Database Security EndSem 2025

Q1 (a) — View Definition Statements

We are given the base relation:

Employees(ename, dept, salary)

We must define two views:

1. **EmployeeNames** → only employee names
2. **DeptInfo** → department and average salary

🔷 1️⃣ View: EmployeeNames

This view should contain only the ename attribute.

SQL Statement:

```
CREATE VIEW EmployeeNames AS

SELECT ename

FROM Employees;
```

Explanation:

- This view hides dept and salary.
- Used for authorization so users can see names without seeing salaries.

🔷 2️⃣ View: DeptInfo

This view lists:

- dept
- average salary for each department

Since we need averages per department, we use GROUP BY.

SQL Statement:

```
CREATE VIEW DeptInfo AS
SELECT dept, AVG(salary) AS avgsalary
```

```
FROM Employees
GROUP BY dept;
```

Explanation:

- AVG(salary) computes average salary.
- GROUP BY dept ensures calculation per department.
- avgsalary is an alias for clarity.

Final Answer (Write in Exam)

```
CREATE VIEW EmployeeNames AS
SELECT ename
FROM Employees;

CREATE VIEW DeptInfo AS
SELECT dept, AVG(salary) AS avgsalary
FROM Employees
GROUP BY dept;
```

Q1 (b)

Requirement

A user needs to know:

- Only the **average department salaries**
- Only for **Toy** and **CS** departments
- Not individual salaries

🔷 Key Idea

Since we already created the view:

```
DeptInfo(dept, avgsalary)
```

which contains:

- dept
- AVG(salary)

We should grant access only to this view — not to Employees table.

🔷 Step 1: Grant SELECT on DeptInfo

```
GRANT SELECT ON DeptInfo TO username;
```

🔷 Step 2: Restrict to Only Toy and CS

Since we want the user to see only Toy and CS departments, there are two clean approaches.

✅ Preferred Answer (Create Restricted View)

Create a more restricted view:

```
CREATE VIEW ToyCSDeptInfo AS
SELECT dept, avgsalary
FROM DeptInfo
WHERE dept IN ('Toy', 'CS');
```

Then grant:

```
GRANT SELECT ON ToyCSDeptInfo TO username;
```

🔷 Why This Is Correct

- User cannot see individual salaries.
- User cannot access Employees.
- User cannot see other departments.
- Only aggregate information for Toy and CS is available.

This follows **principle of least privilege**.

Final Exam Answer (Compact)

Create restricted view:

```
CREATE VIEW ToyCSDeptInfo AS
SELECT dept, avgsalary
FROM DeptInfo
WHERE dept IN ('Toy', 'CS');
```

Grant:

```
GRANT SELECT ON ToyCSDeptInfo TO username;
```

This allows access only to average salaries of Toy and CS departments.

Q1 (c)

Requirement

You want your secretary to:

1. **Fire people** → remove employees
2. **Check who is an employee** → see employee names
3. **Check average department salaries** → see DeptInfo

But you are delegating authority, so she needs proper privileges.

🔷 Step 1: Fire People

Firing means deleting tuples from Employees.

Grant:

```
GRANT DELETE ON Employees TO Secretary;
```

This allows removing employees.

🔷 Step 2: Check Who Is an Employee

To see names only (not salary), she should use the view.

Grant:

```
GRANT SELECT ON EmployeeNames TO Secretary;
```

This prevents salary access.

🔷 Step 3: Check Average Department Salaries

Grant:

```
GRANT SELECT ON DeptInfo TO Secretary;
```

This allows viewing average department salaries.

🔷 Should We Grant SELECT on Employees?

No.

If we grant:

```
GRANT SELECT ON Employees
```

She could see individual salaries, which is not intended.

Final Privileges to Grant

```
GRANT DELETE ON Employees TO Secretary;
GRANT SELECT ON EmployeeNames TO Secretary;
GRANT SELECT ON DeptInfo TO Secretary;
```

Q1 (d)

Question

You do NOT want your secretary to see **individual salaries**.

You granted in (c):

```
GRANT DELETE ON Employees TO Secretary;
GRANT SELECT ON EmployeeNames TO Secretary;
GRANT SELECT ON DeptInfo TO Secretary;
```

Now the question asks:

- Does this completely prevent her from knowing salaries?
- Can she deduce some salaries?
- Can she always find out any salary she wants?

🔷 Key Observation

Secretary has:

1. DELETE on Employees
2. SELECT on EmployeeNames
3. SELECT on DeptInfo (average salary per dept)

She does NOT have SELECT on Employees.

But she **can delete rows**.

This is the trick.

🔷 Can She Infer Some Individual Salaries?

Yes — she possibly can.

Example:

Suppose in CS department:

- Alice: 100
- Bob: 200

Average = 150

If secretary deletes Bob,

Now CS has only Alice.

New average = 100

She now knows Alice’s salary.

So:

She cannot directly read salaries,
but by deleting strategically, she may deduce them.

🔷 Can She Always Find the Salary of Any Individual?

No — not always.

It depends on the data.

If:

- Department has many employees,
- And she cannot isolate one person,

Then she may not uniquely determine salary.

But in small departments or by repeatedly deleting others,
she **may be able to infer it**.

So the protection is NOT guaranteed.

🔷 Why This Happens

Because:

- Aggregate view (AVG)
- Combined with DELETE privilege

- Creates an **inference attack**

This is known as:

Indirect disclosure via aggregate queries + modification rights

---

🔷 Final Answer (Exam Style)

No, the privileges do not completely prevent salary disclosure.
Although the secretary cannot directly read the salary attribute, she can delete employees and observe changes in the average salary from DeptInfo. By strategically deleting employees in a department, she may isolate one individual and deduce that person's salary. However, she cannot always determine any individual's salary — it depends on the number of employees and data distribution in the department.

This is a classic **inference + update privilege problem** (very important concept).

Q1 (e)

Requirement

You want to give your secretary the authority to allow other people to read the EmployeeNames view.

This means:

- She must be able to **grant SELECT** on EmployeeNames to others.
- So she needs the **GRANT OPTION**.

🔷 SQL Statement

Since you are the owner (DBA), you execute:

```
GRANT SELECT
ON EmployeeNames
TO Secretary
WITH GRANT OPTION;
```

🔷 What This Means

- Secretary can SELECT from EmployeeNames.

- Secretary can execute:

```
GRANT SELECT ON EmployeeNames TO SomeUser;
```

- Others will then be able to read employee names.

🔷 Important Concept

WITH GRANT OPTION allows **delegation of privilege**.

Without it:

- Secretary could read the view.
- But could not pass that right to others.

Final Exam Answer

```
GRANT SELECT
ON EmployeeNames
TO Secretary
WITH GRANT OPTION;
```

This allows the secretary to grant SELECT privilege on the EmployeeNames view to other users.

Q1 (f)

Situation

- Secretary has SELECT (with GRANT OPTION) on EmployeeNames.
- She creates two views:

```
CREATE VIEW AtoRNames AS

SELECT ename

FROM EmployeeNames

WHERE ename BETWEEN 'A' AND 'R';

CREATE VIEW HowManyNames AS

SELECT COUNT(*) AS total

FROM EmployeeNames;
```

Now you decide to grant her **INSERT** privilege on EmployeeNames.

🔷 Step 1 — Correct SQL Command

Since you are the owner, you execute:

```
GRANT INSERT
ON EmployeeNames
TO Secretary;
```

(Without GRANT OPTION unless explicitly required.)

🔷 Step 2 — What Does This Actually Mean?

Important:

EmployeeNames is defined as:

```
CREATE VIEW EmployeeNames AS

SELECT ename

FROM Employees;
```

This view:

- Is based on a single base table
- No aggregation
- No GROUP BY
- No joins

So it is **partially updatable**.

🔷 Step 3 — What Happens When Secretary Inserts into EmployeeNames?

Example:

```
INSERT INTO EmployeeNames VALUES ('John');
```

Problem:

The underlying table Employees requires:

(ename, dept, salary)

But the view contains only:

Ename

So:

- INSERT cannot succeed unless
  - The DB allows NULLs for dept & salary
  - Or default values exist

Otherwise, insertion fails.

🔷 Step 4 — Secretary's Actual Privileges After This Grant

After:

GRANT INSERT ON EmployeeNames TO Secretary;

She can:

✓ Attempt to insert via the view
✓ Insert only the ename attribute

But:

- She still CANNOT see salaries.
- She still has no SELECT on Employees.
- She cannot modify dept or salary.
- Insertion will succeed only if base-table constraints allow it.

🔷 Important Correction

Earlier answer did not fully explain:

- Whether insertion is actually possible.
- View updatability limits.
- Dependence on base table constraints.

That was incomplete.

🔷 Final Correct Exam Answer

To grant insert privilege:

GRANT INSERT ON EmployeeNames TO Secretary;

Since EmployeeNames is based on a single table without aggregation, it is updatable. However, because the view contains only the attribute ename, the secretary can only insert values for ename. The insertion will succeed only if the underlying table allows NULL or default values for dept and salary. The secretary still cannot see or modify individual salaries.

Q1 (g)

Situation

- You granted:

```
GRANT SELECT ON EmployeeNames
TO Secretary
WITH GRANT OPTION;
```

- Secretary then grants:

```
GRANT SELECT ON EmployeeNames TO Todd;
```

Now the question is about what happens **if you revoke the privilege from Secretary**, and Todd’s access is involved.

🔷 Key Issue

Todd received the privilege **indirectly** from you through Secretary.

So Todd's privilege depends on Secretary’s authority.

🔷 Case 1: You Revoke From Secretary

If you execute:

```
REVOKE SELECT ON EmployeeNames FROM Secretary;
```

Then:

- Secretary loses privilege.
- Todd also loses privilege automatically.
- Because Todd’s access was derived from Secretary.

This is **cascading revocation**.

🔷 Case 2: Todd Also Gets Privilege Directly From You

If you separately execute:

GRANT SELECT ON EmployeeNames TO Todd;

Then even if you revoke from Secretary:

Todd keeps access.

Because now Todd has an independent grant path.

🔷 Important Concept: Grant Graph

Privileges form a **grant graph**:

You → Secretary → Todd

If Secretary link breaks, Todd loses access
unless Todd also has direct link:

You → Todd

Final Exam Answer

If the secretary grants SELECT on EmployeeNames to Todd using her grant option, then Todd's privilege depends on the secretary's authority. If the privilege is later revoked from the secretary, Todd's privilege is also revoked due to cascading revocation. However, if Todd also received the privilege directly from the owner, he will retain it.

Q1 (h) — View Update on Preceding Schema

We have the base relation:

Employees(ename, dept, salary)

And views:

EmployeeNames(ename)
DeptInfo(dept, avgsalary)

The question concerns whether updates through these views are allowed.

🔷 1️⃣ Updating EmployeeNames View

```
CREATE VIEW EmployeeNames AS
SELECT ename
FROM Employees;
```

This view:

- Is based on a single base table.
- No aggregation.
- No GROUP BY.
- No joins.

Can we update it?

Yes — but only partially.

Example:

```
UPDATE EmployeeNames
SET ename = 'John'
WHERE ename = 'Jack';
```

This updates ename in Employees table.

However:

- You cannot insert a tuple because dept and salary are missing.
- You cannot update salary since it is not present in the view.

So this is a **partially updatable simple view**.

🔷 2️⃣ Updating DeptInfo View

```
CREATE VIEW DeptInfo AS
SELECT dept, AVG(salary) AS avgsalary
FROM Employees
GROUP BY dept;
```

This view uses:

- Aggregation (AVG)
- GROUP BY

Can we update it?

No.

If someone executes:

```
UPDATE DeptInfo
SET avgsalary = 90000
WHERE dept = 'CS';
```

The DBMS cannot determine:

- Which individual employee salaries should change?
- How to modify multiple rows to achieve new average?

There is no deterministic mapping.

Therefore:

This view is **NOT updatable**.

🔷 General Rule for View Updatability

A view is updatable if:

- It is based on a single base table
- No GROUP BY
- No aggregate functions
- No DISTINCT
- No joins
- No set operations

EmployeeNames satisfies these (partially).

DeptInfo violates them (aggregation), so not updatable.

Final Exam Answer

The EmployeeNames view is updatable because it is based on a single relation without aggregation, although updates are limited to attributes present in the view. However, the DeptInfo view is not updatable because it uses aggregation and GROUP BY. Since the database cannot determine how to modify individual salary tuples to achieve a new average, updates on this view are not allowed.

Q1 (i) — Extended Vacation Question

Requirement

You are going on vacation and want to authorize **Joe** to:

1. **Read and modify the Employees relation**
2. **Read and modify the EmployeeNames view**
3. **Be able to delegate authority** (WITH GRANT OPTION)

🔷 Step 1: Grant Privileges on Employees

To allow Joe to read and modify:

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON Employees
TO Joe
WITH GRANT OPTION;
```

Explanation:

- SELECT → read Employees
- INSERT, UPDATE, DELETE → modify Employees
- WITH GRANT OPTION → Joe can delegate to others

🔷 Step 2: Grant Privileges on EmployeeNames View

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON EmployeeNames
TO Joe
WITH GRANT OPTION;
```

This allows Joe to modify and delegate rights on the view as well.

🔷 Can Joe Read DeptInfo?

No — **unless explicitly granted**.

Having privileges on Employees does NOT automatically give:

- SELECT on DeptInfo

Privileges are checked per object.

Since DeptInfo is a separate view, Joe cannot read it unless you grant:

```
GRANT SELECT ON DeptInfo TO Joe;
```

Without this, access is denied.

🔷 Final Answer (Exam-Ready)

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON Employees
TO Joe
WITH GRANT OPTION;

GRANT SELECT, INSERT, UPDATE, DELETE
ON EmployeeNames
TO Joe
WITH GRANT OPTION;
```

Joe cannot read the DeptInfo view unless he is explicitly granted SELECT privilege on that view.

Q1 (j)

Situation

- You granted Joe:

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON Employees
TO Joe
WITH GRANT OPTION;
```

- Joe granted:

```
GRANT SELECT ON Employees TO Mike;
```

Now you return and want to:

- Revoke **Mike's SELECT privilege on Employees**
- But **NOT revoke Joe's privileges** (even temporarily)

🔷 Key Question

Can you remove Mike's privilege without affecting Joe?

🔷 Important Principle

In SQL, **REVOKE removes a privilege granted by the grantor**.

Mike’s privilege came **from Joe**, not from you.

The grant chain is:

You → Joe → Mike

You did NOT directly grant SELECT to Mike.

🔷 Can You Directly Revoke From Mike?

If you execute:

REVOKE SELECT ON Employees FROM Mike;

This revokes the privilege granted by Joe to Mike,
but Joe keeps his own privilege.

So:

✓ Mike loses SELECT
✓ Joe keeps all his rights
✓ No effect on Joe

🔷 Why This Works

Because:

- Revocation removes only the specified privilege for the specified user.
- It does not affect the grantor’s privilege.
- Joe’s authority remains valid.

🔷 When Would Joe Be Affected?

Joe would only be affected if you revoked from Joe directly:

REVOKE SELECT ON Employees FROM Joe;

Then:

- Joe loses privilege.
- Mike also loses privilege (cascading).

But that is NOT what we want.

Final Exam Answer

Yes, this can be done. Since Mike’s SELECT privilege was granted by Joe, you can execute:

REVOKE SELECT ON Employees FROM Mike;

This removes Mike’s privilege without affecting Joe’s privileges. Joe retains all rights previously granted to him.

Q1 (k).

🔷 Situation Summary

- You own Employees
- You granted Joe privileges **WITH GRANT OPTION**
- Joe created a view:

AllNames (based on EmployeeNames)

- Joe also created another table StaffNames
- Joe granted Mike SELECT on AllNames
- Mike granted Susan SELECT on AllNames

Now you want:

Remove Mike and Susan’s rights to see your data
Even if it annoys Joe

So you're willing to revoke some of Joe’s privileges if necessary.

🔷 Key Insight

Mike and Susan's access depends on:

Joe → AllNames (uses EmployeeNames → Employees)

If we break Joe’s ability to read from EmployeeNames,
then:

- AllNames becomes invalid
- Mike and Susan lose access
- Cascade happens

🔷 What REVOKE Statement?

Revoke Joe’s SELECT privilege on EmployeeNames with CASCADE:

```
REVOKE SELECT ON EmployeeNames
FROM Joe
CASCADE;
```

🔷 What Happens After This?

1️⃣ Joe’s Rights on Employees?

Joe still has whatever rights he has on:

- Employees

Unless we revoke those too,
his privileges on Employees remain unchanged.

We revoked only access to EmployeeNames.

2️⃣ What Happens to AllNames?

AllNames is defined using EmployeeNames.

Since Joe loses SELECT on EmployeeNames:

- AllNames becomes invalid
- AllNames is dropped automatically (due to CASCADE)

3️⃣ Mike and Susan?

- Their privilege was on AllNames
- Since AllNames is dropped
- Their privileges disappear
- They lose access

🔷 Final State

After executing:

REVOKE SELECT ON EmployeeNames FROM Joe CASCADE;

Result:

- Joe loses SELECT on EmployeeNames
- Joe still retains privileges on Employees
- AllNames view is dropped
- Mike and Susan lose access
- StaffNames (independent table) remains unaffected

🔷 Final Exam Answer

Execute:

REVOKE SELECT ON EmployeeNames FROM Joe CASCADE;

This revokes Joe’s privilege and causes cascading revocation. The view AllNames, which depends on EmployeeNames, is dropped. Consequently, Mike and Susan lose their rights to read AllNames. Joe retains whatever privileges he still has on Employees, since those were not revoked.

Q5 (a) — How Does Scale-Out Occur in MongoDB?

🔷 Scale-Out Meaning

Scale-out (horizontal scaling) means:

Adding more machines to distribute data and workload instead of upgrading a single machine.

🔷 MongoDB Uses Sharding

MongoDB achieves scale-out through **sharding**.

🔷 How Sharding Works:

1. Data is partitioned into chunks.
2. Chunks are distributed across multiple machines called **shards**.
3. A **mongos router** directs queries to appropriate shards.
4. A **config server** stores metadata about data distribution.

🔷 Shard Key

- A shard key determines how data is distributed.
- Proper shard key ensures:
    - Balanced data distribution
    - Even query load

🔷 Benefits

- Handles large datasets.
- Improves performance.
- Distributes read & write load.
- Provides horizontal scalability.

Q5 (b) — Aggregation in MongoDB and its Usefulness

🔷 What is Aggregation?

Aggregation processes documents to compute:

- Sum
- Average
- Count
- Grouping
- Sorting
- Filtering

MongoDB uses the **Aggregation Pipeline**.

🔷 Aggregation Pipeline

Documents pass through stages like:

- $match → filter
- $group → grouping
- $sort → sorting
- $project → reshape fields
- $limit → limit results

🔷 Example:

```
db.sales.aggregate([
  { $group: { _id: "$dept", total: { $sum: "$amount" } } }
])
```

🔷 Usefulness

- Data analysis
- Reporting
- Business intelligence
- Real-time analytics
- Reduces need for client-side computation

It makes MongoDB powerful for analytical queries.

🔷 Why Aggregation is Useful

✅ 1️⃣ Data Analysis

- Generate reports
- Calculate totals, averages

✅ 2️⃣ Business Intelligence

- Revenue by region
- Monthly trends

✅ 3️⃣ **Server-Side Processing**

- Reduces client-side computation
- Faster and more efficient

✅ 4️⃣ **Real-Time Analytics**

- Works efficiently on large datasets

Q5 (c) — Why is Covered Query Important in MongoDB?

🔷 What is a Covered Query?

A **covered query** is a query where:

- All fields used in the query (filter + projection)
- Are present in an index
- So MongoDB does **NOT need to read the actual documents**

The query is answered **entirely from the index**.

🔷 Normal Query vs Covered Query

❌ Normal Query:

1. MongoDB checks index to find matching documents.
2. Then it reads actual documents from disk.
3. Extracts requested fields.

This involves disk I/O → slower.

✅ Covered Query:

1. MongoDB checks index.
2. All required fields are already in the index.

3. No need to access collection documents.

No extra disk access → faster.

🔷 Conditions for Covered Query

A query is covered if:

1. The filter fields are indexed.
2. The returned (projected) fields are also in the index.
3. _id is excluded if not part of index.

Example:

```
db.users.createIndex({ name: 1, age: 1 })
```

Query:

```
db.users.find(
  { name: "John" },
  { name: 1, age: 1, _id: 0 }
)
```

Here:

- name and age are indexed.
- _id excluded.
- Query is fully covered.

🔷 Why Covered Queries Are Important

✅ 1️⃣ Faster Performance

- No document fetch.
- Reduces disk I/O.

✅ 2️⃣ Lower Latency

- Index access is much faster.

✅ 3️⃣ Reduced Memory Usage

- Only index pages are accessed.

✅ 4 Better for Read-Heavy Workloads

- Improves scalability.

Final Structured Summary

(a)

MongoDB scales horizontally using sharding. Data is partitioned using shard keys and distributed across shards, managed via config servers and mongos routers.

(b)

Aggregation in MongoDB uses the aggregation pipeline to process and transform documents for analytical computations like sum, average, grouping, and sorting. It enables efficient server-side analytics.

(c)

Covered queries improve performance by retrieving results directly from indexes without accessing actual documents, reducing disk I/O and latency.

Q6 (a) — Signature-Based vs. Anomaly-Based IDS (5 Marks)

🔷 1 Signature-Based IDS

Definition:

A signature-based IDS detects intrusions by comparing network/system activity against a database of known attack patterns (signatures).

How it works:

- Maintains a database of known attack signatures.
- Monitors traffic.
- Raises alert if pattern matches a known signature.

Advantages:

- High accuracy for known attacks.
- Low false positive rate.
- Easy to understand and implement.

Disadvantages:

- Cannot detect new or zero-day attacks.
- Requires frequent updates of signature database.

🔷 2 Anomaly-Based IDS

Definition:

An anomaly-based IDS detects intrusions by identifying deviations from normal behavior.

How it works:

- Builds a model of normal user/system behavior.
- Any significant deviation is flagged as suspicious.

Advantages:

- Can detect unknown and zero-day attacks.
- Identifies unusual behavior patterns.

Disadvantages:

- Higher false positive rate.
- Requires training phase.
- Normal behavior changes over time.

🔷 Key Differences

Aspect	Signature-Based	Anomaly-Based
Detects new attacks	No	Yes
False positives	Low	Higher
Requires updates	Signature updates	Model retraining

Q6 (b) — Ways to Improve the Efficiency of Anomaly-Based IDS (10 Marks)

Anomaly-Based IDS detects intrusions by identifying deviations from normal behavior. Its main problem is high false positives and performance overhead. The following methods improve its efficiency:

🔷 1️⃣ Use High-Quality Training Data

- Train IDS using clean, attack-free normal traffic.
- Ensure the training dataset represents real network behavior.
- Avoid contaminated training data.

✅ Improves accuracy and reduces false positives.

🔷 2️⃣ Proper Feature Selection

- Select only relevant features (e.g., packet size, login frequency).
- Remove redundant or noisy attributes.
- Use dimensionality reduction techniques (e.g., PCA).

✅ Reduces computational overhead and increases detection accuracy.

🔷 3️⃣ Adaptive / Online Learning

- Continuously update the normal behavior model.
- Adjust to changes in user activity or traffic patterns.
- Prevent outdated behavior models.

✅ Reduces long-term false positives.

🔷 4️⃣ Threshold Optimization

- Tune detection thresholds carefully.
- Avoid overly sensitive settings.
- Balance between false positives and false negatives.

✅ Improves detection precision.

🔷 5️⃣ Use Hybrid IDS Approach

- Combine anomaly-based and signature-based detection.
- Use signature confirmation for suspected anomalies.

✅ Reduces false alarms and improves reliability.

🔷 6️⃣ Apply Machine Learning Techniques

- Use clustering (K-means), SVM, Neural Networks.
- Improve behavioral modeling and classification.

✅ More intelligent detection of complex attacks.

🔷 7️⃣ Feedback Mechanism

- Allow administrator feedback on alerts.
- Retrain model using corrected classifications.

✅ Improves IDS accuracy over time.

Q6 (c) — Is There Anything Interesting About False Positives? (5 Marks)

🔷 What is a False Positive?

A **false positive** occurs when an IDS incorrectly classifies normal, legitimate activity as an attack.

🔷 Interesting / Important Points About False Positives

1️⃣ Major Practical Problem

In real deployments, excessive false positives are often a bigger problem than missed attacks. Too many alerts reduce usability.

2️⃣ Alert Fatigue

If an IDS generates too many false alarms:

- Administrators become desensitized.
- Alerts may be ignored.
- Real attacks may go unnoticed.

---

3 Trade-off with False Negatives

There is always a trade-off:

- Lower detection threshold → more false positives.
- Higher threshold → more false negatives.

Improving one often worsens the other.

---

4 Higher in Anomaly-Based IDS

Anomaly-based IDS tends to generate more false positives because:

- Normal behavior changes over time.
- Rare but legitimate actions may appear suspicious.

---

5 Cost and Resource Impact

False positives:

- Waste analyst time.
- Increase operational cost.
- Reduce trust in the IDS system.

Q6 (d) — Issues in Testing Intrusion Detection Systems (5 Marks)

Testing IDS is challenging due to several practical and technical issues.

◆ 1 Lack of Realistic and Labeled Datasets

- Real network attack data is hard to obtain.
- Many publicly available datasets (e.g., KDD’99) are outdated.
- Modern attacks may not be represented.

This affects reliability of testing results.

🔷 2️⃣ Class Imbalance Problem

- In real networks, attacks are rare compared to normal traffic.
- Testing datasets often have imbalanced classes.
- High accuracy may be misleading.

Example: 99% normal traffic → system may appear accurate without detecting attacks.

🔷 3️⃣ Evolving Attack Patterns

- Attack methods constantly change.
- Hard to simulate all possible or zero-day attacks.
- Testing may not reflect future threats.

🔷 4️⃣ Performance Metrics Selection

IDS performance depends on:

- Detection rate
- False positive rate
- False negative rate
- Precision / Recall

Choosing correct evaluation metrics is important.

🔷 5️⃣ Environment Dependency

- IDS performance varies by network environment.
- A system tested in one network may not perform similarly in another.
- Results may not generalize well.