**КТ № 14. Связи один-ко-многим, один-к-одному, многие-ко-многим**

**Время на выполнение практической работы:** 4 часа.

**Теоретические сведения к выполнению практической работы:**

**Связь один-ко-многим**

Например, одна компания может производить множество товаров. То есть мы имеем отношение одни-ко-многим (1 компания - много товаров). Для создания подобной связи в Sequelize применяется метод hasMany(). Например:



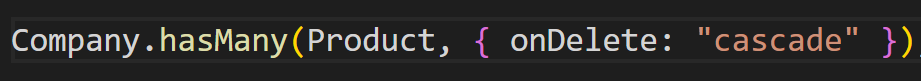
Метод hasMany() вызывается именно у главной модели - Company. А в сам метод передается зависимая модели - Product.

В итоге будут созданы две таблицы, которые описываются следующим SQL-кодом:



По умолчанию в зависимой таблице (то есть products) будет создаваться дополнительный столбец, который называется по имени главной модели плюс суффикс "Id", то есть в данном случае companyId. И через данный столбец строка из companies сможет ссылаться на объект из таблицы products.

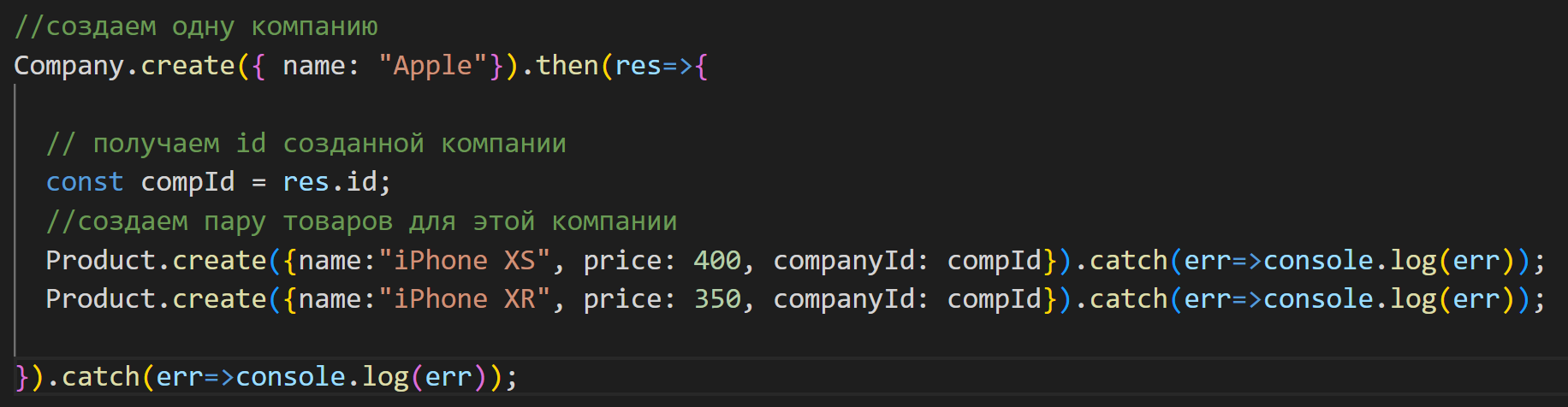
В данном случае в коде для таблицы products мы видим, что при удалении главного объекта из таблицы companies зависимый объекты из таблицы products не удаляются - в столбце companyId для таких строк будет устанавливаться значение NULL, поскольку действует выражение ON DELETE SET NULL. Однако нередко более распространённой стратегией является каскадное удаление - при удалении главного объекта удаляются и все связанные с ним объекты. Для этого в метод hasMany() передается в качестве второго параметра специальный объект, который настраивает отношение между моделями. В частности, параметр "onDelete" позволяет задать действия при удалении:



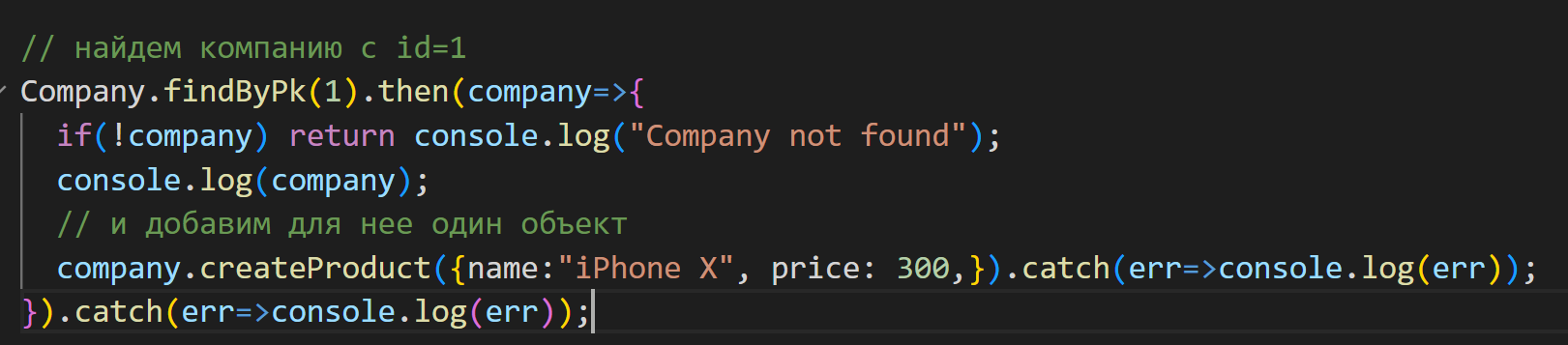
**Основные операции**

Рассмотрим некоторые базовые операции, которые могут вызвать затруднения при работе с моделями со связью один-ко-многим.

При создании объекта зависимой модели нередко требуется указать ссылку на связанную главную модель. Для этого мы можем задействовать свойство модели, которое совпадает с именем столбца-внешнего ключа в соответствующей таблице. Например, в примере выше в таблице companies для связи с таблицей products создаваться столбец companyId. И хотя в модели Company выше явным образом не определено подобное свойство, но оно создается неявно. Например, создание объектов:

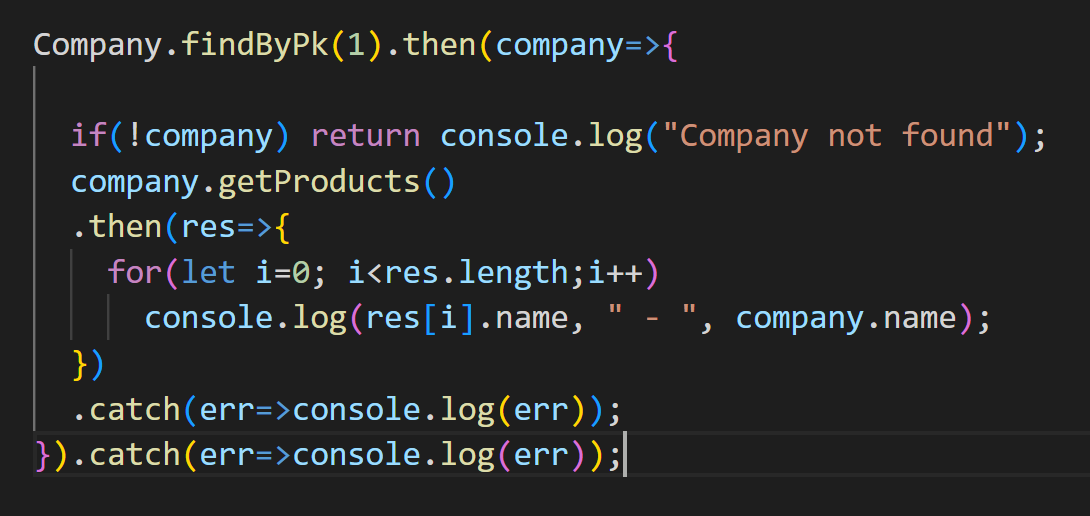


Есть другой способ добавления зависимой модели - через главную модель. У главной модели для этого неявно определяется метод по имени createЗАВИСИМАЯ\_МОДЕЛЬ() (например, createProduct()):

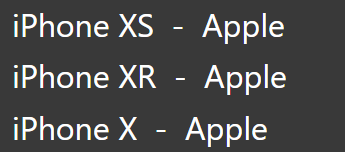


Фактически единственное отличие от первого варианта добавления состоит в том, что в данном случае не надо указывать id главной модели.

Для получения всех связанных объектов зависимой модели у главной модели определяется метод по имени getЗАВИСИМАЯ\_МОДЕЛЬs() (например, getProducts()). Например, получим все товары компании с id=1:



Консольный вывод:

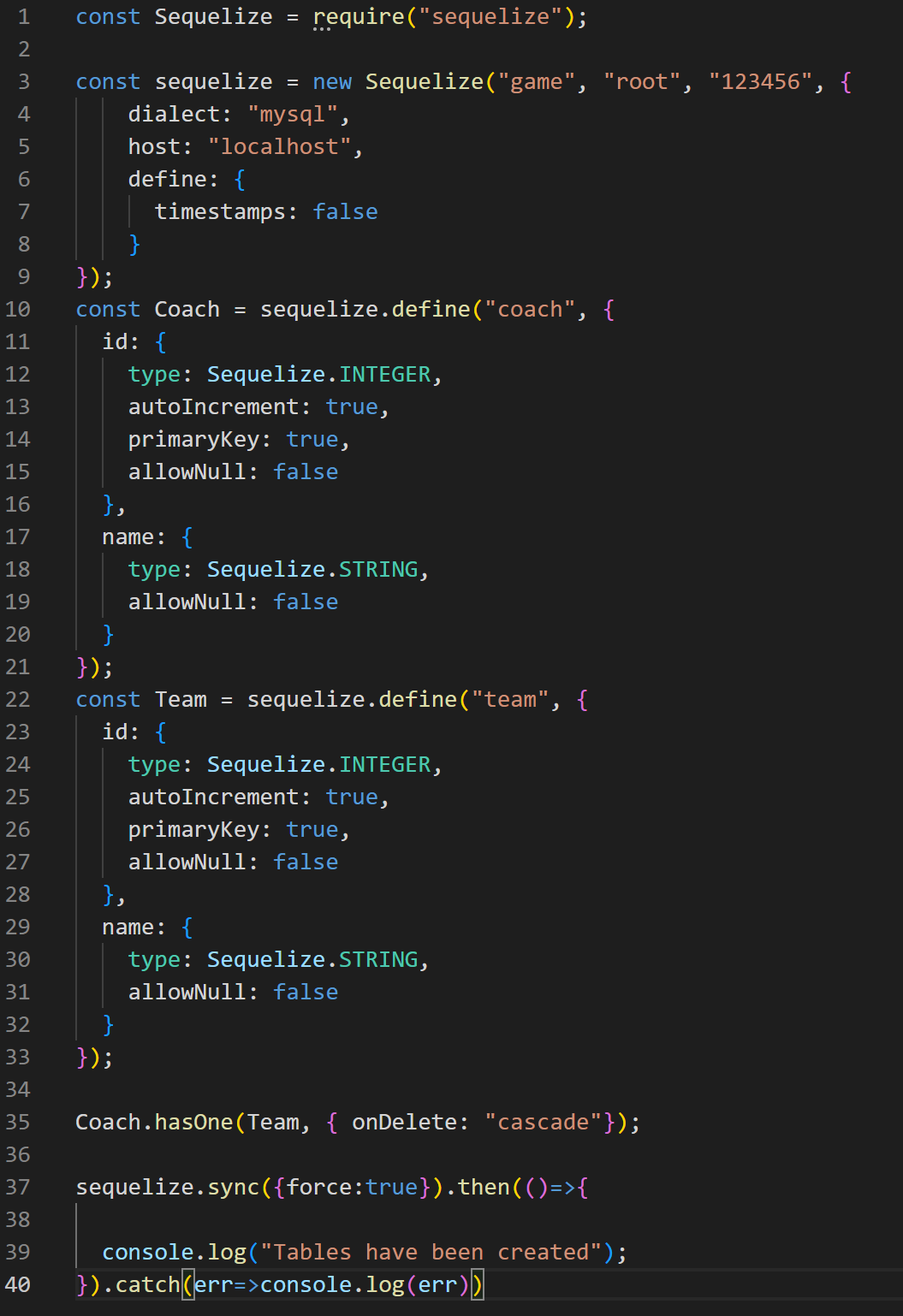


Все остальные операции со связанными зависимыми моделями можно проводить также, как и с обычными моделями.

**Связь один-к-одному**

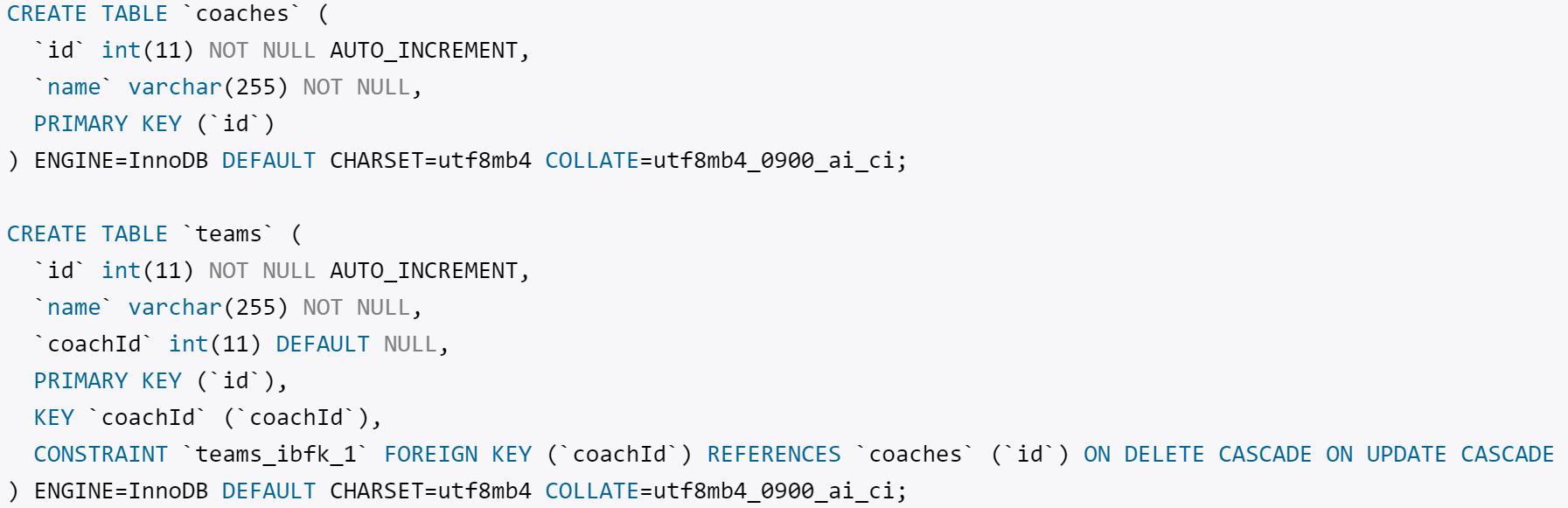
Отношение один к одному указывает, что одна сущность может владеть другой сущностью в единственном экземпляре. Например, у команды может быть только один тренер. С другой стороны, тренер может тренировать одновременно только одну команду.

Для создания подобной связи между моделями применяется метод hasOne(). Например, определим модели тренера и команды:



В данном случае модель тренера (Coach) условно считается главной, а модель команды (Team) зависимой (но в данном случае деление на главную и зависимую модель достаточно условно). Поэтому метод hasOne() вызывается у модели Coach, и в качестве первого параметра передается модель Team. Хотя в данном случае не имеет значения, какая именно модель является главной или зависимой. Второй параметр метода задает конфигурацию связи, в частности, каскадное удаление.

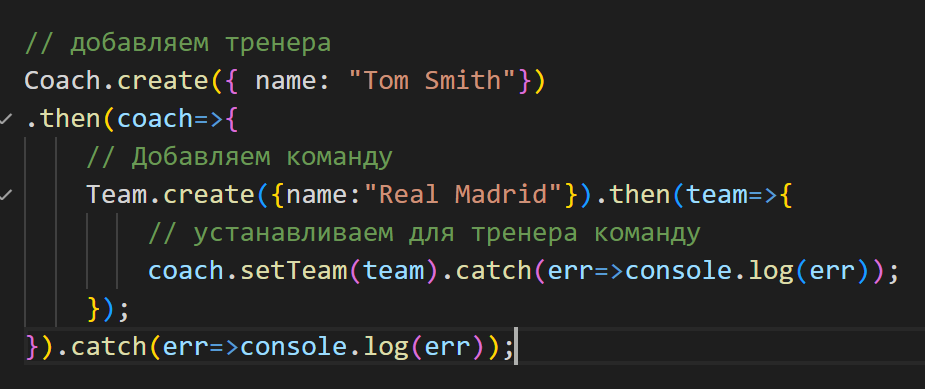
В итоге при выполнении данного кода в MySQL будут созданы следующие таблицы:



Как мы видим, в таблице teams создается дополнительный столбец coachId, через который данная таблица будет связана с таблицей coaches.

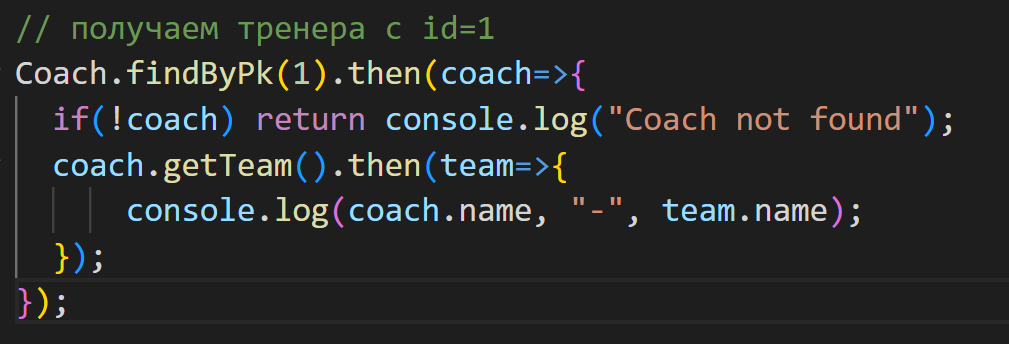
**Добавление и получение связанных данных**

Для установки связанных данных применяется метод setНАЗВАНИЕ\_МОДЕЛИ(). Например, добавим тренера и его команду:



По факту метод setМОДЕЛЬ() будет вызывать SQL-команду UPDATE. То есть к моменту вызова данного метода связываемые сущности уже должны быть в базе данных.

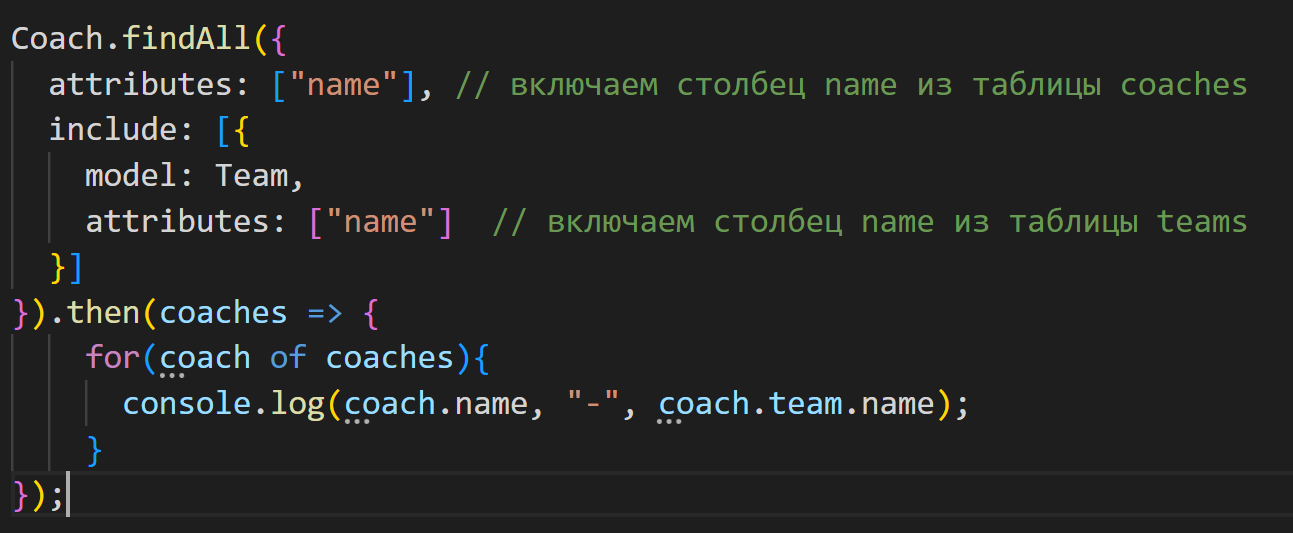
Для получения связанных данных применяется метод getНАЗВАНИЕ\_МОДЕЛИ(). Например, получим тренера и его команду:



В данном случае на консоли мы получим:



Получение всех тренеров с включением связанных данных:



**Связь многие-ко-многим**

Отношение многие-ко-многим предполагает, что сущность одного типа одновременно может иметь связи с множеством сущностей другого типа и наоборот. Например, один студент может посещать несколько университетских курсов. Соответственно один университетский курс может посещаться множеством студентов. То есть есть в данном случае имеем связь многие ко многим.

Физически на уровне базы данных обычно для создания подобной связи создается промежуточная таблица, через которую связываются две основные таблицы. В Sequelize поэтому для связи двух сущностей отношением многие-ко-многим нам надо задать промежуточную модель. Так, возьмем пример с курсами и студентами:



Здесь в качестве промежуточной сущности выступает модель Enrolment - по сути данные успеваемости определенного студента по определенному курсу. В этой модели можно определить различные свойства. Так, в данному случае определено свойство "grade", которое призвано хранить оценку студена по данному курсу. Аналогично в этой модели мы могли бы определить какие-нибудь атрибуты, которые бы связывали студента с курсом, например, дату поступления на данный курс, дату окончания и т.д.

Непосредственно для создания связи многие-ко-многим применяется метод belongsToMany(). Первый параметр метода - сущность, с которой надо установить связь. Второй параметр - объект конфигурации связи, который с помощью параметра through обязательно должен задавать промежуточную сущность, через которую будут связаны обе основные сущности.

В итоге при выполнении данного кода в базе данных будут созданы три таблицы с помощью следующих SQL-команд:

CREATE TABLE IF NOT EXISTS `students` (`id` INTEGER NOT NULL auto\_increment , `name` VARCHAR(255) NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS `courses` (`id` INTEGER NOT NULL auto\_increment , `name` VARCHAR(255) NOT NULL, PRIMARY KEY (`id`)) ENGINE=InnoDB;

CREATE TABLE IF NOT EXISTS `enrolments` (`id` INTEGER NOT NULL auto\_increment , `grade` INTEGER NOT NULL, `studentId` INTEGER, `courseId` INTEGER,

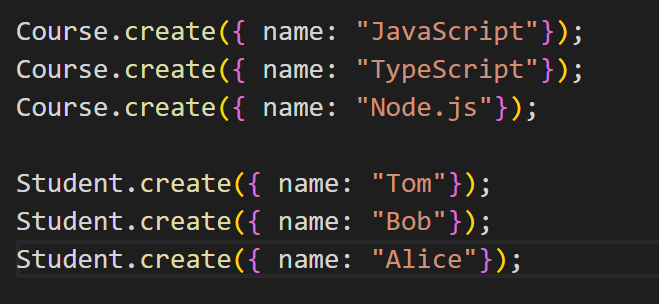
UNIQUE `enrolments\_courseId\_studentId\_unique` (`studentId`, `courseId`), PRIMARY KEY (`id`), FOREIGN KEY (`studentId`) REFERENCES `students` (`id`)

ON DELETE CASCADE ON UPDATE CASCADE, FOREIGN KEY (`courseId`) REFERENCES `courses` (`id`) ON DELETE CASCADE ON UPDATE CASCADE)

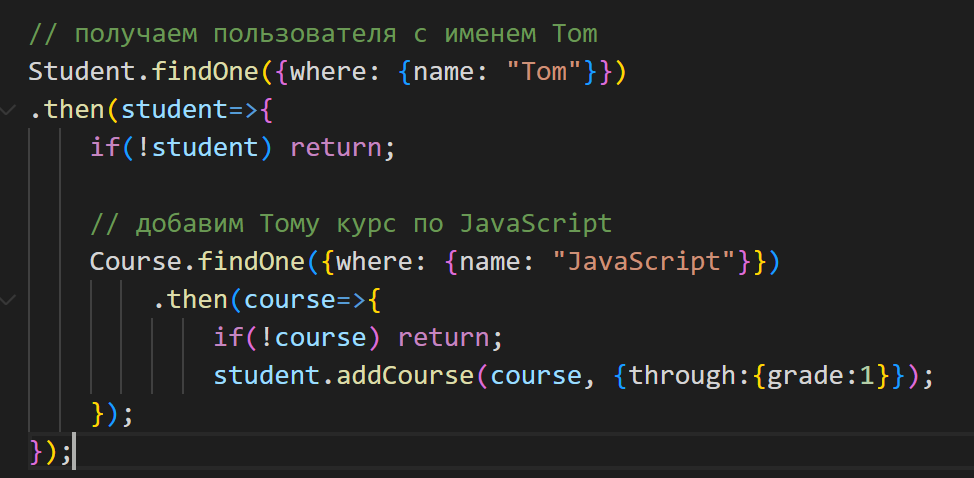
ENGINE=InnoDB;

**Добавление связанных данных**

При установке связи многие ко многим модели могут использовать метод addИМЯ\_МОДЕЛИ() для добавления объектов (например, student.addCourse() и course.addStudent()). Например, пусть у нас будет создано несколько объектов - студентов и курсов:



Добавим студенту с именем Tom курс по JavaScript:

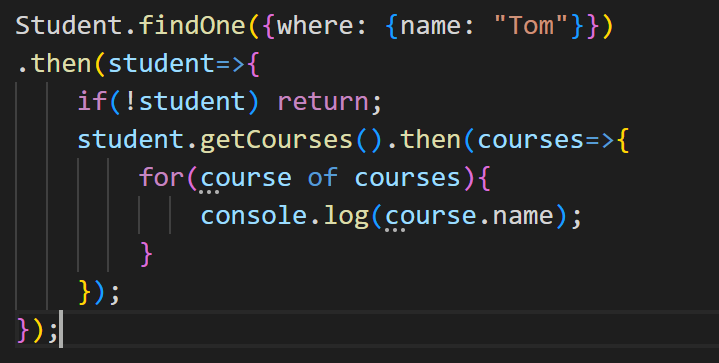


Первым параметром в student.addCourse() передается добавляемый курс. Вторым параметром устанавливается значение для столбца grade в таблице enrolments. В итоге данный метод будет выполнять sql-команду:



**Получение связанных данных**

Для получения связанных данных у каждой из моделей, участвующих в связи, мы можем использовать метод getИМЯ\_МОДЕЛИs(). Например, получим все курсы студента по имени Tom:



Однако в реальности в данном случае мы получаем не просто курс из таблицы courses, а сводные данные на основании таблицы enrolments - выполняемая sql-команда в данном случае будет выглядеть следующим образом:

SELECT `course`.`id`, `course`.`name`, `enrolment`.`id` AS `enrolment.id`, `enrolment`.`grade` AS `enrolment.grade`, `enrolment`.`studentId` AS `enrolment.studentId`, `enrolment`.`courseId` AS `enrolment.courseId` FROM `courses` AS `course` INNER JOIN `enrolments` AS `enrolment` ON

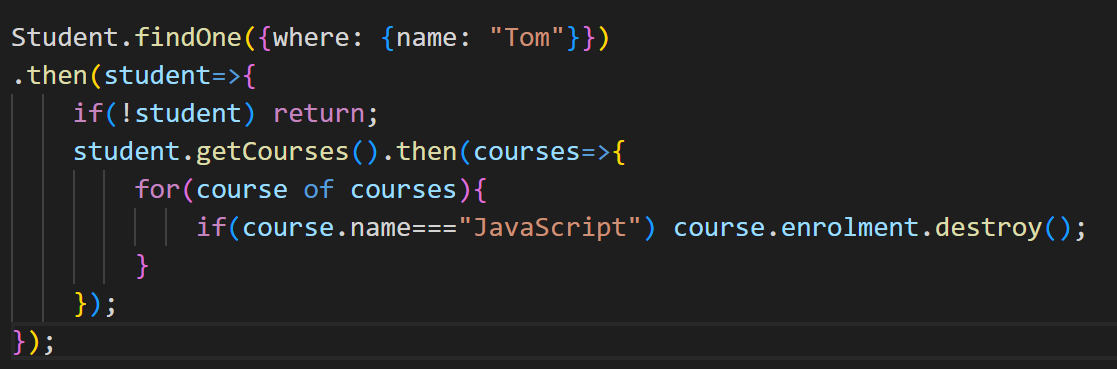
`course`.`id` = `enrolment`.`courseId` AND `enrolment`.`studentId` = 2;

То есть в данном случае мы сможем получить название и id курса, а также id и значение grade объекта Enrolment:



**Удаление связанных данных**

Для удаления связанных данных необходимо получить объект из промежуточной таблицы и удалить его. Например, удалим у студента по имени Tom курс JavaScript:



**Этапы выполнения:**

1. ***Реализуйте в Sequelize связь один-ко-многим;***
2. ***Продемонстрируйте SQL-код (две таблицы);***
3. ***Создайте объект зависимой модели;***
4. ***Добавьте зависимую модель через главную модель;***
5. ***Получите все товары компании с id=1;***
6. ***Реализуйте связь один-к-одному и продемонстрируйте получившиеся таблицы;***
7. ***Установите связанные данные;***
8. ***Получите связанные данные;***
9. ***Реализуйте связь многие-ко-многим и продемонстрируйте получившиеся таблицы;***
10. ***Установите связанные данные;***
11. ***Получите связанные данные;***
12. ***Удалите связанные данные.***