Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский

политехнический университет»

(ПНИПУ)

Кафедра вычислительной математики, механики и биомеханики

Разработка высоконагруженных веб-приложений

Отчет по лабораторным работам

Выполнил:

студент гр. ИТСИ-19-1м

Сай Дарья Владимировна

Пермь 2020

# Лабораторная работа №1. Реализация балансировки и отказоустойчивости с использованием Nginx.

Для реализации данной работы было разработано приложение на Java с использованием Spring Boot. Приложение называется demo и находится в одноименной папке в гит-репозитории. Данное приложение при обращении к нему возвращает ответ типа count = 1 (количество обращений к приложению).

Для запуска нескольких приложений, между которыми можно было бы балансировать нагрузку был создан файл "docker-compose.yml", в котором находится описание трех контейнеров с приложением. Чтоб запустить этот файл необходимо,находясь в директиве с файлом, в командной строке написать "sudo docker-compose up".

Само приложение упаковано в Docker-контейнер. Для запуска приложения из контейнера в терминале необходимо ввести команду "sudo docker run -p 8080:8080 c13518c0c795", находясь в директиве с файлом.

В папке Nginx находится конфигурационный файл "Nginx\_Round\_Robin.conf" в котором находится конфигурация для балансировщика нагрузки и Dockerfile для запуска nginx из контейнера.

# Лабораторная работа №2. Реализация разделяемого хранилища данных с использованием Redis

В данной лабораторной работе необходимо реализовать разделяемое хранилище, которое бы хранило разделяемое состояние между экземплярами веб-сервисов.

В ходе данной работы был установлен Redis на Ubuntu. После чего был подключен клиент jedis для работы из веб-приложения из 1 лабораторной работы.

Jedis jedis = new Jedis("localhost", 6379);

Далее идет проверка есть ли ключ "count" в БД, если нет, то создаем запись "jedis.set("count", "0");", если есть, то запрашиваем счетчик, инкрементируем его "Long c = jedis.incr("count");" и выводим его на форму Long c = jedis.incr("count");

Для проверки запускаем два приложения на портах 8081 и 8082, обновляем страницы на обоих приложениях счетчик увеличивается (рисунок 1).

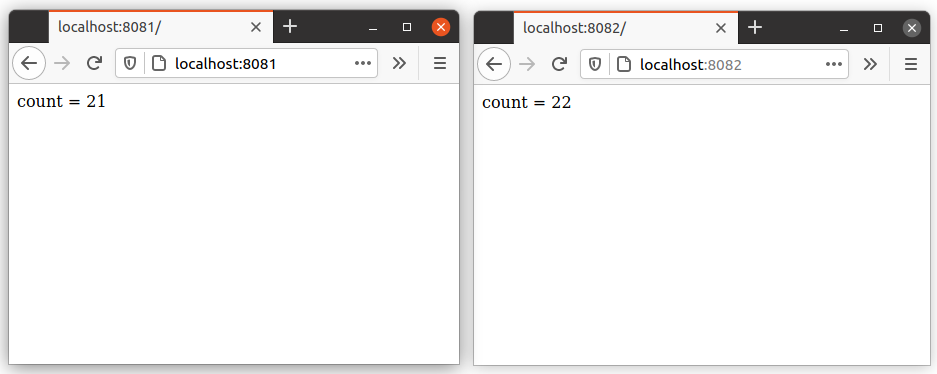
[](https://github.com/TudvasevaD/cloud_web/blob/master/lab2/1.png)

Рисунок 1. Работа двух приложений с разделяемым хранилищем

# Лабораторная работа №3. Реализация балансировки нагрузки с использованием очередей Kafka

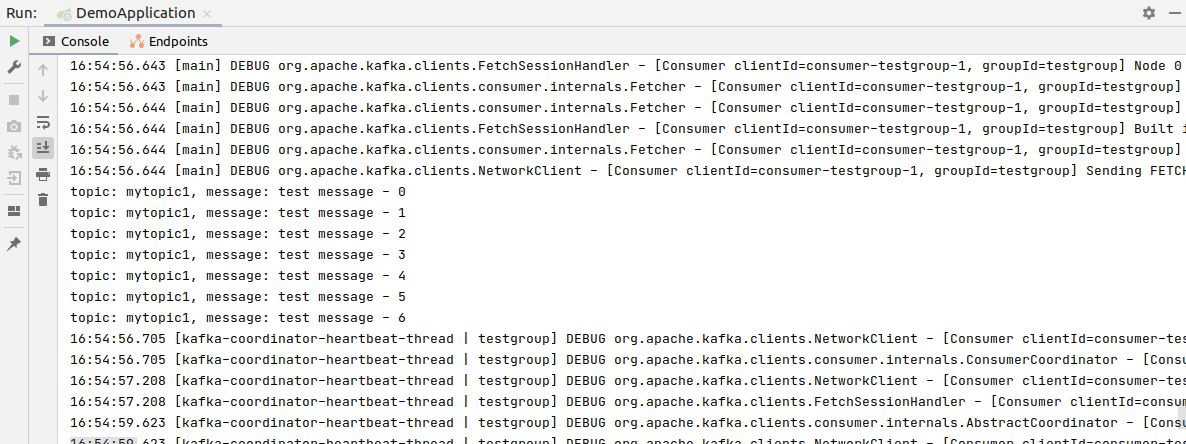
Основная цель – рассмотреть применимость очередей для ограничения максимального количества входящих запросов в единицу времени.

Для реализации данной работы был использован Docker-compose файл c Kafka. Так же было создано два приложения:

* demo – Producer – с неравномерной скоростью посылает сообщения в очередь;
* demo1 – Consumer – читает из очереди сообщения с определенными промежутками времени.

Необходимо было запустить докер-файл с Кафкой, затем demo и demo1. На рисунках ниже представлен вывод в консоль во время работы данных приложений.

Producer: [](https://github.com/TudvasevaD/cloud_web/blob/master/lab3/prod.png)

Consumer: [](https://github.com/TudvasevaD/cloud_web/blob/master/lab3/cons.png)

# Лабораторная работа №4. Реализация партиционирования с использованием Postgres

В данной лабораторной работе рассматривается партиционирование таблиц на примере СУБД Postgres.

В ходе данной работы была установлена СУБД Postgres и создана БД mydb.

Далее была создана таблица measurement:

CREATE TABLE measurement ( city\_id int not null, logdate date not null, peaktemp int, unitsales int ) PARTITION BY RANGE (logdate);

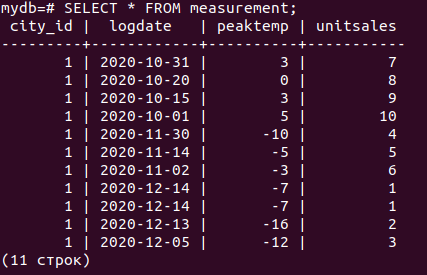
После чего были созданы партиции по месяцам: октябрь, ноябрь и декабрь:

CREATE TABLE measurement\_y2020m10 PARTITION OF measurement FOR VALUES FROM ('2020-10-01') TO ('2020-11-01');

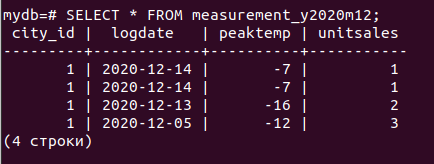
CREATE TABLE measurement\_y2020m11 PARTITION OF measurement FOR VALUES FROM ('2020-11-01') TO ('2020-12-01');

CREATE TABLE measurement\_y2020m12 PARTITION OF measurement FOR VALUES FROM ('2020-12-01') TO ('2021-01-01');

Таблица были заполнены данными, по запросы у на выборку пришел следующий ответ:

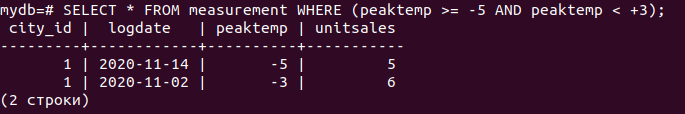
[](https://github.com/TudvasevaD/cloud_web/blob/master/lab4/2.png)

Если сделать запрос на выборку данных только из партиции за декабрь получим следующий результат:

[](https://github.com/TudvasevaD/cloud_web/blob/master/lab4/3.png)

Запрос на выборку данных с температурой от +3 до -5:

SELECT \* FROM measurement WHERE (peaktemp >= -5 AND peaktemp < +3);

[](https://github.com/TudvasevaD/cloud_web/blob/master/lab4/4.png)

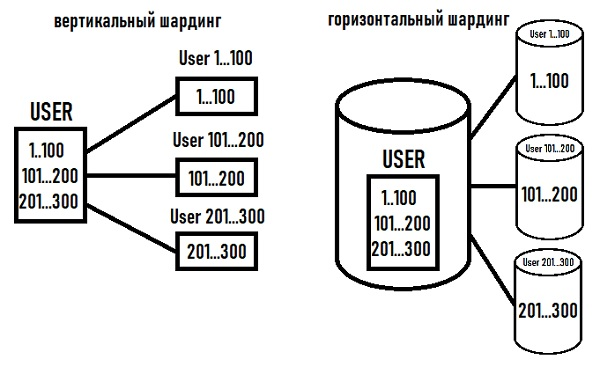
# Доклад. Оптимизация БД

СУБД часто является «узким местом» в производительности веб-приложений, что влияет на быстродействие и устойчивость к высоким нагрузкам. Т.е. на сервер БД может приходить много запросов от различных клиентов и сервер просто не справляется с нагрузкой. Для решения данной проблемы производится оптимизация путем масштабирования.

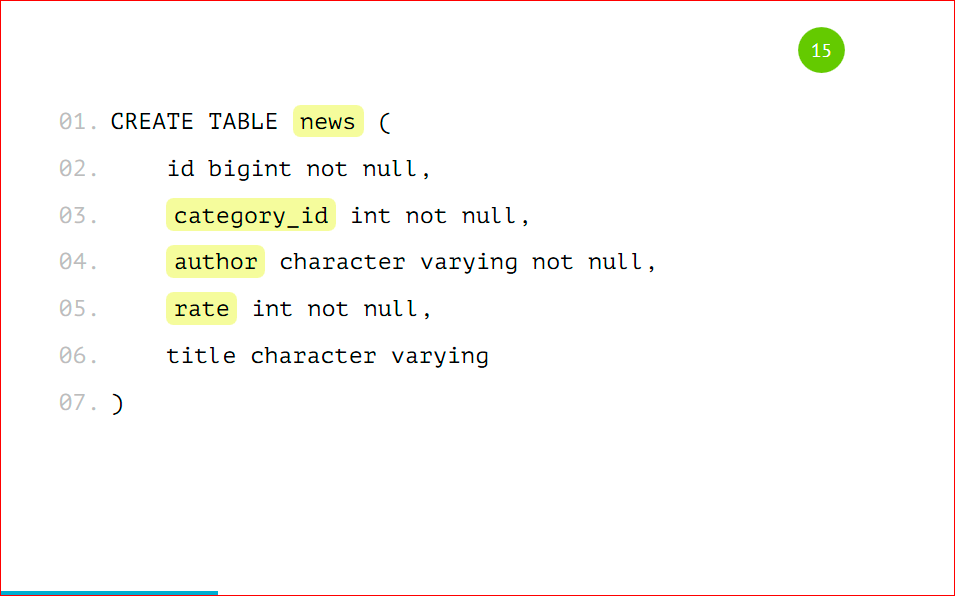
Партиционирование (partitioning) – разбиение таблиц на логические части по выбранным критериям. Делит весь объем операций по обработке данных на несколько независимых и параллельно выполняющихся потоков, что существенно ускоряет работу СУБД. Для правильного конфигурирования параметров партиционирования необходимо, чтобы в каждом потоке было примерно одинаковое количество записей. Например, на новостных сайтах имеет смысл партиционировать записи по дате публикации, так как свежие новости на несколько порядков более востребованы и чаще требуется работа именно с ними, а не со всех архивом за годы существования новостного ресурса.

Партицирование "вертикальный шардинг" - разбитие таблицы базы данных на несколько, по какому-либо принципу. Самый простой пример, у вас база данных ваших пользователей, которую надо разбить (см. картинку ниже). Что нам это дает: прирост в 3-4 раза, просто в исполнении, один сервер.

Партицирование "горизонтальный шардинг" - тоже самое, только таблички лежат в разных базах на других инстансах (это виртуальная машина, которая запускается и работает в облаке). Что нам это дает: сложнее первого варианта, разные сервера. Единственное отличие горизонтального от вертикального в том, что горизонтальное будет разносить данные по разным серверам.

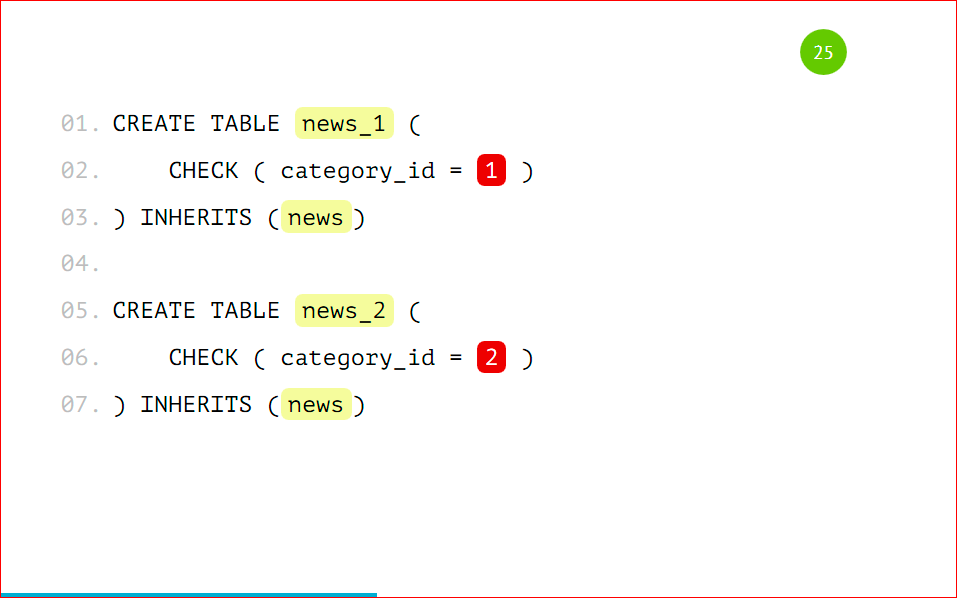


Рассмотрим пример горизонтального шардинга на примере таблицы Новости в БД Postgres. На рисунке ниже вы видите изначальную структуру данной таблицы, у новости есть идентификатор, есть категория, в которой эта новость расположена, есть автор новости, ее рейтинг и какой-то заголовок.



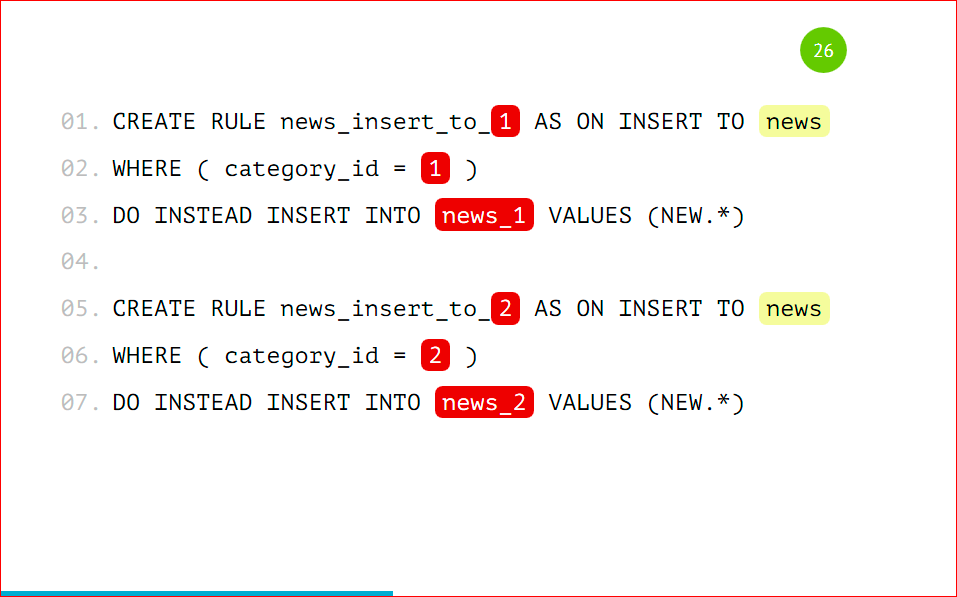
Для разделения данной таблицы на несколько нужно будет сделать 2 действия — это поставить у нашего шарда, например, news\_1, то, что она будет наследоваться таблицей news. News будет базовой таблицей, будет содержать всю структуру, и создавая партицию, будем указывать, что она наследуется нашей базовой таблицей. Наследованная таблица будет иметь все колонки родителя — той базовой таблицы, которую мы указали, а также она может иметь свои колонки, которые мы дополнительно туда добавим. Она будет полноценной таблицей, но унаследованной от родителя, и там не будет ограничений, индексов и триггеров от родителя — это очень важно. Если вы на базовой таблице насоздаете индексы и унаследуете ее, то в унаследованной таблице индексов, ограничений и триггеров не будет.

Далее нужно поставить ограничения. Это будет проверка, что в новую таблицу news\_1 будут попадать только данные, у которых category\_id = 1.

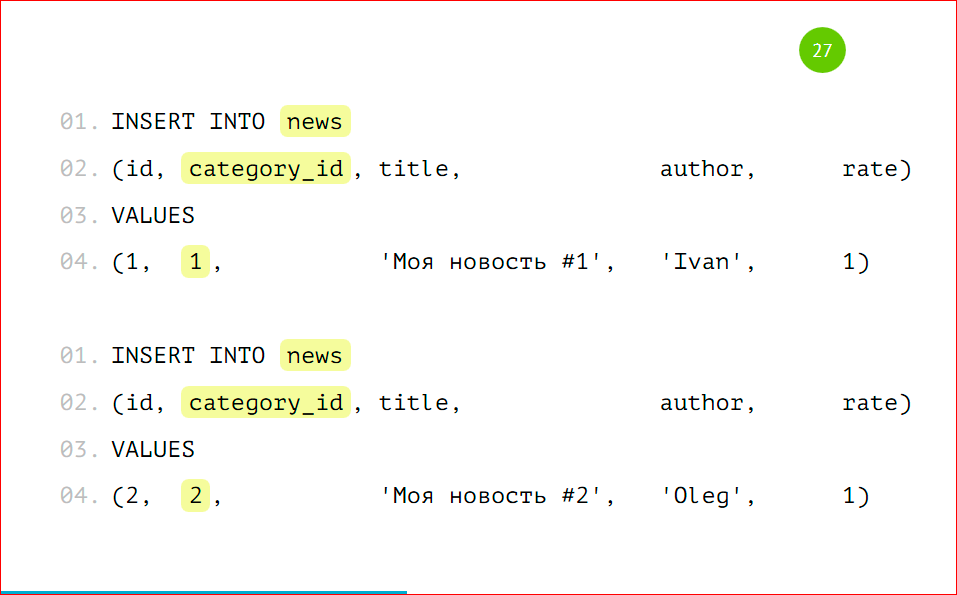


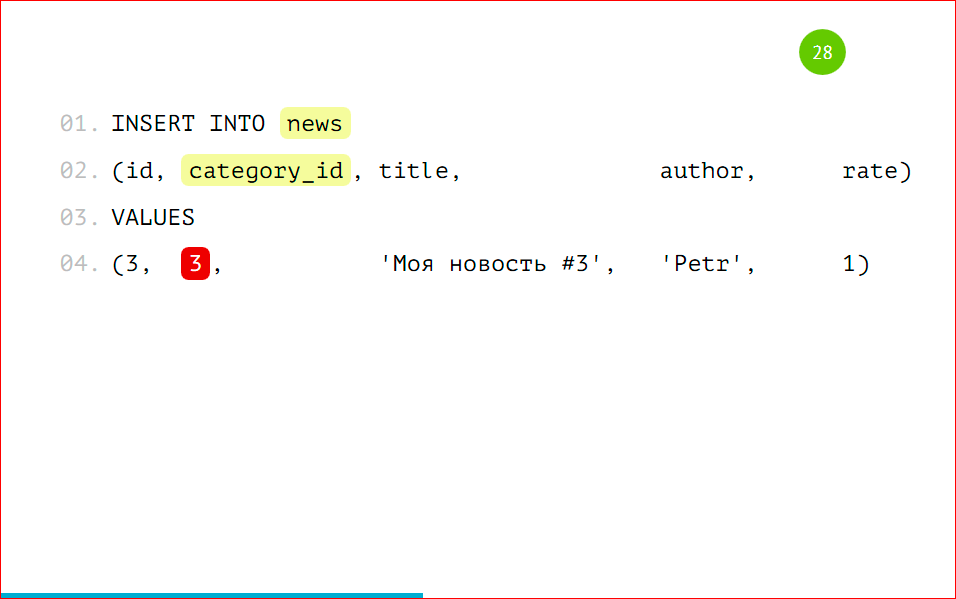
Не стоит создавать партиции по разным полям, т.е. что в 1-ую партицию у нас будут попадать записи с category\_id=1, а во 2-ую — по рейтингу.

Далее на базовую таблицу должны добавить некоторое правило, чтобы, когда мы будем работать с нашей основной таблицей news, вставка на запись с category\_id = 1 попала в нужную партицию. Мы указываем простое правило, называем его как хотим, говорим, что, когда данные будут вставляться в news с category\_id = 1, вместо этого будем вставлять данные в news\_1.



Данные будем вставлять как обычно, будто у нас обычная большая толстая таблица, т.е. мы вставляем запись с category\_id=1 с category\_id=2, можем даже вставить данные с category\_id=3.





Выборка данных:

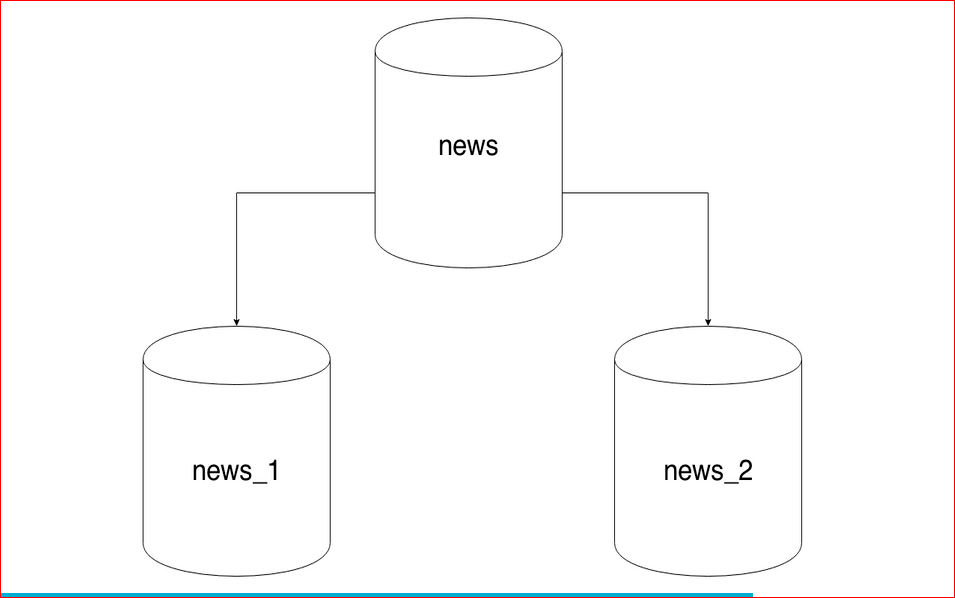


К недостаткам данного метода относится то, что индексы, созданные на основной таблице, не будут унаследованы в дочерней таблице нашей партиции. Это значит, что придется заводить одинаковые индексы на всех партициях, все ограничения, все триггеры дублировать на все таблицы.

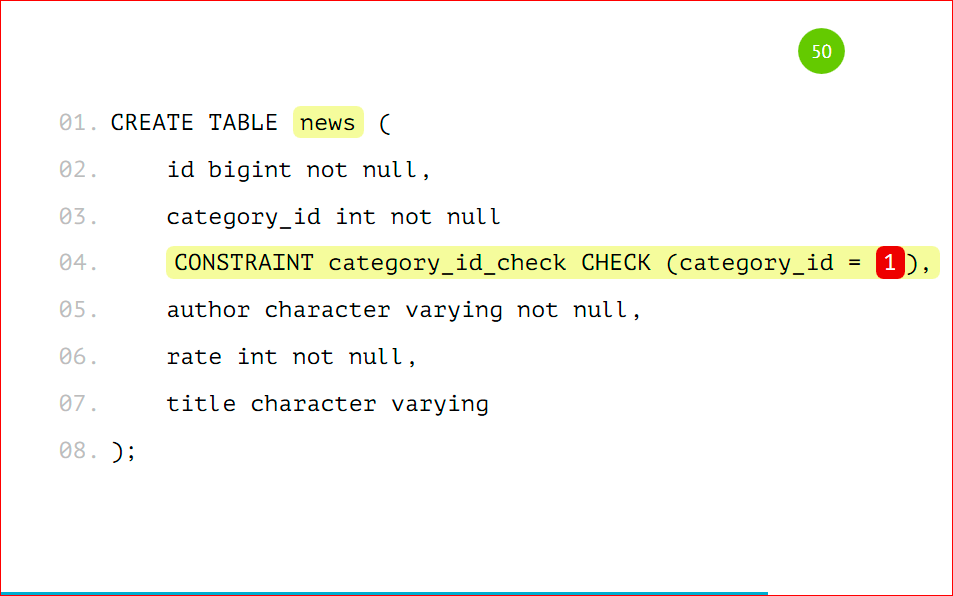
Для решения данной проблемы используют различные улитилиты, например, PartitionMagic, которая позволяет автоматически управлять партициями и не заморачиваться с созданием индексов, триггеров, с несуществующими партициями. Эта утилита open source’ная. Добавляется к БД в виде хранимой процедуры. Мы вызываем процедуру, указываем, что таблица будет news, и партицировать будем по category\_id.

Партицирование применяется на одном инстансе — это тот же самый инстанс базы, где у вас лежала бы большая толстая таблица, но мы ее раздробили на мелкие части. Можно совершенно не менять исходное приложение — оно точно так же будет работать с основной таблицей — вставляем туда данные, редактируем, удаляем. Так же все работает, но работает быстрее. Приблизительно, в среднем, в 3-4 раза быстрее.

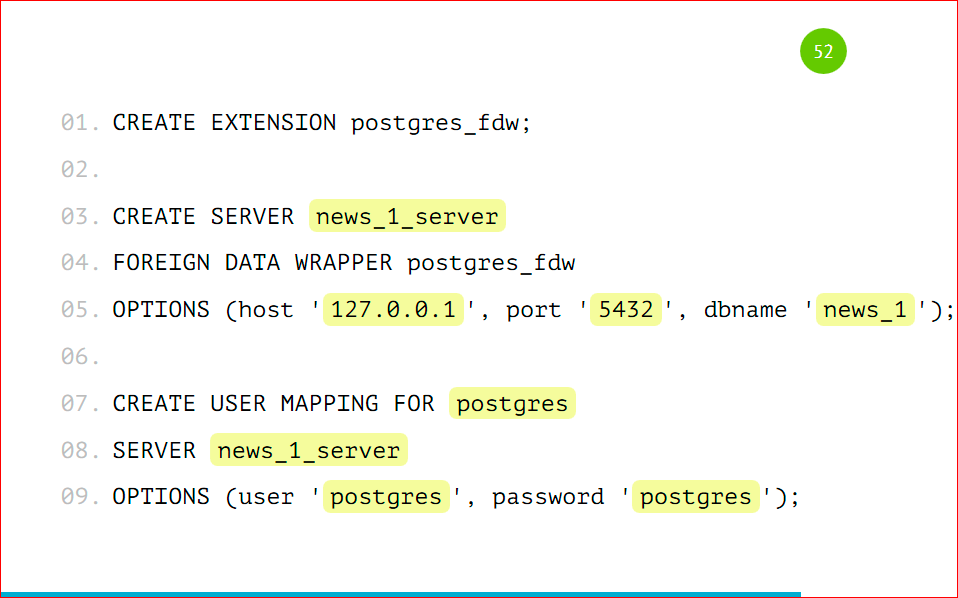
Горизонтальный шардинг — это когда мы данные разносим по нескольким серверам. Рассмотрим такую же структуру с двумя шардами — news\_1 и news\_2, но это будут разные инстансы, третьим инстансом будет основная база, с которой мы будем работать



Создаем таблицу с шардом, которую мы делаем на сервере, который будет выступать шардом с category\_id=1. Единственное, что туда нужно добавить, это CONSTRAINT CHECK, того, что записи будут выпадать только с category\_id=1. Это не унаследованная таблица.



Для настройки шардинга на основном сервере следует написать:

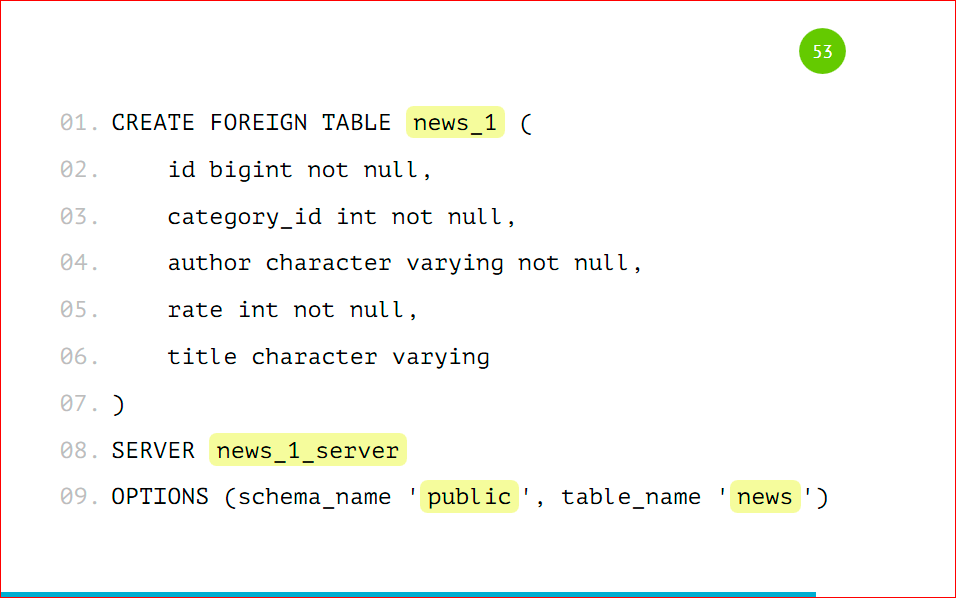


Мы подключаем EXTENSION. EXTENSION идет в Postgres’e из коробки, делается это командой CREATE EXTENSION, называется он postgres\_fdw, расшифровывается как foreign data wrapper.

Далее нам нужно завести удаленный сервер, подключить его к основному, мы называем его как угодно, указываем, что этот сервер будет использовать foreign data wrapper, который мы указали. Таким же образом можно использовать для шарда MySql, Oracle, Mongo… Foreign data wrapper есть для очень многих баз данных, т.е. можно отдельные шарды хранить в разных базах.

В опции мы добавляем хост, порт и имя базы, с которой будем работать, нужно просто указать адрес вашего сервера, порт (скорее всего, он будет стандартный) и базу, которую мы завели. Далее мы создаем маппинг для пользователя — по этим данным основной сервер будет авторизироваться к дочернему. Мы указываем, что для сервера news\_1 будет пользователь postgres, с паролем postgres. И на основную базу данных он будет маппиться как наш user postgres.

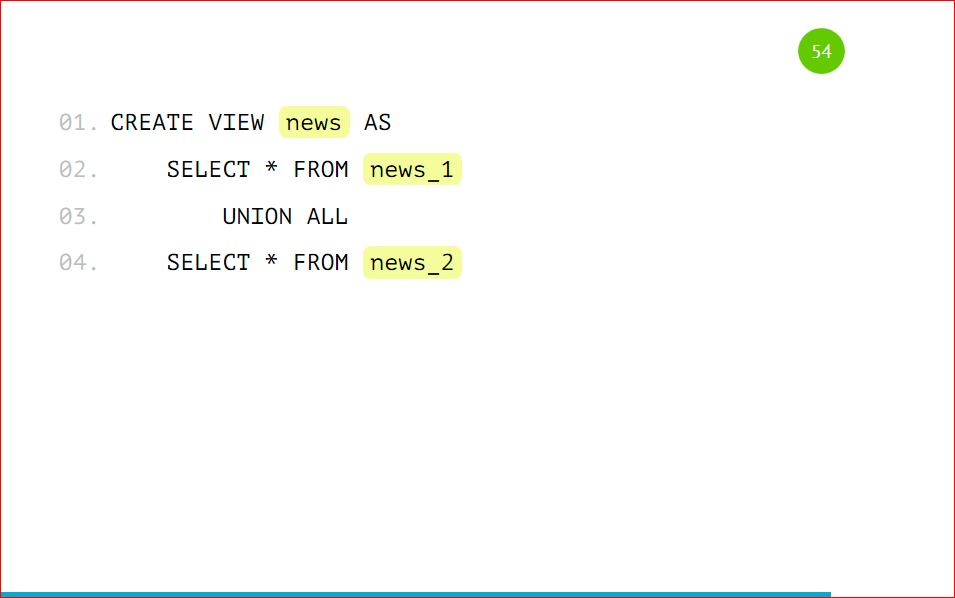
Далее мы заводим табличку на основном сервере:



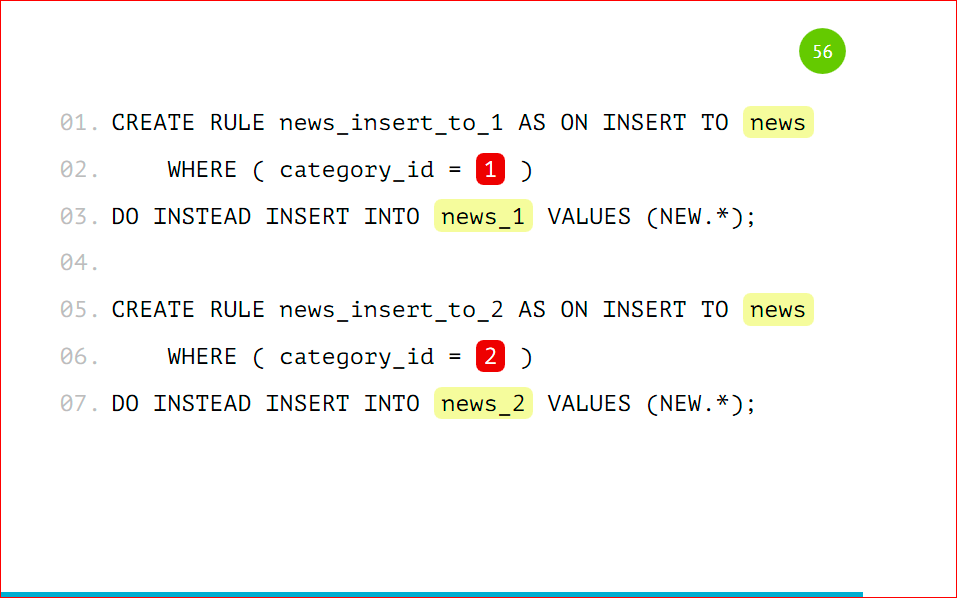
Это будет табличка с такой же структурой, но единственное, что будет отличаться — это префикс того, что это будет foreign table, т.е. она какая-то иностранная для нас, отдаленная, и мы указываем, с какого сервера она будет взята, и в опциях указываем схему и имя таблицы, которую нам нужно взять.

Схема по дефолту — это public, таблицу, которую мы завели, назвали news. Точно так же мы подключаем 2-ую таблицу к основному серверу, т.е. добавляем сервер, добавляем маппинг, создаем таблицу. Все, что осталось — это завести нашу основную таблицу.

Это делается с помощью VIEW, через представление, мы с помощью UNION ALL склеиваем запросы из удаленных таблиц и получаем одну большую толстую таблицу news из удаленных серверов.



Мы заводим основное правило, которое будет срабатывать, если ни одна проверка не сработала, чтобы не происходило ничего. Т.е. мы указываем DO INSTEAD NOTHING и заводим такие же проверки, как мы делали ранее, но только с указанием нашего условия, т.е. category\_id=1 и таблицу, в которую данные вместо этого будут попадать.



Т.е. единственное отличие — это в category\_id мы будем указывать имя таблицы.

Репликация — это синхронное или асинхронное копирование данных между несколькими серверами. Ведущие сервера называют мастерами (master), а ведомые сервера — слэйвами (slave). Мастера используются для изменения данных, а слэйвы — для считывания. В классической схеме репликации обычно один мастер и несколько слэйвов, так как в большей части веб-проектов операций чтения на несколько порядков больше, чем операций записи. Однако в более сложной схеме репликации может быть и несколько мастеров.

Репликация позволяет использовать два или больше одинаковых серверов вместо одного. Операций чтения (SELECT) данных часто намного больше, чем операций изменения данных (INSERT/UPDATE). Поэтому, репликация позволяет разгрузить основной сервер за счет переноса операций чтения на слейв.

Например, создание нескольких дополнительных slave-серверов позволяет снять с основного сервера нагрузку и повысить общую производительность системы, а также можно организовать слэйвы под конкретные ресурсоёмкие задачи и таким образом, например, упростить составление серьёзных аналитических отчётов — используемый для этих целей slave может быть нагружен на 100%, но на работу других пользователей приложения это не повлияет.

В приложении у будет два соединения с базой данных. Одно — для мастера и одно для слейва. Репликация обычно поддерживается самой СУБД (например, MySQL) и настраивается независимо от приложения.

Однако, репликация сама по себе не очень удобный механизм масштабирования. Причиной тому — рассинхронизация данных и задержки в копировании с мастера на слейв. Поэтому, в последовательных операциях необходимо использовать чтение с Мастера, чтобы получить актуальные данные.

Если использовать синхронную репликацию, для обеспечения полного разграничения функций (для мастера только запись, для слейва – только чтение), то это может значительно уменьшить скорость работы MySQL. Синхронный режим не следует использовать в Web приложениях. Зато это отличное средство для обеспечения отказоустойчивости. Вы всегда можете переключиться на слейв, если мастер ломается и наоборот. Чаще всего репликация используется совместно с шардингом именно из соображений надежности.

Шардинг и репликация часто используются совместно. Такая схема часто используется не для масштабирования, а для обеспечения отказоустойчивости. Так, при выходе из строя одного из серверов шарда, всегда будет запасной.

Итак, подведем итоги. Партицирование — это действительно просто, стоит всего лишь несколько действий совершить, все настроить, и оно все будет замечательно работать, не будет просить есть. Можно так же работать с основной таблицей, как мы работали ранее, но при этом у нас все красиво лежит по полочкам и готово к масштабированию, готово к большому количеству данных. Все это работает на одном сервере, и при этом мы получаем прирост производительности в 3-4 раза, за счет того, что у нас объем данных в таблице сокращается, т.к. это разные таблицы.

Шардинг — лишь немного сложнее партицирования, тем, что нужно настраивать каждый сервер по отдельности, но это дает некое преимущество в том, что мы можем просто бесконечное количество серверов добавлять, и все будет замечательно работать.

Репликация - используется в большей мере для резервирования баз данных и в меньшей для масштабирования. Master-Slave репликация удобна для распределения запросов чтения по нескольким серверам. Зачастую репликация используется вместе с шардингом при решении вопросов масштабирования.