Связи между сущностями

Между сущностями можно определить связи «один к одному», «один ко многим» и «многие ко многим».

Связи полиморфны, то есть значением связи может быть объект не только того типа, который указан в определении связи, но и любого из его подтипов.

Если между двумя сущностями есть ассоциация, то к соответствующему полю (свойству) сущности должна быть применена одна из следующих аннотаций моделирования связей: `OneToOne`, `OneToMany`, `ManyToOne`, `ManyToMany`. Для ассоциаций, в которых не указан тип противоположной стороны (например, если используется непараметризованная коллекция), в аннотации необходимо указать сущность, находящуюся с противоположной стороны отношения.

Связи могут быть одно- и двунаправленными. У двунаправленной связи есть владеющая и инверсная стороны. У однонаправленной связи есть только владеющая сторона. Владеющая сторона связи определяет изменения связи в базе данных.

К двунаправленным связям применяются следующие правила:

- Инверсная сторона двунаправленной связи должна ссылаться на владеющую сторону (точнее, на поле или свойство сущности, представляющее противоположную сторону связи) с помощью элемента `mappedBy` аннотаций `OneToOne`, `OneToMany`, или `ManyToMany`.
- В двунаправленных связях вида «один ко многим» владеющей должна быть сторона «многие», вследствие чего в аннотации `ManyToOne` нельзя указать элемент `mappedBy`.
- В двунаправленных связях вида «один к одному» владеющей считается сторона, которая содержит соответствующий внешний ключ.
- В двунаправленных связях вида «многие ко многим» владеющей может быть любая сторона связи.

Провайдер персистентности выполняет объектно-реляционное преобразование связей, включая их загрузку и сохранение в базе данных, а также обеспечение ссылочной целостности, как указано в базе данных (например, с помощью внешних ключей).

Целостность связей во время выполнения, в том числе взаимное соответствие сторон «один» и «многие» двунаправленной связи, должно обеспечивать само приложения.

Если при загрузке сущности из базы данных отсутствуют связанные с ней сущности со стороны «многие», провайдер персистентности должен в качестве значения связи вернуть пустую коллекцию, а не `null`.

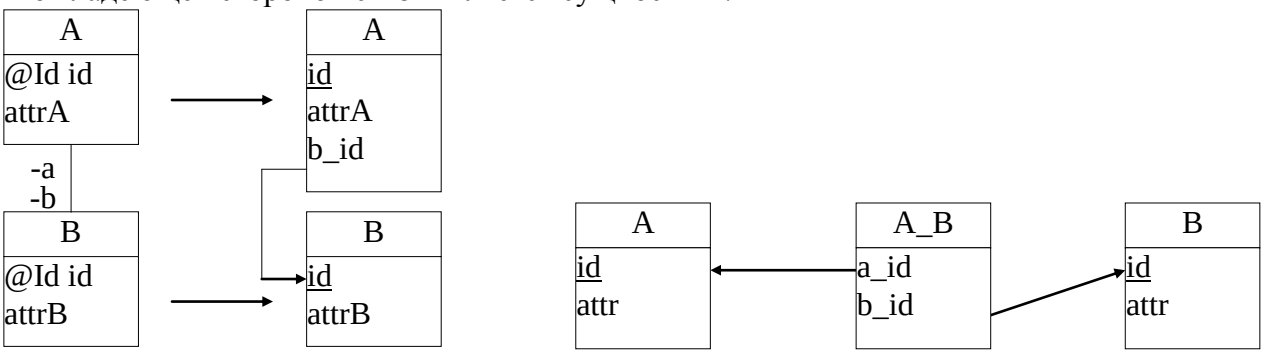
Отображение связей по умолчанию

Во всех рассматриваемых ситуациях сущность A является владельцем связи, сущности A и B отображаются на таблицы A и B, соответственно.

Ситуация	Внешний ключ в таблице A[1]	Промежуточная таблица A_B[2]	Уникальность внешнего ключа на таблицу B[3]
A (1) – (1) B A (1) –> (1) B	+	–	+
A (N) – (1) B A (N) –> (1) B	+	–	–
A (N) – (N) B A (N) –> (N) B	–	+	–
A (1) –> (N) B	–	+	+

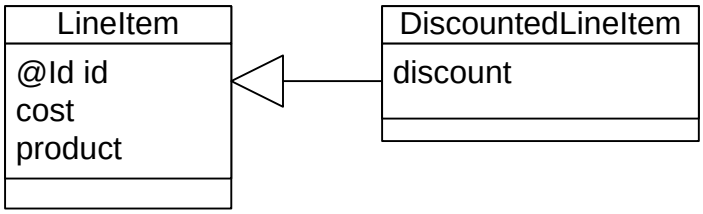
Примечания:
1. Таблица A содержит внешний ключ на таблицу B. Внешний ключ называется *<поле A>_<первичный ключ B>* и совпадает по типу с первичным ключом таблицы B.
2. Существует промежуточная таблицы A_B (сначала идет название сущности-владельца), в которой есть два внешних ключа. Один из них указывает на таблицу A, имеет такой же тип, как первичный ключ таблицы A, и называется *<поле B>_<первичный ключ A>*. Другой указывает на таблицу B, имеет такой же тип, как первичный ключ таблицы B, и называется *<поле A>_<первичный ключ B>*.
3. Для внешнего ключа на таблицу B (независимо от того, определен он в таблице A или в промежуточной таблице A_B) определено ограничение уникальности.

Пример отображения по умолчанию связанных сущностей A и B на таблицы, при условии, что владеющей стороной связи является сущность A:



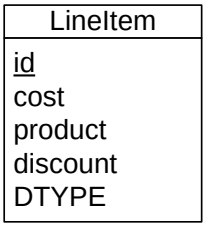
Наследование

Сущности поддерживают наследование, полиморфные ассоциации и полиморфные запросы.
Сущностями могут быть объявлены как конкретные, так и абстрактные классы. Абстрактная сущность отображается как любая другая сущность и может указываться в запросах (при этом действия будут выполняться над экземплярами ее конкретных подклассов).
Сущность может наследовать от класса, который обеспечивает состояние сущности и информацию об отображении, но не является сущностью. Как правило, такой отображенный родительский класс используется для определения состояния и отображения, общих для множества классов сущностей.
Отображенный родительский класс, в отличие от сущности, нельзя указывать в запросах или использовать в других операциях с сущностями. Отображенный родительский класс не может быть инверсной стороной в связи между сущностями.
Отображенному родительскому классу не соответствует отдельная таблица, а указанная в нем информация об отображении применяется к сущностям-наследникам и может быть переопределена в них с помощью аннотаций `AttributeOverride` и `AssociationOverride`.
Для обозначения отображенного родительского класса используется аннотация `MappedSuperclass`.
Классы сущностей могут расширять классы не-сущностей, и наоборот. Родительский класс не-сущность служит только для наследования поведения. Состояние родительского класса не-сущности не является постоянным и не становится постоянным при наследовании его в классе сущности. Классы не-сущности нельзя указывать в запросах или использовать в других операциях с сущностями.
Существует три основные стратегии для отображения иерархии классов в реляционную базу данных. Рассмотрим их на следующем примере:

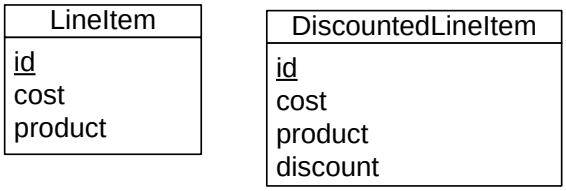


Стратегии отображения иерархии классов:

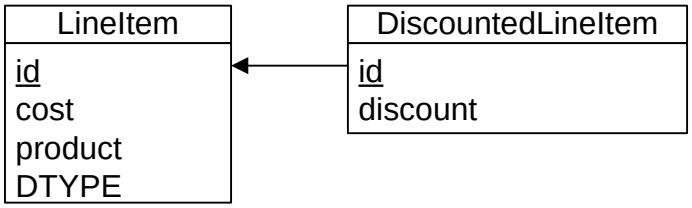
1) единственная таблица на иерархию классов – в таблице имеется столбец-дискриминатор, позволяющий определить точный класс конкретной записи;



2) отдельная таблица на каждый конкретный класс (поддержка данной стратегии необязательна);



3) соединение таблиц подклассов – поля, относящиеся только к конкретному подклассу, отображаются на отдельную таблицу, тогда как общие для иерархии классов поля отображаются на таблицу, соответствующую родительскому классу; для чтения экземпляра подкласса из базы данных выполняется соединение родительской и необходимой дочерней таблицы. Для определения точного класса конкретной записи в таблицу, соответствующую родительскому классу, вводится столбец-дискриминатор.



Каждая из этих стратегий обладает определенными достоинствами и недостатками.

Стратегия	Достоинства	Недостатки
Единственная таблица на иерархию классов	Хорошая поддержка полиморфных связей и полиморфных запросов.	1. Столбцы для состояния подклассов должны допускать неопределенные (null) значения. 2. При добавлении подклассов необходимо изменять структуру таблицы.
Отдельная таблица на каждый конкретный класс		1. Слабая поддержка полиморфных связей. 2. Для полиморфных запросов необходимо использовать операцию объединения (UNION).
Соединение таблиц подклассов	Поддержка полиморфных связей.	Для чтения экземпляра подкласса и полиморфных запросов требуется выполнять соединение таблиц, что может иметь неприемлемую производительность в глубокой иерархии классов.

Для выбора стратегии предназначена аннотация `Inheritance`, которая должна быть применена к корневому классу иерархии.
Если в стратегии отображения иерархии классов требуется столбец-дискриминатор, то его параметры задаются с помощью аннотации `@DiscriminatorColumn`, которая имеет следующие основные элементы:

- `String name` – название столбца (по умолчанию – `DTYPE`);
- `DiscriminatorType discriminatorType` – тип столбца, допускается использовать строковый, символьный или целочисленный тип (по умолчанию используется строковый – `DiscriminatorType.STRING`).

Если столбец-дискриминатор – строкового типа, то его значением по умолчанию является имя сущности (неквалифицированное имя класса сущности). Использование символьного или целочисленного типа позволяет уменьшить размер записи таблицы, но в этом случае для каждого класса в иерархии нужно задать свое значение дискриминатора с помощью аннотации `@DiscriminatorValue`.