

Хранимые процедуры

Хранимые процедуры в PostgreSQL - это функции, содержащие один или несколько SQL-запросов и дополнительной логики на языке PL/pgSQL (или другом поддерживаемом процедурном языке). Эти процедуры "хранятся" в базе данных и могут быть вызваны для выполнения определенных операций именно над данными в базе данных. Это чем-то похоже на функции в языках программирования, только в рамках БД.

Как работают хранимые процедуры?

1. Исполнение сервером баз данных:

Хранимые процедуры выполняются сервером базы данных, что означает, что вся обработка запросов происходит на сервере, а не на клиенте. Это снижает нагрузку на сеть и может улучшить общую производительность за счет сокращения количества передаваемых данных.

2. Возможность изменения данных и структуры БД:

С помощью хранимых процедур можно выполнять любые операции SQL, включая вставку, обновление, удаление и изменение структуры базы данных. Также можно вызывать другие процедуры из текущей процедуры. Они могут динамически изменять таблицы, создавать или удалять индексы, а также модифицировать схему базы данных.

3. Компиляция процедуры:

Когда хранимая процедура создается, сервер базы данных компилирует ее в промежуточный байт-код или план выполнения. При последующих вызовах хранимой процедуры сервер использует уже скомпилированную версию, что ускоряет ее выполнение.

4. Возможность выборки данных:

Хранимые процедуры могут возвращать наборы данных, подобно тому как это делает оператор SELECT. Они могут возвращать одно значение, ряд значений или целые таблицы, которые затем могут быть использованы в приложении или для дальнейшей обработки.

Дополнительные есть:

- Транзакционный контроль. Хранимые процедуры могут управлять транзакциями, позволяя разработчику контролировать целостность данных. Они могут включать команды COMMIT и ROLLBACK для управления транзакциями внутри процедур.
- Параметризация. Хранимые процедуры могут принимать параметры, что делает их гибкими и мощными инструментами для различных задач. Параметры могут быть использованы для передачи данных в процедуру или для настройки ее поведения.
- Обработка ошибок. В PostgreSQL есть возможности для обработки ошибок внутри хранимых процедур. Можно использовать конструкции TRY...CATCH или EXCEPTION для управления исключениями и определения поведения при возникновении ошибок.

Зачем это нужно?

1. Модульность и повторное использование: Хранимые процедуры позволяют централизовать и стандартизировать логику, что облегчает поддержку и повторное использование кода.

Допустим, вы часто выполняете запрос на обновление зарплаты сотрудников на основе их производительности. Каждый раз вам нужно написать следующий SQL-запрос:

```
UPDATE employees SET salary = salary * 1.1 WHERE performance_rating > 8;
```

Но вы можете просто создать процедуру, которую можно выполнить, и результат будет тот же:

```
CREATE OR REPLACE FUNCTION update_salary_based_on_performance() RETURNS VOID AS $$  
BEGIN  
    UPDATE employees SET salary = salary * 1.1 WHERE performance_rating > 8;  
END;  
$$ LANGUAGE plpgsql;
```

И если нужно будет изменить запрос, вы один раз меняете его в процедуре, а не лезете во все куски кода, где эта процедура вызывается.

2. Улучшение производительности: Поскольку они выполняются на сервере, хранимые процедуры могут уменьшить нагрузку на сеть, уменьшая количество обращений к серверу базы данных.

Предположим, вы регулярно запрашиваете сложный отчет, объединяющий данные из нескольких таблиц. Это требует выполнения многочисленных запросов из приложения, что может быть медленным и неэффективным.

Создайте хранимую процедуру, которая собирает все необходимые данные на стороне сервера:

```
CREATE OR REPLACE FUNCTION generate_complex_report() RETURNS TABLE(...) AS $$
BEGIN
    RETURN QUERY
    SELECT ... FROM ... JOIN ... WHERE ...;
END;
$$ LANGUAGE plpgsql;
```

Теперь вам просто нужно вызвать эту процедуру для получения отчета.

3. Безопасность: Хранимые процедуры могут обеспечить дополнительный уровень безопасности, скрывая сложную бизнес-логику от клиентских приложений и ограничивая доступ к базовым таблицам.

Для получения данных о зарплатах сотрудников пользователи могут напрямую запрашивать данные из таблицы `employees`, что представляет угрозу безопасности.

Создайте хранимую процедуру, которая предоставляет доступ к зарплатам только авторизованным пользователям:

```
CREATE OR REPLACE FUNCTION get_employee_salary(employee_id INT) RETURNS NUMERIC AS $$
BEGIN
    IF check_user_permission(current_user) THEN
        RETURN (SELECT salary FROM employees WHERE id = employee_id);
    ELSE
        RAISE EXCEPTION 'Access denied';
    END IF;
END;
$$ LANGUAGE plpgsql;
```

4. Транзакционное управление: Они позволяют объединять несколько SQL-запросов в одной транзакции, что упрощает управление сложными операциями.

Без хранимых процедур (функций) выполнение нескольких связанных операций обновления, вставки или удаления, требует управления транзакциями в коде приложения.

Объедините все эти операции в одну хранимую процедуру, управляя транзакцией внутри нее:

```
CREATE OR REPLACE FUNCTION process_transaction(...) RETURNS VOID AS $$
BEGIN
    BEGIN TRANSACTION;
    -- Ваши SQL-операции здесь
    COMMIT TRANSACTION;
END;
$$ LANGUAGE plpgsql;
```

Это обеспечивает, что все операции либо выполняются полностью, либо откатываются в случае ошибки, поддерживая целостность данных.

Хранимые процедуры, несмотря на свою мощь и гибкость, имеют ряд недостатков, особенно в некоторых конкретных сценариях. Давайте рассмотрим несколько примеров, где использование хранимых процедур может быть неоптимальным или даже вредным:

1. Сложность обслуживания и отладки

Пример: Разработчик создает сложную хранимую процедуру для расчета финансовых отчетов. Со временем требования меняются, и процедуру нужно модифицировать. Однако из-за сложности и отсутствия хорошей документации обновление и отладка занимает значительное время и ресурсы.

2. Проблемы с масштабируемостью

Пример: В высоконагруженной системе используется хранимая процедура для обработки транзакций. Она содержит сложную логику и блокировки. При увеличении нагрузки процедура становится узким местом, приводя к замедлению работы всей системы и трудностям с масштабированием.

3. Зависимость от конкретной СУБД

Пример: Компания использует хранимые процедуры, специфичные для одной СУБД. При необходимости перехода на другую СУБД возникают сложности с переносом логики, так как она глубоко интегрирована в специфические особенности исходной СУБД.

4. Трудности с версионированием

Пример: В системе контроля версий хранится код приложения, но хранимые процедуры версионизируются отдельно или вообще не версионизируются. Это приводит к проблемам при развертывании новых версий приложения и синхронизации изменений в базе данных.

5. Проблемы с безопасностью

Пример: Хранимая процедура предоставляет доступ к чувствительным данным. Неправильное управление правами доступа к процедуре может привести к утечкам данных или несанкционированному доступу.

6. Сложности в тестировании

Пример: Автоматизированное тестирование приложений часто не включает тестирование хранимых процедур, что может привести к пропуску серьезных ошибок в бизнес-логике, реализованной на стороне базы данных.