Прилоржение 3

Изоляция и многоверсионность.

Видимость версий строк

Убедиться в том что версии строки в базах данных существуют в нескольких следующим образом

Ввод [1]:
```
# Для низкоуровневого параллельного подключения
# использую OPM SQLAlchemy

from sqlalchemy import create_engine, text

# А также библиотеку pandas для построения таблиц

import pandas as pd
```

Ввод [2]:
```
# Создадим таблицу

!psql -U postgres -c "DROP TABLE test;"
!psql -U postgres -c "CREATE TABLE test(s text);"
```

```
DROP TABLE
CREATE TABLE
```

Ввод [3]:
```
# Создам движок и два параллельных подключения
# для демонстрации работы принципа многоверсионности

engine = create_engine(f"postgresql+psycopg2://postgres:password(

connectionOne = engine.connect()
connectionTwo = engine.connect()
connectionOne, connectionTwo
```

Out[3]:
```
(<sqlalchemy.engine.base.Connection at 0x7fea7bb31210>,
 <sqlalchemy.engine.base.Connection at 0x7feb0018da10>)
```

Ввод [4]:
```
# И вставим одну строку. Если не начать транзакцию
# явно командой BEGIN, psql выполняет команду
# и немедленно фиксирует результат:

connectionOne.begin()
connectionOne.execute(text("INSERT INTO test VALUES ('Первая вер
connectionOne.commit()
```

Ввод [5]:

```
# Начнем транзакцию и выведем ее номер:

connectionOne.begin()
result = connectionOne.execute(text("SELECT pg_current_xact_id()
pd.DataFrame(data=result.fetchall())
```

Out[5]:

	pg_current_xact_id
0	895

Ввод [6]:

```
# Транзакция видит первую (и пока единственную) версию строки:

result = connectionOne.execute(text("SELECT *, xmin, xmax FROM t
pd.DataFrame(data=result.fetchall())
```

Out[6]:

	s	xmin	xmax
0	Первая версия	894	0

Ввод [7]:

```
# Теперь начнем другую транзакцию в другом сеансе:

connectionTwo.begin()
result = connectionTwo.execute(text("SELECT pg_current_xact_id()
pd.DataFrame(data=result.fetchall())
```

Out[7]:

	pg_current_xact_id
0	896

Ввод [8]:

```
# Транзакция видит ту же единственную версию:

result = connectionTwo.execute(text("SELECT *, xmin, xmax FROM t
pd.DataFrame(data=result.fetchall())
```

Out[8]:

	s	xmin	xmax
0	Первая версия	894	0

Ввод [9]:

```
# Теперь изменим строку во второй транзакции.

connectionTwo.execute(text("UPDATE test SET s = 'Вторая версия';
result = connectionTwo.execute(text("SELECT *, xmin, xmax FROM t
pd.DataFrame(data=result.fetchall())
```

Out[9]:

	s	xmin	xmax
0	Вторая версия	896	0

Ввод [10]:

```
# А что увидит первая транзакция?

result = connectionOne.execute(text("SELECT *, xmin, xmax FROM t
pd.DataFrame(data=result.fetchall())
```

Out[10]:

	s	xmin	xmax
0	Первая версия	894	896

Поскольку изменение не зафиксировано, первая транзакция продолжает видеть первую версию строки. Так работает принцип **многоверсионности**

Ввод [11]:
```
# Теперь зафиксируем изменения во второй ранзакции
# и проверим что увидит первая транзакция.

connectionTwo.commit()

result = connectionOne.execute(text("SELECT *, xmin, xmax FROM t
pd.DataFrame(data=result.fetchall())
connectionOne.commit()
```

Теперь и первая транзакция видит вторую версию строки.

После фиксации первая версия строки больше не видна ни в одной транзакции.

Уровень изоляции Read Commited

Ввод [12]:
```
# Создадим заново таблицу test
# но уже с целочисленным полем n

!psql -U postgres -c "DROP TABLE test;"
!psql -U postgres -c "CREATE TABLE test(n integer);"
```

```
DROP TABLE
CREATE TABLE
```

Ввод [13]:
```
# Создам два параллельных подключения

connectionOne = engine.connect()
connectionTwo = engine.connect()
connectionOne, connectionTwo
```

Out[13]:
```
(<sqlalchemy.engine.base.Connection at 0x7fea7ca41510>,
 <sqlalchemy.engine.base.Connection at 0x7fea7bb30290>)
```

Ввод [14]:
```
# Положим туда данные

connection = engine.connect()
connection.execute(text("INSERT INTO test VALUES (42);"))
connection.commit()
connection.close()
```

Ввод [15]:
```
# Запрос из первой транзакции (по умолчанию уровень изоляции Rea

connectionOne.begin()
result = connectionOne.execute(text("SELECT * FROM test;"))
pd.DataFrame(data=result.fetchall())
```

Out[15]:

	n
0	42

Ввод [16]:
```
# Удаляем строку во второй транзакции и фиксируем изменения:

connectionTwo.execute(text("DELETE FROM test;"))
connectionTwo.commit()
```

Ввод [17]:
```
# Повторим запрос: строк в таблице уже нет

result = connectionTwo.execute(text("SELECT * FROM test;"))
pd.DataFrame(data=result.fetchall())
```

Out[17]: —

Ввод [18]:
```
# Первая транзакция видит произошедшие изменения.
# Таблица пуста

result = connectionOne.execute(text("SELECT * FROM test;"))
pd.DataFrame(data=result.fetchall())
```

Out[18]: —

Уровень изоляции Repeatable Read

Ввод [19]:
```
# Вернем строку на место

connection = engine.connect()
connection.execute(text("INSERT INTO test VALUES (42);"))
connection.commit()
connection.close()
```

Ввод [20]:
```
# Создам заново два параллельных подключения

connectionOne = engine.connect()
connectionTwo = engine.connect()
connectionOne, connectionTwo

connectionOne.execution_options(isolation_level="REPEATABLE READ
connectionOne.begin()
result = connectionOne.execute(text("SELECT * FROM test;"))
pd.DataFrame(data=result.fetchall())
```

Out[20]:

	n
0	42

Ввод [21]:
```
# Удаляем строку во второй транзакции и фиксируем изменения:

connectionTwo.execute(text("DELETE FROM test;"))
connectionTwo.commit()
result = connectionTwo.execute(text("SELECT * FROM test;"))
pd.DataFrame(data=result.fetchall())
```

Out[21]: —

Ввод [22]:

```
# На этом уровне изоляции первая транзакция не видит изменений.

result = connectionOne.execute(text("SELECT * FROM test;"))
pd.DataFrame(data=result.fetchall())
```

Out[22]:

	n
0	42