Lab 03: Making reports with Pandoc

My Favorite Course

Mateusz Mazur (CWL 1)

December 2, 2025

Contents

[@NOTE]: Please note that this text was AI-generated

Introduction

> *Your task is to create a report demonstrating sample transactions in PostgreSQL. The report should include code snippets and explanations of the SQL commands used. You can use the provided template to structure your report.*

This report provides an overview of sample transactions in PostgreSQL. The goal is to demonstrate the use of SQL commands for managing data in a relational database.

Transactions in PostgreSQL

> *What are transactions in PostgreSQL, and how do you use them? Provide examples of starting a transaction, committing a transaction, and rolling back a transaction.*

Transactions in PostgreSQL ensure that a series of operations are executed in a reliable and consistent manner. They follow the ACID properties: Atomicity, Consistency, Isolation, and Durability.

Starting a Transaction

To begin a transaction, use the `BEGIN` command:

```sql
BEGIN;
```

Committing a Transaction

To save the changes made during a transaction, use the `COMMIT` command:

```sql
COMMIT;
```

Rolling Back a Transaction

If an error occurs or you want to discard changes, use the `ROLLBACK` command:

```sql
ROLLBACK;
```

Example: Managing Bank Accounts

> *Show the use of transactions in a sample scenario, such as transferring money between two bank accounts. Include SQL code snippets and explanations. Provide a plot of the database schema.*

This section demonstrates a sample transaction for transferring money between two bank accounts.

Step 1: Create the Table

First, create a table to store account information:

```sql
CREATE TABLE accounts (
    account_id SERIAL PRIMARY KEY,
    account_name VARCHAR(50),
    balance NUMERIC(10, 2)
);
```

Final database schema is presented in fig. 1.

Step 2: Insert Sample Data

Insert initial data into the `accounts` table:

```sql
INSERT INTO accounts (account_name, balance)
VALUES
    ('Alice', 1000.00),
    ('Bob', 500.00);
```

Step 3: Perform a Transaction

Transfer $200 from Alice's account to Bob's account:

Figure 1: Schema of the database used in the exercise

```
BEGIN;

UPDATE accounts
SET balance = balance - 200
WHERE account_name = 'Alice';

UPDATE accounts
SET balance = balance + 200
WHERE account_name = 'Bob';

COMMIT;
```

Step 4: Verify the Results

Check the updated balances:

```
SELECT * FROM accounts;
```

Problems in Transactions

> *Discuss common problem(s) that can occur in transactions, such as dirty reads, non-repeatable reads, and phantom reads. Provide examples and SQL code snippets to illustrate these issues.*

Non-repeatable Reads

Non-repeatable reads occur when a transaction reads the same row twice and gets different data each time due to another transaction modifying that row in between the reads, e.g.:

```
T1: BEGIN;
T1: SET TRANSACTION ISOLATION LEVEL
T1:     READ COMMITTED; -- (default)
```

```
                                                  T2: BEGIN;
T1: -- T1 reads the data for the first time.
T1: SELECT * FROM accounts
T1:     WHERE account_name = 'Alice';
T1: -- Result: 'Alice', 1000.00
                                                  T2: -- T2 updates the data for the same row.
                                                  T2: UPDATE accounts SET balance = balance - 200
                                                  T2: WHERE account_name = 'Alice';
T1: -- T1 reads the data again (just checking).
T1: SELECT * FROM accounts
T1:     WHERE account_name = 'Alice';
T1: -- Result: 'Alice', 1000.00 (the same)
                                                  T2: -- T2 commits the change to the database.
                                                  T2: COMMIT;
T1: -- T1 reads the data for the second time
T1: --      within the SAME transaction.
T1: SELECT * FROM accounts
T1:     WHERE account_name = 'Alice';
T1: -- Result: 'Alice', 800.00
T1: -- The value has changed from 1000.00 to 800.00,
T1: -- constituting a "Non-repeatable read".
T1: COMMIT;
```

Conclusion

> *Summarize the key points covered in the report about transactions in PostgreSQL.*

- This report demonstrated the use of transactions in PostgreSQL to ensure data consistency during operations.
- Transactions are a powerful feature for managing complex database operations safely and reliably.
- Common problems such as non-repeatable reads can occur, and understanding these issues is crucial for effective database management.
- By following best practices and using transactions appropriately, developers can ensure the integrity of their data in PostgreSQL databases.