

Module 5 – Lab 5: Enforcing Data Access Policies

Primary Course SLO: SLO 5 – Implement data access security and monitoring

Environment: PostgreSQL + Terminal

Time: 40–60 Minutes

Purpose (Read First)

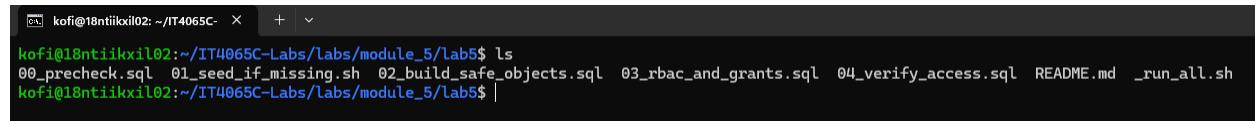
You are now the *Security Administrator*. In previous labs, you built the data structures; now, you will lock them down. You will implement:

- **Role-Based Access Control (RBAC):** Creating distinct roles for Analysts, Stewards, and Admins.
- **Data Masking:** Hiding Personally Identifiable Information (PII) while keeping data queryable.

Step 1: Replace Placeholder Files (Required)

Your repo currently contains placeholder files for Lab 5.

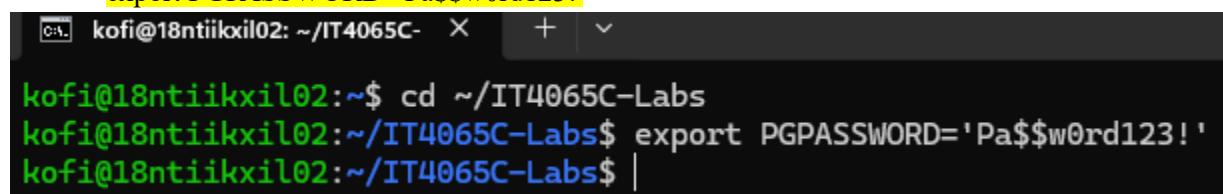
1. **Download:** [Lab5 Source.zip](#) from Canvas.
2. **Navigate:** `~/IT4065C-Labs/labs/module_5/lab5` in your lab environment.
3. **Replace:** Delete the old 7 files and paste the new (7) ones (`_run_all.sh`, `00_precheck.sql`, `01_see_if_missing.sh`, etc.).
Do not rename files. Do not merge content. Replace them entirely.
4. **Verify:** Run `ls`. You must see the `_run_all.sh` script before proceeding.



```
kofi@18ntiikxil02:~/IT4065C-Labs/labs/module_5/lab5$ ls
00_precheck.sql  01_seed_if_missing.sh  02_build_safe_objects.sql  03_rbac_and_grants.sql  04_verify_access.sql  README.md  _run_all.sh
kofi@18ntiikxil02:~/IT4065C-Labs/labs/module_5/lab5$ |
```

Step 2: Establish your Session

1. Navigate to the root course directory
`cd ~/IT4065C-Labs`
2. Run this in your terminal to avoid repeated password prompts:
`export PGPASSWORD='Pa$$w0rd123!'`



```
kofi@18ntiikxil02:~$ cd ~/IT4065C-Labs
kofi@18ntiikxil02:~/IT4065C-Labs$ export PGPASSWORD='Pa$$w0rd123!'
kofi@18ntiikxil02:~/IT4065C-Labs$ |
```

Step 3: Run Policy Enforcement

1. Execute the master script. This will build the views and apply GRANT and REVOKE commands.
`bash labs/module_5/lab5/_run_all.sh`

This script will automatically:

1. Verify required schemas exist
2. Confirm raw data is present
3. Create analytics and masked views
4. Create roles and apply least-privilege access

5. Test access using PASS / FAIL verification

You should see output indicating:

- Views created
- Roles created
- Some queries succeed
- Some queries fail on purpose

Screenshot 1: PASS/FAIL Results

Scroll up to Step 4: Verify access behavior section.

- **Notice:** role_analyst gets an ERROR: permission denied when trying to touch raw data. This is **good**; it means your security is working.
- **Capture a screenshot** of this PASS/FAIL section.

```
psql: kofi@18ntiikxil02: ~/IT4065C- × + | ▾
= Step 4: Verify access behavior (PASS/FAIL) ==
== Verification: Expected PASS/FAIL outcomes (RBAC + masking) ==
Legend: PASS means allowed, FAIL means correctly denied.

connected_as | current_role
postgres     | postgres
(1 row)

--- Role: role_analyst (should access analytics only) ---
SET
PASS: analyst can read v_sales_by_day
rows_from_sales_by_day
1
(1 row)

FAIL (expected): analyst cannot read raw.customers
psql:labs/module_5/lab5/04_verify_access.sql:28: ERROR: permission denied for schema raw
LINE 1: SELECT COUNT(*) FROM raw.customers;
^

FAIL (expected): analyst cannot read masked customer view (not granted)
psql:labs/module_5/lab5/04_verify_access.sql:23: ERROR: permission denied for view v_customers_masked
FAIL (expected): analyst cannot read raw PII view
psql:labs/module_5/lab5/04_verify_access.sql:26: ERROR: permission denied for view v_customers_raw_pii
RESET

--- Role: role_steward (should access analytics + masked customers) ---
SET
PASS: steward can read v_sales_by_day
rows_from_sales_by_day
1
(1 row)

PASS: steward can read v_customers_masked
rows_from_customers_masked
4
(1 row)

FAIL (expected): steward cannot read raw.customers
psql:labs/module_5/lab5/04_verify_access.sql:42: ERROR: permission denied for schema raw
LINE 1: SELECT COUNT(*) FROM raw.customers;
^

FAIL (expected): steward cannot read raw PII view
psql:labs/module_5/lab5/04_verify_access.sql:45: ERROR: permission denied for view v_customers_raw_pii
RESET

--- Role: role_admin (should access everything) ---
SET
PASS: admin can read raw.customers
rows_from_raw_customers
4
(1 row)

PASS: admin can read raw PII view
rows_from_raw_pii_view
4
(1 row)

PASS: admin can read masked customers
rows_from_masked_view
4
(1 row)

PASS: admin can read analytics view
rows_from_sales_by_day
1
(1 row)

RESET
```

Sample

Step 4: Audit the "Privilege Matrix"

Now, look under the hood to see how Postgres stores these permissions.

1. Enter Postgres: `psql -h localhost -U postgres -d it4065c`
2. Audit the Schema: `\dp student_kofi.*`

Decoder Ring for Screenshot 2

Look closely at the **Access privileges** column for these three rows:

- **v_sales_by_day:** You should see role_analyst=r. (r = read/select).
- **v_customers_masked:** You should see role_steward=r.
- **dim_customers:** You should only see role_admin.

Screenshot 2: The Audit Trail

- Capture a screenshot of the \dp output.

```

kofi@10.10.10.102:~/IT4065C-Labs$ psql -h localhost -U postgres -d it4065c
psql (14.20 (Ubuntu 14.20-Ubuntu 22.04.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)
Type "help" for help.

it4065c=# \dp student_kofi.*
                                         Sample
   Schema |      Name       | Type | Access privileges | Column privileges | Policies
-----+-----+-----+-----+-----+-----+-----+
student_kofi| data_classification_register | table | postgres=arwdDxt/postgres + |          |          |
student_kofi| dim_customers | table | postgres=arwdDxt/postgres + |          |          |
student_kofi| dim_products | table | postgres=arwdDxt/postgres + |          |          |
student_kofi| fct_order_items | table | postgres=arwdDxt/postgres + |          |          |
student_kofi| fct_orders | table | postgres=arwdDxt/postgres + |          |          |
student_kofi| olap_sales_by_day | table | postgres=arwdDxt/postgres + |          |          |
student_kofi| oltp_order_detail | table | postgres=arwdDxt/postgres + |          |          |
student_kofi| stg_customers | view  | postgres=arwdDxt/postgres + |          |          |
student_kofi| stg_order_items | view  | postgres=arwdDxt/postgres + |          |          |
student_kofi| stg_orders | view  | postgres=arwdDxt/postgres + |          |          |
student_kofi| stg_products | view  | postgres=arwdDxt/postgres + |          |          |
student_kofi| test_table | table | postgres=arwdDxt/postgres + |          |          |
student_kofi| v_customers_masked | view  | postgres=arwdDxt/postgres + |          |          |
student_kofi| v_customers_raw_pii | view  | postgres=arwdDxt/postgres + |          |          |
student_kofi| v_sales_by_day | view  | postgres=arwdDxt/postgres + |          |          |
                           +-----+-----+-----+-----+-----+-----+
(15 rows)
it4065c=#

```

Step 5: Critical Thinking

Answer in 4–7 sentences each.

1. **RBAC vs. Masking:** In your script output, role_steward was allowed to see v_customers_masked but denied access to raw.customers. Explain why we give the Steward a masked view instead of just giving them restricted access to the raw table.
2. **Interpreting Errors:** Look at the error ERROR: permission denied for schema raw. Even though the analyst has a role, they can't even "see" into the raw schema. How does this **Schema-level** security provide a better "defense in depth" than just table-level security?
3. **The Role of role_admin:** Your \dp output shows role_admin=arwdDxt. Using the decoder ring, explain what happens to the security of the system if a student accidentally uses the role_admin credentials for a basic dashboard.
4. **Monitoring:** We successfully enforced access. If an analyst tried to run SELECT * FROM raw.customers 100 times in a row, the database would stop them every time. Why is it still important to log those 100 **failed** attempts?

Deliverables

Submit a PDF containing:

1. **Screenshot 1:** Verification section showing SET ROLE and PASS/FAIL.
2. **Screenshot 2:** The \dp student_kofi.* table.
3. **Written Responses:** Q1–Q4.