**Транзакции и уровни изоляции***(заполняется спикером урока)*

**Что нужно знать для проверки ДЗ**

1) Знать как делать запросы в PostgreSQL (select, insert, update, delete)

2) Иметь консоль чтобы понять что синтаксис SQL корректный (можно использовать Data Grip или DBeaver или просто консоль)

3) Знать как делать транзакции в SQL, быть знакомым со всеми ключевыми словами

**Чек-лист:**

1) **Шаг 1**:Студент предоставил корректный SQL код по созданию таблиц и запрос с созданием тестового аккаунта корректный

Ответ:

-- Создание базы данных

CREATE DATABASE BankTransactionsDB;

-- Переключение на созданную базу данных

\c BankTransactionsDB;

-- Создание таблицы Accounts

CREATE TABLE Accounts (

AccountID serial PRIMARY KEY,

AccountNumber VARCHAR(255),

Balance DECIMAL

);

-- Создание таблицы Transactions

CREATE TABLE Transactions (

TransactionID serial PRIMARY KEY,

AccountID INT REFERENCES Accounts(AccountID),

TransactionType VARCHAR(255) CHECK (TransactionType IN ('Deposit', 'Withdrawal')),

Amount DECIMAL,

TransactionDate TIMESTAMP

);

В этих запросах используется тип данных serial для автоматического инкрементирования AccountID и TransactionID при вставке новых записей. Также, в таблице Transactions есть внешний ключ (FOREIGN KEY), который ссылается на поле AccountID в таблице Accounts.

2) Был создан тестовый аккаунт:

Ответ: (цифры могут быть другими, но запрос должен работать корректно)  
**INSERT INTO Accounts (AccountNumber, Balance) VALUES ('123456789', 1000.00), ('987654321', 1500.00);**

3) **Шаг 2:**  
Первая транзакция написана корректно:  
Ответ:  
**BEGIN;**

**UPDATE Accounts SET Balance = Balance + 500 WHERE AccountID = 1;**

**INSERT INTO Transactions (AccountID, TransactionType, Amount, TransactionDate) VALUES (1, 'Deposit', 500, NOW()); COMMIT;**

4) Вторая транзакция также написана корректно:  
**BEGIN;**

**UPDATE Accounts SET Balance = Balance - 300 WHERE AccountID = 2;**

**INSERT INTO Transactions (AccountID, TransactionType, Amount, TransactionDate) VALUES (2, 'Withdrawal', 300, NOW());**

**COMMIT;**

5) **Шаг 3**Студент должен будет объяснить в чем разница между READ UNCOMMITTED и READ COMMITTED   
Пояснение:  
READ UNCOMMITTED: Этот уровень изоляции позволяет читающим транзакциям видеть изменения, внесенные другими транзакциями, даже если они еще не зафиксированы. В данном случае, если вы изменяете баланс в одном соединении, то второе соединение сможет прочитать измененные данные еще до завершения транзакции.

**-- Соединение 1**

**BEGIN TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;**

**SELECT Balance FROM Accounts WHERE AccountID = 1;**

**-- Оставьте соединение 1 открытым**

**-- Соединение 2**

**BEGIN;**

**UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;**

**COMMIT;**

**-- Попробуйте выполнить**

**SELECT в соединении 1 SELECT Balance FROM Accounts WHERE AccountID = 1;**

**-- Вернет измененное значение баланса из транзакции 2**

READ COMMITTED: Этот уровень изоляции позволяет читающим транзакциям видеть только те изменения, которые уже зафиксированы. В данном случае, чтение вторым соединением вернет измененные данные только после завершения транзакции изменения.  
**-- Соединение 1**

**BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;**

**SELECT Balance FROM Accounts WHERE AccountID = 1;**

**-- Оставьте соединение 1 открытым -- Соединение 2**

**BEGIN;**

**UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;**

**COMMIT;**

**-- Попробуйте выполнить SELECT в соединении 1**

**SELECT Balance FROM Accounts WHERE AccountID = 1;**

**-- Вернет измененное значение баланса из транзакции 2 только после ее завершения**REPEATABLE READ: Этот уровень изоляции не позволяет читающим транзакциям видеть изменения, внесенные другими транзакциями, пока они сами не завершатся. В данном случае, изменения, внесенные в одной транзакции, не будут видны другой транзакции до завершения первой транзакции.

**-- Соединение 1**

**BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;**

**SELECT Balance FROM Accounts WHERE AccountID = 1;**

**-- Оставьте соединение 1 открытым**

**-- Соединение 2**

**BEGIN;**

**UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = 1;**

**-- Попробуйте выполнить SELECT в соединении 1**

**SELECT Balance FROM Accounts WHERE AccountID = 1;**

**-- НЕ вернет измененное значение баланса из транзакции 2, так как она еще не завершена COMMIT;**

**-- Теперь выполните SELECT в соединении 1**

**SELECT Balance FROM Accounts WHERE AccountID = 1;**

**-- Вернет измененное значение баланса из транзакции 2 после ее завершения**

6) **Шаг 4**

Ответ студента совпал хотя бы с одним из 2-3 пунктов из каждого топика снизу.

**Ответ:**

Изменение уровня изоляции влияет на то, каким образом транзакции взаимодействуют друг с другом при одновременном доступе к данным. Ниже представлены основные выводы и обсуждения:

* READ UNCOMMITTED:
  + Позволяет читающим транзакциям видеть изменения, которые еще не зафиксированы.
  + Может привести к чтению "грязных данных" (данных, которые могут быть отменены или не зафиксированы).
  + Не рекомендуется использовать в банковских системах, где важна высокая степень целостности данных.
* READ COMMITTED:
  + Позволяет читающим транзакциям видеть только зафиксированные изменения.
  + Уменьшает вероятность чтения "грязных данных", но все еще допускает некоторую степень несогласованности в данных.
  + Может быть более предпочтительным в банковских системах, где важна баланс между целостностью данных и производительностью.
* REPEATABLE READ:
  + Предотвращает чтение данных, измененных другими транзакциями до завершения текущей транзакции.
  + Обеспечивает более высокую степень изоляции, но может привести к долгосрочным блокировкам.
  + Предпочтительно в системах, где важна максимальная целостность данных и невозможность чтения "грязных данных".

### **Предпочтительные уровни изоляции в банковских системах:**

* В банковских системах, где целостность данных является критическим аспектом, READ COMMITTED или REPEATABLE READ могут быть более предпочтительными, поскольку они обеспечивают более высокий уровень изоляции данных.
* READ COMMITTED может быть разумным компромиссом между целостностью данных и производительностью, учитывая, что уровень REPEATABLE READ может привести к долгосрочным блокировкам.

### **Стратегии управления конфликтами и блокировками:**

* Оптимистическое управление конфликтами: Позволяет транзакциям выполняться независимо, а затем проверяет наличие конфликтов перед их фиксацией. Могут использоваться механизмы версионирования данных.
* Пессимистическое управление конфликтами: Использует блокировки для предотвращения конфликтов. Допускает меньше конфликтов, но может привести к долгосрочным блокировкам, особенно при высокой нагрузке.
* Блокировка по чтению и записи: Применяется для обеспечения уровня изоляции. Блокировка чтения может блокировать запись, и наоборот. Требует внимательного управления, чтобы избежать долгосрочных блокировок.

В реальных банковских системах, выбор уровня изоляции и стратегии управления конфликтами зависит от баланса между целостностью данных, производительностью и требованиями к конкретному бизнес-сценарию.

**Итого, минимум, чтобы принять ДЗ**

- студент сделал карту пользовательских историй, она отличается от примера спикера

**Само домашнее задание из примера (для вашего понимания):**

Сделать User Story Map для своего продукта в Miro

Переложить ее на верхнеуровневый Roadmap на 3-6 мес в таблице (как если бы была задача показать топ-менеджменту крупными мазками, над чем ведется работа подразделения)

Прочитать все приложенные к уроку статьи и сказать, какая понравилась больше всего ;)

Если нет своего продукта, то вы можете:

1. Использовать рабочий проект – работайте над реальными кейсами своего продукта.
2. Придумать свою идею.
3. Если придумать совсем не получается, то можете использовать кейсы, предлагаемые в воркшопе.