkpoint. It helps reduce recovery time by providing a known starting point from which to begin the recovery process in case of a failure.
5. **Undo and Redo Logging**: Two common techniques used in database recovery:
   * Undo Logging: Before a transaction modifies data, a log record of the original data values is written to the transaction log. This allows for rollback of changes if the transaction fails.
   * Redo Logging: After a transaction modifies data, a log record of the modified data values is written to the transaction log. This allows for replay of changes during recovery to ensure durability.
6. **Recovery Manager**: The recovery manager is responsible for ensuring that the database can recover to a consistent state after a system failure. It uses the transaction log to undo or redo transactions as necessary.
7. **Types of Failures**:
   * System Failure: Failure of hardware, software, or both.
   * Transaction Failure: A transaction may fail due to errors or violations of integrity constraints.
   * Media Failure: Failure of storage media where the database resides.
8. **Recovery Techniques**:
   * Restart Recovery: The simplest recovery technique where the system restarts and all active transactions are rolled back.
   * Undo/Redo Recovery: Involves analyzing the transaction log to undo incomplete transactions and redo committed transactions.
   * Restore from Backup: Involves restoring the database from a previous backup and applying transaction logs to bring it up to date.

**Types of database failures**

Database failures can occur due to various reasons, leading to disruptions in database operations and potential data loss. Here are some common types of database failures:

1. **System Failures**:
   * Hardware Failures: Failures of physical components such as disks, memory modules, or CPUs can lead to database downtime or data corruption.
   * Software Failures: Bugs, errors, or crashes in the database management system (DBMS) software can cause unexpected behavior or system crashes.
   * Operating System Failures: Issues with the underlying operating system, such as kernel panics or resource exhaustion, can impact database availability and performance.
   * Network Failures: Connectivity issues or network outages can disrupt communication between database servers, clients, or storage systems.
2. **Transaction Failures**:
   * Deadlocks: Transactions may become deadlocked when they are unable to proceed due to conflicting resource locks held by other transactions.
   * Constraint Violations: Violations of integrity constraints or business rules within transactions can lead to transaction failures.
   * Aborted Transactions: Transactions may be aborted due to user cancellation, application errors, or system interruptions.
3. **Media Failures**:
   * Disk Failures: Physical damage, data corruption, or disk failures can result in the loss of database files or data corruption.
   * Storage System Failures: Failures in storage systems such as SANs (Storage Area Networks) or NAS (Network Attached Storage) can impact database availability and data integrity.
4. **Natural Disasters**:
   * Floods, fires, earthquakes, or other natural disasters can physically damage data centers or infrastructure, leading to data loss or extended downtime.
5. **Human Errors**:
   * Accidental Data Deletion: Users or administrators may inadvertently delete or modify data, leading to data loss or corruption.
   * Misconfiguration: Incorrect configuration settings or parameter changes can negatively impact database performance, stability, or security.
6. **Security Breaches**:
   * Unauthorized Access: Intrusions, hacking attempts, or unauthorized access to the database can compromise data confidentiality, integrity, or availability.
   * Data Theft: Theft of sensitive data from the database, whether through cyberattacks or insider threats, can result in data breaches and compliance violations.

**Types of database recovery**

Database recovery refers to the process of restoring a database to a consistent and usable state after a failure or corruption. There are several types of database recovery techniques and strategies:

1. **Point-in-Time Recovery (PITR)**:
   * Point-in-Time Recovery allows you to restore a database to a specific moment in time, typically before the occurrence of a failure or corruption. It involves restoring a database backup and then applying transaction logs up to the desired point in time to replay the transactions that occurred after the backup was taken.
2. **Crash Recovery**:
   * Crash recovery is the automatic recovery process that occurs when a database management system (DBMS) restarts after an unexpected system failure or crash. During crash recovery, the DBMS uses transaction logs and other recovery mechanisms to ensure that transactions are either committed or rolled back to maintain data consistency.
3. **Rollback Recovery**:
   * Rollback recovery involves undoing the changes made by incomplete or aborted transactions. Transaction logs are used to identify and rollback any transactions that were in progress but not committed at the time of a failure. This ensures that the database returns to a consistent state before the failure occurred.
4. **Redo Recovery**:
   * Redo recovery involves reapplying the changes made by transactions that were committed but not yet permanently written to disk at the time of a failure. Transaction logs are used to replay these committed transactions to ensure that all changes are durably stored and that the database is consistent.
5. **Backup and Restore**:
   * Backup and restore is a fundamental database recovery technique that involves creating regular backups of the database and its transaction logs. In the event of a failure or data loss, you can restore the database from a recent backup and then apply transaction logs to recover any lost data or changes since the backup was taken.
6. **Database Replication**:
   * Database replication involves maintaining multiple copies of the database across different servers or locations. In the event of a failure, one of the replicated copies can be promoted to serve as the primary database, minimizing downtime and data loss.
7. **High Availability (HA) and Failover**:
   * High availability architectures use redundant hardware, clustering, or virtualization techniques to ensure continuous availability of the database even in the event of hardware failures or system crashes. Failover mechanisms automatically switch to a standby database or server in the event of a failure to minimize downtime.
8. **Disaster Recovery (DR)**:
   * Disaster recovery plans outline procedures for recovering databases and IT systems in the event of catastrophic failures, such as natural disasters, fires, or prolonged outages. DR strategies often include offsite backups, data replication, and failover mechanisms to ensure business continuity.

**Recovery Techniques**

Database recovery techniques ensure data consistency and integrity, allowing the system to recover to a consistent state after failures. Here are some common recovery techniques:

1. **Transaction Logs**:
   * Transaction logs record all changes made to the database, including committed and uncommitted transactions. During recovery, these logs are used to replay committed transactions and roll back uncommitted transactions to restore the database to a consistent state.
2. **Undo Logging**:
   * Undo logging involves recording the old values of modified data before changes are made by a transaction. In the event of a failure, these old values can be used to roll back the changes made by incomplete transactions, ensuring data consistency.
3. **Redo Logging**:
   * Redo logging involves recording the new values of modified data after changes are made by a transaction. In the event of a failure, these redo log records can be used to reapply committed transactions that were not yet durably stored to disk, ensuring durability and consistency.
4. **Checkpointing**:
   * Checkpointing involves periodically writing database changes from memory to disk, along with a record of the latest checkpoint in the transaction log. This ensures that recovery can start from a known consistent state after a failure, reducing the amount of redo and undo operations needed.
5. **Immediate Update**:
   * Immediate update techniques immediately update the database on disk as transactions are committed, ensuring that changes are durably stored and reducing the need for extensive recovery operations in the event of a failure.
6. **Deferred Update**:
   * Deferred update techniques delay the modification of the database on disk until after transactions are committed. Changes are first recorded in the transaction log, and then durably stored during a checkpoint or commit operation. This simplifies recovery by reducing the need for undo operations.
7. **Shadow Paging**:
   * Shadow paging involves maintaining a shadow copy of the database that is updated transactionally. During recovery, the system can revert to the last consistent shadow copy, discarding any incomplete transactions and ensuring data consistency.
8. **Database Replication**:
   * Database replication involves maintaining multiple copies of the database on different servers or locations. In the event of a failure, one of the replicated copies can be promoted to serve as the primary database, ensuring continuous availability and minimizing downtime.
9. **Backup and Restore**:
   * Backup and restore techniques involve creating regular backups of the database and its transaction logs. In the event of a failure, the database can be restored from a recent backup, and transaction logs can be applied to recover any lost data or changes since the backup was taken.

**Buffer Management**

Buffer management is a crucial aspect of database systems, responsible for efficiently handling data storage and retrieval. Here's an overview:

1. **Buffer Pool**:
   * A buffer pool is a portion of memory allocated by the database management system (DBMS) to cache frequently accessed data pages from the disk. These data pages are temporarily stored in memory, improving performance by reducing the need for frequent disk I/O operations.
2. **Page Replacement Algorithms**:
   * Buffer management involves selecting which pages to keep in the buffer pool when new pages need to be loaded. Various page replacement algorithms are used for this purpose, including:
     + Least Recently Used (LRU): Removes the least recently used page from the buffer pool when space is needed.
     + First-In-First-Out (FIFO): Removes the oldest page from the buffer pool when space is needed.
     + Clock (or Second Chance): Similar to FIFO but uses a clock hand to track pages and give them a "second chance" before eviction.
     + Most Recently Used (MRU): Removes the most recently used page from the buffer pool when space is needed.
     + Random: Selects a page randomly from the buffer pool for eviction.
3. **Buffer Management Operations**:
   * Page Fetch: When a query requests data that is not in the buffer pool, the corresponding data page is fetched from the disk into the buffer pool.
   * Page Replacement: When the buffer pool is full and a new page needs to be loaded, a page replacement algorithm selects a victim page to be evicted from the buffer pool to make space for the new page.
   * Page Flush: Modified data pages in the buffer pool are periodically flushed back to disk to ensure durability and consistency. This process involves writing dirty pages (those that have been modified) back to their corresponding disk locations.
4. **Dirty Page Management**:
   * Dirty pages are those that have been modified in memory but not yet written back to disk. Buffer management includes mechanisms for tracking and managing dirty pages, ensuring that changes are durably stored on disk to maintain data consistency.
   * Write-Ahead Logging (WAL): A common technique used in buffer management is to log changes to dirty pages in a write-ahead log before writing the changes back to disk. This ensures that modifications are recoverable in the event of a crash or failure.
5. **Buffer Pool Sizing**:
   * Proper sizing of the buffer pool is essential for optimizing database performance. A larger buffer pool can accommodate more data pages in memory, reducing the frequency of disk I/O operations and improving query response times. However, an excessively large buffer pool can lead to memory contention and performance degradation.

**Database Security**

Database security involves protecting sensitive data stored in a database from unauthorized access, misuse, or modification. Here are key components and practices:

1. **Authentication**:
   * Authentication ensures that only authorized users can access the database. This typically involves verifying the identity of users through credentials such as usernames, passwords, or other authentication methods like multi-factor authentication (MFA).
2. **Authorization**:
   * Authorization controls what actions authenticated users are allowed to perform within the database. This includes defining user roles, privileges, and permissions to access specific data, execute certain queries, or perform administrative tasks.
3. **Encryption**:
   * Encryption protects data by converting it into a scrambled format that can only be decrypted with the appropriate encryption key. Database encryption can be applied at various levels, including encrypting data at rest (stored on disk) and data in transit (moving between the client and server).
4. **Access Control**:
   * Access control mechanisms restrict access to sensitive data based on user roles, privileges, and permissions. This includes implementing fine-grained access controls to ensure that users can only access the data they need for their specific tasks, while preventing unauthorized access to sensitive information.
5. **Audit Logging**:
   * Audit logging tracks and records database activity, including login attempts, queries executed, data modifications, and administrative actions. Audit logs provide a trail of events for monitoring and investigating security incidents, compliance auditing, and forensic analysis.
6. **Database Activity Monitoring (DAM)**:
   * DAM solutions monitor database activity in real-time to detect and alert on suspicious or unauthorized activities, such as unauthorized access attempts, data breaches, or unusual query patterns. DAM helps organizations proactively identify and respond to security threats.
7. **Database Patching and Updates**:
   * Regularly applying patches and updates to the database management system (DBMS) and associated software components is essential for addressing security vulnerabilities and ensuring that the database remains protected against known threats and exploits.
8. **Data Masking and Redaction**:
   * Data masking and redaction techniques hide sensitive data by replacing it with anonymized or pseudonymized values in non-production environments or when sharing data with third parties. This helps protect confidential information while still allowing authorized users to work with realistic data sets.
9. **Database Backup and Recovery**:
   * Regularly backing up the database and implementing robust disaster recovery procedures are essential for data protection and business continuity. In the event of a security breach, data corruption, or other disasters, database backups enable organizations to restore data to a previous known good state.
10. **Security Training and Awareness**:
    * Educating database administrators, developers, and end-users about security best practices, policies, and procedures is crucial for creating a security-aware culture. Training programs can help prevent common security mistakes, such as weak passwords, improper data handling, or social engineering attacks.

**Goals of database security**

The goals of database security are to ensure the confidentiality, integrity, and availability of data stored in a database. These goals are often referred to as the CIA triad:

1. **Confidentiality**:
   * Confidentiality ensures that sensitive data is protected from unauthorized access, disclosure, or theft. The goal is to prevent unauthorized users or attackers from viewing or accessing sensitive information stored in the database. This is achieved through measures such as access controls, encryption, and data masking.
2. **Integrity**:
   * Integrity ensures that data remains accurate, consistent, and trustworthy throughout its lifecycle. The goal is to prevent unauthorized modification, deletion, or tampering of data stored in the database. This is achieved through measures such as data validation, access controls, checksums, and digital signatures.
3. **Availability**:
   * Availability ensures that data is accessible and usable when needed by authorized users. The goal is to prevent disruptions, downtime, or denial-of-service attacks that could impact the availability of the database or its services. This is achieved through measures such as redundancy, fault tolerance, backup and recovery procedures, and disaster recovery planning.

In addition to the CIA triad, database security also aims to achieve other important objectives, including:

1. **Authentication**:
   * Authentication verifies the identity of users and ensures that only authorized individuals are granted access to the database. This is typically achieved through mechanisms such as usernames, passwords, biometrics, or multi-factor authentication.
2. **Authorization**:
   * Authorization controls what actions authenticated users are allowed to perform within the database. This includes defining user roles, privileges, and permissions to access specific data, execute certain queries, or perform administrative tasks.
3. **Auditing and Compliance**:
   * Auditing tracks and records database activity for monitoring, compliance, and forensic analysis purposes. This includes logging login attempts, queries executed, data modifications, and administrative actions to detect and investigate security incidents or compliance violations.
4. **Data Privacy**:
   * Data privacy ensures that personal or sensitive information is handled in accordance with legal and regulatory requirements, such as GDPR, HIPAA, or PCI DSS. This includes implementing measures to protect personally identifiable information (PII), healthcare data, financial data, or other sensitive data types.
5. **Security Awareness**:
   * Security awareness programs educate database administrators, developers, and end-users about security best practices, policies, and procedures to prevent common security mistakes and mitigate security risks.

**NoSQL Database**

A NoSQL database is a type of database that provides a mechanism for storage and retrieval of data that is modeled in ways other than the tabular relations used in relational databases.

NoSQL databases are categorized based on their data model. Here are the main types of NoSQL databases:

1. **Document-Oriented Databases**:
   * Document-oriented databases store data in flexible, semi-structured documents, typically using formats like JSON or BSON (Binary JSON). Each document can contain nested fields and arrays, allowing for complex and hierarchical data structures. Examples include MongoDB, Couchbase, CouchDB, and Amazon DocumentDB.
2. **Key-Value Stores**:
   * Key-value stores are the simplest form of NoSQL databases, where data is stored as a collection of key-value pairs. Each key is unique and maps to a corresponding value, similar to a dictionary or hash table. Key-value stores are highly scalable and efficient for simple read and write operations. Examples include Redis, Amazon DynamoDB, Riak, and Berkeley DB.
3. **Column-Family Stores**:
   * Column-family stores organize data into columns grouped by column families or column families. Each row can have a different set of columns, and columns are stored together, allowing for efficient retrieval of related data. Column-family stores are well-suited for applications requiring high write throughput and analytical queries. Examples include Apache Cassandra, HBase, and ScyllaDB.
4. **Graph Databases**:
   * Graph databases are designed for representing and querying graph-like structures, consisting of nodes, edges, and properties. They are optimized for traversing relationships between nodes and performing complex graph queries. Graph databases excel in applications such as social networks, recommendation systems, and network analysis. Examples include Neo4j, Amazon Neptune, JanusGraph, and ArangoDB.
5. **Multi-Model Databases**:
   * Multi-model databases support multiple data models within a single database engine, allowing developers to choose the most appropriate model for each use case. They provide flexibility and versatility by combining the strengths of different data models. Examples include ArangoDB, Couchbase, OrientDB, and MarkLogic.

**Applications of NoSQL**

 **Web Applications**:

* NoSQL databases are commonly used in web applications to handle large volumes of unstructured or semi-structured data, such as user profiles, session data, content management, and user-generated content. They provide scalability, flexibility, and high performance, making them suitable for handling dynamic and rapidly changing data.

 **Big Data and Real-time Analytics**:

* NoSQL databases are well-suited for storing and analyzing large volumes of data generated by big data applications, IoT devices, sensors, and social media platforms. They can handle high velocity, variety, and volume of data, enabling real-time analytics, event processing, and data-driven decision-making.

 **Content Management Systems (CMS)**:

* NoSQL databases are used in content management systems to store and manage structured and unstructured content, such as articles, blogs, images, videos, and user comments. They provide scalability and flexibility for handling diverse content types and high traffic loads.

 **E-commerce and Retail**:

* NoSQL databases are used in e-commerce and retail applications for product catalogs, inventory management, order processing, and customer data management. They can handle large product catalogs, fluctuating inventory levels, and complex customer interactions, providing a seamless shopping experience.

 **Gaming**:

* NoSQL databases are used in online gaming applications for player profiles, game state management, leaderboards, and social interactions. They provide scalability and low-latency access to data, supporting massive multiplayer games and real-time gaming experiences.

 **IoT (Internet of Things)**:

* NoSQL databases are used in IoT applications for collecting, storing, and analyzing data from connected devices, sensors, and machines. They can handle high volumes of time-series data, event streams, and sensor data, enabling real-time monitoring, predictive maintenance, and anomaly detection.

 **Social Networking and Collaboration**:

* NoSQL databases are used in social networking platforms and collaboration tools for storing user profiles, social graphs, messages, comments, and media files. They provide scalability and flexibility for handling social interactions, user-generated content, and personalized recommendations.

 **Mobile Applications**:

* NoSQL databases are used in mobile applications for offline data synchronization, caching, and storing user preferences, settings, and application data. They provide fast read and write access, low-latency data retrieval, and support for mobile device constraints.