Объекты модели также переопределяют Ruby-методы `id()` и `hash()`, чтобы те ссылались на первичный ключ модели. Это означает, что объекты модели с правильными идентификаторами могут использоваться в качестве ключей хэша. Это также означает, что несохраненные объекты модели не могут использоваться в качестве надежных ключей хэша (поскольку у них еще нет правильного идентификатора).

И еще одно заключительное замечание: Rails рассматривает два объекта модели эквивалентными (используя метод `==`), если они являются экземплярами одного и того же класса и имеют один и тот же первичный ключ. Это означает, что несохраненные объекты модели могут сравниваться в качестве эквивалентных, даже если у них разные значения свойства. Если вам когда-нибудь придется сравнивать несохраненные объекты модели (что случается довольно редко), вам может потребоваться переписать метод `==`.

Как вы увидите, столбцы `id` также играют важную роль в установке связей между таблицами.

Определение связей в моделях

Active Record поддерживает три типа связей между таблицами: «один к одному», «один ко многим» и «многие ко многим». Эти связи указываются путем добавления к моделям объявлений `has_one`, `has_many`, `belongs_to` и еще одного объявления с удивительным названием — `has_and_belongs_to_many`.

Связи типа «один к одному»

Связь типа «один к одному» (или, точнее, связь «один к нулю или к одному») реализуется с использованием внешнего ключа в одной строке одной таблицы, который ссылается не более чем на одну строку в другой таблице. Эта связь может осуществляться между заказами (`orders`) и выставленными счетами (`invoices`): для каждого заказа не может быть больше одного счета (рис. 19.1).

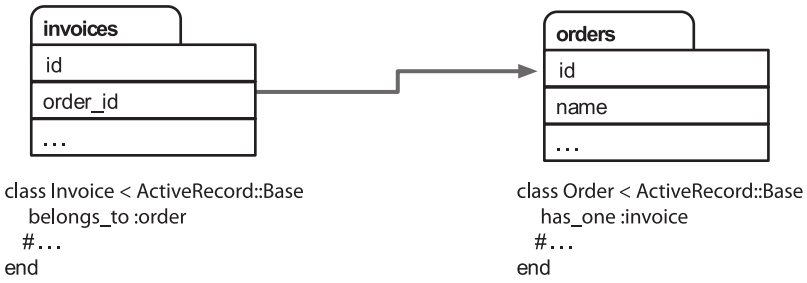

Рис. 19.1. Связь «один к одному»

Из примера видно, что эта связь указана в Rails добавлением к модели `Order` объявления `has_one` и добавлением к модели `Invoice` объявления `belongs_to`.

Здесь проиллюстрировано одно важное правило: модель для таблицы, имеющей внешний ключ, *всегда* содержит объявление `belongs_to`.

Связи типа «один ко многим»

Связь «один ко многим» позволяет представлять коллекцию объектов. Например, в заказе может быть любое число связанных с ним товарных позиций. В базе данных все строки товарных позиций, принадлежащих конкретному заказу, содержат столбец внешнего ключа, ссылающийся на этот заказ (рис. 19.2).

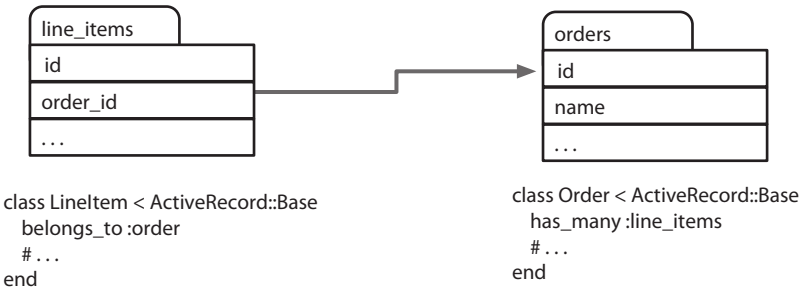

Рис. 19.2. Связь «один ко многим»

В Active Record родительский объект (тот самый, который логически содержит коллекцию дочерних объектов) использует для объявления своих связей с дочерней таблицей `has_many`, а для дочерней таблицы, чтобы указать на ее родителя, используется объявление `belongs_to`. В нашем примере класс `LineItem` принадлежит заказу: `belongs_to :order`, а класс, относящийся к таблице `orders`, содержит множество товарных позиций: `has_many :line_items`.

Опять-таки учтите: поскольку товарная позиция (`line item`) содержит внешний ключ, у нее имеется объявление о принадлежности `belongs_to`.

Связи типа «многие ко многим»

В завершение нужно определить категории наших товаров. Товар может принадлежать многим категориям, и каждая категория может содержать множество товаров. Это и является примером связи «многие ко многим». Похоже на то, что каждая из взаимосвязанных сторон содержит коллекцию элементов другой стороны (рис. 19.3).

В Rails это отношение выражается добавлением объявления `has_and_belongs_to_many` к обеим моделям. Связи «многие ко многим» являются симметричными, обе связываемые таблицы объявляют свою связь друг с другом, используя сокращение `habtm`.

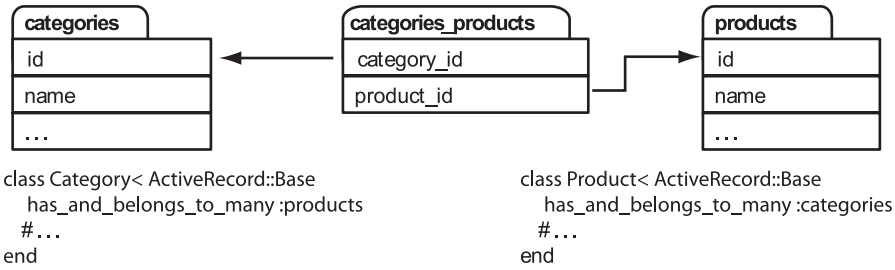

Рис. 19.3. Связь «многие ко многим»

Rails реализует связи «многие ко многим», используя промежуточную объединительную таблицу. В ней содержатся пары внешних ключей, связывающие две заданные таблицы. Active Record предполагает, что имя этой объединительной таблицы является объединением имен двух заданных таблиц, следующих в алфавитном порядке. В нашем примере объединяются таблицы `categories` и `products`, поэтому Active Record будет искать объединительную таблицу по имени `categories_products`.

Объединительные таблицы можно также определить непосредственным образом. В приложении Depot мы определили объединительную модель `LineItems`, которая объединяла `Products` либо с `Carts`, либо с `Orders`. Ее самостоятельное объявление также предоставило нам место для хранения дополнительных свойств, в частности `quantity` (количество).

После определений данных следующим вполне естественным побуждением будет доступ к данным, содержащимся в базе данных, — давайте к нему и приступим.

19.3. Создание, чтение, обновление, удаление (CRUD — Create, Read, Update, Delete)

Такие названия, как SQLite и MySQL, подчеркивают, что весь доступ к базе данных осуществляется путем использования языка структурированных запросов —Structured Query Language (SQL). В большинстве случаев Rails все возьмет на себя, но это целиком и полностью зависит от вас. Вскоре будет показано, что вы можете предоставлять выражения или даже полные SQL-инструкции, выполняемые базой данных.

Если вы уже знакомы с языком SQL, при чтении этого раздела обратите внимание на то, как Rails предоставляет места для знакомых выражений, таких как `select`, `from`, `where`, `group by` и т. д. Если вы еще не знакомы с SQL, вы узнаете, что одной из сильных сторон Rails является то, что вы можете отложить углубление в подобные области до тех пор, пока вам на самом деле не понадобится получить доступ к базе данных на этом уровне.