Итак, последняя база данных на сегодня – это Redis. Redis отличается от того, что у нас уже было: и от Mongo-и, и от Postgres-а, хотя бы потому что он является базой данных «ключ-значение». То есть он ведет себя приблизительно как (НРЗ 00:16), но только лежит на отдельном сервере, к нему надо соединяться, брать команды… Выполнять, вернее, команды и получать значения, выставлять значения по ключу. В принципе, это всё, что Redis умеет, но работает он очень быстро, потому что все значения лежат в оперативной памяти – так Redis написан.

Почему есть смысл его использовать как отдельную базу данных? – Потому что, во-первых, у Redis-а своя сессия, то есть, когда ваше приложение закроется и откроется заново, в Redis-е что-то будет лежать. В Redis-е можно для кэшей соответственно, определять сохранение на жёсткий диск…, вернее, не для кэшей, а для значений, которые там лежат. Оговорка по Фрейду, мы его используем, в основном, для кэшей. Если там хранить какие-нибудь значения, можно настроить так, что Redis раз в какой-то промежуток времени будет эти значения сливать на жёсткий диск и потом, если вдруг не дай бог, машина reboot-нется или Redis упадёт и начнёт стартовать заново – ваши не потеряются, ваши данные будут восстановлены, прочитаны с диска.

Для работы с Redis-ом я использую client TablePlus – это визуальный клиент для Redis-а. Для Redis-а вообще, их, в принципе, не сильно много. TablePlus – это было то, что я в принципе, просто нашёл как (НРЗ 01:31) client с самым большим количеством like-ов от коллег. И попробуем пользоваться им для того, чтобы смотреть на Redis. Нормально…, в нормальных реалиях, в нормальных ситуациях Redis-ом пользуются из консоли, в основном, выполняя просто команды Redis-а, которые я здесь и буду выполнять. Давайте посмотрим уже сразу и на…, как этим пользоваться. Будет простая операция: положим ключик в «k1»…, то есть, вернее, значение 1 в ключик «k1» и потом по этому адресу его просто и достанем.

Соответственно, connect-имся, можно видеть, что здесь нет context-а. Авторы библиотеки Redis решили, что нам можно по умолчанию connect-иться к context.Background(). Если вам нужен какой-то определенный context, вы вызываете команду DialContext – это то же самое, то есть уже всё тут указывается. И вызывается context, который вам нужен.

Устанавливаем сеть, то есть сеть в Redis-е – её можно менять. Никогда не видел, чтоб кто-то использовал что-то другое, кроме как «tcp», но такая опция у вас есть. Это address – тут всё, как обычно: никаких логинов, паролей я не использую, поэтому здесь просто такой самый простой тип соединения.

Потом проверяем на ошибки, опять же здесь неправильная функция defer, то есть здесь, по-хорошему, надо сделать также, как в Mongo: такую функцию. Давайте сейчас, прям, может быть, я её и сделаю. То есть у нас тут c.Close. Если да, не равно нулю, то значит, так panic. Да, так лучше. И здесь у нас тоже есть ошибка. У нас есть команда Do, которая выполняет… Вернее, не команда, а метод Do, который выполняет команду к Redis-у. Я уже говорил, что Redis – он командно-ориентированный, соответственно команды там бывают разные. И авторы driver-а именно пошли самым простым путем: просто есть такой Do, у Do есть имя команды, потому что оно всегда в 1 экземпляре существует, потом какая-то куча аргументов, потому что на 1-ю команду «SCAN» аргументов большего 1, у команды «SET» аргументов ровно 2. У каких-то команд нет аргументов, у каких-то команд – 1 аргумент и так далее.

Здесь мы можем наблюдать всякие ошибки, если таковые есть. Если, например, connection с Redis-ом закрылся, или Redis не способен обработать вставку или забор. То есть какие-то вещи тут выдаются. И 2-й…, 1-й, вернее – это результат выполнения команд, то есть то, что Redis нам отправил как ответ. Нас в команде «SET» не особо не интересует ответ, потому что команда «SET» - она…, она

просто…, она если у вас…, она без ошибок выполнилась – она выполнилась. То есть у вас в Redis-е что-то появилось.

И дальше мы здесь выполняем команду «GET». Соответственно, в чём прикол? У Redis-а есть этот reply – этот интерфейс. И авторы Redis-а себя немножко загнали в ловушку с тем, что они не возвращают типы. Дженериков в Go ещё не завезли…, вернее, уже завезли в версии 1.18. Подождём немножко, и будут вам дженерики, но пока их не было на момент создания библиотеки. И пришлось такую городить штуку: то есть у reply надо проверять, значит, type и делать switch case, то есть здесь у самого ответа Redis-а проверяется тип этого ответа и cast-уются в определенные в структуры. То есть вам нужно обязательно знать, что у вас возвращается по ключу, иначе cast будет неправильным. Давайте посмотрим, как это работает в случае с Redis-ом.

Я здесь (НРЗ 05:58) с названиями элементов, то есть RedisSimple. Смотрим. Нет ошибки, в первом случае мы вызвали команду «GET». Команда «GET» получала значение по ключу «k1», у нас там лежит 1. Вернула нам, соответственно, 1. И команда «INGR», инкрементирует переменную по этому адресу и возвращает её инкрементированное значение, то есть аналогично команде, если мы напишем это. И здесь тоже cast-уется в Int, без cast-а в Int у вас ничего не выйдет. То есть Do будет жаловаться, что нельзя тут ничего назначить и так далее.

И соответственно, здесь выдастся 2. Если мы сейчас посмотрим в нашем табличном сборщике, вот он там определенный (НРЗ 06:49). По умолчанию Redis не ставит никакой (НРЗ 06:52), и здесь показывается ключ, значение, тип можно видеть, что он здесь STRING, потому что типы в Redis-е – они не примитивные. Всё что не SET, не LIST и не HASH, то есть не какие-то вычисляемые значения или не переборные…, не множественные значения, соответственно, всё это у нас STRING.

И Time to live – интересная штука, определяет, сколько этот ключ будет жить. Redis часто использует для кэшей, поэтому функционал для кэшей – сколько жить какой-то конкретной записи, он в Redis встроен. Как-то так. Давайте посмотрим, как теперь в Redis засунуть строку.

Как засовывается строка в Redis? – Да точно также: то есть здесь мы создаем клиента и выполняет команды Do. Я здесь хочу ещё показать, кроме того, что как засунуть строку, потому что здесь особо ничего интересного: такой же «SET», такие же аргументы…, такой же аргумент, но только здесь строка, а не…, это не важно. Здесь мы, когда получаем, мы здесь тоже cast-уем String, потому что возвращается нам опять же тип byte, нам его надо откастовать в String. Если он возвращается в String – надо просто вернуть String и так далее. Зачем этим заниматься, когда можно просто обернуть?

Что здесь интересно? – Здесь я использую команду «EXPIRE», команда «EXPIRE» как раз ставит тот самый Time to live, про который я уже говорил и позволяет строке через какое-то время успешно из Redis-а удалиться. Здесь мы получаем строковое значение и, если мы посмотрим сейчас в нашей TablePlus – мы видим, что tti у нас 993, то есть 7 секунд я вам про этот tti рассказывал. Вот уже ещё 5 секунд прошло, ещё 2 и так далее. Если мы переделаем вставку, то есть ещё раз вставим значение в этот ключ, tti у нас обновится. Вот, в принципе, как-то так.

Соответственно, со всеми этими ключами и значениями возникает интересный вопрос: а что будет, если ключа нет? Мы запросили что-то, а ключа нет, что делать? То есть, что будет Redis делать? Да ничего, на самом деле, делать не будет. То есть в данном случае он вернет просто значение по умолчанию – об этом позаботится наш cast, вернёт нам пустое значение и отдаст нам ошибку. Давайте её…, на неё посмотрим.

Да, тут Println, уберем 2-й метод, про который я ещё не рассказал. И, соответственно, возвратился нуль, но при этом мы про это не узнаем, потому что у нас здесь пустая строка. В данном случае надо проверять ошибки для того, чтобы знать, что у вас в ключике ничего не сохранено и, значит, надо туда что-то сохранить. Но для того, чтобы получить функционал от непосредственно client-а, то есть здесь мы что делаем? Мы жмякаем такие кнопочку «обновить» – он раз, все ключи выдал. Чтобы такое получить: вам нужно несколько больше усилий приложить, чем, например, в Postgres-е, который, в принципе, для этого предназначен и в Redis-е, всё-таки, с этим Do command немножко сложнее.

То есть что мы делаем? Мы вызываем команду Do со «SCAN»-ом. Команда «SCAN» как раз сканирует наши ключики, начиная с какого-то определенного offset-а. Redis не умеет отдавать огромные пачки ключей сразу... Вернее умеет, но это долго, поэтому здесь существует такая… такой offset и, значит, если у нас что-то случилось – мы паникуем, если нет, то мы получаем наш offset. Опять же, тут даже комментарий есть, что ключи – это от multi bulk-reply. Это, значит, несколько разделенных групп ключей, которые мы можем перебрать, и пока у нас этот итератор существует, мы что-то сделаем. Если нет, то мы выходим из цикла.

К сожалению, да, получить весь массив…, размер массива ключей в Redis-е, как это можно, например, сделать в Postgres-е на уровне самой базы данных, она примерно знает сколько там ключей лежат. Redis знает достаточно условно. Плюс, если там несколько client-ов пишут, то Redis не знает вообще. Поэтому здесь надо делать таким образом. Давайте посмотрим, как это работает. Здесь будет один проход. У нас только здесь такое есть значение. strVal и k1, только такие значения. И потом вы можете из каждого получить что-то, что там лежит.

И соответственно, последний вопрос, который нас будет волновать – я там показывал, выпендривался, здесь типы. Что у нас здесь есть не только STRING,

но и всякие составные типы. Уже всё, уже ничего не вываливается, короче, вот здесь: SET, ZSET, LIST, HASH. Давайте посмотрим, как с LIST-ом работать. То есть это обычный связанный список в Redis-е. Иногда он нужен, когда вам нужно, например, хранить какие-то значения, добавлять в конец, удалять из конца, например, для каких-то распаковщиков или паковщиков данных для из кэшей, потому что они-то как раз пользуются листами для построения своей структуры.

Просто, любая команда, любое действие в Redis-овском driver-е выполняется с помощью команды Do. Значит, вы можете просто зайти в Google, найти в Redis-е в документации как это делается. Увидеть: ага, RPUSH, потом LSET – изменит list, то есть поставит элементы в какую-то позицию. И LRANGE вернет list. Вот, собственно, я просто посмотрел в документацию и здесь сделал то же самое.

Вот у нас «RPUSH»-и, которые кидают: «one», «two», «three». И дальше я делаю «LRANGE», оборачиваю в redis.Values, потому что это многострочное значение, также как и SCAN, но здесь мне не надо делать SCAN-ы, мне здесь надо просто засунуть это всё. Всё это будет у меня в массиве, мой лист. И мне потом просто надо будет по нему пройтись. То есть если мы сейчас это запустим, то увидим как раз вывод redis.String, потому что оно возвращает ошибку и переменную. И в нашей TablePlus мы увидим LIST, тип LIST.

Как-то так работается с Redis-ом. Ещё раз, цель: не показать всё, что умеет Redis, Mongo и Postgres. Показать просто, что это всё базы данных. Они что-то имеют…, их driver-а общее, но имеют и много различий. То есть это…, не базы данных – это не монолитная штука. Это просто то, что хранит ваши данные на протяжении какого-то определенного промежутка времени. И с этим тоже надо иметь в виду, когда…, это тоже надо иметь в виду, когда вы с этими базами данных работаете.

Например, Postgres можно…, в Postgres-е можно хранить связанные значения, большие. Если вы планируете несколько таблиц связывать, если вам нужна структура, если вам нужна валидация. То есть всегда стандартный фиксированный набор элементов.

В Mongo-е хранить как раз те данные, которые…, для которых вам не нужна особая структура, вам просто нужно что-то туда засунуть, а потом что-то оттуда вынуть. И вы точно знаете, что вы оттуда вынимаете.

Соответственно, Redis нам нужен, когда нам что-то нужно в память быстро положить и там хранить, а потом забрать. У него самый простой интерфейс, на мой взгляд, у этого driver-а, потому что у самого Redis-а интерфейс самый простой: у него нет никаких подключений, как у Mongo-и. нет никаких комплексных схем, как у Postgres-а и так далее.

То есть все базы данных, которые я перечислил в этой лекции – они будут иметь свои специфические driver-а, специфические задачи, но в целом как-то из Go работается с ними вот так.