1. **Поясните понятие «ORM»**

**Object-Relational Mapping** - технология программирования, которая позволяет работать с SQL-базой данных, как с набором программных объектов. **Mapping**: база данных – объект contextDB, таблица – коллекция объектов, строка в таблице – объект, структура таблицы – класс.

**ORM** – это методика или техника программирования, предназначенная для преобразования между **несовместимыми типами данных в объектно-ориентированных языках программирования.** Суть заключается в том, что создается некая абстракция - “**виртуальная объектная база**”, запросы к которой, преобразуются в **SQL команды,** т.е. больше не нужно писать **SQL-запросы к базе данных вручную**.

**ORM фреймворк** может быть написан на каком-либо объектно-ориентированном языке и представлять обертку над некой реляционной базой данных. **Классы будут соответствовать таблицам в базе, а экземпляры этих классов – конкретным строкам таблицы**.

**Преимущества**

* **Независимость от вида базы данных**

Это, пожалуй, главнейшая особенность и преимущество использования **ORM** в приложении. Так как нет необходимости писать специфический код под конкретный вид базы данных.

* **Моделирование предметной области**

При использовании **ORM** для построения приложения, бизнес-логика приложения работает с объектами языка, а не с самой структурой базы данных. Это возможно благодаря соответствию между бизнес-моделью и самой базой данных.

* **Меньше кода и больше эффективности**

**ORM** добавляет дополнительный слой абстракции, который позволяет разработчикам сконцентрироваться на их бизнес-логике, не отвлекаясь на создание сложных запросов к базе данных. Это приводит к сокращению количества необходимого для написания кода и увеличивает продуктивность разработчика.

* **Управление зависимостями**

**ORM** предоставляет свободное управление зависимостями в базе данных. Связанные объекты загружаются автоматически, когда вызов методов преобразуется в соответствующий **SQL запрос**.

* **Параллелизм, кэширование и транзакции**

**ORM** поддерживает возможность параллельной работы, позволяя нескольким пользователям одновременно изменять один и тот же объект.

Другая особенность – объекты могут быть сохранены в кэше, сокращая нагрузку на базу и в целом увеличивая скорость работы приложения.

Изменения, вносимые в объект, могут быть ограничены в рамках данной транзакции, которая может быть сохранена или возвращена обратно в первоначальное состояние. В каждый момент времени могут быть активными множество транзакций, но все эти транзакции изолированы друг от друга.

**Недостатки**

* **Накладные расходы**

**ORM фреймворк** добавляет слой абстракции, ускоряющий процесс разработки и снижающий сложность создания конечного продукта. Однако, ничто не проходит даром – приложение начинает потреблять больше памяти, и нагрузка на процессор также увеличивается.

* **Производительность**

Некоторые операции, такие как массовая вставка, обновление, удаление и т.д., медленнее, когда выполняются посредством **ORM.** Поэтому, в таких случаях лучше использовать чистый **SQL**.

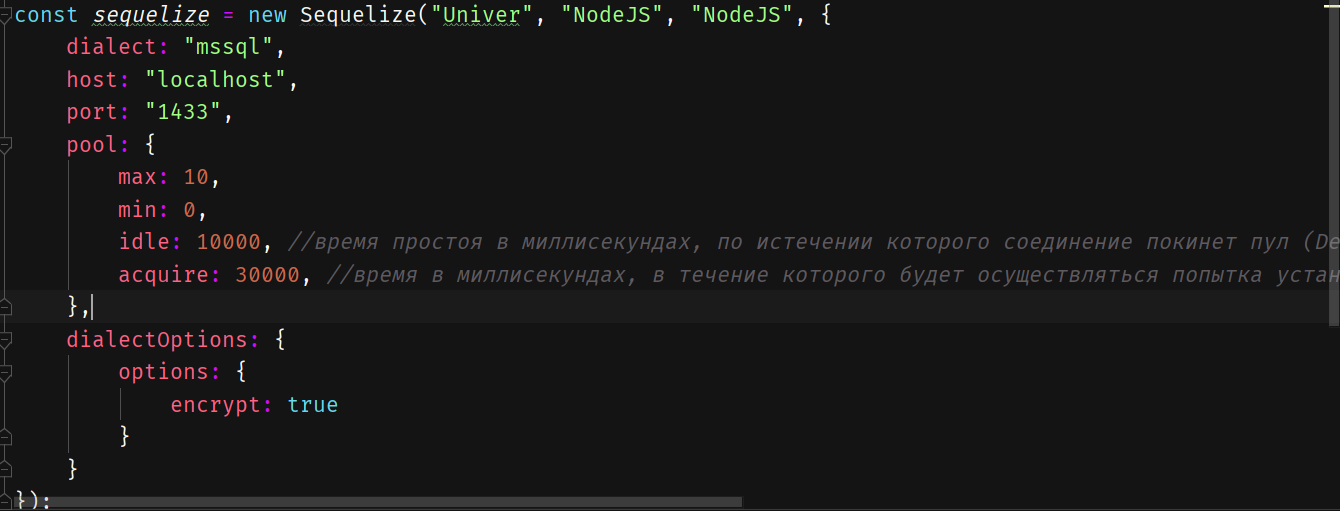
* **Поверхностное знание SQL**

Несмотря на то, что **ORM** облегчает жизнь, это часто приводит к тому, что разработчики не очень стремятся учить **SQL** или разбираются в нем слабо.

1. **Какой дополнительный пакет необходимо установить для работы sequelize c Microsoft SQL Server?**

Npm i tedious

1. **Поясните понятие «sequelize-схема».**



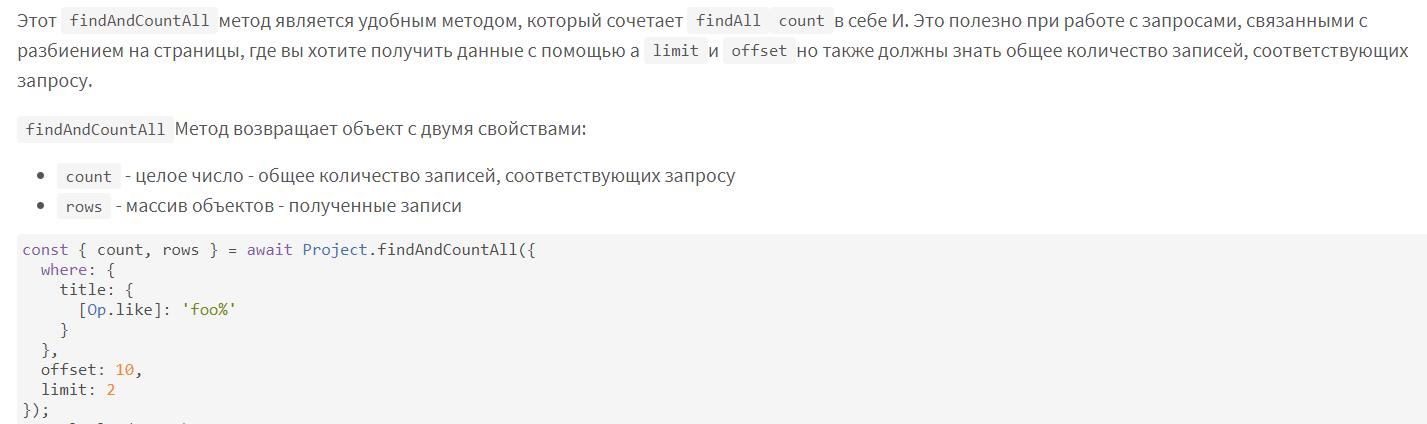
1. **Какая функция sequelize позволяет выполнить SELECT-запрос?**

**findAll** генерирует стандартный SELECT запрос, который извлекает все записи из таблицы (если только они не ограничены чем-то вроде whereпредложения, например).

**findByPk** получает только одну запись из таблицы, используя предоставленный первичный ключ.

**findOne** получает первую найденную запись (которая удовлетворяет необязательным параметрам запроса, если они предусмотрены).

**findOrCreate** создаст запись в таблице, если он не может найти ее, выполняющую параметры запроса. В обоих случаях он возвращает экземпляр (найденный экземпляр или созданный экземпляр) и логическое значение, указывающее, был ли этот экземпляр создан или уже существовал.



1. **Какая функция sequelize позволяет выполнить INSERT-запрос?**

create(), findOrCreate()

1. **Какая функция sequelize позволяет выполнить UPDATE-запрос?**

Update()

1. **Какая функция sequelize позволяет выполнить DELETE-запрос?**

Destroy()

1. **Поясните, что такое «хуки»?**

Хуки (также известные как события жизненного цикла) - это функции, которые вызываются до и после выполнения вызовов в sequelize. Например, если вы хотите всегда устанавливать значение для модели перед ее сохранением, вы можете добавить beforeUpdate.

Триггеры представляют собой функции, которые выполняются (если они определены) до/после/во время действий с данными. Список самых популярных триггеров в порядке их выполнения:

beforeValidate(данные, опции) - выполняется перед валидацией;

afterValidate(данные, опции) или validationFailed(данные, опции, ошибка) - выполняется после успешной или неуспешной проверки валидации соответственно;

beforeCreate(данные, опции) - вызывается перед созданием записи;

beforeDestroy(данные, опции) - выполняется перед удалением записи;

beforeUpdate(данные, опции) - вызывается перед обновлением записи;

beforeSave(данные, опции) - вызывается перед сохранением записи;

afterCreate(данные, опции) - вызывается после создания записи;

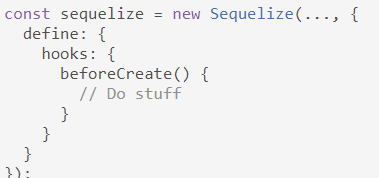
afterDestroy(данные, опции) - выполняется после удаления записи;

afterUpdate(данные, опции) - вызывается после обновления записи;

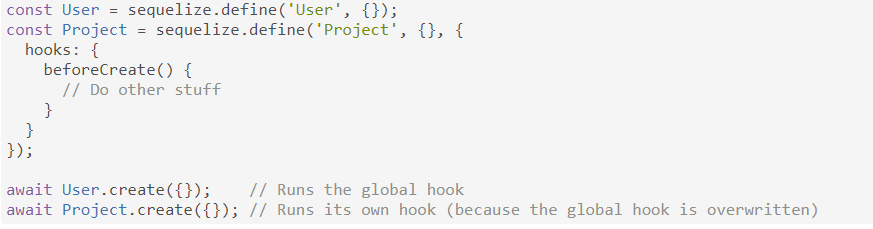
afterSave(данные, опции) - вызывается после сохранения записи.

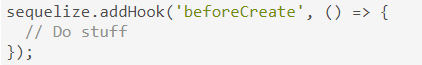
2 вида хуков: локальные и глобальные. Сперва вызываются локальные, а потом глобальные.

Глобальные хуки - это хуки, которые запускаются для всех моделей. Они могут определять поведение, которое вы хотите для всех ваших моделей. Они могут быть определены двумя способами, которые имеют несколько различную семантику:



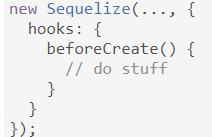
Это добавляет хук по умолчанию ко всем моделям, который запускается, если модель не определяет свою собственную beforeCreate хук





Этот хук всегда запускается, независимо от того, указывает ли модель свой собственный хук или нет beforeCreate хук.

Постоянные хук также могут быть определены в опциях, переданных конструктору Sequelize

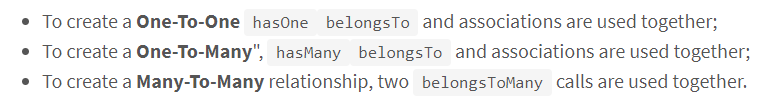


Обратите внимание, что вышеприведенное не совпадает *с крючками по умолчанию*, упомянутыми выше. Этот вариант использует define опцию конструктора. А вот этот-нет.

1. **Поясните назначение функции sequelize *Схема*.hasMany.**

Создание связей между таблицами осуществляется с использованием моделей. Рассмотрим установление следующих типов связей:

* один к одному;
* один ко многим;
* многие ко многим.



Эта A.hasOne(B) связь означает, что существует взаимно однозначное отношение между A и B, причем внешний ключ определяется в целевой модели (B).

Эта A.belongsTo(B) связь означает, что существует взаимно однозначное отношение между A и B, причем внешний ключ определяется в исходной модели (A).

A.hasMany(B) Ассоциация означает, что существует отношение "один ко многим" Между A и B, причем внешний ключ определяется в целевой модели (B).

Эти три вызова вызовут Sequelize для автоматического добавления внешних ключей к соответствующим моделям (если они уже не присутствуют).

A.belongsToMany(B, { through: 'C' }) Ассоциация означает, что существует отношение "многие ко многим" между A и B, используя таблицу C в качестве [соединительной таблицы](https://en.wikipedia.org/wiki/Associative_entity), которая будет иметь внешние ключи (aId bId например, и). Sequelize автоматически создаст эту модель C (если она уже не существует) и определит для нее соответствующие внешние ключи.

1. **Поясните понятие «raw query».**

Поскольку часто встречаются случаи использования, в которых просто проще выполнять необработанные / уже подготовленные SQL-запросы, вы можете использовать этот sequelize.queryметод.

По умолчанию функция возвращает два аргумента - массив результатов и объект, содержащий метаданные (например, количество затронутых строк и т. д.).

Задаем запрос вручную, самописный запрос, когда очень сложно реализовать через методы.

1. **Поясните понятие «code first».**

Сперва определяем модели. Метод sync() синхронизирует структуру базы данных с определением моделей. Например, если для какой-то модели отстуствует соответствующая таблица в БД, то эта таблица создается.

Если в бд есть подобная таблица, но она не соответствует определению модели, то мы можем использовать параметр {force: true}, чтобы удалить таблицы и создать их заново, но уже с нужной нам структурой

1. **Поясните понятие «транзакция».**

Sequelize по умолчанию не использует транзакции. Однако для готового к производству использования Sequelize вам определенно следует настроить Sequelize для использования транзакций.

Sequelize поддерживает два способа использования транзакций:

* Неуправляемые транзакции: фиксация и откат транзакции должны выполняться пользователем вручную (путем вызова соответствующих методов Sequelize).
* Управляемые транзакции: Sequelize автоматически откатит транзакцию, если возникнет какая-либо ошибка, или зафиксирует транзакцию в противном случае. Кроме того, если CLS (локальное хранилище продолжения) включен, все запросы в обратном вызове транзакции автоматически получат объект транзакции.