Lab 6

Views, Roles, String, Sets & Dates

CSE 4308

DATABASE MANAGEMENT SYSTEMS LAB

Prepared by: Md. Tariquzzaman
Academic Year: 2024-2025

Contents

Lab Workflow

Repeat this subsection for every single lab. If anything in this section fails in IUT lab PCs, check the commands given at the end of the lab manual.

1. Connect to the internet by running nmtui in terminal, add a ethernet connection, download the files from the given local server url.

2. Stay connected to the ethernet during the whole lab period.

3. After finishing the lab, and giving viva, you can connect back to DSL, and submit the solutions to your google classroom. Make sure you remove your DSL connection after that.

4. **Create Your Role and Database**

   Inside the PostgreSQL shell, create your own role (user) and database.

   1. Login to PostgreSQL as ‘postgres‘:

   Log in again to PostgreSQL using ‘postgres‘:

```
sudo -i -u postgres psql
```

   2. Create Your Role and Database

   Important: If the role or database already exists, we must drop them first to avoid conflicts.

```
-- Terminate active connections to the database
SELECT pg_terminate_backend(pid)
FROM pg_stat_activity
WHERE datname = 'cse4308_12345';

-- Drop database if it exists
DROP DATABASE IF EXISTS cse4308_12345;

-- Drop role if it exists
DROP ROLE IF EXISTS alice;

CREATE ROLE alice LOGIN;
ALTER ROLE alice WITH PASSWORD 'alice123';

CREATE DATABASE cse4308_12345 OWNER alice;

GRANT ALL PRIVILEGES ON DATABASE cse4308_12345 TO alice;
```

   This script will:

   - Terminate active sessions to allow safe dropping.
   - Drop the existing `cse4308_12345` database and `alice` role if present.
   - Create a new `alice` role with a password.
   - Create the database `cse4308_12345` owned by `alice`.
   - Grant all permissions to `alice`.

3. Exit PostgreSQL:

```
\q
```

5. **Connect Using Your Newly Created User**

Now connect to your own database as the ‘alice‘ user.

1. Connect to Your Database:

Use the following command to log in as the ‘alice‘ user:

```
psql -U alice -d cse4308_12345
```

2. Verify the Connection:

Verify that you are connected to the correct database with the following command:

```
\conninfo
```

Expected output:

```
You are connected to database "cse4308_12345" as user "alice".
```

3. Exit PostgreSQL:

```
\q
```

6. **Load the Lab SQL File (MUST be run outside psql)**

Load the SQL file that sets up the tables and inserts initial data.

1. Load the SQL File:

Run the following command to load the lab SQL file:

```
psql -U alice -d cse4308_12345 -f /home/cse/Downloads/lab4.sql
```

This will: - Drop old tables (if they exist). - Create new tables. - Insert initial rows of data.

7. **Reconnect and Verify Tables**

After loading the SQL file, reconnect to PostgreSQL to verify that the tables have been created.

1. Reconnect to Your Database:

```
psql -U alice -d cse4308_12345
```

2. Check Tables:

Verify the tables have been created by listing them:

```
\dt
```

3. View Table Data:

Verify the data by running the following SQL queries:

```
SELECT * FROM <table_name>
```

8. Sample Solution file: Open a text editor and save it in the appropriate submission name format
`fullid_CSE4308_L4_T1.sql`. You will use this text editor to write your solutions, make sure the save it after writing each solution. The structure of this file will be:

```
-- Task 1
INSERT INTO students (student_id, name, major, gpa)
VALUES (104, 'David', 'ME', 3.60);
--Task 2
SELECT name, gpa FROM students WHERE major = 'CSE';
```

1 Introduction

This lab module demonstrates advanced data manipulation techniques in PostgreSQL. We will cover character string processing for data cleaning, temporal operations for date logic, set-theoretic operators for combining datasets, and Data Control Language (DCL) for security management.

2 Lab Environment Setup

Before executing the functional demonstrations, you must create the schema and populate the tables. Execute the following SQL block to initialize your workspace.

2.1 DDL and Data Insertion

```sql
-- 1. AcademicStaff Table
CREATE TABLE AcademicStaff (
    id SERIAL PRIMARY KEY,
    full_name VARCHAR(100),
    email_addr VARCHAR(100),
    joining_date DATE
);

INSERT INTO AcademicStaff (id, full_name, email_addr, joining_date) VALUES
(101, 'ALICE munn', 'alice.m@university.edu', '2018-01-15'),
(102, 'bob roberts', 'bob.roberts@gmail.com', '2019-11-20'),
(103, 'Dr. chaRLie', 'charlie.c@yahoo.com', '2020-03-10'),
(104, 'DAVID smith', 'david.s@university.edu', '2021-07-01'),
(105, 'eve O''Connor', 'eve.oconnor@outlook.com', '2022-02-14'),
(106, 'frank LIN', 'frank.lin@protonmail.com', '2023-08-30'),
(107, 'Grace HOPPER', 'g.hopper@cs.edu', '2024-01-05'),
(108, 'helen keller', 'helen.k@org.net', '2024-09-12');

-- 2. SystemLogs Table
CREATE TABLE SystemLogs (
    log_id INT PRIMARY KEY,
    raw_entry TEXT
);

INSERT INTO SystemLogs (log_id, raw_entry) VALUES
(1, '   System_Start   '),
(2, '##CRITICAL_ERROR##'),
(3, 'User_Login_Success'),
(4, '   WARN: Low_Memory '),
(5, '__Connection_Timeout__');

-- 3. Course Enrollment Tables
CREATE TABLE Course_Alpha (student_id INT);
CREATE TABLE Course_Beta (student_id INT);

INSERT INTO Course_Alpha (student_id) VALUES
(1001), (1002), (1003), (1004), (1005);

INSERT INTO Course_Beta (student_id) VALUES
(1004), (1005), (1006), (1007), (1008);
```

Listing 1: Environment Setup Script

3 Scalar Functions: String Processing

String functions are essential for *Data Normalization*—the process of organizing data to appear similar across all records.

3.1 Case Conversion

Inconsistent casing often occurs during manual data entry.

- `UPPER(str)` / `LOWER(str)`: Converts entire string case.
- `INITCAP(str)`: Capitalizes the first letter of each word (Title Case).

```
SELECT
    full_name AS original,
    UPPER(full_name) AS upper_case,
    INITCAP(full_name) AS normalized_case
FROM AcademicStaff
WHERE id <= 104;
```

Result Set:

original	upper_case	normalized_case
ALICE munn	ALICE MUNN	Alice Munn
bob roberts	BOB ROBERTS	Bob Roberts
Dr. chaRLie	DR. CHARLIE	Dr. Charlie
DAVID smith	DAVID SMITH	David Smith

3.2 String Sanitization (TRIM)

Sanitization removes unwanted characters, such as leading spaces or specific symbols.

- `TRIM([BOTH|LEADING|TRAILING] 'char' FROM str)`: If 'char' is omitted, whitespace is removed.

```
SELECT
    raw_entry,
    TRIM(raw_entry) AS standard_trim,
    TRIM(BOTH '#' FROM raw_entry) AS hash_removal
FROM SystemLogs
WHERE log_id IN (1, 2, 4);
```

Result Set:

raw_entry	standard_trim	hash_removal
' System_Start '	'System_Start'	' System_Start '
'##CRITICAL_ERROR##'	'##CRITICAL_ERROR##'	'CRITICAL_ERROR'
' WARN: Low_Memory '	'WARN: Low_Memory'	' WARN: Low_Memory '

3.3 Substring Extraction Pattern Matching

Used to parse specific information from larger strings (e.g., extracting domains from emails).

- `SUBSTR(str, start, length)`: Extracts segment starting at index *start*.
- `LEFT(str, n)` / `RIGHT(str, n)`: Takes n characters from ends.
- `POSITION(substring IN string)`: Returns the index of a character.

```sql
-- We use POSITION to find the '@' symbol, then extract everything after it.
SELECT
    email_addr,
    POSITION('@' IN email_addr) as at_symbol_index,
    SUBSTR(email_addr, POSITION('@' IN email_addr) + 1) AS domain_only
FROM AcademicStaff
WHERE id BETWEEN 101 AND 103;
```

Listing 2: Dynamic Domain Extraction

Result Set:

email_addr	at_symbol_index	domain_only
alice.m@university.edu	8	university.edu
bob.roberts@gmail.com	12	gmail.com
charlie.c@yahoo.com	10	yahoo.com

3.4 Pattern Replacement

```sql
SELECT
    raw_entry,
    REPLACE(raw_entry, '_', ' ') AS clean_text
FROM SystemLogs
WHERE log_id = 3;
```

Result: Converts `User_Login_Success` → `User Login Success`.

4 Temporal Data Management

4.1 Intervals and Aging

PostgreSQL provides the `AGE` function to calculate the symbolic difference between two dates.

```
-- Calculating Tenure. Assumed Current Date: 2025-01-01
SELECT
    INITCAP(full_name) AS name,
    AGE('2025-01-01', joining_date) AS tenure
FROM AcademicStaff
WHERE id >= 106;
```

Result Set:

name	tenure
Frank Lin	1 year 4 mons 2 days
Grace Hopper	11 mons 27 days
Helen Keller	3 mons 20 days

4.2 Field Extraction (EXTRACT)

The `EXTRACT` function retrieves sub-fields (Year, Month, Day, Quarter, Week) from a date source. This is useful for aggregation (e.g., "Sales per Quarter").

```
SELECT
    joining_date,
    EXTRACT(YEAR FROM joining_date) AS join_year,
    EXTRACT(QUARTER FROM joining_date) AS fiscal_qtr
FROM AcademicStaff
WHERE id <= 104;
```

5 Relational Set Operators

Set operators combine the result sets of two or more queries. **Rules for Set Operations:**

1. **Union Compatibility:** The queries must return the same number of columns.
2. **Type Compatibility:** Corresponding columns must have compatible data types.

5.1 UNION vs UNION ALL

- `UNION`: Combines sets and **removes duplicates**. (Logical OR)
- `UNION ALL`: Combines sets and **keeps duplicates**. Faster performance.

```
-- Returns unique list of all students enrolled in ANY course
SELECT student_id FROM Course_Alpha
UNION
SELECT student_id FROM Course_Beta;
```

Result Count: 8 Rows (1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008).

5.2 INTERSECT

Returns only rows that appear in **BOTH** result sets. (Logical AND).

```
-- Find students taking BOTH courses
SELECT student_id FROM Course_Alpha
INTERSECT
SELECT student_id FROM Course_Beta;
```

Result Set:

student_id
1004
1005

5.3 EXCEPT (Difference)

Returns rows from the first query that are **NOT** present in the second query. Note: In Oracle, this is called `MINUS`.

```
-- Students in Alpha who are NOT taking Beta
SELECT student_id FROM Course_Alpha
EXCEPT
SELECT student_id FROM Course_Beta;
```

Result Set:

student_id
1001
1002
1003

6 Database Views

A View is a "virtual table" defined by a query. It does not store data itself (unless Materialized) but fetches it from the base tables when queried. Views are critical for simplifying complex queries and implementing security by hiding sensitive columns.

6.1 Standard Views

```
-- Create a view that hides the ID and raw Joining Date
CREATE VIEW View_PublicDirectory AS
SELECT
    INITCAP(full_name) AS display_name,
    email_addr
FROM AcademicStaff;

-- Query the View just like a table
SELECT * FROM View_PublicDirectory WHERE email_addr LIKE '%edu';
```

6.2 Updatable Views WITH CHECK OPTION

In PostgreSQL, simple views (1:1 mapping to a table without aggregates) are automatically updatable. The `WITH CHECK OPTION` clause is a powerful constraint that prevents a user from performing an update or insert through the view if the new row would vanish from the view (i.e., it violates the view's `WHERE` clause).

```
-- 1. Create a Restricted View for 'edu' emails only
CREATE OR REPLACE VIEW View_Edu_Staff AS
SELECT id, full_name, email_addr
FROM AcademicStaff
WHERE email_addr LIKE '%edu'
WITH CHECK OPTION;

-- 2. Valid Insert (Succeeds)
INSERT INTO View_Edu_Staff (full_name, email_addr)
VALUES ('John Doe', 'john.d@university.edu');

-- 3. Invalid Insert (Fails due to Check Option)
-- Fails because the email does not end in 'edu', so the row
-- would effectively disappear from this view immediately.
INSERT INTO View_Edu_Staff (full_name, email_addr)
VALUES ('Jane Doe', 'jane.d@gmail.com');
```

6.3 Materialized Views

Unlike standard views, a **Materialized View** physically saves the result of the query to the disk. This drastically improves read performance for expensive queries (e.g., complex joins or aggregations) but requires manual refreshing when the base data changes.

```
-- Define a heavy analytical query physically
CREATE MATERIALIZED VIEW MatView_StaffAnalysis AS
SELECT
    EXTRACT(YEAR FROM joining_date) as year,
    COUNT(*) as total_recruits
```

```
FROM AcademicStaff
GROUP BY EXTRACT(YEAR FROM joining_date);

-- Querying is instant (no recalculation)
SELECT * FROM MatView_StaffAnalysis;

-- Refreshing the data after base table updates
REFRESH MATERIALIZED VIEW MatView_StaffAnalysis;
```

7 Data Control Language (DCL)

DCL handles permissions and access control. In a production environment, permissions are rarely assigned to individual users; instead, **Role-Based Access Control (RBAC)** is used via Group Roles.

7.1 Role Management Attributes

```
-- 1. Create a concrete user (able to log in)
CREATE ROLE alice_operator
LOGIN
PASSWORD 'securePass123';

-- 2. Create a Group Role (abstract container, cannot log in)
CREATE ROLE faculty_group NOLOGIN;
```

7.2 Role Inheritance (RBAC)

Instead of granting permissions to `alice_operator` directly, we grant permissions to the `faculty_group`, and then make Alice a member of that group. This makes managing large teams significantly easier.

```
-- 1. Assign privileges to the Group
GRANT SELECT, INSERT ON Course_Alpha TO faculty_group;
GRANT SELECT ON View_PublicDirectory TO faculty_group;

-- 2. Add User to the Group (Alice inherits privileges)
GRANT faculty_group TO alice_operator;
```

7.3 Column-Level Security (Least Privilege)

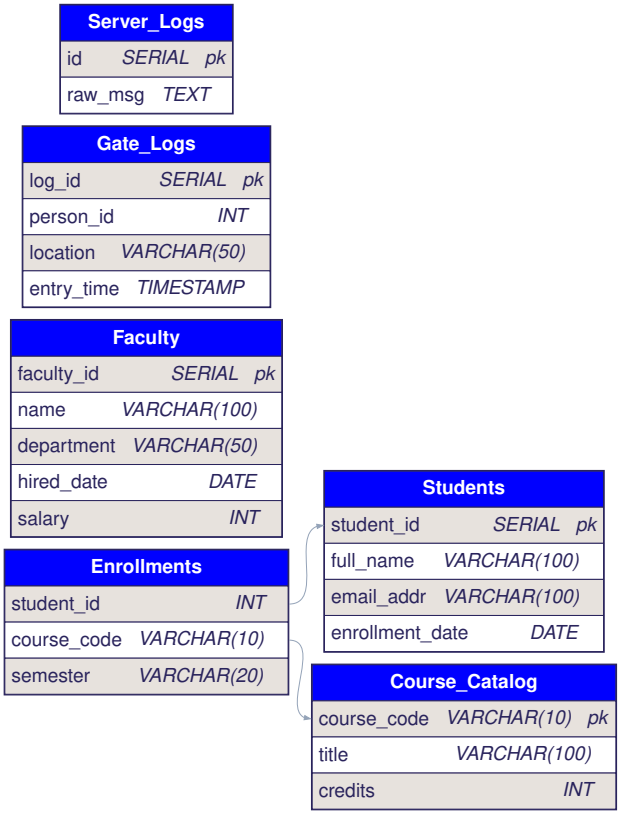
You can restrict access to specific columns. For example, a user might need to verify a log entry exists but should not be allowed to see the sensitive `raw_entry` content.

```
-- Create a restricted auditor role
CREATE ROLE auditor NOLOGIN;

-- Allow reading ONLY the log_id, effectively hiding raw_entry
GRANT SELECT (log_id) ON SystemLogs TO auditor;
```

7.4 Revoking Access

```
-- Revoke specific privilege
REVOKE INSERT ON Course_Alpha FROM faculty_group;

-- Remove a user from a group
REVOKE faculty_group FROM alice_operator;
```

8 Lab Tasks



1. Extract the domain name from the email addresses. You must use `POSITION` to find the '@' symbol and `SUBSTR` to get the text after it.

2. Find the names of students who enrolled in the year 2023. You must use `EXTRACT(YEAR ...)` in your `WHERE` clause.

3. Find the Student IDs of those who are taking 'CS102' but are NOT taking 'CS101'.

4. Display the `full_name` of students and the `domain` of their email, but only for students who have visited the 'Server Room'.

5. Create a **Materialized View** named `MatView_Enrollment_Stats` that counts how many students are in each course.