Лабораторная работа №11 Транзакции в базах данных

Цель работы:

Изучить понятие транзакции, свойства ACID, освоить команды управления транзакциями в PostgreSQL. Научиться применять транзакции для обеспечения целостности данных.

Программное обеспечение: PostgreSQL, DBeaver
Время: 2 академических часа

Теоретическая часть

1. Что такое транзакция?

Транзакция - это последовательность операций с базой данных, которая выполняется как единое целое. Либо все операции выполняются успешно, либо ни одна из них.

2. Свойства ACID

- **Atomicity (Атомарность)** - транзакция выполняется полностью или не выполняется совсем
- **Consistency (Согласованность)** - транзакция переводит базу данных из одного согласованного состояния в другое
- **Isolation (Изолированность)** - параллельные транзакции не должны влиять друг на друга
- **Durability (Долговечность)** - результаты выполненной транзакции должны сохраняться после сбоев

3. Команды управления транзакциями в PostgreSQL

```
BEGIN;          -- Начало транзакции
COMMIT;         -- Подтверждение транзакции
ROLLBACK;       -- Откат транзакции
SAVEPOINT;      -- Создание точки сохранения
ROLLBACK TO;    -- Откат к точке сохранения
```

Подготовка базы данных

Создание таблиц для демонстрации транзакций

```
SET search_path TO ваш_логин;

-- Таблица банковских счетов
CREATE TABLE bank_accounts (
    account_id SERIAL PRIMARY KEY,
    account_number VARCHAR(20) UNIQUE NOT NULL,
    client_name VARCHAR(100) NOT NULL,
    balance DECIMAL(12,2) DEFAULT 0.00,
    created_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Таблица операций по счетам
CREATE TABLE account_transactions (
    transaction_id SERIAL PRIMARY KEY,
    account_id INTEGER REFERENCES bank_accounts(account_id),
    transaction_type VARCHAR(20) NOT NULL,
    amount DECIMAL(12,2) NOT NULL,
    description TEXT,
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    related_account_id INTEGER,
    status VARCHAR(20) DEFAULT 'completed'
);

-- Таблица заказов (для демонстрации конкурентного доступа)
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    order_number VARCHAR(20) UNIQUE NOT NULL,
    customer_name VARCHAR(100) NOT NULL,
    total_amount DECIMAL(10,2),
    status VARCHAR(20) DEFAULT 'new',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Таблица товаров (для демонстрации конкурентного доступа)
CREATE TABLE products (
```

```
    product_id SERIAL PRIMARY KEY,
    product_name VARCHAR(100) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    stock_quantity INTEGER NOT NULL,
    reserved_quantity INTEGER DEFAULT 0,
    last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Заполнение таблиц тестовыми данными

```
-- Банковские счета
INSERT INTO bank_accounts (account_number, client_name, balance) VALUES
('ACC001', 'Иванов Иван', 50000.00),
('ACC002', 'Петров Петр', 30000.00),
('ACC003', 'Сидорова Мария', 75000.00),
('ACC004', 'Козлов Алексей', 25000.00),
('ACC005', 'Смирнова Анна', 100000.00);

-- Товары
INSERT INTO products (product_name, price, stock_quantity) VALUES
('Ноутбук', 50000.00, 10),
('Смартфон', 30000.00, 20),
('Планшет', 25000.00, 15),
('Монитор', 15000.00, 8),
('Клавиатура', 2000.00, 50);

-- Заказы
INSERT INTO orders (order_number, customer_name, total_amount) VALUES
('ORD001', 'Иванов', 50000.00),
('ORD002', 'Петров', 60000.00),
('ORD003', 'Сидорова', 30000.00);
```

Практические задания

Задание 1: Базовые операции с транзакциями

1.1 Простая транзакция с COMMIT

```
-- Начало транзакции
BEGIN;
```

```sql
-- Изменение данных
UPDATE bank_accounts
SET balance = balance + 10000
WHERE account_number = 'ACC001';

-- Подтверждение транзакции
COMMIT;

-- Проверка изменения
SELECT * FROM bank_accounts WHERE account_number = 'ACC001';
```

1.2 Транзакция с ROLLBACK

```sql
BEGIN;

-- Сохраняем исходное состояние
SELECT balance FROM bank_accounts WHERE account_number = 'ACC002';

-- Пытаемся выполнить некорректную операцию
UPDATE bank_accounts
SET balance = balance - 50000  -- Пытаемся снять больше, чем есть
WHERE account_number = 'ACC002';

-- Откатываем транзакцию
ROLLBACK;

-- Проверяем, что баланс не изменился
SELECT balance FROM bank_accounts WHERE account_number = 'ACC002';
```

Задание 2: Транзакция с несколькими операциями

2.1 Перевод денег между счетами

```sql
BEGIN;

-- Снимаем деньги с первого счета
UPDATE bank_accounts
SET balance = balance - 5000,
    last_updated = CURRENT_TIMESTAMP
WHERE account_number = 'ACC001';
```

```sql
-- Фиксируем операцию в журнале
INSERT INTO account_transactions
(account_id, transaction_type, amount, description)
VALUES
(1, 'withdrawal', 5000, 'Перевод на счет ACC002');

-- Добавляем деньги на второй счет
UPDATE bank_accounts
SET balance = balance + 5000,
    last_updated = CURRENT_TIMESTAMP
WHERE account_number = 'ACC002';

-- Фиксируем операцию в журнале
INSERT INTO account_transactions
(account_id, transaction_type, amount, description, related_account_id)
VALUES
(2, 'deposit', 5000, 'Перевод со счета ACC001', 1);

COMMIT;

-- Проверяем балансы
SELECT account_number, balance FROM bank_accounts
WHERE account_number IN ('ACC001', 'ACC002');
```

Задание 3: Использование SAVEPOINT

3.1 Транзакция с точками сохранения

```sql
BEGIN;

-- Операция 1
UPDATE bank_accounts
SET balance = balance + 5000
WHERE account_number = 'ACC003';

-- Создаем точку сохранения
SAVEPOINT after_first_update;

-- Операция 2 (с ошибкой)
UPDATE bank_accounts
```

```sql
SET balance = balance - 60000  -- Пытаемся снять слишком много
WHERE account_number = 'ACC004';

-- Если операция 2 некорректна, откатываемся к точке сохранения
ROLLBACK TO after_first_update;

-- Продолжаем с корректной операцией
UPDATE bank_accounts
SET balance = balance - 5000
WHERE account_number = 'ACC004';

COMMIT;
```

Задание 4: Вложенные транзакции через SAVEPOINT

4.1 Сложная бизнес-логика с откатами

```sql
BEGIN;

-- Операция 1: Резервирование товара
UPDATE products
SET reserved_quantity = reserved_quantity + 1
WHERE product_id = 1;

SAVEPOINT reserve_product;

-- Операция 2: Создание заказа
INSERT INTO orders (order_number, customer_name, total_amount)
VALUES ('ORD004', 'Новый клиент', 50000.00);

SAVEPOINT create_order;

-- Операция 3: Списание денег
UPDATE bank_accounts
SET balance = balance - 50000
WHERE account_number = 'ACC005';

-- Проверяем, что баланс не отрицательный
DO $$
BEGIN
    IF (SELECT balance FROM bank_accounts WHERE account_number =
```

```
'ACC005') < 0 THEN
        RAISE EXCEPTION 'Недостаточно средств';
    END IF;
END $$;

-- Если все ок, коммитим
COMMIT;
```

Задание 5: Обработка ошибок в транзакциях

5.1 Транзакция с обработкой исключений

```
BEGIN;

-- Пытаемся выполнить несколько операций
UPDATE bank_accounts SET balance = balance - 10000 WHERE account_number
= 'ACC001';
UPDATE bank_accounts SET balance = balance + 10000 WHERE account_number
= 'ACC999'; -- Несуществующий счет

-- Если возникла ошибка
EXCEPTION WHEN OTHERS THEN
    -- Откатываем транзакцию
    ROLLBACK;
    -- Выводим сообщение об ошибке
    RAISE NOTICE 'Ошибка: %', SQLERRM;
END;
```

Задание 6. Самостоятельно

6.1 Реализация паттерна "Корзина покупок"

Создайте транзакцию, которая:

1. Резервирует товары на складе
2. Создает заказ
3. Списывает деньги со счета покупателя
4. Если любой шаг не удается, откатывает все изменения

Контрольные вопросы

1. Что такое транзакция в контексте баз данных?

2. Объясните каждое из свойств ACID своими словами.
3. В чем разница между COMMIT и ROLLBACK?
4. Что такое точка сохранения (SAVEPOINT) и для чего она используется?
5. Что произойдет, если не завершить транзакцию явно?
6. Как можно обнаружить "подвисшие" транзакции?
7. В чем разница между пессимистичной и оптимистичной блокировкой?
8. Что такое взаимоблокировка (deadlock) и как ее избежать?

Требования к отчету

1. **SQL-скрипты** всех выполненных заданий
2. **Скриншоты** выполнения команд с результатами
3. **Объяснение** наблюдаемых эффектов в каждом задании
4. **Ответы на контрольные вопросы**
5. **Анализ** проблем, возникших при выполнении заданий