# Reflection on Module 9: Working with Raw SQL in pgAdmin

## Introduction

In this exercise, I containerized a FastAPI application alongside PostgreSQL and pgAdmin using Docker Compose, then used pgAdmin’s Query Tool to manually create, insert, query, update, and delete records. This hands‑on work reinforced my understanding of SQL commands and database relationships.

## What I Did

1. Docker Compose Setup  
 - Defined three services (`api`, `db`, `pgadmin`) in `docker-compose.yml`.  
 - Built and ran containers with `docker compose up --build`.

2. Database Schema Creation  
 - Created a `users` table with a primary key and timestamp.  
 - Created a `calculations` table with foreign‑key reference to `users`.

3. Data Manipulation  
 - Inserted sample users (`alice`, `bob`, `hany`) and calculation records.  
 - Queried tables and joined `users` to `calculations`.  
 - Demonstrated an `UPDATE` on one row and a `DELETE` on another.

4. Documentation  
 - Captured screenshots of each SQL command and its result.  
 - Compiled evidence into a Word document with captions.

## Smooth Parts

• Docker Compose Magic: Spinning up all services with one command was seamless—containers launched and networked automatically.

• pgAdmin Intuitiveness: The Query Tool’s editor and output panes made it easy to write and execute SQL.

• SQL Syntax Familiarity: Basic `CREATE`, `INSERT`, and `SELECT` statements felt straightforward thanks to previous work with Python’s SQLAlchemy.

## Challenges & Resolutions

1. Foreign‑Key Constraint Violations  
 - Issue: Initial `INSERT` into `calculations` failed because no matching `users` rows existed.  
 - Resolution: Split the inserts—added `users` first, confirmed via `SELECT`, then inserted `calculations`.

2. Unhealthy FastAPI Container Status  
 - Issue: Docker showed the `api` container as 'unhealthy' due to its default healthcheck.  
 - Resolution: Recognized pgAdmin and Postgres as the focus and deferred fixing the API healthcheck.

3. Repeated INSERTs & Auto‑Increment IDs  
 - Issue: Running the script multiple times incremented the `id` sequence, causing `UPDATE id = 1` to affect no rows.  
 - Resolution: Queried current `id` values with `SELECT … ORDER BY id`, then targeted the correct `id` for the `UPDATE`.

## Key Takeaways

• Order Matters: Always insert parent records before child records when foreign keys are involved.

• Check Context: Verify you’re connected to the intended database/schema in pgAdmin before running DDL.

• Value of Screenshots: Captured output provides clear evidence of successful SQL operations for grading.

• Hands‑On SQL: Writing raw SQL deepened my grasp of one‑to‑many relationships and CRUD basics.

## Conclusion

This module solidified my practical skills in container orchestration with Docker Compose and direct SQL manipulation in pgAdmin. I feel more confident troubleshooting foreign‑key issues, adjusting queries based on actual data, and documenting my workflow for future reference.