Database Security MidSem 2025

Q1)

Ans: (i) Role Hierarchy and Permission Assignment

🔷 Step 1: Understanding the Requirement

From the question:

- TAs and CIs of a course can **read and write** that course's grades.
- CIs can additionally **submit final grades**.
- All faculty can **read grades of every course**.
- We are given specific roles and permissions for only 3 courses: 123, 456, 789.

🔷 Role Hierarchy (RH)

A role hierarchy is used when a senior role should inherit permissions of a junior role.

Since:

- CIs can do everything TAs can do for a course (read & write)
- Plus CIs can submit final grades

Therefore:

$$123CI \geq 123TA$$

$$456CI \geq 456TA$$

$$789CI \geq 789TA$$

This means:

- A CI automatically inherits read and write permissions from TA
- CI additionally has submit permission

No other hierarchy is needed because:

- Faculty (FacGrad) only has read permissions and does not logically dominate CI or TA roles.

🔷 Permission Assignment (PA)

Now we assign permissions according to system rules.

1️⃣ TA Roles

Since TAs can read and write grades of their course:

- 123TA → {read123, write123}
- 456TA → {read456, write456}
- 789TA → {read789, write789}

2️⃣ CI Roles

CIs can submit final grades.

Because of hierarchy, they inherit read and write automatically.

Thus we assign:

- 123CI → {submit123}
- 456CI → {submit456}
- 789CI → {submit789}

Due to inheritance, each CI effectively has:

- read
- write
- submit
  for its course.

3️⃣ Faculty Role (FacGrad)

The problem states:

All faculty members can read grades for every course.

So:

FacGrad → {read123, read456, read789}

Faculty are not given write or submit permissions.

(ii) Static Separation of Duty (SSD)

🔷 What is Static Separation of Duty?

Static SoD constraints restrict **role assignment** to a user.

👉 It controls what roles a user is allowed to be assigned —
not what roles they activate in a session (that is dynamic SoD).

🔷 From the Question — What Must Be Enforced Statically?

From the problem statement:

1. Graduate students and faculty are disjoint groups
2. No graduate student can be a Course Instructor (CI)
3. No graduate student can be a TA for more than two courses

All of these are **assignment-level restrictions**, so they are enforced using **Static SoD constraints**.

🔷 SSD Constraint 1: Graduate vs Faculty Disjointness

The question states:

Graduate students and faculty are disjoint groups.

This means:

- A user cannot be both a graduate student and faculty.
- In role terms, TA roles (graduate students) must not overlap with faculty role (FacGrad).

Constraint:

User cannot be assigned both:

FacGrad and any of {123TA,456TA,789TA}

This ensures:

- No faculty member is treated as graduate TA.
- No graduate student is treated as faculty.

🔷 SSD Constraint 2: Graduate Cannot Be CI

The question says:

No graduate student can be a Course Instructor.

TA roles represent graduate students.

So we enforce:

A user assigned any TA role **cannot be assigned any CI role**.

Constraint:

{123TA,456TA,789TA} mutually exclusive with {123CI,456CI,789CI}

Meaning:

- A graduate student (TA) cannot also be assigned CI role of any course.

🔷 SSD Constraint 3: At Most Two TA Roles Per Graduate Student

The question states:

No graduate student can be a TA for more than two courses.

This is a cardinality constraint.

Constraint:

$$\text{Cardinality}(\{123TA,456TA,789TA\}) \leq 2$$

Meaning:

- A user can hold at most 2 of these TA roles.
- Example: 123TA & 456TA allowed
- 123TA, 456TA & 789TA together NOT allowed.

This is purely static because it restricts assignment, not activation.

🔷 Why All These Are Static

They depend only on:

- What roles a user is assigned.

They do NOT depend on:

- Session
- Login
- Active roles

Therefore they are Static SoD constraints.

Final Clean Exam Answer (Structured)

Static Separation of Duty Constraints

1. **Graduate and Faculty Disjointness**
   FacGrad mutually exclusive with {123TA,456TA,789TA}
2. **Graduate Cannot Be CI**
   {123TA,456TA,789TA} mutually exclusive with {123CI,456CI,789CI}
3. **At Most Two TA Roles per Graduate Student**
   $\text{Cardinality}(\{123TA,456TA,789TA\}) \leq 2$

(iii) Dynamic Separation of Duty (DSD)

🔷 What is Dynamic Separation of Duty?

Dynamic SoD restricts **role activation within a session**, not assignment.

👉 A user may be assigned multiple roles
👉 But cannot activate certain roles together at the same time (same login session)

🔷 What Does the Question Require?

The problem states:

The system must not permit anyone to perform in more than one Course-Instructor role at any given time (e.g., login session).

This means:

- A user may be assigned multiple CI roles
- But during one session, they can activate only one CI role

This prevents confusion in submitting final grades, especially if the same instructor teaches multiple courses.

🔷 Why This Is Dynamic (Not Static)

Because:

- The restriction is about **"at any given time"**
- It talks about login/session behavior
- It does NOT forbid assignment of multiple CI roles
- It only restricts simultaneous activation

Therefore, it is a **Dynamic SoD constraint**.

🔷 DSD Constraint Definition

CI Roles:

$$\{123CI, 456CI, 789CI\}$$

Dynamic Constraint:

$$\text{Cardinality}(\{123CI, 456CI, 789CI\}) \leq 1$$

in any session.

🔷 What This Means Practically

If a professor is assigned:

- 123CI
- 456CI

They can:

✓ Log in as 123CI
✓ Log out and log in as 456CI

But they **cannot activate both roles in the same session simultaneously**.

🔷 Why This Constraint Is Needed

Without this:

- An instructor teaching multiple courses could accidentally submit final grades for the wrong course.
- It ensures clarity and prevents operational confusion.

Final Exam-Ready Answer

Dynamic Separation of Duty Constraint

At most one of the following roles may be activated in any session:

$$\text{Cardinality}(\{123CI, 456CI, 789CI\}) \leq 1$$

This ensures that although a user may be assigned multiple CI roles, they cannot act in more than one CI role during the same login session.

Q2)

Ans: SQL Security Setup for Video Store Database

🔷 Assumptions (Clearly Stated)

Since exact duties are not specified, we assume:

- **SPIELBERG (Purchasing Department)**
  Handles inventory, pricing, and new item releases.
  → Needs access to ITEM, possibly FILM and STUDIO.
- **SHYAMALAN (Shipping Department)**
  Handles order fulfillment and shipment.
  → Needs access to CART, SHOPPER (shipping address only), and ITEM.
- **MARSHALL (Customer Service Manager)**
  Handles customer information and complaints.
  → Needs full access to SHOPPER and limited access to order data.

We apply **principle of least privilege**.

🔷 1️⃣ SPIELBERG — Purchasing Department

Needs:

- Read and update inventory
- Insert new items
- View film and studio info

SQL Grants:

GRANT SELECT, INSERT, UPDATE ON ITEM TO SPIELBERG;

GRANT SELECT ON FILM TO SPIELBERG;

GRANT SELECT ON STUDIO TO SPIELBERG;

Explanation:

- Cannot delete items (avoids data loss)
- Only operational purchasing access

🔷 2️⃣ SHYAMALAN — Shipping Department

Needs:

- View customer shipping address
- View cart orders
- Update inventory when item ships

Since credit card data must be protected, we restrict column access.

SQL Grants:

GRANT SELECT (SHNUM, FNAME, LNAME, SHIPADDR)

ON SHOPPER TO SHYAMALAN;

GRANT SELECT ON CART TO SHYAMALAN;

GRANT SELECT, UPDATE (INVENTORY)

ON ITEM TO SHYAMALAN;

Explanation:

- No access to CRCARD (credit card field)
- Cannot change pricing
- Only updates inventory after shipment

No GRANT OPTION here (not managerial).

🔷 3 MARSHALL — Customer Service Manager

Needs:

- View and update customer data
- Handle complaints and order queries
- As manager, may delegate privileges

SQL Grants:

GRANT SELECT, UPDATE ON SHOPPER

TO MARSHALL WITH GRANT OPTION;

GRANT SELECT ON CART

TO MARSHALL WITH GRANT OPTION;

Explanation:

- Full control over customer data
- No access to payroll (ROLES.SALARY)
- Manager can delegate access to subordinates

Q3) (a) — Scenario where Mandatory Access Control (MAC) prevents a breach that Discretionary Access Control (DAC) cannot

Consider a **military intelligence system** with classified documents labeled:

- Top Secret
- Secret
- Confidential
- Unclassified

Users (officers) also have security clearances at these levels.

🔷 Scenario

Suppose:

- Alice has **Top Secret** clearance.
- Bob has only **Confidential** clearance.
- Alice accesses a Top Secret document about a military mission.

Under **Discretionary Access Control (DAC)**:

- Alice owns or has access to the document.
- She could grant read access to Bob.
- If Bob is compromised or careless, classified information leaks.
- The system allows this because DAC lets users propagate permissions.

Thus, DAC **cannot prevent information leakage once a legitimate user chooses to share it.**

🔷 How Mandatory Access Control (MAC) Prevents This

Under **Mandatory Access Control (e.g., Bell–LaPadula model):**

- Access is controlled by system-enforced security labels.
- Policies are centrally enforced.
- Users cannot override access rules.

Key rules:

- **No Read Up** (cannot read higher classification)
- **No Write Down** (cannot send high-level data to lower level)

In this case:

- Bob (Confidential) cannot read Top Secret data.
- Alice (Top Secret) cannot write/share that data to Confidential level.
- Even if Alice wants to share, the system blocks it.

🔷 Why DAC Cannot Prevent This

Because in DAC:

- Object owners can grant permissions at their discretion.
- There is no automatic control of information flow between levels.
- Information may leak intentionally or accidentally.

🔷 Conclusion

Mandatory Access Control prevents unauthorized information flow through system-enforced classification levels, ensuring that sensitive information cannot flow from higher to lower security levels — something discretionary access control cannot guarantee.

(b) Given Policy

- Doctors:
    - Can read and write **patient records**
    - Can read and write **prescriptions**

- Nurses:
    - Can read and write **prescriptions**
    - Must learn **nothing about patient record contents**
- Prevent information flow from patient records → prescriptions.

This is fundamentally an **information flow control problem**.

(i) Capture Policy in a Lattice Model

We use a **security lattice** where labels represent confidentiality levels.

🔷 **Step 1: Define Security Labels**

Let:

- Lp = Patient Records (High sensitivity)
- Lr = Prescriptions (Lower sensitivity)

Define ordering:

$$Lr \leq Lp$$

Meaning:

- Patient Records are more confidential than Prescriptions.

🔷 **Step 2: Assign Labels to Objects**

- Patient records → label Lp
- Prescriptions → label Lr

🔷 **Step 3: Assign Labels to Subjects**

- Doctors → Lp (High clearance)
- Nurses → Lr (Lower clearance)

🔷 **Step 4: Enforce Bell–LaPadula Rules (Confidentiality Model)**

Bell–LaPadula rules:

1. **No Read Up**
2. **No Write Down**

Apply to this scenario:

🔷 **Nurses**

- Label = Lr
- Cannot read patient records (Lp)
  Because that would be **read up** → forbidden.

✓ Nurses learn nothing about patient records.

🔷 **Doctors**

- Label = Lp
- Can read patient records (same level).
- Can write prescriptions?

Here is the key:

If prescriptions are lower level,
doctor writing prescription would be **write down**.

Bell–LaPadula forbids write-down → prevents leakage.

Thus:

- Doctors cannot leak patient record data into prescriptions unless policy explicitly allows controlled declassification.

Therefore:
Information flow from patient records → prescriptions is prevented.

🔷 **Lattice Diagram (Exam Sketch)**

L_p (Patient Records, Doctor)
↑
|
L_r (Prescriptions, Nurse)

Higher cannot write to lower.
Lower cannot read higher.

(ii) Most Appropriate Security Model

The most appropriate model is:

✅ **Bell–LaPadula Model**

Why?

- This is a **confidentiality requirement**
- The main goal is preventing information flow from high (patient records) to low (prescriptions)
- Bell–LaPadula enforces:
    - No Read Up
    - No Write Down
- Exactly matches policy requirement.

Other models like Biba focus on integrity, not secrecy.

(iii) Prevent Doctor Prescribing for Herself

Now we add constraint:

A doctor must not prescribe medicine for herself.

This is NOT a confidentiality problem.

It is a **conflict-of-interest / integrity constraint.**

We augment the model using:

Option 1: Clark–Wilson Model (Best Answer)

Clark–Wilson enforces:

- Well-formed transactions
- Separation of duty
- Certification rules

Add rule:

- A doctor cannot issue prescription where:

Doctor_ID = Patient_ID

This can be implemented as:

- A transaction constraint
- Or database integrity constraint
- Or separation-of-duty rule

Option 2: Add Dynamic Separation of Duty

- Doctor writing prescription role
- Doctor patient role
- Cannot activate both for same identity

(c) — Inference Attack Using Aggregate Queries

Given

- Table:
  Students(SID, SNAME, DEPTID, LOANAMOUNT)
- Allowed aggregate queries: COUNT, SUM, MEAN
- Query result is returned **only if 4–7 tuples** are selected.
- Direct query on Erika alone (1 tuple) is rejected.

We must:

- Find a **tracker**
- Use **at most 4 queries**
- Determine **Erika's loan amount**

🔷 Key Idea: Tracker Attack

A **tracker** is a condition that:

- Includes the target person (Erika)
- Has valid size (4–7 people)
- Can be paired with another valid set
- Their difference isolates the target

We use set difference logic via arithmetic.

🔷 Step 1: Choose a Valid Group Containing Erika

Suppose in some department there are:

- Erika

- A
- B
- C
- D

Total = 5 people (valid, 4–7 allowed)

Query 1

SELECT SUM(LOANAMOUNT)

FROM Students

WHERE DEPTID = 'CS';

Assume this group contains exactly those 5.

Let result =

$$S1=L(E)+L(A)+L(B)+L(C)+L(D)$$

🔷 **Step 2: Choose Subset Excluding Erika**

Construct another group of 4 people:

- A
- B
- C
- D

Size = 4 → valid

Query 2

SELECT SUM(LOANAMOUNT)

FROM Students

WHERE DEPTID = 'CS'

AND SNAME <> 'Erika';

Let result =

$$S2=L(A)+L(B)+L(C)+L(D)$$

🔷 Step 3: Subtract

$$S1 - S2 = L(\text{Erica})$$

We now know Erika's loan amount.

🔷 Why This Works

- Each query individually satisfies 4–7 rule.
- System protects only small/large result sizes.
- It does **not protect against overlapping query inference**.
- By subtracting valid aggregates, we isolate a single individual.