Chandigarh Engineering College- CGC, Landran, Mohali
Department of Computer Science and Engineering

Assignment No.2

Subject and Subject code: Database Management Systems (BTCS 501-18)
Semester: 5th

Date on which assignment given: 24/9/2024 **Date of submission of assignment:** 30/9/2024

Total Marks: 10

Course Outcomes

CO1	Describe relational algebra expressions for a query and optimize the Developed expressions.
CO2	Design the databases using ER method and normalization.
CO3	Construct the SQL queries for Open source and Commercial DBMS.
CO4	Illustrate various methods of organizing data and transaction properties.
CO5	Implement the optimization techniques for security handling and enhance knowledge about advance databases.

Assignment related to COs		Marks Distribution	Relevance to CO No.	Level to Bloom Taxonomy
Q1	How would you make comparison between DAC, MAC, and RBAC access control methods?	2.5	CO5	L2
Q2	Discuss Object oriented and object relational databases in detail and also discuss their comparison.	2.5	CO4	L3
Q3	What is intrusion detection system? Explain.	2.5	CO5	L4
Q4	Write a note on Logical databases.	2.5	CO5	L4

Solution for Assignment 2

Ans: When comparing DAC (Discretionary Access Control), MAC (Mandatory Access Control), and RBAC (Role-Based Access Control), it's important to examine how each method defines and enforces access to resources, their security strengths, and use cases. Here's a breakdown:

1. Discretionary Access Control (DAC):

- **Definition**: In DAC, the owner of a resource (like a file or folder) has control over who can access or modify it. The owner decides the permissions for other users.
- **Key Characteristics**:
  - **Ownership**: Resource owners can assign permissions to users (read, write, execute).
  - **Flexibility**: Access control decisions are made by the individual resource owner.
  - **Implementation**: Commonly implemented in operating systems like Windows and Unix/Linux (e.g., file permissions).

2. Mandatory Access Control (MAC):

- **Definition**: MAC is a more restrictive access control model in which access decisions are made by the system based on security policies, not individual users. Resources have labels (e.g., security clearances) that determine access based on predefined rules.
- **Key Characteristics**:
  - **Centralized control**: Access control decisions are made by an authority (e.g., the system administrator), not individual users.
  - **Labels and Classifications**: Resources are tagged with security labels (like classification levels—Top Secret, Secret, etc.), and users are granted access based on these labels.
  - **No User Discretion**: Users cannot change permissions or share resources arbitrarily.

3. Role-Based Access Control (RBAC):

- **Definition**: RBAC grants permissions based on the roles that users assume within an organization, rather than the specific identity of the user. Each role has a set of permissions, and users are assigned to roles.
- **Key Characteristics**:
  - **Role-Based**: Access rights are defined by the roles, such as "admin," "manager," or "employee," and users are assigned roles according to their responsibilities.
  - **Centralized Management**: Admins assign roles to users, simplifying the management of permissions across large organizations.
  - **Least Privilege**: Users are given only the permissions necessary for their role.

Comparison Summary:

Feature	**DAC (Discretionary Access Control)**	**MAC (Mandatory Access Control)**	**RBAC (Role-Based Access Control)**
Control Type	User-based	System-based	Role-based
Access Decisions	Made by the owner of the resource	Made by the system/administrator	Made based on the user's role

Feature	DAC (Discretionary Access Control)	MAC (Mandatory Access Control)	RBAC (Role-Based Access Control)
Flexibility	High (users can change permissions)	Low (centralized control, no discretion)	Medium (based on roles, not individuals)
Security Level	Low to Medium (depends on user behavior)	High (enforces strict policies)	Medium to High (depends on roles setup)
Management Complexity	Low (simple to set up)	High (complex configurations)	Medium (requires role definitions)
Best for	Small to medium organizations	High-security environments (military, government)	Medium to large organizations with clear role definitions

Ans: Object-Oriented Databases (OODB) vs. Object-Relational Databases (ORDB)

In database management systems (DBMS), there are various ways to model and organize data. **Object-Oriented Databases (OODB)** and **Object-Relational Databases (ORDB)** represent two different approaches to managing data, each with its strengths and trade-offs. Below is a detailed discussion of each type of database, followed by a comparison of the two.

1. Object-Oriented Databases (OODB)

An **Object-Oriented Database (OODB)** integrates object-oriented programming (OOP) concepts with database management. In an OODB, data is represented as objects, similar to how data is handled in object-oriented programming languages (e.g., Java, C++, and Python). It uses the principles of OOP, such as inheritance, encapsulation, and polymorphism, to model complex data.

Key Features:

1. **Objects**: The fundamental unit in OODB is the "object," which encapsulates both **data** (attributes) and **methods** (functions that operate on the data). These objects correspond to real-world entities, like a `Car`, `Customer`, or `Employee`.
2. **Encapsulation**: Data and the functions that manipulate the data are encapsulated within an object, making it easier to manage complex data structures.
3. **Inheritance**: Objects can inherit properties and behaviors from other objects, facilitating code reuse and a hierarchical data structure.
4. **Polymorphism**: Methods can behave differently based on the object that invokes them, even if the method has the same name.
5. **Complex Data Types**: OODBs allow the use of complex data types (like collections, arrays, and sets) as native types, which are often difficult to manage in relational databases.
6. **Relationships**: OODBs support relationships between objects in the form of references or pointers, reflecting relationships like associations, aggregations, and compositions that are common in object-oriented systems.
7. **Persistence**: Objects in an OODB are persistent, meaning that they survive beyond the application's runtime and can be retrieved and manipulated in future sessions.

2. Object-Relational Databases (ORDB)

An **Object-Relational Database (ORDB)** is a type of database management system that blendsfeature of object-oriented databases with relational databases. ORDBs extend the relational model by incorporating some aspects of object orientation, such as support for complex data types, inheritance, and polymorphism.

Key Features:

1. **Relational Model**: ORDBs maintain the core of the relational model, where data is stored in tables (relations). This ensures that SQL (Structured Query Language) is still the primary method for querying the database.
2. **User-Defined Types (UDTs)**: ORDBs allow users to define custom complex data types (e.g., an object type like `Employee` with attributes such as `name`, `ID`, and `address`). These types are stored and manipulated within the database.
3. **Inheritance**: In ORDBs, tables can inherit properties from other tables, which is similar to inheritance in object-oriented programming.
4. **Methods and Functions**: You can define methods (procedures or functions) that operate on complex types, just as you would in object-oriented programming.
5. **Support for Complex Data Types**: ORDBs support complex types such as arrays, sets, lists, and multi-valued attributes, which are typically difficult to manage in pure relational databases.
6. **SQL Extensibility**: While an ORDB extends SQL with new capabilities (such as object-oriented features), it still retains the core relational structure and capabilities (e.g., JOINs, GROUP BY).
7. **Relationship Support**: Like relational databases, ORDBs maintain relationships between tables, but they can also manage relationships between more complex data types.

Comparison of OODB and ORDB

Feature	**Object-Oriented Database (OODB)**	**Object-Relational Database (ORDB)**
Data Model	Object-based (objects, methods)	Relational with object-oriented extensions
Underlying Technology	Object-oriented programming principles	Relational model with object extensions
Support for Complex Types	Excellent (supports any object)	Good (supports complex data types via UDTs)
Inheritance	Direct inheritance between objects	Inheritance of tables (limited)
Query Language	Custom (may use object query languages)	SQL (extended for objects)
Performance	Can be slower with large datasets	Better suited for relational operations but can be slower for complex objects
Data Storage	Objects are stored directly in the database	Data is stored in tables; complex types in special columns
Flexibility	Very flexible, more like programming	Flexible within the context of relational model
Compatibility with SQL	Not directly compatible with SQL	Fully compatible with SQL, with extensions
Adoption	Low adoption in mainstream databases	More widely adopted than OODBs (e.g., PostgreSQL, Oracle)
Use Cases	CAD systems, multimedia, simulations	Business applications, enterprise systems, e-commerce

Ans: Intrusion Detection System (IDS)

An **Intrusion Detection System (IDS)** is a security technology designed to monitor network or system activities for malicious or abnormal behavior, potential security breaches, and unauthorized access attempts. It helps organizations detect and respond to possible security incidents, such as attacks, intrusions, or policy violations, in real-time or after the fact.

An IDS is essentially a monitoring tool that provides visibility into network and system activities, often by analyzing traffic, logs, and system calls. When it detects suspicious or anomalous behavior that might indicate a security threat, the IDS generates alerts to inform administrators about potential incidents.

Key Features of an IDS:

- **Monitoring**: Continuously observes network or system traffic for signs of suspicious activity.
- **Detection**: Identifies potential security breaches, including attacks, unauthorized access, or violations of security policies.
- **Alerting**: Notifies system administrators or security personnel when suspicious activity is detected, often with detailed information on the nature of the potential threat.
- **Reporting**: Provides logs and reports that can be analyzed to understand the nature of the attack and assist in improving security defenses.

Types of Intrusion Detection Systems:

1. **Network-Based IDS (NIDS)**:
    - A **Network-Based IDS** monitors network traffic in real-time and analyzes it for signs of potential malicious activities such as DDoS attacks, port scanning, or protocol misuse.
    - **Deployment**: It is typically placed at strategic points in a network (e.g., behind firewalls or at network perimeters) to monitor all inbound and outbound traffic.
    - **Functionality**: NIDS captures network packets and inspects them for known attack signatures or anomalies based on statistical models.
    - **Example Tools**: Snort, Suricata, Cisco Secure IDS.

2. **Host-Based IDS (HIDS)**:
    - A **Host-Based IDS** is installed on individual hosts (servers, workstations) to monitor and analyze activities on the host itself, such as file accesses, system calls, user activities, and application behavior.
    - **Deployment**: HIDS are deployed on critical servers or endpoints that need extra protection from attacks targeting system-level vulnerabilities.
    - **Functionality**: It typically checks for unauthorized file modifications, unusual system calls, and discrepancies in the system's normal behavior.
    - **Example Tools**: OSSEC, Tripwire, Samhain.

3. **Hybrid IDS**:
    - **Hybrid IDS** combines the features of both Network-Based and Host-Based IDS to provide a more comprehensive security monitoring solution. It can analyze both network traffic and host-level activities.
    - **Example Tools**: Security Information and Event Management (SIEM) systems, such as Splunk, often combine IDS capabilities with other monitoring tools.

How IDS Works:

An IDS typically operates in the following sequence:

1. **Traffic Capture**: The IDS collects data, either by analyzing network traffic (in the case of a NIDS) or system activities (in the case of a HIDS).
2. **Pre-Processing**: Raw data is filtered to remove irrelevant information and reduce the volume of data to be analyzed.
3. **Detection**: The system analyzes the captured data based on predefined rules or statistical models to detect potential threats.
4. **Alerting**: If a potential intrusion or suspicious activity is detected, the IDS generates an alert, typically with details about the event, its severity, and the affected systems.
5. **Response**: Depending on the configuration, the IDS may initiate an automatic response (like blocking the malicious activity or blocking the user), or it may simply notify administrators for manual investigation.

Ans: Logical Databases: A Note

A **logical database** refers to the **logical view** or **structure** of data as perceived by the users, independent of how the data is physically stored or organized on storage devices. It focuses on the **relationships between data** elements and how they are grouped together in an abstract, conceptual way. The concept of a logical database allows users to interact with data without needing to understand or manage the underlying physical details.

Key Concepts of Logical Databases:

1. **Logical View vs. Physical View**:
    - **Logical View**: Represents how users or applications see the data and how it is logically structured. It is concerned with **what data** is stored, **how it is grouped**, and how different pieces of data relate to each other.
    - **Physical View**: Refers to how data is actually stored and organized on physical storage devices (such as hard drives, SSDs, or cloud storage). It deals with the **implementation details** like indexing, storage methods, and retrieval mechanisms.

    The **logical schema** defines the structure, while the **physical schema** defines how the data is stored and accessed.

2. **Data Models and Logical Design**:
    - Logical databases are typically represented using various **data models** (such as the **Relational Model**, **Hierarchical Model**, **Network Model**, or **Object-Oriented Model**).
    - The **logical schema** of a database is often designed using a **data model** that defines the relationships between tables, entities, attributes, and other components in a database. In relational databases, for example, tables, rows, and columns form the logical view.
3. **Database Schema**:
    - The **logical schema** refers to the **design or blueprint** of the database that defines the structure of the data. It includes **tables**, **relationships**, **views**, **constraints**, and other entities that reflect how data is organized logically.
    - In a **relational database**, this would involve defining tables, columns, primary keys, foreign keys, and relationships between tables.
4. **Independence from Physical Storage**:
    - A logical database design is **independent of the physical storage details**. This is known as **data independence**, which allows database designers and end-users to focus on **data relationships and usability** without worrying about how or where the data is stored.
    - Data independence is essential in modern database systems, allowing changes to be made to the physical structure (e.g., optimizing storage, changing file formats) without affecting the logical structure or user queries.

Components of a Logical Database:

1. **Entities and Attributes**:
    - Entities represent real-world objects or concepts (e.g., Employee, Customer, Product), while attributes define the properties or characteristics of those entities (e.g., Employee ID, Name, Salary).
2. **Relationships**:
    - Relationships describe how entities are related to each other (e.g., a Customer might place an Order, or an Employee might work in a Department). In logical databases, these relationships are defined using **foreign keys** and **primary keys** in relational models.
3. **Normalization**:
    - **Normalization** is a technique used in relational databases to organize data in a way that reduces redundancy and dependency, ensuring the logical schema is efficient and minimizes anomalies during data operations like insertions, updates, and deletions.

4. **Views**:
    - A **view** is a virtual table that represents a specific subset or combination of data from one or more tables. Views are part of the logical database, providing different perspectives or ways to interact with the underlying data.
5. **Constraints**:
    - Constraints define the rules or restrictions that apply to data in a logical database (e.g., **Primary Key**, **Foreign Key**, **Unique**, **Not Null**). These rules ensure data integrity and maintain consistency

Advantages of Logical Databases:

1. **Data Independence**:
    - Logical databases provide **data independence** by abstracting the physical storage details. Changes to how data is physically stored do not require changes to the logical schema or affect user queries.
2. **Flexibility and Scalability**:
    - Logical databases allow users to focus on the **structure and organization** of data without worrying about the underlying physical considerations, making it easier to adapt to future changes or scale the system.
3. **Improved Data Integrity**:
    - Logical databases, especially those using the relational model, typically have built-in constraints and rules (like primary and foreign keys) that help maintain data integrity and consistency.
4. **Easier Data Access**:
    - The logical structure makes it easier to organize and query the data. Users can create views or complex queries without dealing with how the data is physically stored.