Module 5: Software Assurance + Secure SQL (SQLi Defense)

Student: Moisés Junca
Course: EN.605.256.82.SP26 Modern Software Concepts in Python
Module: Module 5 - Software Assurance + Secure SQL (SQLi Defense)
Date: February 21, 2026

---

1. Installation Instructions

Method 1: pip + venv (Traditional)
```bash
cd module_5
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
pip install -e .
```

Method 2: uv (Fast Alternative)
```bash
cd module_5
pip install uv
uv venv
source .venv/bin/activate
uv pip sync requirements.txt
pip install -e .
```

Running the Application
```bash
export DB_NAME=gradcafe
export DB_USER=gradcafe_user
export DB_PASSWORD=your_password
python src/app.py
```

---

2. Dependency Graph Summary

The dependency graph (`dependency.svg`) visualizes the module structure and external dependencies of the GradCafe Analytics application. The core `app.py` module depends on Flask for the web framework, providing routing and request handling capabilities. It imports `db_utils.py` for database connection management, which in turn depends on `psycopg` (psycopg3) for PostgreSQL connectivity. The application also utilizes `query_data.py` and `load_data.py` modules for data analysis and ETL operations respectively. External dependencies include `os` and `threading` for system operations and concurrency, while development tools like `pylint` and `pydeps` ensure code

quality and dependency analysis. This modular architecture separates concerns and promotes maintainability.

---

3. SQL Injection Defenses

What Changed
Before (Vulnerable):
```python
col = "gpa"
cur.execute(f"SELECT AVG({col}) FROM applicants WHERE {col} IS NOT NULL")
```

After (Secure):
```python
from psycopg import sql

col = "gpa"
query = sql.SQL(
    "SELECT AVG({col}) FROM applicants WHERE {col} IS NOT NULL"
).format(col=sql.Identifier(col))
cur.execute(query)
```

Why It's Safe
1. **SQL Composition:** Dynamic SQL parts (column names) are constructed using `psycopg.sql.SQL` and `sql.Identifier()`, which properly quotes identifiers and prevents injection
2. **Parameter Binding:** User values use parameterized queries (`%s` placeholders), ensuring they're treated as data, not code
3. **LIMIT Enforcement:** All queries have explicit LIMIT clauses (LIMIT 1 for aggregates, LIMIT 5/10 for GROUP BY), with a `clamp_limit()` helper that enforces a maximum of 100
4. **Separation of Concerns:** SQL statement construction is separated from execution, making injection attacks impossible

---

4. Least-Privilege Database Configuration

Permissions Granted
```sql
CREATE USER gradcafe_user WITH PASSWORD 'secure_password';
GRANT CONNECT ON DATABASE gradcafe TO gradcafe_user;
GRANT SELECT ON TABLE applicants TO gradcafe_user;
```

Why These Permissions
- **Read-Only Access:** The application only performs SELECT queries for analytics; no INSERT/UPDATE/DELETE needed

```
- **No Superuser:** User cannot create/drop databases, tables, or modify
schema
- **No DDL Rights:** User cannot ALTER tables, DROP objects, or execute
administrative commands
- **Single Table:** Access limited to `applicants` table only, following
principle of least privilege
- **Connection Only:** User can connect to the database but has minimal
privileges beyond SELECT

### Environment Variables Used
- `DB_HOST`: Database server hostname
- `DB_PORT`: Database port (default: 5432)
- `DB_NAME`: Database name (gradcafe)
- `DB_USER`: Least-privilege username (gradcafe_user)
- `DB_PASSWORD`: User password (not committed to repo)

All credentials are stored in `.env` (gitignored) with `.env.example` showing
required variables.

---

## 5. Security Checklist

✅ **Pylint 10/10:** Zero warnings or errors
✅ **SQL Injection Fixed:** All queries use `sql.Identifier()` or
parameterized binding
✅ **LIMIT Enforcement:** All queries have explicit LIMIT clauses (max 100)
✅ **Environment Variables:** DB credentials externalized, `.env.example`
provided
✅ **Least Privilege:** Read-only DB user with minimal permissions
✅ **Dependency Scan:** Snyk test passed, Flask upgraded to 3.1.3
✅ **CI/CD:** GitHub Actions with 4 jobs (Pylint, pydeps, Snyk, Pytest)
✅ **Reproducible Environment:** `setup.py` + pip/uv instructions

---

## 6. GitHub Actions Workflow

The CI pipeline runs 4 parallel jobs on every push:

1. **Pylint:** Enforces 10/10 code quality (`--fail-under=10`)
2. **Dependency Graph:** Generates `dependency.svg` using pydeps + Graphviz
3. **Snyk:** Scans dependencies for vulnerabilities
4. **Pytest:** Runs full test suite with 100% coverage requirement

See `actions_success.png` for proof of successful run.
```