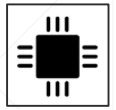
CTE запросы как основа бизнес-логики

Камиль Исламов

План



- Сравнение некоторых вариантов обработки данных

- Возможности Common Table Expressions у PostgreSQL

- Варианты использования CTE в хранимых процедурах

- Реализация бизнес-логики на многоуровневых CTE

- Итоги

Некоторые способы многоэтапной обработки данных в приложениях

- Получение и сохранение данных из БД через ORM, обработка в приложении

- Получение и сохранение данных SQL-запросами, обработка в приложении

- Получение, обработка и сохранение данных в коде хранимых процедур, вызов процедур в приложении

- Получение, обработка и сохранение данных с применением CTE SQL запросов, вызываемых в приложении

Некоторые преимущества ORM

- Можно работать с БД без навыков работы с SQL
- Унификация кода приложения при обращении к БД
- Теоретическая возможность игнорировать особенности конкретной БД
- Минимизация трудозатрат при миграция между СУБД
- Простота обучения персонала и развёртывания
- Привычные методологии работы с контролем версий

Некоторые преимущества работы без ORM





- Выигрыш производительности сложных запросов
- Оптимизация обращений к БД
- Уменьшение кода в приложениях при использовании логики в БД
- Использование специфических преимуществ СУБД
- Возможность изменения логики без изменения программного кода
- Возможность распределение ресурсов при разработке кода (приложение – БД)

Варианты обработки данных в функциях

- Отдельные SQL запросы для каждого элемента логики

- Оптимизация SELECT запросов с применением CTE

- Использование CTE запросов для некоторых изменений данных

- Обширное использование CTE запросов на изменение данных

- Разработка единого запроса-транзакции на основе CTE-цепочек

Возможности и области применения CTE

- Один из вариантов оптимизации SQL запросов

- Удобная альтернатива временным таблицам

- Возможность упорядочивания сложных SQL запросов на блоки

- Один из способов сокращения объёма SQL кода

- Возможность эффективного распределения этапов обработки данных

Дополнительный функционал CTE PostgreSQL

- Функционал *Рекурсии*

- Нет необходимости в управлением «материализацией»

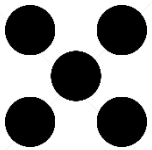
- *Insert* функционал, возвращающий вновь созданные строки

- *Delete* функционал, возвращающий удалённые строки

- *Update* функционал, возвращающий изменённые строки

- *On Conflict* функционал, возвращающий задействованные строки

Описание идеологии CTE-цепочек



- Подготовка входящих данных
  - Предварительное преобразование форматов данных
  - Предварительная проверка корректности данных

- Сбор задействованных существующих данных
  - Запросы данных из справочников
  - Дополнение данными из соответствующих Foreign таблиц

- Выполнение основных бизнес-операций
  - Промежуточные блоки подготовки данных
  - Основные действия по изменению данных

- Завершающие бизнес-операции
  - Журналирование выполненных действий
  - Подготовка комплекта выходных данных

Варианты обмена данными между CTE блоками

- Простые результаты SELECT запросов из таблиц

- Обработанные результаты на основе предыдущих CTE блоков

- Виртуальные таблицы из UNNEST массивов входящих параметров

- Вновь созданные строки из INSERT => Returning запросов

- Только что удалённые строки из DELETE => Returning запросов

- Обновлённые строки из UPDATE => Returning запросов

- Задействованные строки из ON CONFLICT запросов

Преимущества CTE в функциях

- Удобный способ передавать в параметры таблицы целиком
  - Подготовить комлект массивов в приложении
  - Каждый столбец как параметр типа массив (int[], text[], …)
  - CTE запрос вида *as (**SELECT** unnest(col1), unnest(col2))*

- Преобразование переменных-массивов в виртуальные таблицы
  - Подготовить один или несколько переменных-массивов
  - Обеспечить одинаковую размерность
  - CTE запрос вида *as (**SELECT** unnest(array1), unnest(array2))*

CTE в функциях: работа с JSON

- Преобразование входящего единого JSON-параметра
  - Предварительная сборка требуемых полей из JSON
  - Конструкция *json_to_record(json) as (col1 int, col2 text)*
  - Формирование виртуальной таблицы из *as(select col1, col2 from json_to_record)*

- Подготовка возвращаемой JSON-переменной
  - Формирование соответствующих CTE блоков
  - Каждая строка из CTE блока «оборачивается» *row_to_json()*
  - Единый JSON формируется через *json_build_object()* или *json_agg()*

- Промежуточные преобразования данных в CTE цепочке
  - Возможность группировки, сортировки и фильтрации в отдельный CTE блок

CTE при обработке JSON-данных приложения

- Архитектура при «классической» парадигме через переменные
  - JSON ⇨ Разбор в приложении ⇨ Подстановка под переменные функции ⇨ Выполнение

- Архитектура на базе преобразования CTE
  - JSON ⇔ Вызов функции ⇔ Выполнение CTE *json_to_record(json) as (col1 int, col2 text)*

Преобразование JSON при использовании CTE

***json_to_record**(json) as (col1 **int**, col2 **text**)*

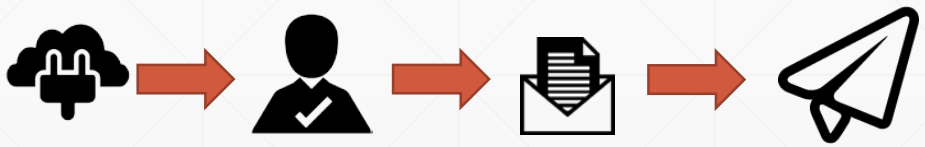
- Системное преобразование данных без «ручного» *cast* через ***::***
  - Преобразование даты
  - Поддержка JSON-совместимых переменных

- Возможность системного преобразования массива в таблицу
  - Поддерживается массивные типы, вида *text[]*, *real[]*
  - *UNNEST* для соответствующих переменных JSON массива
  - Преобразование в таблицу CTE запросом вида

```
with virt1 as (
    SELECT UNNEST(jsar1) col1 FROM json_to_record(json) as (jsar1 text[])
)
```

SQL функция на чтение на базе CTE и JSON

1. Приложение формирует JSON параметр с текущими данными клиента
2. CTE блок преобразования входящего JSON-параметра
3. Цепочка SELECT блоков для подготовки результирующего JSON
4. CTE блок журналирования и финальный SELECT



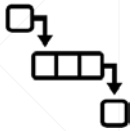
SQL функция на изменение на базе CTE и JSON

1. Приложение формирует JSON параметр с новыми данными
2. CTE блок преобразования входящего JSON-параметра
3. Цепочка UPDATE/INSERT/ON CONFLICT блоков для изменения данных
4. Цепочка SELECT блоков для подготовки результирующего JSON

Применение CTE цепочек в функциях

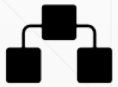
- *SQL функции* в виде одной CTE цепочки
  - Простые Read-only методы некритичных операций
  - Вспомогательный атомарный функционал внутри транзакции
  - Основной функционал приложения без применения сложной логики
- *PL/PGSQL функции* с несколькими CTE цепочками
  - Для широкого применения констант внутри функции
  - В случаях, когда цепочки не линейны и выполняют разные сценарии
  - Требуется использование массивов с промежуточной обработкой

SQL функции в виде одной CTE цепочки



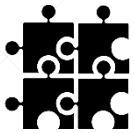
- Концептуальная архитектура реализации
  - Create Function language SQL
  - Одна CTE цепочка, выполняющая взаимозависимые действия

- Преимущества
  - Возможность запуска запроса в приложении без функции
  - Повышенная производительность за счёт минимума SQL операций



- Особенности
  - Усложнение цепочек для вариантных логических схем обработки данных
  - Невозможность эффективно формировать и обрабатывать массивы

PG/PLSQL функции с несколькими CTE цепочками

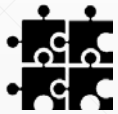


- Концептуальная архитектура реализации
  - Классическая PG/PLSQL функция с необходимыми переменными
  - Отдельные CTE цепочки, выполняющая соответствующий функционал

- Преимущества
  - Использование оптимизированных CTE цепочек с удобным обменом данными
  - Уменьшение количества SQL операций благодаря CTE цепочкам
  - Удобство чтения и сопровождения кода при оптимальной архитектуре цепочек

- Особенности
  - Необходимость оптимизации CTE цепочек в разрезе функциональности

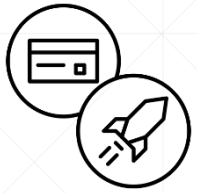
Логика работы взаимосвязанных CTE цепочек



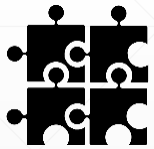
Этап обработки	Описание	Результат
Адаптация входящих данных	Преобразование JSON, массивов и текстовых переменных в требуемые	CTE блоки, вычисленные на базе входящих JSON, массивов, массивов JSON и текста
Получение требующихся существующих данных	Обращение к существующим таблицам для валидации и последующей обработки данных	CTE блоки, сформированные из предыдущих при объединении с данных из существующих таблиц
Обработка всех необходимых данных	Формирование необходимых CTE блоков, реализующих основную бизнес-логику	CTE блоки с подготовленными и обработанными данными
Подготовка результатов	Формирование JSON или SET of Records	Не-CTE запросы, содержащие JSON или результирующий SQL

CTE цепочки как основа бизнес-логики

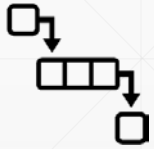
- Возможность сформировать транзакцию как один SQL запрос



- CTE цепочка как SQL функция
  - Планирование цепочки движения данных
  - Реализация бизнес логики с множеством взаимосвязанных действий
  - Высокая эффективность для логики с линейным алгоритмом



- CTE цепочки внутри PL/PGSQL функций
  - Возможность оптимизации многокомпонентной бизнес-логики по блокам
  - Удобный инструмент работы с массивами и JSON
  - Эффективная альтернатива созданию временных таблиц



Преимущества использования CTE цепочек

- Удобный инструмент оптимизации любых SQL запросов

- Бизнес-логика реализована преимущественно в приложении
  - Оптимизация структуры и объёмов SQL кода
  - Широкое применение представлений на базе CTE цепочек
  - Минимизация обращений к БД с применением не Read-only цепочек

- Бизнес-логика реализована в БД
  - Сокращение количества PL/PGSQL функций в пользу SQL функций
  - Оптимизация применения курсоров и переменных в PL/PGSQL функциях
  - Внедрение эффективные решений с применением массивов

Пример логики CTE цепочки в SQL функции

1. Блок преобразует JSON в строку таблицы

2. SELECT блок запрашивает существующие данные по ИД из блока 1

3. ON CONFLICT блок создаёт или обновляет строку по данным из блока 1 и 2

4. UPDATE блок актуализирует глобальную статистику по данным из блока 3

5. INSERT блок создаёт запись в журнале по данным из блока 3 и 4

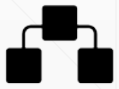
6. SELECT блок формирует результирующий JSON по данным из блоков 4 и 5

Использование SQL функций как CTE цепочки

- Для каждой транзакции запускается одна SQL функция

- При необходимости, CTE блоки функции задействуют дополнительные функции, возвращающие значение или Set of Record (CTE блок)

- Преимущества
  - Минимальное количество SQL операций
  - Возможность унификации кода для схожих функциональностей



- Варианты масштабирования
  - Вызов удалённых процедур, например PL-Proxy
  - Использование FDW для прямого доступа к разным инстансам, БД

Возможности CTE цепочек для приложений

- Множество функциональных действий в одной SQL операции
  - Базовое журналирование

  - Реализация бизнес-логики с минимальным обменом данных с приложением
  - Оптимизация стратегии обращения к БД

- Сокращение времени блокировок в БД за счёт перераспределения промежуточных обработок данных между приложением и БД

- Оптимизация архитектуры приложения
  - Освобождение приложения от функций, оптимизированных для БД
  - Сокращение количества кода

Особенности приложений, применяющих CTE

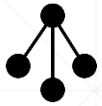
- Проектирование с учётом оптимизации транспортирования данных
  - Логическая упаковка обращений к БД в единую транзакцию-запрос
  - Проектирование архитектуры приложения с минимумом переноса данных

- Эффективное перераспределения функций БД/приложение
  - Удобная обработка массивов возможностями БД
  - Оптимизация возможных преобразований типов данных
  - Использование возможностей JSON для обмена данными

  - Выполнение финансово-критичных операций в единой среде, оптимизируя риски утечки данных и неточности округлений

Преимущества CTE для реализации бизнес-логики

- Эффективный инструмент оптимизации SELECT SQL запросов



- Оптимизация использования памяти при внедрении CTE цепочек

- Дополнительные возможности бизнес-логики UPD, DEL, INS цепочек CTE

- Повышение производительности за счёт перехода на SQL-функции

- Возможность использования парадигмы «Транзакция как SQL-запрос»

Вопросы

- kam@kambox.ru

- @ikamil