Faculty of Computer Science and Engineering

*University Ss. Cyril and Methodius*

Documentation of the project in Databases

**DBLearnStar – Transactional Operations**

Student: Genti Nuhiu, 221525 Mentor: Prof. Vangel Ajanovski

Skopje, 2025

Contents

[1. Description 3](#_Toc208007566)

[2. How submissions are made 3](#_Toc208007567)

[3. Submission example 3](#_Toc208007568)

[3.1. Description (written by admin): 3](#_Toc208007569)

[3.2. Student submission: 4](#_Toc208007570)

[3.3. Result: 4](#_Toc208007571)

[4. Execution Flow 4](#_Toc208007572)

[5. Setting up a database 5](#_Toc208007573)

[5.1. Prepare seed SQL file on your project folder. (schema and data.sql) 5](#_Toc208007574)

[5.2. Run Podman container 7](#_Toc208007575)

[5.3. Connect and test 7](#_Toc208007576)

[5.4. Database information 8](#_Toc208007577)

# Description

Transactional operations in SQL refer to the concept of treating a sequence of one or more SQL statements as a single, indivisible unit of work. This unit of work, known as a transaction, ensures data integrity and consistency within a database by adhering to the ACID properties: Atomicity, Consistency, Isolation, and Durability.

In order for this to work efficiently in the environments of DBLearnStar, we decided to use sandbox. The sandbox feature allows you to submit SQL solutions in a safe, isolated environment. Your queries are tested against a cloned database and validated with automated test cases.

# How submissions are made

A test problem consists of three main parts:

* **Correct query (reference solution):** Written in the description part with annotations “@”. This is the query that represents the expected solution. It is not executed by the client; it is used by the evaluator for comparison.
* **Student query:** Written in the box on the right hand side. This can be any valid SQL statement or sequence of statements (e.g., SELECT, UPDATE, INSERT, DDL changes).
* **Test cases:** Each test case has a name and a query. They are written in the following format: test-name: query. Multiple test cases can be provided, each separated by @. These queries are executed after your solution to verify correctness.

# Submission example

## Description (written by admin):

**@**

begin;

update student set last\_name=’test’;

end;

**@**

test-1: select \* from student

**@**

test-2: select \* from enrollment;

**@**

*The red part represents the correct query, and the blue part represents the test cases which will test the correctness of the student query.*

## Student submission:

begin;

update student set last\_name=’test’;

end;

## Result:

The table below represents the result after the student query and correct query have been executed in cloned databases. The test cases are tested against both databases, and if a difference has been found the result “false” will be returned, otherwise “true”.

**A screenshot of a computer

AI-generated content may be incorrect.**

# Execution Flow

* Two sandbox databases are cloned from a template. Both start with the same initial data.
* Correct and student SQL block are executed inside the separate sandboxes.
* Each test case query is run and compared with the expected output from the correct query.
* Differences (if any) are reported back by returning “false”.
* After the result has been returned, the sandbox databases are deleted.

# Setting up a database

When creating a test instance, it is necessary to have a database on which the queries will be tested. Let’s see a database consisting of three entities: student, course and enrollment.

## Prepare seed SQL file on your project folder. (schema and data.sql)

DROP TABLE IF EXISTS enrollment;

DROP TABLE IF EXISTS student;

DROP TABLE IF EXISTS course;

CREATE TABLE student (

student\_id SERIAL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(120) UNIQUE NOT NULL,

enrolled\_at DATE NOT NULL DEFAULT CURRENT\_DATE

);

CREATE TABLE course (

course\_id SERIAL PRIMARY KEY,

code VARCHAR(16) NOT NULL UNIQUE,

title VARCHAR(120) NOT NULL,

credits INTEGER NOT NULL CHECK (credits BETWEEN 1 AND 10)

);

CREATE TABLE enrollment (

student\_id INTEGER NOT NULL REFERENCES student(student\_id) ON DELETE CASCADE,

course\_id INTEGER NOT NULL REFERENCES course(course\_id) ON DELETE CASCADE,

enrolled\_on DATE NOT NULL DEFAULT CURRENT\_DATE,

grade VARCHAR(2),

PRIMARY KEY (student\_id, course\_id)

);

INSERT INTO student (first\_name, last\_name, email, enrolled\_at) VALUES

('Alice', 'Ng', 'alice.ng@example.com', '2025-09-01'),

('Boris', 'Ivanov', 'boris.ivanov@example.com', '2025-09-01'),

('Cara', 'Lopez', 'cara.lopez@example.com', '2025-08-28'),

('Driton', 'Krasniqi','driton.k@example.com', '2025-08-30'),

('Elena', 'Petrova','elena.p@example.com', '2025-08-29');

INSERT INTO course (code, title, credits) VALUES

('CS101', 'Intro to Computer Science', 6),

('MATH1', 'Discrete Mathematics', 5),

('DB201', 'Relational Databases', 6),

('AI100', 'Intro to AI', 4);

INSERT INTO enrollment (student\_id, course\_id, enrolled\_on, grade) VALUES

(1, 1, '2025-09-01', NULL),

(1, 3, '2025-09-01', NULL),

(2, 1, '2025-09-01', 'B+'),

(2, 2, '2025-09-01', NULL),

(3, 3, '2025-08-30', 'A'),

(3, 4, '2025-08-30', NULL),

(4, 2, '2025-08-31', 'B'),

(5, 4, '2025-08-31', 'A-');

## Run Podman container

# Create persistent storage volume

podman volume create pgdata

# Start PostgreSQL 16 container

podman run -d \

--name pg-university \

-e POSTGRES\_USER=uni\_admin \

-e POSTGRES\_PASSWORD=uni\_pass \

-e POSTGRES\_DB=university \

-p 5432:5432 \

-v pgdata:/var/lib/postgresql/data:Z \

-v ./pg-seed:/docker-entrypoint-initdb.d:ro,Z \

docker.io/library/postgres:16

This will create a database called “University”, a database user with username *uni\_admin* and password *uni\_pass*, and will auto-run the seed SQL (tables and dummy-data).

## Connect and test

* From host:

psql "postgresql://uni\_admin:uni\_pass@localhost:5432/university"

* Inside container:

podman exec -it pg-university psql -U uni\_admin -d university

* Sanity query:

SELECT s.first\_name, s.last\_name, c.code, e.grade

FROM enrollment e

JOIN student s ON s.student\_id = e.student\_id

JOIN course c ON c.course\_id = e.course\_id

ORDER BY s.student\_id, c.course\_id;

## Database information

**JDBC URL:** jdbc:postgresql://localhost:5432/university

**Database URL:** postgresql://uni\_admin:uni\_pass@localhost:5432/university

**Database name:** university

**Database user:** uni\_admin

**Database password:** uni\_pass

**Host:** localhost

**Port:** 5432