Итак, Mongo. Для Mongo-и я, значит, не беру client, который у меня встроен в ide, тут тоже можно добавить соединения – тут их куча, в том числе и MongoDB. Я беру отдельно стоящий клиент, он мне удобнее. Это ваше право, каким клиентом пользоваться. Я люблю Robo 3T либо у него есть Studio 3T – это будет платный клиент с большим количеством фишек. Мне достаточно этого для запросов и какой-то отладки.

Тут слева у нас базы данных, это моё соединение, соединение тоже с localhost-ом, потому что напоминаю: всё крутится здесь в docker-е, вот оно Mongo. И, соответственно, здесь есть 2 базы данных, в них есть коллекции. Коллекции – это некий аналог таблиц, но опять же, очень отдаленный. В данном случае у нас есть, например, коллекции, которые называются numbers – в ней лежат 2 объекта: 1-й объект – это наша машина, отголоски с нашего Postgres-ового хранилища, 2-й наш объект – это какой-то item, хлеб с какой-то ценой. То есть что у них есть общее? Да ничего у них нет общего, кроме поля _id. Соответственно, поле _id – это поле, добавляемое Mongo-й. Как я говорил: почти в каждой базе данных надо знать, что у неё лежит где, поэтому она использует либо неявные, либо явные id-шники. Mongo использует явные id-шники.

Также mongo строит сама index-ы, index-ы вы можете построить по любому элементу. Они помогают mongo-е делать запросы, то есть без index-а, также, кстати, как и в Postgres-е, Mongo будет искать, перебирая всё подряд. С index-ом ей будет проще, потому что она будет знать, где примерно…, я не буду углубляться сейчас в сложности построения index-ов, где примерно лежит искомая строка или искомое число, в каких…, каком множестве объектов и это множество успешно найти. Гораздо быстрее, чем, если просто перебрать все объекты подряд.

Здесь нам не очень актуально. У нас там 2 объекта, в этих примерах их не будет много, просто мы осваиваемся с интерфейсом. Интерфейс также в себя включает

строку запросов – про неё чуть позже. И давайте мы теперь посмотрим, как из Golang-а в Mongo добавить именно уже какие-то данные?

Итак, давайте посмотрим, как вставляются данные в Mongo. То есть здесь начинается всё, естественно, с установления соединений. Как можно увидеть, создается context. Context у нас создается с Timeout-ом, создается, соответственно, Connect к Mongo-е. И тут что интересно? То есть мы можем видеть, что первым идёт context, потом какие-то опции соединения. То есть как в Postgres-е, в postgres-е была просто у нас строчка, в Mongo-е у нас, соответственно, какие-то опции.

Опции в mongo-е существуют по той причине, что клиент в Mongo-е – он несколько более сложный, это обусловлено самой Mongo-й, то есть она отгружает больше на клиент, чем Postgres, поэтому у неё есть куча настроек: есть настройки Pool-а, которые строят в (НРЗ 03:15), здесь настройки работы с ReplicaSet-ом. Есть всякие preference-ы у неё, конфигурация вследствие защищенных соединений и так далее.

Здесь я не использую никаких опций, как в качестве примера. И здесь мы, соответственно, просто берем и ставим url. То есть, по сути, повторяя эту строчку от Postgres-а, только с другими параметрами. Тут у нас Mongo: логин пароль, наш localhost. Напоминаю, всё в docker-е. И port, у Postgres-а была 5432, у Mongo-и по умолчанию 27017 – тоже не столь важно.

Здесь дальше defer у меня правильно написанный, то есть который ещё и ошибки отслеживает и умеет о них как-то пользователя рапортовать. В Postgres-е я просто на это забил и просто закрываю соединения. Здесь всё-таки несколько правильней написан этот defer, то есть здесь так лучше и закрывать, не panic-а,

может быть, какие-нибудь log-и выводить и прочее. Как-то дать пользователю вашего приложения знать о том, что что-то случилось.

Дальше всё различие идёт между Mongo и Postgres-ом. В Postgres-е у нас была таблицы, здесь у нас Collection-ы. Напоминаю, что Collection никакой структурой он нас не обеспечивает, никакой схемой и Collection-ы все содержат какие-то просто документы, разделенные только вашим желанием, с id-шниками какими-то. Но и здесь это, в принципе, всё отражение своё находит, то есть операция вставки принадлежит не базе данных, как в Postgres-е, где у нас соединение держало операции любые и выборки, и вставки. Здесь вам не нужно работать с несколькими таблицами, то есть несколькими коллекциями, их как-то соединять в подавляющем большинстве случаев, поэтому вы здесь просто работаете на уровне уже самой коллекции. И на уровне коллекции вы можете ставить, прочитать, удалить и так далее.

Здесь у нас формат jsonb… Вернее, формат bson, то есть binary json, потому что, если вы сейчас в client-е, например, откроете документ для редактирования – вы увидите здесь обычный json, то есть реально, просто json. Но, по сути, хранится-то он несколько в другом формате. Разницы для вас по большому счету, не будет.

То есть здесь мы используем, значит, bson для того, чтобы взаимодействовать с Mongo-driver-ом – это штука, которая поставляется вместе с driver-ом Mongo-и, и использует самую простую его репрезентацию. То есть это словарь с словарями, которые… То есть словарь структур, который представляет собой такие уровни-вставки. То есть, на самом деле, это просто список, у которого есть ключ и значения, и так всё это делится структурно. И мы после вставки может прочитать id вставленного элемента. Тот самый ObjectID, который вы видели в клиенте. Давайте вставим, запустим. Мы получили этот ObjectID. Если мы сейчас в client-е

обновим коллекцию. Да, тут вызвал команду… Вот мы увидим тот самый наш pi. Как-то так, давайте теперь какие-то рассмотрим операции посложнее.

Следующим пунктом нашего меню является вставка объекта. Вставка объекта происходит ровно таким же образом, как вставка нашего bson.D, в данном случае Mongo-driver не делает никакой разницы в отличие от Postgres-a. То есть для Mongo-driver-a какая разница, что cast-овать в документ. Это как раз следует из того, что у нас Postgres проверяет схему, проверяет структуру, поэтому он работает на уровне запросов с какими-то mapping-ами. Mongo-driver работает на уровне: получи документ и вставь его.

Значит, документ у нас – это наш объект в данном случае. У нас Document-oriented база, но, по сути, эти документы – они объекты. Я создал здесь Classic Number, у которого есть метки, теги для Unmarshaler-a…, вернее, для Marshaler-a, на самом деле, bson-овского. Внутри insert-a, у нас если не является документ уже bson-ом, то происходит его Marshaling структуры этой. И Marshalling у нас происходит согласно заголовкам. И всё, что нам нужно сделать – это просто передать наш номер и наслаждаться зрелищем.

Передаём номер, получаем id-шник объекта. Напоминаю, что здесь была у нас переменная «epsilon». И у нас появился здесь 4-й объект. Можно заметить, что они также добавляются, в принципе, по порядку, их index-ы генерируются, и соответственно, как-то так. Вот мы добавили объект, и он сам уже у нас cast-ился в json.

Мы посмотрели, как добавляются элементы, и мы здесь видим список. То есть такой a-ля запрос на получение такого списка. Здесь у нас можно фильтровать по каким-то полям, можно написать name: «pi», например, и здесь на выведется 1

объект. Можно описать brand…, ой, простите… Можно описать brand: «BMW», он выведет нам brand, всё круто.

Как теперь нам такой провернуть…, такой же трюк и с Golang-ом? Всё очень просто. Вот наш метод с Golang-а, тут всё вообще без изменений. Можно будет вынести в отдельную функцию, просто лень. Здесь у нас есть коллекция, опять же. Опять же, это коллекция «numbers». И здесь что происходит? Здесь мы берем и собираем…, и запускаем метод Find.

Метод Find – это, в принципе, то же самое, что этот find и эти квадратные…, фигурные скобочки в bson.D – это то же самое, что и фильтр там. То есть сейчас я ни по чему не фильтрую, и у меня здесь будут все результаты. Если б мне надо было отфильтровать, например, по имени «pi», это бы выглядело как-то так. Если мне надо бы отфильтровать по имени «pi» и brand-у «BMW», то это бы выглядело как-то так. Если предположить, что эти 2 поля могут соединиться в 1 элементе, потому что в противоположном случае у вас ничего не найдется.

Но я ничего не фильтрую, потому что здесь задача: просто показать, как это делается. Делается точно также как в той библиотечке Postgres-а, которую мы разбирали в этой…, в pgxscan, в Select-е. Соответственно, такой общий метод, как я уже сказал, для всех элементов. То есть мы получаем Next, если он есть, мы его сразу опять же, преимущество Mongo: мы его можешь сразу за-Marshal-ить в наш Number. И есть 2-й способ получить все элементы – это взять и вызвать у Current-а…, у результата вызвать Current. Это текущее поле bson.Raw и вызвать его элементы. Элементы – это непосредственно разбитые уже по ключу и значениям элементы объекта, как можно из названия догадаться.

То есть здесь я показываю 2 способа, соответственно, элемент – это массив. Этот массив можно перебирать, этот массив состоит из записей, ключ-значение, просто

такой bson.D формат. В num у вас засунется как раз то, что там лежит. Поскольку у меня сейчас элементы неоднородны, в каких-то случаях в Number. Напоминаю, Number у нас там «name» и «value», в каких-то случаях что-то будет, в каких-то случаях чего-то…, ничего там не будет. Давайте посмотрим.

Можно видеть, что 1-й случай: вот наши элементы как раз, наш массив, что видно, что в id у нас Objectid, здесь просто строковое значение. Видно, что тут «brand»: «BMW». «License plate», нет ни name, ни value, здесь ничего нет. И ладно, и так далее, то есть до первого значения, которое что-то в себе содержит. Можно увидеть, что Mongo ещё делает специальное уточнение для касто-типов правильного. То есть здесь Double – это важно, для типа Double у Mongo-driver-а зарезервирован тип float64. Если я сделаю здесь float32, точности здесь не хватает, то есть размера самого элемента, размера самого примитивного типа. И ничего туда не запишется, тут будет «pi», но будет 0.

Можно проверить, как это всё сработает, то есть я меняю щас тип. Если мы перезапустим: 0 и 0, потому что Mongo хранит Double-ы. Float32 – это неправильный тип, соответственно, если тип либо значение не совпадает – Mongo-driver ничего не map-ит.

Вот, получить 1 элемент можно с помощью функции FindOne, для того, чтобы этим не заниматься, этим вызовом res.Next и так далее. Тут всё пакуется, на самом деле, через метод Find. И через SingleResult, который оборачивает в себе cursor-ы и roll-элементы, и специальные образы возвращает в результат. И в результате вы уже можете задекодировать его в номер.

Давайте посмотрим на него, то есть здесь я как раз указываю фильтр, потому что, когда вы делаете FindOne, вам нужно какой-то представить, какой результат вы получаете. Результат без какого-то и какой-то фильтрации получить

предопределенный возможно, только если вы имеете только 1 столбец в коллекции… Вернее, 1 документ в коллекции, что практически не реально. Вот мы видим, нашёлся «pi», и распечатался Number.

Как можно увидеть, отличий сильных с Postgres-ом нет. В этом и есть цель того, чем я сейчас занимаюсь: не показать все фичи Mongo-driver-а или все Postgres-driver-а. А просто показать, что, в принципе, работается с ними похожим образом и базово после освоения одного driver-а, подобный driver вы можете освоить достаточно легко: driver Mysql, driver Oracle-а, driver Postgres-а. После Mongo-и то же самое есть, например, можно сделать для RienDB и так далее.