Laboratory Work 7: SQL Views and Roles

Objective

To understand and practice creating and managing SQL views (regular and materialized), implementing role-based access control, and learning how to secure and organize database objects effectively.

Prerequisites

- Completion of Laboratory Work 6 (SQL JOINs)
- Basic SQL knowledge (SELECT, INSERT, UPDATE, DELETE)
- Understanding of database security concepts
- Access to a PostgreSQL database system

Theoretical Background

What are Views?

A view is a named query that provides another way to present data in database tables. Views are virtual tables based on one or more base tables. When you create a view, you essentially create a query and assign it a name, making it useful for wrapping commonly used complex queries.

Types of Views

1. **Regular Views**: Virtual tables that don't store data physically
2. **Materialized Views**: Views that cache query results physically for faster access
3. **Updatable Views**: Views that allow INSERT, UPDATE, and DELETE operations

What are Roles?

PostgreSQL manages database access permissions using roles. A role can be either a database user or a group of database users, depending on how it's configured. Roles can own database objects and control access to them.

Part 1: Database Setup (Use Lab 6 Tables)

You will use the same database setup from Laboratory Work 6 (employees, departments, and projects tables). If you haven't created them yet, run the setup scripts from Lab 6 Part 1.

Part 2: Creating Basic Views

Exercise 2.1: Simple View Creation

Create a view that shows employee names with their department information (name, salary, department name, and location). Include only employees who are assigned to a department.

Test your view:

sql
```
SELECT * FROM employee_details;
```
Question: How many rows are returned? Why doesn't Tom Brown appear?

Exercise 2.2: View with Aggregation

Create a view named `dept_statistics` showing department statistics including department name, employee count, average salary, maximum salary, and minimum salary. Include all departments even if they have no employees.

Test your view:

sql
```
SELECT * FROM dept_statistics
ORDER BY employee_count DESC;
```

Exercise 2.3: View with Multiple Joins

Create a view named `project_overview` showing complete project information including project name, budget, department name, location, and the count of employees in that department (team size).

Exercise 2.4: View with Filtering

Create a view named `high_earners` for employees with salaries greater than $55,000, showing employee name, salary, and department name.

Question: What happens when you query this view? Can you see all high-earning employees?

Part 3: Modifying and Managing Views

Exercise 3.1: Replace a View

Modify the `employee_details` view to include a salary grade column that shows:

- 'High' for salary > 60000
- 'Medium' for salary > 50000
- 'Standard' for all others

Exercise 3.2: Rename a View

Rename the `high_earners` view to `top_performers`.

Verify:

sql
SELECT * FROM top_performers;

Exercise 3.3: Drop a View

Create a temporary view named `temp_view` showing employees with salary less than $50,000, then drop it.

Part 4: Updatable Views

Exercise 4.1: Create an Updatable View

Create a simple updatable view named `employee_salaries` on the employees table showing emp_id, emp_name, dept_id, and salary.

Exercise 4.2: Update Through a View

Update John Smith's salary to $52,000 through the `employee_salaries` view.

Verify the update:

sql
SELECT * FROM employees WHERE emp_name = 'John Smith';
Question: Did the underlying table get updated?

Exercise 4.3: Insert Through a View

Try inserting a new employee named 'Alice Johnson' (emp_id: 6, dept_id: 102, salary: 58000) through the view.

Question: Was the insert successful? Check the employees table.

Exercise 4.4: View with CHECK OPTION

Create a view named `it_employees` showing only IT department employees (dept_id = 101) with LOCAL CHECK OPTION.

Try to insert an employee from a different department:

sql
```
-- This should fail
INSERT INTO it_employees (emp_id, emp_name, dept_id, salary)
VALUES (7, 'Bob Wilson', 103, 60000);
```
Question: What error message do you receive? Why?

Part 5: Materialized Views

Exercise 5.1: Create a Materialized View

Create a materialized view named `dept_summary_mv` that shows for each department:

- dept_id, dept_name
- Total number of employees
- Total salaries (use 0 if no employees)
- Total number of projects
- Total project budget (use 0 if no projects)

Load data immediately (WITH DATA).

Query the materialized view:

sql
```
SELECT * FROM dept_summary_mv ORDER BY total_employees DESC;
```

Exercise 5.2: Refresh Materialized View

Insert a new employee named 'Charlie Brown' (emp_id: 8, dept_id: 101, salary: 54000). Query the materialized view before and after refreshing it.

Question: What's the difference before and after refresh?

Exercise 5.3: Concurrent Refresh

Create a unique index on `dept_summary_mv` using dept_id, then refresh the materialized view concurrently.

Question: What's the advantage of CONCURRENTLY option?

Exercise 5.4: Materialized View with NO DATA

Create a materialized view named `project_stats_mv` showing project name, budget, department name, and count of assigned employees. Create it WITH NO DATA.

Try to query it:

sql
```
SELECT * FROM project_stats_mv;
```
Question: What error do you get? How do you fix it?

Part 6: Database Roles

Exercise 6.1: Create Basic Roles

Create the following roles:

- A basic role named `analyst` (no login)
- A role named `data_viewer` with LOGIN and password 'viewer123'
- A user named `report_user` with password 'report456'

View all roles:

sql
```
SELECT rolname FROM pg_roles WHERE rolname NOT LIKE 'pg_%';
```

Exercise 6.2: Role with Specific Attributes

Create the following roles with their respective attributes:

- `db_creator` - can create databases, has login, password 'creator789'
- `user_manager` - can create roles, has login, password 'manager101'
- `admin_user` - superuser with login, password 'admin999' (use carefully!)

Exercise 6.3: Grant Privileges to Roles

Grant the following privileges:

- SELECT privilege on employees, departments, and projects tables to `analyst`
- ALL PRIVILEGES on `employee_details` view to `data_viewer`
- SELECT and INSERT privileges on employees table to `report_user`

Exercise 6.4: Create Group Roles

Create group roles for different teams and assign members:

1. Create group roles: `hr_team`, `finance_team`, `it_team`
2. Create individual users: `hr_user1` (password: 'hr001'), `hr_user2` (password: 'hr002'), `finance_user1` (password: 'fin001')
3. Assign hr_user1 and hr_user2 to hr_team
4. Assign finance_user1 to finance_team
5. Grant SELECT and UPDATE on employees to hr_team
6. Grant SELECT on dept_statistics to finance_team

Exercise 6.5: Revoke Privileges

Perform the following revocations:

- Revoke UPDATE privilege on employees from hr_team
- Revoke hr_team membership from hr_user2
- Revoke ALL PRIVILEGES on employee_details from data_viewer

Exercise 6.6: Modify Role Attributes

Make the following modifications:

- Add LOGIN and password 'analyst123' to the analyst role
- Make user_manager a SUPERUSER
- Remove password from analyst role (set to NULL)
- Add a connection limit of 5 to data_viewer

Part 7: Advanced Role Management

Exercise 7.1: Role Hierarchies

Create a role hierarchy:

1. Create a parent role `read_only` and grant SELECT on all tables in schema public
2. Create child roles: `junior_analyst` (password: 'junior123') and `senior_analyst` (password: 'senior123')
3. Grant read_only membership to both junior and senior analysts
4. Grant additional INSERT and UPDATE privileges on employees to senior_analyst only

Exercise 7.2: Object Ownership

Transfer ownership of database objects:

1. Create a new role `project_manager` with LOGIN and password 'pm123'
2. Transfer ownership of the `dept_statistics` view to project_manager
3. Transfer ownership of the `projects` table to project_manager

Check ownership:

sql
```
SELECT tablename, tableowner
FROM pg_tables
WHERE schemaname = 'public';
```

Exercise 7.3: Reassign and Drop Roles

Properly remove a role by reassigning its objects:

1. Create a role named `temp_owner` with LOGIN
2. Create a table `temp_table` with an id column (INT)
3. Transfer ownership of temp_table to temp_owner
4. Reassign all objects owned by temp_owner to postgres
5. Drop all objects owned by temp_owner
6. Drop the temp_owner role

Exercise 7.4: Row-Level Security with Views

Create role-specific views that restrict data access:

1. Create `hr_employee_view` showing employees only from HR department (dept_id = 102)
2. Grant SELECT on this view to hr_team
3. Create `finance_employee_view` showing only emp_id, emp_name, and salary for all employees
4. Grant SELECT on this view to finance_team

Part 8: Practical Scenarios

Exercise 8.1: Department Dashboard View

Create a comprehensive dashboard view named `dept_dashboard` for department managers showing:

- Department name and location
- Employee count
- Average salary (rounded to 2 decimals)
- Number of active projects
- Total project budget
- Budget per employee (rounded to 2 decimals, handle division by zero)

Exercise 8.2: Audit View

Create an audit view for high-value projects:

1. First, add a `created_date` column to projects table with default CURRENT_TIMESTAMP
2. Create view `high_budget_projects` showing projects with budget > 75000
3. Include project name, budget, department name, created date
4. Add an approval_status column that shows:
    - 'Critical Review Required' for budget > 150000
    - 'Management Approval Needed' for budget > 100000
    - 'Standard Process' otherwise

Exercise 8.3: Create Access Control System

Set up a complete access control system with multiple role levels:

Level 1 - Viewer Role:

- Create role `viewer_role`
- Grant SELECT on all tables and views in schema public

Level 2 - Entry Role:

- Create role `entry_role`
- Grant viewer_role membership
- Grant INSERT on employees and projects

Level 3 - Analyst Role:

- Create role `analyst_role`
- Grant entry_role membership
- Grant UPDATE on employees and projects

Level 4 - Manager Role:

- Create role `manager_role`
- Grant analyst_role membership
- Grant DELETE on employees and projects

Create Users:

- Create users: alice (password: 'alice123'), bob (password: 'bob123'), charlie (password: 'charlie123')
- Assign alice to viewer_role, bob to analyst_role, charlie to manager_role