Bases de Dados

Lab 08 : Desenvolvimento de Aplicações e Transações

4ª Parte – Atualizações Concorrentes e Transações

1. Abra uma linha de comando para o sistema Postgres e use a base de dados "bank" (istxxxxxx).

```
psql -U istXXXXXX -h db.tecnico.ulisboa.pt
```

2. Escreva uma consulta para obter os dados das contas do cliente *Cook*.

```
SELECT * FROM depositor natural join account WHERE
customer_name='Cook';
```

3. O cliente *Cook* pretende transferir 500€ da conta A-102 para a conta A-101. Inicie uma transação com o comando `START TRANSACTION;`

```
     START TRANSACTION;
```

4. Coloque 500€ na conta A-101.

```
UPDATE account SET balance=500+(SELECT balance from account
WHERE account_number='A-101') WHERE account_number='A-101';
UPDATE account SET balance=(SELECT balance from account WHERE
account_number='A-102')-500 WHERE account_number='A-102';
```

5. Consulte os saldos das contas do cliente *Cook*. Neste momento quanto dinheiro tem o cliente *Cook* no banco?

```
SELECT SUM(balance) from depositor natural join account WHERE
customer_name='Cook';
```

1200

```
SELECT * FROM depositor natural join account WHERE
customer_name='Cook';
```

```
 account_number | customer_name | branch_name |  balance
----------------+---------------+-------------+-----------
```

```
A-101           | Cook           | Downtown     | 1000.0000
A-102           | Cook           | Uptown       |  200.0000
```

6. Mantendo a transação em aberto, abra uma segunda linha de comando do sistema e ligue-se à mesma base de dados.

```
psql -U istXXXXXX -h db.tecnico.ulisboa.pt
```

7. Consulte os saldos das contas do cliente *Cook*. Afinal quanto dinheiro tem o cliente *Cook* no banco?

```
SELECT SUM(balance) from depositor natural join account WHERE
customer_name='Cook';
```

1200

```
SELECT * FROM depositor natural join account WHERE
customer_name='Cook';
```

```
account_number | customer_name | branch_name | balance
---------------+---------------+-------------+----------
      A-101    | Cook          | Downtown    | 500.0000
      A-102    | Cook          | Uptown      | 700.0000
```

8. Na primeira linha de comando, onde está a correr a transação, retire 500€ da conta A-102.

```
UPDATE account SET balance=(SELECT balance from account WHERE
account_number='A-102')-500 WHERE account_number='A-102';
```

9. Na segunda linha de comando consulte os saldos das contas do cliente *Cook*.

```
SELECT * FROM depositor natural join account WHERE
customer_name='Cook';
```

```
 account_number | customer_name | branch_name | balance
```

```
----------------+---------------+-------------+----------
 A-101          | Cook          | Downtown    | 500.0000
 A-102          | Cook          | Uptown      | 700.0000
```

10. Na primeira linha de comando confirme a transação (instrução `COMMIT`)

```
COMMIT;
```

11. Na segunda linha de comando consulte novamente os saldos das contas do cliente *Cook*. Confirme os resultados da transação.

```
SELECT * FROM depositor natural join account WHERE
customer_name='Cook';
```

```
 account_number | customer_name | branch_name |  balance
----------------+---------------+-------------+-----------
 A-101          | Cook          | Downtown    | 1000.0000
 A-102          | Cook          | Uptown      | -300.0000
```

5ª Parte – Implementação de Transações em Flask

1. Implemente o mecanismo de transações em Python.

2. O cliente *Cook* acabou de abastecer 25€ de gasolina numa estação de serviço e vai pagar com multibanco (conta A-101). Ao mesmo tempo a sua esposa, que tem um cartão multibanco para a mesma conta, vai levantar 50€ numa máquina multibanco do outro lado da cidade.

3. Abra uma linha de comando para a base de dados e inicie uma transação.

```
START TRANSACTION;
```

4. Abra uma janela do browser para a página

http://web.ist.utl.pt/istxxxxxx/app.cgi/accounts

5. Na primeira transação retire 25€ da conta (abastecimento).

```
UPDATE account set balance=(SELECT balance from account WHERE
account_number='A-101')-25 WHERE account_number='A-101';
```

6. No browser retire 50€ da conta (levantamento). O que se passa quando tenta fazer isto? Porquê?

   Fica a aguardar a resposta da função que recebe os dados via POST.
   Como nessa função tentamos fazer primeiro um UPDATE e só depois é que damos instruções para a página carregar, mostrando a query de SQL. Enquanto o UPGRADE não for executado o browser vai ficar a aguardar. Isto acontece porque aquela linha está bloqueada devido a uma transação em curso.

7. A linha telefónica da estação de serviço está intermitente e a ligação cai. O pagamento que estava a ser feito tem de ser cancelado. Cancele essa transação. (Garantir o ROLLBACK)

```
ROLLBACK;
```

8. Ao mesmo tempo que faz a alínea 17, repare no que acontece no browser. Como explica este fenómeno?

   No momento que executamos ROLLBACK na linha de comandos, terminamos a transação que estava a bloquear a linha que queremos atualizar com a nossa query SQL. Como o bloqueio da linha foi levantado, a função que tentava executar o UPGRADE foi executada e começou executar as instruções seguintes, neste caso gerar a página com a nossa query SQL.

9. Consulte novamente os saldos das contas do cliente *Cook*.

```
SELECT * FROM depositor natural join account WHERE
customer_name='Cook';
```

```
 account_number | customer_name | branch_name |  balance
----------------+---------------+-------------+-----------
 A-102          | Cook          | Uptown      | -300.0000
 A-101          | Cook          | Downtown    |  950.0000
```

No final o seu ficheiro app.cgi deve ficar como o código seguinte:

```
#!/usr/bin/python3

from wsgiref.handlers import CGIHandler
from flask import Flask
from flask import render_template, request

## Libs postgres
import psycopg2
import psycopg2.extras

app = Flask(__name__)

## SGBD configs
DB_HOST="db.tecnico.ulisboa.pt"
DB_USER="istxxxxxx"
DB_DATABASE=DB_USER
DB_PASSWORD="xxxxxxxx"
DB_CONNECTION_STRING = "host=%s dbname=%s user=%s password=%s" %
(DB_HOST, DB_DATABASE, DB_USER, DB_PASSWORD)


## Runs the function once the root page is requested.
## The request comes with the folder structure setting ~/web as
the root
@app.route('/')
def list_accounts():
  dbConn=None
  cursor=None
  try:
    dbConn = psycopg2.connect(DB_CONNECTION_STRING)
    cursor = dbConn.cursor(cursor_factory =
psycopg2.extras.DictCursor)
    query = "SELECT * FROM account;"
    cursor.execute(query)
    return render_template("index.html", cursor=cursor)
  except Exception as e:
```

```
    return str(e) #Renders a page with the error.
  finally:
    cursor.close()
    dbConn.close()

@app.route('/accounts')
def list_accounts_edit():
  dbConn=None
  cursor=None
  try:
    dbConn = psycopg2.connect(DB_CONNECTION_STRING)
    cursor = dbConn.cursor(cursor_factory =
psycopg2.extras.DictCursor)
    query = "SELECT account_number, branch_name, balance FROM
account;"
    cursor.execute(query)
    return render_template("accounts.html", cursor=cursor,
params=request.args)
  except Exception as e:
    return str(e)
  finally:
    cursor.close()
    dbConn.close()

@app.route('/balance')
def alter_balance():
  try:
    return render_template("balance.html", params=request.args)
  except Exception as e:
    return str(e)


@app.route('/update', methods=["POST"])
def update_balance():
  dbConn=None
  cursor=None
  try:
```

```
    dbConn = psycopg2.connect(DB_CONNECTION_STRING)
    cursor = dbConn.cursor(cursor_factory =
psycopg2.extras.DictCursor)
    query = f'''UPDATE account SET
balance={request.form["balance"]} WHERE account_number =
'{request.form["account_number"]}';'''
    cursor.execute(query)
    return query
  except Exception as e:
    return str(e)
  finally:
    dbConn.commit()
    cursor.close()
    dbConn.close()


CGIHandler().run(app)
```

Estrutura de ficheiros

No servidor sigma deverá ter a seguinte estrutura de ficheiros para garantir que este guião funciona.

```
~
|
-web
|  |-templates
|  |      |-accounts.html
|  |      |-balance.html
|  |      |-index.html
|  |- app.cgi
|  |- test.cgi
```

Os ficheiros.cgi não deve ter permissões de escrita para o owner nem nenhum dos diretórios onde se encontra. Se necessário execute na linha de comandos **chmod -R 755 ~/web** que aplica recursivamente as permissões à directoria.