# Sir Sajjad – Final Exam – 5th Semester - DBMS

## Transaction in a Database:-

In a database, a transaction is a unit of work that is performed on the data in the database. Transactions are used to ensure that the data in the database remains in a consistent state, even if something goes wrong during the transaction.

Transactions typically involve multiple operations on the data, and they are either all completed or none of them are completed. This is known as the atomicity property of transactions. For example, if a transaction involves transferring money from one bank account to another, the transaction would include both a debit operation on the first account and a credit operation on the second account. If either of these operations fails, the entire transaction is rolled back and the database is returned to its previous state.

Transactions also have the properties of isolation and durability. Isolation means that the changes made by one transaction are not visible to other transactions until the first transaction is committed. This helps to prevent race conditions and other concurrency issues. Durability means that once a transaction is committed, the changes it made to the database are permanent and will not be lost, even if the system shuts down or experiences an error.

Transactions are an important feature of databases, as they help to ensure the integrity and consistency of the data stored in the database.

Top of Form

Bottom of Form

Indexing:-

Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique which is used to quickly locate and access the data in a database.

Query stored Procedure:

A stored procedure is a pre-compiled set of Structured Query Language (SQL) statements that are stored in a database and can be called by name. Stored procedures are often used to encapsulate a set of SQL statements that are used frequently, or to perform a specific task that requires multiple SQL statements to be executed.

To call a stored procedure, you can use the **EXEC** or **EXECUTE** statement followed by the name of the stored procedure and any required parameters. For example:

Stored procedures can be used to improve the performance of a database by reducing the amount of SQL code that needs to be parsed and compiled. They can also help to enforce data integrity and improve security by encapsulating complex business logic in the database, rather than in the application code.

Sql Injection:

SQL injection is a type of security vulnerability that occurs when an attacker is able to execute malicious SQL statements on a database. This can be done by inserting malicious code into a SQL statement that is sent to the database, in an attempt to trick the database into executing unintended commands or returning sensitive data.

SQL injection attacks are possible because of the way that some applications construct SQL statements. If an application dynamically creates a SQL statement based on user input, and does not properly validate or escape that input, it is possible for an attacker to insert malicious code into the statement. For example, consider the following SQL statement:

Active attack

An active attack in SQL is an attack in which the attacker actively modifies or manipulates the data or structure of a database. This can be done in a variety of ways, including:

* Inserting, updating, or deleting data in the database
* Adding or dropping tables or other database objects
* Modifying the schema or structure of the database
* Executing arbitrary commands on the database

Active attacks can have serious consequences, such as loss or corruption of data, unauthorized access to sensitive information, or disruption of database operations.

Some examples of active attacks in SQL include:

* SQL injection: As described above, SQL injection is a type of attack in which an attacker inserts malicious code into a SQL statement in an attempt to trick the database into executing unintended commands.
* Data manipulation: An attacker could use SQL commands to modify the data in a database, such as changing the values of certain fields or deleting rows.
* Table manipulation: An attacker could use SQL commands to create, alter, or drop tables in a database, potentially altering the structure of the database or disrupting database operations.

To prevent active attacks in SQL, it is important to properly validate and escape user input, and to use prepared statements and parameterized queries whenever possible. It is also important to implement proper security measures, such as strong authentication and authorization controls, to prevent unauthorized access to the database.

Passive Attack

In the context of database security, a passive attack is a type of security threat that involves the unauthorized access or monitoring of sensitive information, without actively altering or modifying the data. Passive attacks are often difficult to detect, as they do not involve any changes to the data or the database itself.

Examples of passive attacks on databases include:

1. Eavesdropping: An attacker listens in on communications between the database and other systems, in order to gather sensitive information.
2. Traffic analysis: An attacker analyzes the patterns of communication between the database and other systems, in order to infer sensitive information.
3. Data mining: An attacker accesses and analyzes data stored in the database, in order to gather sensitive information.

To prevent passive attacks on databases, it is important to implement strong security measures such as encryption, authentication, and access controls. It is also important to regularly monitor the database for suspicious activity, and to implement appropriate measures to detect and respond to any security threats.

Top of Form

Bottom of Form

Active attack vs Passive attack:

In the context of database security, an active attack is a type of security threat that involves the unauthorized modification or alteration of data in the database. Active attacks are typically more noticeable and easier to detect than passive attacks, as they involve changes to the data or the database itself.

Examples of active attacks on databases include:

1. SQL injection: An attacker inserts malicious code into a database query in order to gain unauthorized access or modify data.
2. Data tampering: An attacker modifies data in the database, either to disrupt the database or to cover their tracks.
3. Denial of service (DoS): An attacker floods the database with requests or traffic, in order to disrupt access or make the database unavailable.

In contrast, a passive attack is a type of security threat that involves the unauthorized access or monitoring of sensitive information, without actively altering or modifying the data. Passive attacks are often difficult to detect, as they do not involve any changes to the data or the database itself. Examples of passive attacks on databases include eavesdropping, traffic analysis, and data mining.

To prevent active and passive attacks on databases, it is important to implement strong security measures such as encryption, authentication, and access controls. It is also important to regularly monitor the database for suspicious activity, and to implement appropriate measures to detect and respond to any security threats.

Commit vs Roll back:

In a database management system (DBMS), a **COMMIT** operation is used to end a transaction and save the changes made during the transaction to the database. A **ROLLBACK** operation is used to undo the changes made during a transaction and restore the database to its state before the transaction began.

A transaction is a logical unit of work that contains one or more SQL statements. Transactions are used to ensure that the database remains in a consistent state, even if an error occurs during the execution of a SQL statement.

For example, consider a banking database that has a **customers** table and an **accounts** table. If a user wants to transfer money from one account to another, the DBMS would need to perform the following steps:

1. Check that the accounts exist and that the user has sufficient funds in the source account.
2. Deduct the amount of the transfer from the source account.
3. Add the amount of the transfer to the destination account.

If any of these steps fail, the transaction should be rolled back to prevent the database from becoming inconsistent. If all steps are successful, the transaction should be committed to save the changes to the database.

Db Sharding vs Distributed:

Database sharding and distributed databases are two approaches that can be used to scale a database to support large amounts of data and handle high levels of concurrency.

Database sharding is a technique that involves horizontally partitioning a database into smaller, independent units called shards. Each shard is stored on a separate server or group of servers, and is responsible for storing and managing a subset of the data. When a request is made to the database, the request is routed to the appropriate shard based on the key value of the data being accessed.

Distributed databases, on the other hand, involve storing data across multiple servers or machines, and using a software layer to manage the data and ensure that it is consistent and available to all users. Distributed databases can be scaled out by adding more servers or machines to the system, and can be designed to automatically replicate data and balance workloads across the servers.

Both database sharding and distributed databases can be used to scale a database to handle large amounts of data and high levels of concurrency, but they differ in how they partition and manage the data. Sharding is often used to scale read-heavy workloads, while distributed databases are better suited to support complex transactions and data consistency requirements.











