**СЛАЙД 3**

С ростом популярности СУБД возникает вопрос об оптимизации ее работы. Так как один из самых распространенных способов увеличения производительности — параллельное выполнение, следует рассмотреть оптимизацию многопоточной программы.

Так, при доступе к БД объемом 100.000 записей многопоточная программа примерно в 1000 раз работает быстрее; в тоже время однопоточная программа показывает нестабильную работу на больших данных.

Несмотря на указанное преимущество многопоточности, узким горлышком все равно остается операция подключения, одна из самых дорогостоящих (т.к. процесс подключения к БД занимает время и память от 2 до 3 МБ).

**СЛАЙД 4**

Данная работа будет основываться на объектно-реляционной СУБД PostgreSQL 14-ой версии, занимающей 4-ое место в рейтинге СУБД на начало 22го года. Выбор аргументирован такими преимуществами как доступность исходного кода и кроссплатформенность.

PostgreSQL содержит инструменты для реализации многопоточности. Однако, обратившись к документации, можно выделить следующее ограничение: «два потока не должны пытаться одновременно работать с одним объектом PGconn. В частности, не допускается параллельное выполнение команд из разных потоков через один объект соединения.»

**СЛАЙД 5**

Одной из наиболее сильных сторон PostgreSQL является архитектура, основанная на модели «клиент-сервер». Выделяют 3 основные подсистемы: клиентская часть, серверная часть и хранилище данных.

Клиентская часть состоит из пользовательского приложения и библиотеки libpq. Данная библиотека содержит набор функций, с помощью которых создается соединение с сервером.

Соединение принимается процессом-демоном *postmaster*, который в дальнейшем с помощью системного вызова *fork()* создаст новый серверный процесс для обслуживания данного соединения, а также ряд служебных процессов.

Третья часть сформирована из хранилища данных и средств его управления.

**СЛАЙД 6**

Далее будут рассмотрены различные подходы повышения эффективности выполнения запроса. 1ый – использование пула соединений. Так как в PostgreSQL отсутствует встроенный пул, внешний может быть реализован на основе средств libpq или в качестве внешней службы.

**СЛАЙД 7**

Преимущество использование пула – это увеличение пропускной способности транзакции до 60%.

Главным недостатком пула на основе библиотеки – затраты на его разработку. В том числе необходимость изменения архитектуры приложения. Недостаток пула в качестве внешней службы – однопоточная реализация самих служб. Встроенный же пул доступен только в коммерческой версии.

**СЛАЙД 8**

Методы, описанные выше, позволяют повысить скорость выполнения запросов путем сокращения числа соединений. Принципиально другим подходом повышения производительности выполнения запроса — его оптимизация в рамках одного потока. Такой оптимизацией может стать распараллеливание запросов.

Распараллеливание — это возможность построения таких планов запросов, которые будут задействовать несколько ядер.

Описанный метод применим только к ограниченному числу запросов (запросы, обрабатывающие большой объемом данных, но возвращающие пользователю всего несколько строк).

3 минуты (3:15)

**СЛАЙД 9**

Разрабатываемый метод состоит из двух этапов обработки запроса: 1ый – отправка запроса серверу, включающий а) формирование очереди запросов и б) отправку запроса из очереди; 2й – получение ответа от сервера. Рассмотрим более подробно каждый из этапов.

**СЛАЙД 10**