Data Engineering 0: основы баз данных и SQL - Семинар 4

Небольшой оффтоп: Подходы к репликации данных

Интро на 10-15 минут про подходы к репликации данных:

- Слепок данных раз в какое-то время
- Сохранять в несколько мест одновременно изменения данных в приложении
- Отправлять в асинхронно изменения в очередь сообщений, чтобы они когда-то потом применились во втором хранилище
- Можно копировать изменения не на уровне приложения, а на уровне БД. Это происходит через отправку изменений WAL в зависимые хранилища

ACID

- Atomicity (Атомарность)
  Транзакция либо выполнилась целиком, либо отменилась целиком
- Consistency (Согласованность)
  Транзакция не может привести БД из корректного состояния в некорректное
- Isolation (Изолированность)
  Транзакции не мешают друг другу
- Durability (Долговечность)
  После выполнения транзакции, изменения точно сохраняются навсегда

WAL – Write Ahead Log

- механизм ведения журнала изменений данных. Обеспечивает D из аббревиатуры ACID.
Принцип: все изменения данных сначала пишутся в журнал/лог, а только потом применяются к основным данным.

Зачем нужен:

1. По WAL можно восстановить БД в случае сбоя
2. WAL можно использовать для репликации данных между БД
3. Производительность – запись в WAL быстрее изменения самих данных

MVCC – Multi-Version Concurrency Control

- механизм управления параллелизмом в PostgreSQL. Помогает избавиться от лишних блокировок в случае параллельных транзакций. Работает через хранение нескольких версий одних и тех же данных.

Принципы:

1. Каждая строка в БД может иметь несколько версий. Когда транзакция изменяет строку, она создает новую версию этой строки, а старая версия остается доступной для других транзакций, которые начали до изменения.
2. Снимки / Snapshots. Каждая транзакция видит состояние данных на момент ее старта.
3. Старые версии строк удаляются через autovacuum. Чистим ненужное и улучшаем производительность

Autovacuum

- фоновый процесс в PostgreSQL. Чистит ненужные данные в БД, улучшая ее перформанс.

Назначение:

1. Очистка старых версий строк. Они появляются из-за MVCC.
2. Анализ таблиц, обновление статистики использования и перестройка планов запросов
3. Спасает от переполнения транзакций

Как они работают в команде

1. Транзакция меняет данные
2. MVCC создает новую версию строки
3. В WAL записывается изменение
4. Регулярно БД создает контрольную точку (checkpoint). В этот момент все изменения WAL применяются к реальным данным. После этого часть данных из WAL становится ненужными. По умолчанию, такое происходит раз в 5 минут.
5. Autovacuum удаляет уже не использующиеся версии данных

Демонстрация разницы между READ COMMITED и REPEATABLE READ

Поднимаем локально PostgreSQL и подключаемся к нему по инструкции из семинара 2.

Создаем 2 таблицы: accounts и transactions. В таблице accounts будем вести текущие балансы пользователей, а в таблицу транзакций будем дописывать информацию про обработанные переводы.

```
CREATE TABLE accounts (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    balance DECIMAL(10, 2) NOT NULL
);

CREATE TABLE transactions (
    id SERIAL PRIMARY KEY,
    from_account INT REFERENCES accounts(id),
    to_account INT REFERENCES accounts(id),
    amount DECIMAL(10, 2) NOT NULL,
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

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

В этом примере при параллельном запуске транзакций size у transactions сначала X записей, а затем X + 1. Баланс у Алисы тоже меняется

```
BEGIN;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

UPDATE accounts
SET balance = balance - 200.00
WHERE name = 'Alice';

SELECT PG_SLEEP(4);

UPDATE accounts
SET balance = balance + 200.00
WHERE name = 'Bob';

INSERT INTO transactions (from_account, to_account, amount)
VALUES ((SELECT id FROM accounts WHERE name = 'Alice'),
        (SELECT id FROM accounts WHERE name = 'Bob'),
        200.00);

COMMIT;
```

```
BEGIN;

SET TRANSACTION ISOLATION LEVEL READ COMMITTED ;

DO $$
DECLARE
    account_name VARCHAR := 'Alice';
    transfer_amount DECIMAL := 100.00;
    current_balance DECIMAL;
    transactions_count DECIMAL;
BEGIN

    FOR i IN 1..10 LOOP

        SELECT count(*) INTO transactions_count
        FROM TRANSACTIONS;
        RAISE NOTICE 'Количество записей в transactions: %', transactions_count;

        SELECT balance INTO current_balance
        FROM ACCOUNTS
        WHERE name = 'Alice';
        RAISE NOTICE 'Баланс Алисы: %', current_balance;

        SELECT PG_SLEEP(1) INTO account_name;

    END LOOP;

END;
$$ LANGUAGE plpgsql;

COMMIT;
```

В этом примере при параллельном запуске транзакций size у transactions всегда X записей.
При повторном запуске уже становится X + 1 запись

```
BEGIN;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Сначала уменьшаем баланс Алисы
UPDATE accounts
SET balance = balance - 200.00
WHERE name = 'Alice';

SELECT PG_SLEEP(4);

-- Затем увеличиваем баланс Боба
UPDATE accounts
SET balance = balance + 200.00
WHERE name = 'Bob';

-- Записываем транзакцию
INSERT INTO transactions (from_account, to_account, amount)
VALUES ((SELECT id FROM accounts WHERE name = 'Alice'),
        (SELECT id FROM accounts WHERE name = 'Bob'),
        200.00);

COMMIT;
```

```
BEGIN;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;

DO $$
DECLARE
    account_name VARCHAR := 'Alice';
    transfer_amount DECIMAL := 100.00;
    current_balance DECIMAL;
    transactions_count DECIMAL;
BEGIN

    FOR i IN 1..10 LOOP

        SELECT count(*) INTO transactions_count
        FROM TRANSACTIONS;
        RAISE NOTICE 'Количество записей в transactions: %', transactions_count;

        SELECT balance INTO current_balance
        FROM ACCOUNTS
        WHERE name = 'Alice';
        RAISE NOTICE 'Баланс Алисы: %', current_balance;

        SELECT PG_SLEEP(1) INTO account_name;

    END LOOP;

END;
$$ LANGUAGE plpgsql;

COMMIT;
```

ДЗ

В рамках этого домашнего задания нужно показать умение работать с уровнями изоляции транзакций. Оформить ДЗ нужно в виде PDF-файла с запросами к БД, скринами результатов их выполнения и Вашими текстовыми комментариями.

Задача 1
Нужно показать разницу между REPEATABLE READ и SERIALIZABLE. В качестве примера описания можно ориентироваться на демонстрацию с семинара (описана выше)

Задача 2
В рамках задачи 2 вам нужно будет работать с моделью данных интернет-магазина, которую вы описывали ранее. Для каждого пункта нужно написать sql-запрос и выбрать минимально необходимый уровень изоляции транзакций и обосновать свой выбор:

1. **Получение товаров с пагинацией**
   Пользователь листает товары по категории с разбивкой на страницы (например, LIMIT + OFFSET).
   a. Может ли "скакать" пагинация, если другой пользователь будет удалять/добавлять товары
   b. Какой уровень изоляции нужен?
2. **Получение содержимого корзины и его итоговой суммы**
   Пользователь смотрит, что у него в корзине и сколько это будет стоить.
   a. Другой пользователь одновременно с этим может изменить цену какого-то товара из корзины
   b. Какой уровень изоляции минимально необходим?
3. **Проверка наличия товара перед оформлением заказа**
   Приложение проверяет, что нужное количество товара есть в наличии, и затем создаёт заказ.
   a. Два пользователя одновременно заказывают один и тот же товар, хотя в наличии всего одна штука
   b. Какой уровень изоляции предотвратит overselling (продажа товаров больше, чем есть на самом деле)?
4. **Агрегирование данных (например, общее количество товаров в категории) Вы делаете SELECT COUNT(*) по фильтру.**
   a. Одновременно с этим другой пользователь удаляет/добавляет товары в категорию
   b. Какой уровень изоляции даст вам точный, воспроизводимый результат?