The FEDERAL POLYTECHNIC, MUBI

**DEPARTMENT OF COMPUTER SCIENCE**

AN ASSIGNMENT ON

**DATABASE DESIGN II**

COURSE CODE: COM322

SUBMITTED BY

**GROUP 2**

ST/CS/HND/22/002

ST/CS/HND/22/009

ST/CS/HND/22/016

ST/CS/HND/22/023

ST/CS/HND/22/030

ST/CS/HND/22/037

ST/CS/HND/22/044

ST/CS/HND/22/051

ST/CS/HND/22/058

ST/CS/HND/22/065

ST/CS/HND/22/072

ST/CS/HND/22/079

ST/CS/HND/22/086

ST/CS/HND/22/093

SUBMITTED TO

THE COURSE LECTURER

AUGUST, 2023

**QUESTION 1:**

Designing form reports and triggers in an object-oriented database involves considering the unique characteristics of object-oriented data models and the need for flexibility and extensibility in the database design. Here's a step-by-step guide on how to design form reports and triggers in an object-oriented database:

**1. Object-Oriented Data Model Design:**

Begin by designing the object-oriented data model that represents the structure and relationships between the data entities. The data model should include classes, attributes, and methods, representing real-world entities and their behaviors. Define the classes and their relationships, considering inheritance, composition, and association among objects.

**2. Define Forms and Reports:**

Identify the specific forms and reports that need to be generated based on the requirements of the application. Forms are used for data input and editing, while reports present information in a structured format. Determine the attributes and methods needed for each form and report, including the data elements to be displayed, filters, sorting options, and aggregate functions for reports.

**3. Implement Data Entry Forms:**

Design data entry forms that allow users to input data into the object-oriented database. These forms should correspond to the classes in the data model, enabling users to create and update objects directly within the database. Consider the user interface design and ensure that data validation and integrity checks are implemented to maintain data consistency.

**4. Create Object-Oriented Reports:**

Design reports that leverage the object-oriented data model to extract and present information from the database. Utilize object-oriented querying mechanisms, such as polymorphism and object traversal, to retrieve data based on the specified criteria. The reports should be dynamic and capable of adapting to changes in the data model.

**5. Implement Triggers:**

Triggers in an object-oriented database are event-driven actions that respond to changes in data. Design triggers to enforce business rules, maintain data integrity, and perform automated actions when specific events occur. Triggers can be defined at the class level, reacting to changes in objects, or at the attribute level, responding to modifications to specific attributes.

**6. Handle Object Evolution:**

One of the key advantages of an object-oriented database is its ability to handle object evolution. Design the database to accommodate changes in the object model over time, such as adding new attributes or methods to classes. Ensure that triggers and forms remain functional even when the data model undergoes modifications.

**7. Maintain Data Security:**

Implement access control mechanisms to ensure data security in the object-oriented database. Define user roles and permissions to restrict data access and modification based on user privileges. Triggers can also be utilized to enforce data security rules and prevent unauthorized actions on objects.

**8. Test and Optimize:**

Thoroughly test the designed forms, reports, and triggers to ensure they function as intended and meet the requirements of the application. Optimize the performance of the database by indexing attributes, improving query efficiency, and minimizing resource consumption.

**9. Provide User Training and Support:**

Finally, provide user training and support for interacting with the forms and reports in the object-oriented database. Familiarize users with the data entry forms and reporting functionalities to ensure efficient utilization of the database's capabilities.

By following these steps and considering the principles of object-oriented database design, you can create a robust and efficient system for handling form reports and triggers in an object-oriented database environment.

**QUESTION 2:**

**Class Diagram:**

A class diagram is a visual representation of the static structure of a system in object-oriented modeling. It shows the classes, their attributes, methods, and the relationships between classes. Class diagrams provide a clear overview of the classes in a system and how they interact with each other.

**Key Elements of a Class Diagram:**

Class: A class is represented as a rectangle with three compartments. The top compartment contains the class name, the middle compartment lists the attributes (data members) of the class, and the bottom compartment contains the methods (operations) that the class can perform.

**Attributes**: Attributes are the data members of a class, representing the properties or characteristics of objects belonging to the class. Each attribute is listed in the middle compartment of the class rectangle, along with its data type.

**Methods:** Methods are the operations or actions that a class can perform. They define the behavior of the class. Each method is listed in the bottom compartment of the class rectangle, along with its parameters and return type.

**Relationships:** Relationships between classes depict how classes are related to each other. Common types of relationships include association, aggregation, composition, inheritance, and dependency.

**Association:** Association represents a bi-directional relationship between two classes, indicating that instances of one class are associated with instances of the other class. It is typically represented by a line connecting the classes, with optional arrowheads to indicate the direction of the association.

**Aggregation:** Aggregation represents a "whole-part" relationship between classes, where one class (the whole) contains or is composed of instances of another class (the part). It is represented by a diamond shape on the side of the whole class.

**Composition:** Composition is a stronger form of aggregation, indicating a "strong" whole-part relationship where the parts cannot exist independently of the whole. It is represented by a filled diamond shape on the side of the whole class.

**Inheritance:** Inheritance represents an "is-a" relationship between classes, where one class (the subclass or derived class) inherits the properties and behaviors of another class (the superclass or base class). It is depicted by an arrow with an open triangle pointing to the superclass.

**Activity Diagram:**

An activity diagram is a visual representation of the dynamic behavior of a system or process, showing the flow of activities and actions from start to end. It is commonly used to model workflows, business processes, or the behavior of software systems.

**Key Elements of an Activity Diagram:**

Initial Node: Represents the starting point of the activity diagram. It is denoted by a solid black circle.

**Activity State:** Represents an activity or action that occurs in the system. It is represented by a rounded rectangle with the name of the activity.

**Decision Node:** Represents a decision point in the workflow, where the flow branches based on certain conditions. It is represented by a diamond shape with incoming and outgoing control flows.

**Merge Node:** Represents the merging of multiple flows back into a single flow after a decision point. It is depicted by a diamond shape with multiple incoming flows and a single outgoing flow.

**Fork Node:** Represents the simultaneous execution of multiple activities. It is represented by a horizontal bar with multiple outgoing control flows.

**Join Node:** Represents the synchronization of multiple flows into a single flow after concurrent activities. It is depicted by a horizontal bar with multiple incoming control flows.

**Final Node:** Represents the end point of the activity diagram. It is denoted by a solid black circle with a border.

Activity diagrams help in understanding the flow of activities, decision points, and concurrency in a process or system, providing a visual representation of the overall behavior and logic. They are especially useful for modeling complex business processes and software workflows.

**QUESTION 3:**

Unified Modeling Language (UML) is a standardized notation for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system, including database systems. UML can be used in database diagramming to model the structure, relationships, and behavior of a database, providing a clear and standardized representation of the database design. Here's how UML is used in database diagrams:

**1. Class Diagrams:** UML class diagrams are commonly used to model the static structure of a database. In the context of a database, classes represent database tables, and their attributes correspond to the table columns. Class diagrams provide an overview of the entities in the database and the relationships between them. Associations in class diagrams represent the relationships between tables, such as one-to-many, many-to-many, or one-to-one relationships.

**2. Object Diagrams:** Object diagrams in UML can be used to represent instances of classes (database records). Object diagrams show the data stored in the database tables and how the entities are related at runtime. These diagrams can be useful for illustrating specific examples or scenarios of the database's data.

**3. Activity Diagrams:** Activity diagrams can be employed to model the flow of database-related processes or transactions. They show the sequence of actions and the decision points in a process, often associated with database interactions, such as data retrieval, insertion, deletion, and updates. Activity diagrams provide a visual representation of the logic and steps involved in a database operation.

**4. Sequence Diagrams:** Sequence diagrams can be used to model the interactions and communication between objects in a database system. These diagrams show the sequence of messages exchanged between objects during a specific scenario, highlighting the flow of control and data within the database system.

**5. Use Case Diagrams:** Use case diagrams can help identify the various interactions between actors (users or external systems) and the database system. They represent the functional requirements of the database, showing how users interact with the system to perform specific tasks.

**6. State Machine Diagrams:** State machine diagrams can be used to model the state transitions and behaviors of database entities or records. These diagrams represent the lifecycle of a record, showing how it evolves through different states based on specific events or conditions.

**7. Package Diagrams:** Package diagrams can be used to organize and represent the various components and modules of a database system. They provide a high-level view of the database structure, grouping related classes and modules into packages, making it easier to manage the complexity of a large database system.

Conclusion: UML provides a versatile and standardized set of diagrams that can be used to model and document different aspects of a database system. By using UML in database diagramming, developers, database administrators, and stakeholders can gain a better understanding of the database design, structure, and behavior, facilitating communication and collaboration during the development and maintenance of database systems.

**REFERENCES**

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional.

Fowler, M. (2002). UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition). Addison-Wesley Professional.

Ambler, S. W. (2002). The Object Primer: Agile Model-Driven Development with UML 2.0. Cambridge University Press.

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). The Unified Modeling Language User Guide (2nd Edition). Addison-Wesley Professional.

Lamsweerde, A. V. (2000). Formal Specification: Techniques and Applications. Wiley.

Pressman, R. S. (2014). Software Engineering: A Practitioner's Approach (8th Edition). McGraw-Hill Education.