Self-Study

1. Class Diagram

What is a Class Diagram ?

A class diagram is a type of static structure diagram in the Unified Modeling Language (UML) that describes the structure of a system by showing its classes, their attributes, operations (methods), and the relationships among objects. It is a visualization of the system's object-oriented design.

What are Class Diagram Components ?

1. **Classes:** Represented by a rectangle defining name, attributes and methods of a class
2. **Relationships:** Arrows and lines connecting classes and represent the relationship between the classes:

    a. ***Association:*** A general relationship.
    b. ***Aggregation:*** A "has-a" relationship.
    c. **Composition:** A stronger "has-a" relationship where parts cannot exist without the whole.
    d. **Inheritance:** A "is-a" relationship.
    e. **Dependency:** A weaker relationship where one class depends on another.

What is a class diagram used for ?

It models the structure of a system in object-oriented design defining the classes, their attributes, methods, and relationships to serve as a foundation for coding by providing a clear design for developers and to communicate the system's structure to stakeholders and developers.

What is the difference between Class Diagram and ERD ?

It. The primary difference is that the class diagram primarily focuses on Object-oriented programming concepts (encapsulation, inheritance, polymorphism) and the application logic and how objects interact in the software. It's critical for developers building the system's functionality. While ERD focuses on Database schema design (normalization, keys, relationships) and the data layer, ensuring the database is structured efficiently to store and retrieve data.

2. Built-In Stored Procedure

What are Built-in stored procedures ?

Built-in stored procedures are pre-defined procedures that are provided by the system and included in database management systems (DBMS). These procedures are intended to carry out regular administrative, management, or utility functions and are written in either the native language of the DBMS or a compiled format. To make database operations simpler, they can be executed by users with appropriate privileges and are usually kept in the database itself.

What are Built-in stored procedures used for ?

A. **Database Administration**: Managing users, roles, permissions, or backups.
B. **Metadata Retrieval**: Querying information about database objects (tables, indexes, schemas).
C. **Object Modification:** Some built-in stored procedures facilitate the recreation or modification of database objects like tables, indexes, or views.
D. **Performance Tuning**: Analyzing query performance or optimizing database settings.
E. **Security Management**: Configuring authentication or auditing.
F. **Maintenance**: Rebuilding indexes, updating statistics, or managing database logs.

Examples of Built-in stored procedures in SQL Server

SQL Server provides several system stored procedures, prefixed with sp_ and stored in the master database or system catalogs.

A. **sp_help:** Retrieves information about database objects (e.g., tables, columns).
B. **sp_who:** Displays information about current users and sessions.
C. **sp_adduser:** Adds a user to the current database.
D. **sp_msforeachtable:** Executes a command on every table in the database.

3. SQL Injection

What is SQL Injection?

SQL Injection is a security vulnerability that allows an attacker to manipulate a web application's database by injecting malicious SQL code into input fields or parameters. This happens when user inputs are not

properly sanitized or validated, enabling attackers to execute unauthorized SQL queries. These queries can retrieve, modify, or delete data, bypass authentication, or even execute administrative operations on the database.

How Does SQL Injection Happen?

SQL Injection occurs when an application directly includes user input into SQL queries without proper validation or escaping. This could happen through one of these cases:

1. **Unfiltered User Input**: An application accepts input and directly
2. **Dynamic SQL Queries**: Developers build SQL queries by concatenating strings with user input, making it easier for attackers to inject malicious code.
3. **Insecure APIs or Libraries**: Using outdated or insecure database APIs that don't support parameterized queries can expose applications to injection risks.

How Can a Developer Secure a System from SQL Injection?

Preventing SQL Injection requires a multi-layered approach across different stages of software development. Below are best practices categorized by development levels:

1. Design Phase

A. **Use Prepared Statements and Parameterized Queries:** Design database interactions to use parameterized queries or prepared statements, which separate SQL code from user input. Most modern frameworks like Entity Framework support parameterized queries by default.

B. ***Adopt an ORM***: Use Object-Relational Mapping (ORM) tools to abstract database queries, reducing the risk of manual query construction.
C. **Define Strict Input Requirements**:
    - Specify allowed input formats (e.g., alphanumeric, specific length) during the design phase to guide validation implementation.

2. Development Phase

A. **Input *V*alidation:**
B. ***A*void Dynamic SQL**:
C. **Escape Special Characters**:
D. **Use Least Privilege**:
E. **Implement Secure *A*PIs**:

3. Testing Phase

A. **Security Testing:** Perform static and dynamic application security testing to identify SQL injection vulnerabilities.
B. **Fuzz Testing:** Test input fields with unexpected or malicious inputs to ensure the application handles them securely.
C. **Penetration Testing**: Engage ethical hackers to attempt SQL injection and identify weaknesses.