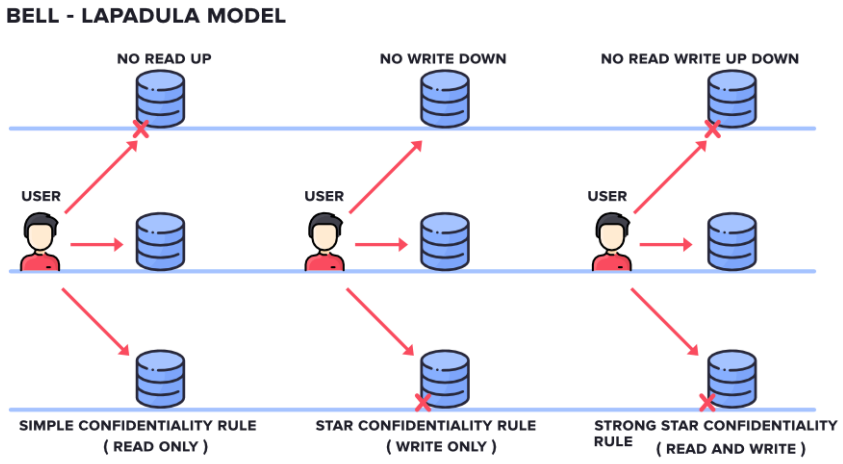
Advanced Database Management Systems

Module 1

Module 2

Explain Bell-LaPadula Model in detail:

Aspect	Description
Purpose	Ensures **data confidentiality** (prevents unauthorized disclosure of information)
Developed By	David Bell and Leonard LaPadula (1973) for U.S. Department of Defense
Main Focus	Controlling **read/write access** based on security levels
Security Levels	Top Secret > Secret > Confidential > Unclassified
Subjects	Active entities (users, processes) that access data
Objects	Passive entities (files, databases, documents) containing data
Simple Security Property (No Read Up)	A subject **cannot read data** at a higher security level — prevents unauthorized reading
Star (*) Property (No Write Down)	A subject **cannot write data** to a lower security level — prevents data leakage
Strong Star Property	A subject can read and write **only** at the same security level
Discretionary Security Property	Access also controlled by an **access matrix** (discretionary access control)
Goal	Maintain **confidentiality** — keep secrets secret
Limitation	Does **not handle integrity** or **availability**; only focuses on confidentiality
Real-world Example	Military systems or government databases with classified documents



Explain the concept of Role Based Access Control

Purpose: RBAC controls access by assigning users to **roles**, and roles are associated with **permissions**. It simplifies administration by grouping privileges by job function rather than per-user.

Aspect	Description	Example
Definition	A security model where access permissions are assigned based on a user's **role** within an organization, not on the individual user directly.	A "Manager" role can view and edit reports, while a "Staff" role can only view them.
Key Idea	Users → assigned to **roles** → roles have **permissions** → permissions allow **actions** on resources.	User → "Rohan" → assigned "Admin" role → can modify all user accounts.
Main Components	1. **Users** - Individuals using the system.2. **Roles** - Job functions (e.g., Admin, Analyst).3. **Permissions** - Allowed operations (e.g., read, write).4. **Sessions** - Mapping of users to activated roles.	—
Advantages	- Simplifies permission management.- Ensures consistency.- Easier to audit and maintain security.	—
Disadvantages	- Not flexible for exceptions.- Role explosion (too many roles for complex organizations).	—
Real-World Use	Common in databases, operating systems, and enterprise applications.	E.g., Hospital system: Doctors can access patient records; nurses can only view them.

Basic elements

- **Users (U)** — human users or processes.
- **Roles (R)** — job functions (e.g., `HR_Clerk`, `Manager`, `DBA`).
- **Permissions (P)** — approval to perform operations on objects (e.g., `read payroll`, `write payroll`).
- **Sessions (S)** — a mapping of a user to a set of roles active during a login/transaction.
- **Role assignment relations:**
  - `UA ⊆ U × R` (user↔role assignments)
  - `PA ⊆ P × R` (permission↔role assignments)

Key principles / features

- **Least privilege:** assign minimal roles to cover job needs.
- **Separation of duties (SoD):** prevent one user from holding conflicting roles (e.g., `PaymentRequester` and `PaymentApprover`). Two types:
  - **Static SoD (SSD):** user cannot be assigned both roles ever.
  - **Dynamic SoD (DSD):** user can have both roles but cannot activate both in the same session.
- **Role hierarchy (inheritance):** roles can inherit permissions (e.g., `Manager` > `Employee`), reducing duplication.
- **Constraints:** business rules limiting assignments or role activation.

How it works (example)

- Define roles: `HR_Read`, `HR_Edit`, `Payroll_Admin`.
- Assign permissions to roles: `HR_Edit` → {`modify_employee_record`}.
- Assign users to roles: `Alice` → `HR_Edit`.
- When Alice logs in, she activates a session with `HR_Edit` and gets the associated permissions.

Advantages

- **Scalability:** adding a new user = assign roles, not dozens of permissions.
- **Manageability:** central role definitions map to business functions.
- **Auditing & compliance:** easier to review who can do what.
- **Supports SoD and least privilege** natively.

Variants / standards

- **RBAC0, RBAC1, RBAC2, RBAC3** (NIST): increasing features (role hierarchies, constraints, etc.).
- Implementations: Active Directory groups (approximate RBAC), enterprise IAM systems, cloud IAM roles.

Limitations

- Role explosion if not designed carefully (too many fine-grained roles).
- Complex role engineering needed to reflect business processes.
- Not designed to express object-level discretionary ownership rules (can be combined with DAC).

Explain in detail Discretionary Access Control and Mandatory access Control

Discretionary Access Control (DAC)

Definition: Access decisions are based on **identity/ownership** and policies set **by owners** of objects. The owner chooses who can access an object (hence "discretionary").

Mechanisms

- **Access Control Lists (ACLs):** each object has a list of users/groups and allowed operations.
- **Capability lists:** each subject keeps a list of capabilities (rights) it possesses.
- **Owner-based control:** the creator/owner can grant/revoke access.

Example systems

- **UNIX file permissions (owner/group/other)** and ACL extensions.
- **Windows NTFS ACLs** (granular rights settable by file owner).

Characteristics and pros

- **Flexible:** owners can share resources easily.
- **Fine-grained control:** per-object, per-user permission settings.
- **User productivity:** convenient for collaboration.

Limitations and risks

- **Less central control:** owners might make unsafe choices (accidental disclosure).
- **Susceptible to Trojan horse attacks:** a subject with access can run code that leaks data.
- **Difficult to enforce enterprise policy uniformly** across many owners.

Mandatory Access Control (MAC)

Definition: Access decisions are **centrally enforced** by the system based on **security labels** and a global policy; users/owners cannot change labels or override policy. MAC is used when strict, non-discretionary control is required (e.g., military, classified systems).

Mechanisms

- Objects and subjects are labeled (e.g., `TopSecret`, compartments). A central policy enforces allowed flows (often using models like BLP for confidentiality or Biba for integrity).

Example systems

- **Military Multilevel Security (MLS)** systems.
- **SELinux** and **AppArmor** implement strong MAC-like policies on Linux (labels and mandatory policy).
- Some government OS configurations or DBMSs that implement MLS.

Characteristics and pros

- **Strong centralized policy** ensures consistent enforcement.
- **Good for high-assurance environments** (confidentiality/integrity requirements).

- **Owners cannot override** — reduces accidental leakage risk.

Limitations

- **Less flexible:** legitimate sharing is harder; admin overhead is higher.
- **Requires label/clearance management** which can be complex.
- **May be overkill** for regular commercial environments.

Aspect	DAC	MAC	RBAC
Who controls policy	Object owner (discretionary)	Central authority (system)	Central roles; admin-managed
Typical use case	General-purpose OS, collaboration	Military, classified, high-assurance systems	Enterprise access management, business roles
Flexibility	High	Low	Medium (structured flexibility)
Ease of admin	Can be chaotic at scale	Centralized but heavy	Scales well if roles well-designed
Ability to enforce SoD / enterprise policy	Weak	Strong (with admin effort)	Strong (via constraints)
Examples	UNIX, Windows ACLs	SELinux, MLS systems	Active Directory groups, IAM roles

Module 3

Explain Different Parallel Database Architecture

Type	Description	Data Storage / Processing	Example / Analogy
1. Shared Memory Architecture	All processors share a **single memory and disk**. Each processor has local cache but can access global memory.	One copy of the database in shared memory. All CPUs can read/write.	Like multiple chefs sharing one kitchen.
2. Shared Disk Architecture	Each processor has **its own memory**, but all share the **same disk(s)**.	Processors read/write to shared disk; coordination via lock manager.	Many chefs have their own counters but use the same fridge.
3. Shared Nothing Architecture	Each processor has **its own CPU, memory, and disk** — no sharing at all. Communication through a network.	Data is **partitioned** across nodes. Each node processes its local part independently.	Each chef has their own kitchen and ingredients; they just coordinate via messages.
4. Hierarchical (or Hybrid) Architecture	Combines features of the above architectures — often used in large systems.	Clusters of shared-memory or shared-disk systems connected in a shared-nothing style.	Like a chain of restaurants — each branch (cluster) runs independently but follows the same master plan.

Aspect	Shared Memory	Shared Disk	Shared Nothing
Scalability	Low (memory bus becomes bottleneck)	Moderate	High
Fault Tolerance	Low	Moderate	High
Communication Speed	Very Fast	Fast	Slower (via network)
Data Sharing	Easy	Easy	Difficult (data partitioned)
Example Systems	IBM System X	Oracle RAC	Teradata, Google BigQuery

What are the parameters for measuring the performance of parallel processing systems

Parameter	Meaning / Definition	Formula / Notes	Goal
1. Speedup (S)	Measures how much faster a parallel system performs a task compared to a single processor.	$S = \frac{T_1}{T_p}$ where T_1 = time on 1 processor, T_p = time on p processors	Higher is better (ideal = number of processors used)
2. Efficiency (E)	Indicates how effectively the processors are utilized.	$E = \frac{S}{p} = \frac{T_1}{p \times T_p}$	Close to 1 (100%) is ideal
3. Scalability	Ability of the system to maintain performance as the number of processors increases.	Depends on how performance grows with added processors.	Should remain high when more nodes are added.
4. Throughput	Number of tasks or transactions completed per unit time.	-	Higher throughput = better system performance.
5. Load Balancing	How evenly the work is distributed among processors.	-	Balanced load prevents idle processors.
6. Latency / Response Time	Time taken to respond to a single request or task.	-	Lower latency = better responsiveness.
7. Communication Overhead	Time spent coordinating and transferring data between processors.	-	Should be minimized.

Intra-query Parallelisms

Aspect	Description
Meaning	Intra-query parallelism means **executing parts of a single query in parallel** to reduce response time.
Goal	Speed up **complex or long-running queries** (like large joins or aggregations) by dividing the work among processors.
Where it's used	In data warehouses and analytical systems where one query scans large volumes of data.

Type	Explanation	Example
1. Intra-operation Parallelism	Parallelizes a single operation (like a scan, join, or sort).	A table scan divided among multiple disks or processors.
2. Inter-operation Parallelism	Parallelizes different operations in a query plan that can be executed simultaneously.	One processor sorts while another performs a join.
3. Pipelined Parallelism	Output of one operation is directly streamed as input to the next — overlapping execution.	As soon as tuples are selected, they start being aggregated — without waiting for the full selection to finish.

Benefit	Description
Reduced query response time	Faster results for large analytical queries.
Improved resource utilization	All processors work simultaneously.
Scalable performance	Better use of multiple CPUs/disks.

Module 4

Explain the difference between Centralized Databases and Distributed Databases

- Centralized Databases have all the data in a single server
- Distributed Databases have the data spread across multiple Servers
- There is less data redundancy in centralized servers and it is overall more cheaper to operate
- There is more data safety and recovery capabilities in the event of an error in distributed systems as multiple copies exist across different locations.
- Centralized Databases are easier to manage since all the data exists in a single location

Explain different commit protocols
a transaction is a logical unit of work (like transferring money or updating records) that must either happen entirely or not at all — that's the Atomicity in ACID
Now, here's the issue commit protocols solve:

When a transaction involves multiple sites or databases (in distributed systems), how do we make sure all sites agree to commit or roll back — even if some fail or messages get lost?

That's where commit protocols come in.

Protocol	Phases	Coordinator Failure Safe?	Blocking?	Notes	Phases Involved
1 Phase Commit	1	❌	❌	Fast but unsafe	Directly Commit
2 Phase Commit	2	⚠ (can block)	✅	Most common	Prepare Phase, Commit /Abort Phase
3 Phase Commit	3	✅	❌	Complex but non-blocking	CanCommit? PreCommit DoCommit

Explain centralized and distributed database and give comparison between the two

Feature	Centralized DB	Distributed DB
Location of Data	Single site	Multiple sites
Failure Impact	Total failure	Partial failure possible
Data Access Speed	Fast (no network delay)	Slower (depends on network)
Scalability	Limited	High
Reliability	Low (single point of failure)	High (redundancy)
Management	Simple	Complex
Commit Protocols	Simple (local commit)	Complex (e.g., 2PC, 3PC)

Module 5

Explain unstructured, semi structured and structured data, with examples

Type	Structure	Example	Storage	Querying	Example
Structured	Fixed schema	SQL tables	RDBMS	SQL	Student Record Database
Semi-Structured	Flexible schema	JSON, XML	NoSQL	Key-value / document queries	XML data models
Unstructured	No schema	Text, image, video	File systems, Hadoop	AI/NLP tools	images, videos, text

Explain hierarchical data model

Aspect	Description
Structure	Tree-like model with parent-child relationships (one parent, many children)
Data Organization	Records connected using pointers or links
Root Node	Single entry point to all data (top of the hierarchy)
Relationship Type	One-to-many (1:N) only
Example	University → Department → Professor
Data Access Method	Navigational — must start from root and traverse down
Advantages	Fast for 1:N relationships; simple structure; strong data integrity
Disadvantages	Inflexible; difficult for many-to-many relationships; changes require reorganization
Real-world Examples	IBM IMS, Windows Registry, XML-based storage

Aspect	Description
Use Cases	Organizational charts, file systems, directory structures

Explain the XML Syntax include (xml declaration, tags, nesting of elements, root elements, xml attribute etc)

Feature	Description	Example
XML Declaration	Appears at top; defines version & encoding	`<?xml version="1.0" encoding="UTF-8"?>`
Tags	Define start and end of data elements	`<name>Rohan</name>`
Nesting of Elements	Elements can be placed inside others to show hierarchy	`<student><name>Rohan</name></student>`
Root Element	One top-level element wrapping all others	`<university>...</university>`
Attributes	Extra info inside start tag as name-value pairs	`<student id="101">`