DBS_MidSem2025

Exported on: 2/22/2026, 7:43:05 PM

User

Explain b) (i)

Gemini

To prepare an **Access Control List (ACL)** for the print server, we treat the server as a single object and list which users (subjects) have permission to perform specific methods (operations).

In an ACL-based system, the object maintains a list of users and their associated rights. Based on the scenario provided, here is the breakdown for each user:

Access Control List (ACL) for Print Server

Subject (User)	Allowed Operations (Rights)
Alice (Manager)	`print`, `queue`, `topQueue`, `start`, `stop`, `restart`, `status`, `readConfig`, `setConfig` (All operations)
Bob (Technician)	`start`, `stop`, `restart`, `status`, `readConfig`, `setConfig`
Cecilia (Power User)	`print`, `queue`, `topQueue`, `restart`
David (Ordinary User)	`print`, `queue`
Erica (Ordinary User)	`print`, `queue`
Fred (Ordinary User)	`print`, `queue`
George (Ordinary User)	`print`, `queue`

Export to Sheets

Key Logic used for the ACL

- **Alice:** As the manager, she is explicitly granted rights to perform **all** operations listed in the scenario.
- **Bob:** His role as a technician limits him to system-level maintenance (`start`, `stop`, `restart`) and configuration management (`status`, `readConfig`, `setConfig`). He does not have `print` or `queue` rights.
- **Cecilia:** Her "Power User" status gives her standard user rights (`print`, `queue`) plus administrative queue management (`topQueue`) and the ability to `restart` the server.

- **Ordinary Users:** David, Erica, Fred, and George are grouped together with the most restrictive access: only being able to `print` files and view the `queue`.

Would you like me to represent this as a **Capability List** (where each user holds their own set of permissions) to see the difference from an ACL?

User

Answer (ii)

Gemini

To transition from an **Access Control List (ACL)** to **Role-Based Access Control (RBAC)**, we first identify logical groups of permissions (roles) based on the job functions described in the scenario.

1. Identification of Roles and Permissions

We can define four distinct roles based on the operational requirements:

- **Ordinary User (OU):** Can perform basic printing and queue viewing tasks.
  - Permissions: `print`, `queue`.
- **Power User (PU):** Inherits Ordinary User capabilities and manages queue priorities or system resets.
  - Permissions: `print`, `queue`, `topQueue`, `restart`.
- **Service Technician (ST):** Focuses on system maintenance and configuration.
  - Permissions: `start`, `stop`, `restart`, `status`, `readConfig`, `setConfig`.
- **Manager (MG):** Requires full administrative control over every function.
  - Permissions: All (`print`, `queue`, `topQueue`, `start`, `stop`, `restart`, `status`, `readConfig`, `setConfig`).

---

2. Role Hierarchy

A hierarchy allows higher-level roles to automatically inherit the permissions of lower-level roles, simplifying management.

- **Ordinary User** is the base role.
- **Power User** inherits from **Ordinary User**.
- **Manager** inherits from both **Power User** and **Service Technician**, giving the Manager the union of all possible permissions.

> **Note:** The **Service Technician** role is often kept separate from the **Power User** hierarchy in this scenario because their duties (hardware/config) are distinct from standard user printing tasks, though the **Manager** ultimately inherits both.

---

3. Permission Mapping

Role	Inherits From	Direct Permissions	Total Effective Permissions
Ordinary User	None	`print`, `queue`	`print`, `queue`
Power User	Ordinary User	`topQueue`, `restart`	`print`, `queue`, `topQueue`, `restart`

Role	Inherits From	Direct Permissions	Total Effective Permissions
Service Technician	None	`start`, `stop`, `restart`, `status`, `readConfig`, `setConfig`	`start`, `stop`, `restart`, `status`, `readConfig`, `setConfig`
Manager	Power User, Service Technician	None (inherits all)	All operations

Export to Sheets

Would you like me to show how the users (Alice, Bob, etc.) would be mapped to these roles for part **(iii)**?

User

Answer part (iii), (i have provided the image of question above)

Gemini

To answer part **(iii)**, we look at how the **Role-Based Access Control (RBAC)** system we defined in part **(ii)** handles organizational changes like departures, role changes, and new hires.

In an RBAC system, instead of editing permissions for every individual, we simply update their **User-to-Role assignments**.

Updated User-to-Role Assignments

User	Action taken	Assigned Role	Effective Permissions
Alice	No change	**Manager**	All operations
Bob	**Removed**	None	No access (Left the company)
Cecilia	No change	**Power User**	`print`, `queue`, `topQueue`, `restart`
David	No change	**Ordinary User**	`print`, `queue`
Erica	No change	**Ordinary User**	`print`, `queue`
Fred	No change	**Ordinary User**	`print`, `queue`
George	**Promoted**	**Service Technician**	`start`, `stop`, `restart`, `status`, `readConfig`, `setConfig`
Henry	**New Hire**	**Ordinary User**	`print`, `queue`
Ida	**New Hire**	**Power User**	`print`, `queue`, `topQueue`, `restart`

Export to Sheets

Key Changes Explained

- **George's Promotion:** In a standard ACL, we would have had to manually add six new permissions to George and remove his old ones. In RBAC, we simply change his label from "Ordinary User" to **"Service Technician"**.
- **Handling New Hires (Henry & Ida):** Henry is instantly granted the correct baseline access by being assigned the **Ordinary User** role. Ida is granted the same complex set of privileges as Cecilia simply by being assigned the **Power User** role, ensuring consistency and reducing administrative error.

- **Bob's Departure:** By removing Bob's user account from the system, all associated **Service Technician** rights are revoked instantly.

Would you like me to explain how this change would be significantly more difficult if we were still using the **Access Control List (ACL)** from part (i)?

User

Answer (b) Signature vs Anomaly based IDS

Gemini

In the context of Intrusion Detection Systems (IDS), the primary difference between signature-based and anomaly-based detection lies in how they identify potential threats.

Signature-based vs. Anomaly-based IDS

Feature	Signature-Based IDS	Anomaly-Based IDS
Method	Compares traffic against a database of known attack patterns (signatures).	Compares current activity against a baseline of "normal" behavior.
New Threats	Effectively blind to Zero-Day attacks or new variants until the database is updated.	Capable of detecting new, unknown, or Zero-Day attacks.
Accuracy	High accuracy for known attacks with very few false positives.	Higher rate of false positives as legitimate but unusual behavior may be flagged.
Efficiency	Fast and efficient for high-volume traffic using pre-defined patterns.	Computationally expensive as it must constantly build and update behavioral profiles.

Export to Sheets

Ways to Improve Anomaly-Based IDS Efficiency

To improve the performance and accuracy of an anomaly-based system, consider the following methods:

1. **Dynamic Baseline Updates:** Regularly retrain the model on the most recent "clean" network traffic to account for evolving legitimate usage patterns (e.g., seasonal changes in traffic or new software updates).
2. **Multivariate Analysis:** Instead of monitoring single variables (like just packet size), correlate multiple data points (e.g., source IP, time of day, and protocol type) to reduce noise and sharpen detection.
3. **Machine Learning Refinement:** Utilize advanced algorithms that can differentiate between a "rare but benign" event and a malicious one more effectively than simple threshold-based triggers.

The "Interesting" Nature of False Positives

What makes false positives interesting in Anomaly-Based IDS—often referred to as the **Base Rate Fallacy**—is that even a system with a very low false positive rate (e.g., 1%) can become unusable if the total volume of traffic is high.

Because actual attacks are extremely rare compared to the billions of legitimate packets, a 1% error rate results in thousands of "cries for help" for every one real attack. This can lead to **alert fatigue**, where security teams eventually ignore the IDS because it "cries wolf" too often, potentially allowing a real intrusion to slip through unnoticed.

Would you like me to go into more detail on how **Zero-Day** vulnerabilities specifically favor one of these methods over the other?

User

Answer Q2) (a)

Gemini

To define the views for the **Employees** relation (comprising fields `ename`, `dept`, and `salary`), you can use the following SQL `CREATE VIEW` statements:

a. View Definition Statements

1. EmployeeNames View

This view is designed for basic identification, restricting access to only the employee names.

SQL

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

2. DeptInfo View

This view provides aggregated data for authorization reasons, hiding individual salary details while showing departmental averages.

SQL

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

Key Details of the Definitions

- **EmployeeNames**: Uses a simple projection to isolate the `ename` attribute.
- **DeptInfo**: Uses the `GROUP BY` clause to calculate the `avgsalary` for each unique `dept` found in the base table.

Would you like to move on to part **(b)** and discuss which specific privileges should be granted to access these department salaries?

User

Answer (b)

Gemini

To allow a user to see **only** the average department salaries for the "Toy" and "CS" departments, you should grant **SELECT** privileges on the specific view that contains that aggregated data, rather than the base table.

b. Privileges for Average Salaries

The user should be granted the **SELECT** privilege on the **DeptInfo** view. However, since they only need access to specific departments, the authorization should be restricted.

Depending on the database system, this can be handled in two ways:

1. **Grant on the View with a WHERE clause:** If you want to be precise, you would create a restricted view or ensure the user's query is filtered.
2. **SQL Command:**

   SQL

   ```sql
   GRANT SELECT ON DeptInfo TO [username];
   ```

Why this is the correct approach:

- **Security:** By granting access to the **DeptInfo** view instead of the **Employees** table, you ensure the user cannot see individual salaries or other sensitive fields like names associated with those salaries.
- **Principle of Least Privilege:** The user only gets the "average" calculation they need, and nothing more.

Would you like me to provide the SQL commands for part **(c)** to authorize your secretary's specific administrative tasks?

User

Answer part (c)

Gemini

To authorize your secretary for the tasks specified, you need to grant privileges that allow for record deletion, data retrieval from the views, and potentially data retrieval from the base table for specific checks.

c. Privileges to Grant to the Secretary

The secretary requires the following privileges:

- **DELETE on Employees:** To allow the secretary to "fire" people (remove their records from the system).
- **SELECT on EmployeeNames:** To allow the secretary to check who is currently an employee by viewing names.
- **SELECT on DeptInfo:** To allow the secretary to check the average department salaries.

The SQL Commands

SQL

```sql
-- Grant permission to remove records (firing)
GRANT DELETE ON Employees TO Secretary;

-- Grant permission to view employee names
GRANT SELECT ON EmployeeNames TO Secretary;

-- Grant permission to view average department salaries
GRANT SELECT ON DeptInfo TO Secretary;
```

Summary of Granted Rights

Task	Object	Privilege
Fire people	`Employees` (Table)	`DELETE`
Check who is an employee	`EmployeeNames` (View)	`SELECT`
Check average salaries	`DeptInfo` (View)	`SELECT`

Export to Sheets

Would you like to proceed to part **(d)** to analyze whether these privileges accidentally allow the secretary to see individual salaries?

User

Answer part (d)

Gemini

Based on the privileges granted in part (c), here is the analysis of whether your secretary can determine individual salaries:

Does the answer to (c) ensure privacy?

No, it does not strictly ensure privacy in all cases. While you did not grant `SELECT` access to the `salary` column of the `Employees` table, the secretary can potentially deduce individual salaries through the **DeptInfo** view depending on the number of employees in a department.

Specific Scenarios

- **Can the secretary find out salaries of *some* individuals? Yes.** If a department has **only one employee**, the average salary shown in the `DeptInfo` view is identical to that individual's actual salary. For example, if "Toy" only has one employee named "Woody," the `avgsalary` for "Toy" is Woody's exact salary.
- **Can the secretary find out the salary of *any* individual? No.** If a department has many employees, the secretary only sees the collective average. Without access to the base `Employees` table or a more granular view, they cannot distinguish one person's pay from another's within a large group.

The "Inference Attack" Risk

The secretary could also use their **DELETE** privilege to perform an inference attack:

1. Check the `avgsalary` for a department.
2. **Fire** (Delete) one person from that department.
3. Check the new `avgsalary`.
4. Use the difference between the two averages and the employee count to calculate the exact salary of the person who was fired.

Would you like to see the **SQL command** for part **(e)** to allow the secretary to delegate read access to others?

User

Answer part (e)

Gemini

To allow your secretary to grant other people the authority to read the `EmployeeNames` view, you must grant the privilege using the **WITH GRANT OPTION**. This clause enables the recipient to delegate that specific privilege to other users.

e. The SQL Command

SQL

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

Understanding the Command

- **GRANT SELECT**: This gives the secretary the right to view the data within the `EmployeeNames` view.
- **ON EmployeeNames**: Specifies the object (the view) the permission applies to.
- **TO Secretary**: Specifies the user receiving the permission.
- **WITH GRANT OPTION**: This is the crucial addition that allows the secretary to then execute `GRANT SELECT ON EmployeeNames TO [SomeOtherUser]`.

Security Implications

By providing the **GRANT OPTION**, you are essentially trusting the secretary to act as a sub-administrator for that specific view. They can now extend read access to anyone in the company without needing your further intervention.

Would you like to move on to part **(f)** to see the command for granting **INSERT** privileges and what that means for the secretary's power?

User

Answer part (f)

Gemini

To give your secretary the right to insert tuples into the `EmployeeNames` view, you would use the following SQL command.

f. The SQL Command and Privilege Description

SQL

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

Description of Privileges

After this command is executed, the secretary has the following set of privileges:

- **INSERT on EmployeeNames:** The secretary can now add new rows to the `EmployeeNames` view. In many database systems, an insert into a view is translated into an insert into the underlying base table (`Employees`). Since `EmployeeNames` only contains the `ename` column, any inserted rows would likely have `NULL` values for `dept` and `salary` in the base table, unless default values are defined.
- **Existing Privileges:** The secretary retains all previously granted rights, including:
    - **DELETE on Employees:** The power to remove records.
    - **SELECT on EmployeeNames (with Grant Option):** The power to read the view and delegate that read access to others.
    - **SELECT on DeptInfo:** The power to view departmental average salaries.

Secretary's Capability with New Views

The secretary has also defined two internal views: `AtoRNames` and `HowManyNames`. Because these are derived from `EmployeeNames` (which the secretary has `SELECT` access to), the secretary can successfully query these to filter names or count them.

Would you like to move on to part **(g)** to see what happens to Todd's access if the secretary's privileges are revoked?

User

Answer part (g)

Gemini

In a standard SQL environment, when a user's privileges are revoked, the revocation follows a **cascading** effect for any privileges that were granted via the `WITH GRANT OPTION`.

g. Impact on Todd's Privileges

When you revoke the secretary’s privileges, **Todd's privileges are also automatically revoked**.

Why this happens:

- **Chain of Authorization:** Todd received his access to the `EmployeeNames` relation directly from the secretary.
- **Dependency:** Todd’s permission is dependent on the secretary’s authority to grant it. Once the secretary no longer has the `GRANT OPTION` (or the privilege itself), the "link" in the chain is broken.
- **Cascading Revoke:** Most relational database management systems (RDBMS) use a `CASCADE` behavior by default during a `REVOKE` command. This ensures that no "orphaned" privileges remain in the system that weren't authorized by a currently valid source.

The Resulting State

- **Secretary:** Can no longer read, insert, or delete data, and cannot grant access to others.
- **Todd:** Loses all access to the `EmployeeNames` relation immediately. If Todd tries to query the table, he will receive an "insufficient privileges" error.

Would you like me to wrap up with the final part **(h)** regarding a view update that cannot be implemented?

User

Answer part (h), aslo this is not the last part, there are 3 more parts i will provide those after this answer

Gemini

In database management, a view update cannot be implemented if the change to the view cannot be translated into a unique, unambiguous change to the underlying base table.

h. Example of a Non-Updatable View

An example of a view update that cannot be implemented on the provided schema is an attempt to update the `avgsalary` column in the `DeptInfo` view.

The Attempted Command:

SQL

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

Why it cannot be implemented:

- **Ambiguity in the Base Table:** The `avgsalary` field in the view is an **aggregate value** calculated using the `AVG()` function. There is no single "average" column in the base `Employees` table to update.
- **Infinite Possibilities:** To change a department's average to 90,000, the database wouldn't know how to adjust the individual salaries of the employees in that department. It could increase one person's salary by a lot, increase everyone's by a little, or even decrease some while increasing others.
- **Formal Restriction:** Most SQL standards and RDBMS engines strictly prohibit updates on views that use **aggregate functions** (`SUM`, `AVG`, `COUNT`), **`GROUP BY`** clauses, or **`DISTINCT`** keywords because the mapping back to specific source tuples is not one-to-one.

I'm ready for the next three parts whenever you'd like to upload them! Would you like me to analyze the next image?

User

Answer (i)

Gemini

To authorize your boss, Joe, according to the requirements in part **(i)**, you need to provide full access to the base tables and views while enabling him to delegate those powers to others.

i. Authorization for Joe

The following SQL statements grant Joe the necessary permissions with the ability to delegate:

SQL

```sql
-- Grant read and modify access on the base table with delegation rights
GRANT SELECT, INSERT, UPDATE, DELETE ON Employees TO Joe WITH GRANT OPTION;

-- Grant read access on the EmployeeNames view with delegation rights
GRANT SELECT ON EmployeeNames TO Joe WITH GRANT OPTION;
```

Can Joe read the `DeptInfo` view?

No. In standard SQL authorization, privileges are not automatically inherited across different objects. Even though Joe has full access to the underlying `Employees` table, he does not automatically gain access to a view created by someone else (the DBA) unless it is explicitly granted to him. To allow him to see it, you would need an additional statement: `GRANT SELECT ON DeptInfo TO Joe;`

Logic behind the SQL

- **Permissions:** Joe needs to "read and modify" the `Employees` relation, which translates to `SELECT` (read) and `INSERT/UPDATE/DELETE` (modify).
- **Delegation:** The `WITH GRANT OPTION` is essential here because Joe is "too far up the management hierarchy" to do the work himself; this allows him to pass these rights to his own assistants.

---

j. Revoking Mike but not Joe

No, you cannot do this easily in standard SQL using a single command. In SQL, the `REVOKE` command typically follows a cascading pattern. If you revoke a privilege from the person who granted it (Joe), all privileges that Joe granted to others (like his secretary Mike) are automatically revoked as well.

However, to revoke **only** Mike's access while keeping Joe's intact, you would execute:

SQL

```sql
REVOKE SELECT ON Employees FROM Mike;
```

This works because you are the owner (DBA) and can revoke rights from anyone in the chain. Since you are not revoking from Joe, Joe's rights—and his ability to grant them to someone else in the future—remain unaffected.

---

k. Final Revocation Analysis

1. The REVOKE statement to execute: To remove Mike and Susan's access while potentially "annoying" Joe by stripping his delegation power, you would execute:

SQL

```sql
REVOKE SELECT ON Employees FROM Joe CASCADE;
```

(Note: If you only want to stop the delegation but let Joe keep reading, you would use `REVOKE GRANT OPTION FOR SELECT ON Employees FROM Joe CASCADE;` *)*

2. What rights does Joe have on `Employees` after this? If you used the `CASCADE` command above, Joe loses his `SELECT` privilege entirely. If you only revoked the `GRANT OPTION`, he can still read the data but can no longer authorize others.

3. What views are dropped as a consequence?

- `All-Names`: This view is dropped because it was defined by Joe using `EmployeeNames`. Since his access to the underlying source is revoked, the dependent view becomes invalid.
- `StaffNames`: This relation is **not** dropped. The prompt describes it as "another relation," implying it is a base table Joe created. Revoking access to your data does not delete Joe's own independent tables.

Would you like me to clarify the difference between **RESTRICT** and **CASCADE** in these final revocation steps?