Глава 8

Проектирование баз данных

Можно сказать, что информационные системы являются многопользовательскими системами по определению. Это свойство само по себе требует наличия базы данных, с которой могут одновременно работать многие пользователи. Прикладные программы зависят от базы данных, однако, обратное утверждение неверно. Вывод очевиден — надлежащий проект базы данных, который может объединить и поддерживать прикладные программы, является необходимым условием реализации информационной системой предусмотренных функциональных возможностей.

В языке UML диаграмма классов определяет структуры данных, требуемые приложением. Структуры данных, которые постоянно находятся в базе данных, моделируются с помощью классов-сущностей, а также как отношения между классами-сущностями. Классы-сущности необходимо отобразить в структуры данных, распознаваемые базой данных. Эти структуры данных изменяются в зависимости от базовой модели базы данных, которая может быть объектно-ориентированной, объектно-реляционной или реляционной.

В этой главе рассматривается отображение объектов в базы данных. Здесь объясняется, каким образом осуществляются преобразования классов-сущностей, ассоциаций, агрегаций и обобщений в структуры данных, имеющиеся в распоряжении каждой из трех моделей баз данных. Рамки книги не позволяют коснуться вопроса "интеграции схемы", т.е. интеграции перекрывающихся структур базы данных, которые возникают в результате требований со стороны многих прикладных программ, конкурирующих за одни и те же ресурсы базы данных.

8.1. Уровень постоянных объектов базы данных

На протяжении всей книги мы постоянно проводили различие между разработкой клиентского приложения и проектированием сервера баз данных. Мы подчеркивали, что модель классов и пакеты классов *BCED* (разд. 6.1.3.2) отражают классы приложения, а не структуры хранения баз данных.

Классы-сущности представляют постоянные объекты базы данных для приложения, однако, *они не являются* постоянными классами для базы данных. *Классы базы данных* инкапсулируют взаимодействие между приложением и базой данных, однако они ни в коей

мере не являются постоянными классами. Нам по-прежнему необходимо спроектировать *уровень архитектуры для постоянных объектов базы данных* (*persistent data base layer*).

Архитектурный уровень постоянных объектов базы данных может поддерживаться различными типами СУБД: реляционными (например, Sybase, DB2, Oracle8), объектно-реляционными (например, UniSQL, Oracle8) или объектными (например, ObjectStore, Versant). Маловероятно, чтобы модель хранения для новой системы принадлежала к одному из устаревших типов моделей наподобие иерархической (например, IMS), сетевой (например, IDMS), инвертированной или аналогичной модели (например, Total, Adabas). В некоторых случаях, но не в случае действительно современных приложений ИС, уровень постоянного хранения может быть реализован с использованием простых плоских файлов.

8.1.1. Модели данных

Специалисты по базам данных выработали свое собственное представление о мире моделирования. Базы данных *хранят данные*. Исторически специалисты по базам данных концентрировались на *моделях данных* (т.е. на языке UML — моделях состояния). Современные возможности баз данных по *хранению и выполнению программ* распространили эту перспективу на *модели поведения* (ориентированные на триггеры и хранимые процедуры), однако моделирование данных по-прежнему остается альфой и омегой разработки баз данных.

Модель данных (*data model*) (называемая также *схемой базы данных* (*database schema*) — это абстракция, которая представляет структуры базы данных в терминах более понятных, чем эти страшные биты и байты. Широко известная классификация уровней модели данных выделяет три уровня абстракции.

1. Внешняя (концептуальная модель).
2. Логическая модель данных.
3. Физическая модель данных.

Внешняя схема (*external schema*) представляет обобщенную концептуальную модель данных, необходимую для единственного приложения. Поскольку база данных обычно поддерживает много приложений, конструируются несколько внешних схем. Затем они интегрируются в виде одной концептуальной модели данных. Наиболее известным методом концептуального моделирования данных являются ER-диаграммы (entity-relationship — сущность связь) [51].

Логическая схема (*logical schema*) (иногда называемая также глобальной концептуальной схемой) представляет модель, которая отражает структуры хранения СУБД. Именно глобальная интегрированная модель поддерживает текущие и перспективные приложения, которым требуется доступ к информации, хранимой в базе данных.

Физическая схема (*physical schema*) отражает специфику конкретной СУБД. Она определяет способ фактического хранения данных в энергонезависимых устройствах памяти, как правило, это дисковые накопители. Физическая схема касается таких вопросов, как использование индексов и кластеризация данных для повышения эффективности обработки данных.

Конструкторские CASE-средства (т.е. средства, направленные на проектирование и реализацию систем) предоставляют разработчикам единый метод моделирования данных для логической и физической схемы. Обычно такие средства трактуют подобную комбинированную модель как физическую модель данных.

На рис. 8.1 показано, каким образом в UML моделируется отношение приложения и модели постоянно хранимых объектов базы данных. Классы пакетов-сущностей представляют "бизнес-объекты" приложения. Диаграмма классов UML (для классов-сущностей) может заменить ER-диаграмму в качестве альтернативного средства концептуального моделирования баз данных.

Пакеты баз данных не "направляют" процесс моделирования баз данных, они скорее направляются им. Пакет баз данных изолирует модель приложения от модели базы данных, проектируется одновременно с определением уровня архитектуры постоянно хранимых объектов базы данных или после определения этого уровня. Пакет баз данных разъединяет классы-сущности от схемы базы данных. Он устанавливает отображение между объектами и базой данных.

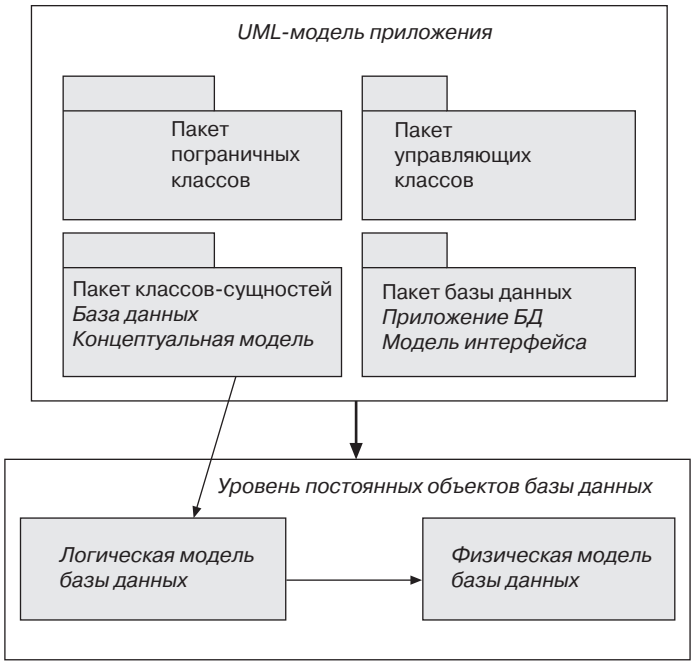


Рис. 8.1. UML и модели постоянно хранимых объектов

8.1.2. Отображение объектов в базу данных

Отображение между приложением и базой данных, за которое отвечает пакет баз данных, может быть очень запутанным вопросом (разд. 6.1.3). В основе трудностей отображения лежат две фундаментальные причины. Во-первых, структуры хранения базы данных могут не иметь практически ничего общего с объектно-ориентированной парадигмой. Во-вторых, база данных почти никогда не проектируется для единственного приложения.

Первая причина равнозначна преобразованию классов пакета сущностей (разд. 5.2.4 и 6.1.3.2) в *отличные от объектно-ориентированных структур*, обычно — реляционные таблицы. Даже если целевая база данных является объектной базой данных, особенности базы данных потребуют аккуратного преобразования.

Вторая причина требует оптимального проектирования базы данных *для всех приложений*, а не только для одного приложения, рассматриваемого в данный момент. Приложениям следует назначить приоритеты с точки зрения их значения для бизнес-процессов, так что структуры базы данных настраиваются на те приложения, которые наиболее важны для организации. Не менее важно, чтобы проектировщик базы данных всегда смотрел вперед, предвидел будущие требования к данным со стороны перспективных приложений и проектировал базу данных таким образом, чтобы адаптироваться к этим требованиям.

Уровень постоянно хранимых объектов базы данных имеет все шансы оказаться *реляционной базой данных*. Технология реляционных баз данных превалирует на рынке. Что касается баз данных больших предприятий, то их переход на *объектную технологию* может совершаться эволюционным путем и должен обязательно включать промежуточный (если не конечный) этап освоения *объектно-реляционной технологии*.

Мы начнем рассмотрение с представления идеального, хотя и маловероятного, сценария, в котором в качестве постоянного хранилища используется чисто *объектная база данных*. Затем обсудим отображение из объектной в *объектно-реляционную базу данных*. И наконец, мы рассмотрим модель *реляционной базы данных*, отличающуюся наиболее жесткими ограничениями и, поэтому, наиболее трудную для отображения модель.

8.2. Модель объектной базы данных

Модель объектной базы данных (ОБД) доставляет меньше всего хлопот при отображении структур данных между прикладной программой и базой данных. В действительности, главенствующей целью объектной СУБД является прозрачная интеграция базы данных с языком программирования, на котором написано приложение.

Консорциум ODMG (Object Data Management Group — Группа по управлению объектными данными) стандартизировал модель ОБД. Организации, входящие в состав ODMG, представляют всех основных поставщиков ПО СУБД. Совсем недавно ODMG изменила направления своей деятельности, сконцентрировав основные усилия на *отображении объектов в реляционные и другие типы баз данных*. Эти усилия нашли свое выражение в разработке стандартного *API-интерфейса объектной памяти* (*Object Storage API*), который способен работать с любыми постоянными источниками данных. В сущности, этот стандарт можно использовать в качестве пакета баз данных (рис. 8.1) для отображения между приложением и базой данных. Последний стандарт (январь 2000 года) получил название объектного стандарта данных (Object Data Standard): ODMG 3.0 [58].

Стандарт определяет, что система управления объектными базами данных (СУОБД) не обеспечивает отдельного языка для баз данных (наподобие SQL) для манипулирования данными *в пределах* среды языка программирования. Вместо этого он предусматривает появление объектов базы данных в языке программирования приложений в качестве обычных объектов языка программирования. Другими словами, язык программирования расширяется за счет объектов базы данных, которые реализуют функции постоянного хранения объектов, управления транзакциями, навигационные запросы (т.е. запросы, которые позволяют "перемещаться" вдоль отношений) и т.д.

Кроме того, стандарт включает отдельный язык запросов для доступа к базе данных *вне* среды языка программирования. Обычно такой язык называют "объектным

SQL" (OSQL). Язык OSQL расширяет возможности запросов реляционного SQL за счет навигационных запросов и возможности обработки более сложных типов данных, таких как шаблоны (разд. 6.2.2.5).

8.2.1. Элементарные типы модели ОБД

Базовыми элементарными типами модели объектной базы данных выступают объект и литерал [58], [20]. Каждый *объект* обладает идентификатором (OID) (разд. 2.1.1.3). У *литерала* OID отсутствует – роль идентификатор литерала играет его значение.

В ОБД проводится различие между *классом* (*реализацией*) и *типом* (*спецификацией*). Тип может обладать несколькими классами. Например, тип Employee (Сотрудник) может быть реализован в виде Smalltalk-класса и/или Java-класса. Семантика типа позволяет отделить спецификацию от ее различных реализаций. Спецификация абстрактного поведения типа называется интерфейсом. Непосредственно материализовать интерфейс нельзя.

Класс ОБД обладает *свойствами* и *операциями*. Свойство может быть *атрибутом* или *отношением* (т.е. атрибутом, который связывает объект с одним или несколькими другими объектами).

8.2.1.1. Типы литералов и объектов

Одним из главных преимуществ ОБД является встроенная поддержка типов литералов и объектов. Это делает ОБД естественной платформой реализации для объектно-ориентированной разработки ИС. Если переход разработчиков ИС к использованию ОБД и не приобрел массового характера, то только из-за того, что ОБД присущи некоторые другие недостатки (наподобие слабой многопользовательской поддержки), а также благодаря коммерческому и политическому влиянию, оказываемому более мощными производителями СУБД.

Стандарт ODMG 3.0 предусматривает следующие *типы литералов* [58]:

- атомический (простой);
- структурный;
- коллекция (шаблон);
- пустой (null).

Атомические литералы принадлежат к одному из следующих подтипов:

- числовой
  - short (короткое целое со знаком);
  - long (длинное целое со знаком);
  - unsigned short (короткое без знака);
  - unsigned long (длинное без знака);
  - float ((десятичное) число с плавающей точкой одинарной точности);
  - double (число с плавающей точкой двойной точности);
- буквенно-цифровые и специальные символы
  - char (одиночный символ);
  - string (строка символов);
  - boolean (т.е. логические значения "истина" или "ложь");

- octet (двоичная строка для представления "сырых" данных);
- enum (перечислимый список допустимых значений).

Структурированный литерал представляет собой предопределенную структуру данных, состоящую более чем из одного атомического литерала. Структурированный литерал может принадлежать одному из следующих типов:

- дата (например, 9 июля 2001);
- время (например, 11:14);
- временная метка (например, 9 июля 2001 11:14:56);
- интервал (например, 11:14, 11:19).

Литерал-коллекция представляет собой шаблон на литералах, т.е. параметризованный тип (разд. 6.2.2.5), при этом значения формальных параметров являются литеральными значениями. Существуют следующие типы литералов-коллекций:

- set<t> (тип `set` (множество), при этом все элементы принадлежат одному литеральному типу `t`; например, `set<dept_name>`, где `dept_name` принадлежит типу string);
- bag<t> (мультимножество (множество, допускающее дублирование элементов));
- list<t> (упорядоченное (отсортированное) множество);
- array<t> (упорядоченная совокупность элементов, количество которых устанавливается динамически, а каждый элемент можно отыскать по его позиции в совокупности);
- dictionary<t,v> (неупорядоченная последовательность (индекс) пар значений ключей, не содержащая повторяющихся ключей).

Пустой литерал (null) можно задать для любого другого литерального типа (например, string или list<>). Пустой литерал означает пустое значение. Применительно к реляционным базам данных значение null представляет одну из следующих возможностей: "неопределенное в данный момент значение" (например, "Я не знаю даты твоего рождения") или "неприменимое значение" (например, "Если ты мужчина, у тебя не может быть девичьей фамилии"). Пустое значение — это не нулевое значение и не символ пробела, это специальный поток битов, который обозначает пустое значение. (В поле базы данных такое значение выглядит именно как `null`. *Прим. ред.*).

Стандарт ODMG 3.0 предусматривает следующие *типы объектов* [58]:

- атомический объект;
- структурный объект;
- коллекция (набор возможных значений, аналогичный коллекции литералов, однако, параметры принимают значения объектов; например, `set<Dept>`, где `Dept` — это класс).

На рис. 8.2 приведен пример объявления типов. Класс Employee обладает пятью свойствами. Одно из этих свойств (`emp_name`) — структурированный объект. Значением свойства `emp_name` является OID экземпляр класса `PersonName`. Класс `PersonName` не показан, но, судя по всему, состоит из таких атрибутов, как `family_name` (фамилия), `first_name` (имя) и `middle_initial` (инициал).

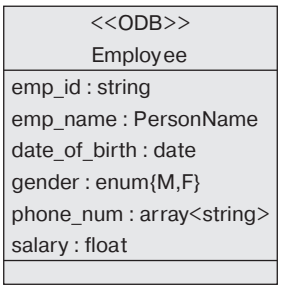


Рис. 8.2. Объявление типов в объектной базе данных

8.2.1.2. Отношения и инверсии

Модель ОБД строится на модельных элементах, рассмотренных в предыдущем разделе. Что касается трех типов отношений (т.е. ассоциации, агрегации и обобщения), модель ОБД непосредственно поддерживает ассоциацию и обобщение. *Агрегация* поддерживается только с помощью ограничения ассоциации.

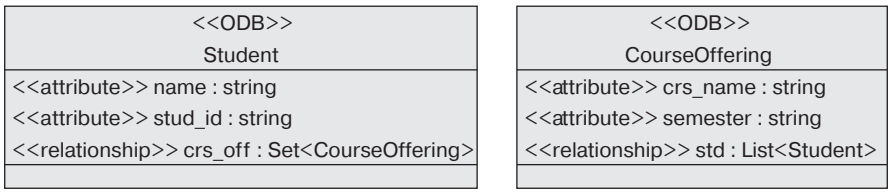
Ассоциация реализуется с использованием типов объектов-коллекций, в частности, множества (`Set<>`) и списка (`List<>`). На практике ассоциации отвечают за переход от реляционно-ориентированного доступа к базе данных, характеризуемого значением, назад к навигационному доступу к базе данных. (Здесь употребляется выражение "переход... назад", поскольку старомодные сетевые базы данных использовали навигацию в качестве присущего им modus operandi.)

На рис. 8.3 показано, каким образом ассоциация типа "многие ко многим" между классами `Student` и `CourseOffering` представляется в модели ОБД. Графическая модель проводит различие между атрибутами и отношениями. Между классами не изображается никаких линий, поскольку свойства отношений (`crs_off` и `std`) уже реализуют эту ассоциацию.

Ключевое слово `inverse`, показанное в определении схемы, накладывает на ассоциацию требование *ссылочной целостности* (*referential integrity*) и исключает возможность появления *зависших указателей* (*dangling pointers*) (т.е. указателей, указывающих на несуществующие (удаленные) объекты. *Прим. ред.*) Например, чтобы добавить объект `Student` к объекту `CourseOffering`, программист может либо добавить объект `CourseOffering` (если быть точным — OID объекта) к множеству курсов `Set<CourseOffering>`, или добавить объект `Student` к списку студентов `List<Student>`. После того, как программист изменяет один конец ассоциации, противоположный конец (инверсный) автоматически изменяется СУОБД.

Хотя это и не показано в приведенных выше определениях классов, модель ОБД позволяет задавать *ключи* (*key*) — уникальные идентифицирующие значения для объектов классов. В отличие от реляционных баз данных ключ не является единственным или даже главным средством идентификации объектов (этой цели служат значения OID). Однако, иногда для поиска данных в ОБД в качестве дополнительной возможности можно воспользоваться значением ключа.

Ключ может быть *простым* (*simple*) или *составным* (*compound*) (в последнем случае он состоит более, чем из одного атрибута). Поскольку у класса может быть несколько ключей, для того, чтобы отличать эти ключи друг от друга, можно использовать последовательные номера (или другой визуальный метод).




```
class Student
{
    attribute     string                name;
    attribute     string                stud_id;
    relationship  Set<CourseOffering>    crs-off
                  inverse   CourseOffering: :std;
};
class CourseOffering
{
    attribute     string                crs_name;
    attribute     string                stud_id;
    relationship  List<Student>         std
                  inverse   Student: :crs_off;
};
```

Рис. 8.3. Представление ассоциации в модели объектной базы данных

8.2.1.3. Наследование типа ISA и EXTENDS

Объектная модель ODMG определяет два типа отношений обобщения. Это отношения типа ISA и EXTENDS. *Отношение типа ISA* соответствует (попросту говоря) данному нами ранее определению наследования интерфейса (разд. 5.3.3). *Отношение типа EXTENDS* соответствует наследованию реализации (разд. 5.3.4). (*ISA* означает не что иное, как *is a* — "является одним из представителей", а *EXTENDS* означает "расширяет". Из приведенного примера ясно, что "класс сотрудников" является представителем "класса личностей", причем не единственным, и в то же время класс "менеджеров" расширяет класс "сотрудников". *Прим. ред.*).

На рис. 8.4 показана расширенная диаграмма UML, содержащая отношения ISA и EXTENDS. Класс "сотрудников" `EmployeeClass` наследует определение интерфейса от класса "личностей" `PersonInterface`. Класс `PersonInterface` — абстрактный класс. Класс "менеджеров" `ManagerClass` наследует реализацию, включая объявление метода `age` (возраст) от класса `EmployeeClass`.

Стандарт ODMG использует ключевое слово *interface* при определении абстрактного класса, наподобие `PersonInterface`. Ключевое слово *class* используется только для определения *конкретного класса*, который может быть материализован, например, такого как `ManagerClass` и `EmployeeClass`.

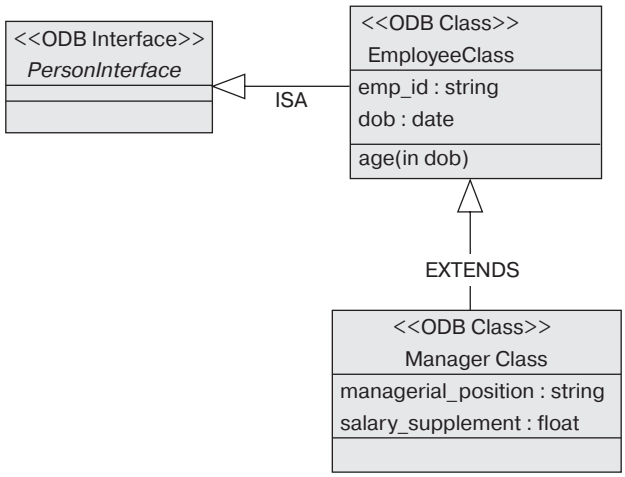


Рис. 8.4. Наследование в модели объектной базы данных

8.2.1.4. Встроенные операции

Поставляемые объектные СУБД оснащены *встроенными операциями*, поддерживающими типы литералов и объектов. К наиболее интересным операциям относятся те, которые поддерживают структурированные типы и коллекции. Чтобы воспользоваться преимуществами интерфейсов СУОБД при разработке проектных моделей, системный проектировщик должен быть знаком с этими интерфейсами.

Ниже приводится перечень некоторых сигнатур поддерживаемых сегодня (или в перспективе) интерфейсов для структурированных типов и коллекций [58]. Назначение операций довольно очевидно из их сигнатур.

- *Date*
  - ushort day_of_year()
  - Month month_of_year()
  - Weekday day_of_week()
  - boolean is_leap_year()
  - boolean is_greater(in Date a_date)
  - boolean is_between(in Date a_date, in Date b_date)
  - Date add_days(in long days)
  - long subtract_date(in Date a_date)
- *Time*
  - ushort millisecond()
  - boolean is_equal(in Time a_time)
  - boolean is_between(in Time a_time, in Time b_time)
  - Time subtract_interval(in Interval an_interval)

  - Interval subtract_time(in Time a_time)
- *Timestamp*
  - ushort millisecond()
  - ushort month()
  - boolean is_between(in Timestamp a_stamp, in Timestamp b_stamp)
  - Timestamp plus(in Interval an_interval)
  - Timestamp minus(in Interval an_Interval)
- *Interval*
  - ushort second()
  - Interval plus(in Interval an_interval)
  - boolean is_less_or_equal(in Interval an_interval)
  - boolean is_zero()

Объекты-коллекции совместно используют несколько операций. Поэтому СУОБД обычно определяют суперкласс с именем `Collection`. Специфические классы коллекций наследуют общее поведение от суперкласса `Collection` (как всегда, унаследованная операция может быть при необходимости уточнена или замещена). К общим операциям относятся следующие.

- unsigned long cardinality()
- boolean is_ordered()
- boolean contains_element(in any element)
- void insert_element(in any element)
- void remove_element(in any element) raises(ElementNotFound)
- *Set*
  - Set create_union(in Set other_set)
  - boolean is_subset_of(in Set other_set)
  - boolean is_proper_superset(in Set other_set)
- *Bag*
  - unsigned long occurrences_of(in any_element)
  - Bag create_intersection(in Bag other_bag)
- *List*
  - any retrieve_element_at(in unsigned long index) raises(InvalidIndex)
  - void insert_element_after(in any element, in unsigned long index) raises(InvalidIndex)
  - void insert_element_first(in any element)
  - void remove_last_element() raises(ElementNotFound)
  - List concat(in List other_list)
  - void append(in List other_list)

- *Array*
  - void remove_element_at(in unsigned long index) raises(InvalidIndex)
  - any retrieve_element_at(in unsigned long index) raises(InvalidIndex)
  - void resize(in unsigned long new_size)
- *Dictionary*
  - any lookup(in any key) raises(KeyNotFound)
  - boolean contains_key(in any key)

8.2.2. Отображение в ОБД

Отображение UML-моделей в ОБД осуществляется относительно беспрепятственно. Основная задача ОБД состоит в том, чтобы предоставить объектно-ориентированную реализацию для объектно-ориентированной модели. Фактически, мы моделируем проект ОБД с помощью обозначенных как стереотипы диаграммы классов для выражения свойств и ограничений ОБД.

Процесс отображения необходимо выполнить для тех фрагментов моделей состояний и поведения, которые относятся к постоянным объектам. По существу отображение ограничено аспектами состояния и поведения классов, входящих в пакет сущностей.

В этой задаче превалирует аспект отображения состояний. Отображение поведения обычно осуществляется как часть архитектурного проектирования (разд. 6.1), проектирования кооперации (разд. 6.2) и проектирования программ клиент/сервер (Chapter 9). В частности, именно в рамках архитектурного проекта должно быть принято начальное решение в отношении того, где должны выполняться различные процессы — на клиенте или на сервере. Это решение влияет на детализированный проект кооперации и проектирование программ клиент/сервер.

8.2.2.1. Отображение классов-сущностей

Внимательный читатель наверняка обратил внимание на неудобное для работы с UML-моделями предположение, касающееся того, что атрибуты классов определены на *элементарных типах данных* и на некоторых *встроенных структурированных типах данных* (`Date`, `Currency`). Ассоциативные роли предполагают, что при проектировании будут использоваться *коллекции* (шаблоны), однако, решение по поводу коллекций обычно откладывается до тех пор, пока не прояснится вопрос их поддержки со стороны выбранной базы данных и среды программирования.

Но что можно ответить на простые вопросы вроде: “Что если у сотрудника несколько телефонных номеров? Каким образом моделировать эту ситуацию в ходе анализа? Действительно ли необходимо держать отдельный класс для телефонных номеров?” Вероятно, можно задать и другие трудные вопросы, например, такие: “Можно моделировать имя сотрудника в виде одного атрибута, обладающего внутренней структурой, отображающей тот факт, что имя состоит из фамилии, собственно имени и инициала посередине? Действительно ли необходимо держать отдельный класс для имен сотрудников?”

Теоретически UML не препятствует расширению системы типов за счет определения новых классов (в ходе анализа) и использования шаблонов (в ходе проектирования). Практически, чтобы избежать разрастания количества маловажных классов, не следует делать этого до тех пор, пока не прояснится вопрос о поддержке, которую

можно получить со стороны платформы реализации для расширяемой системы типов, а равно и для встроенных структурированных типов и коллекций. В результате, вопросы, наподобие приведенных выше, зачастую попросту игнорируются вплоть до этапа проектирования.

Попробуем прояснить эти и аналогичные вопросы, взяв некоторые из наших моделей анализа и отобразив их в схеме классов ОБД. В этом разделе рассматриваются классы, отличные от классов-сущностей. В последующих разделах мы также рассмотрим отношения между классами-сущностями.

Пример 8.1. Управление контактами с клиентами

Обратитесь к спецификациям классов для приложения *Управление контактами с клиентами* в примере 4.6, рис. 4.3 (разд. 4.2.1.2.3). Рассмотрите классы `Contact` и `Employee`. Заметим, что приведенные выше вопросы сформулированы применительно к этим двум классам.

Класс `Contact` обладает атрибутами `family_name` и `first_name`, однако, в нем отсутствует понятие "контактного имени" (т.е. имен контактного лица). Аналогично `Employee` содержит атрибуты `family_name`, `first_name` и `middle_name`, но у нас нет возможности запросить в базе данных "имя работника", поскольку подобного понятия просто не существует.

Класс `Contact` обладает также атрибутами `phone`, `fax` и `email`. Модель в ее настоящем виде не позволяет контактному лицу иметь более одного номера телефона, факса или адреса электронной почты — на практике подобное предположение довольно нереально.

Наша задача в данном примере заключается в отображении классов-сущностей `Contact` и `Employee` в проект ОБД. Это отображение должно быть обращено на указанные проблемы.

Один из вариантов искомого отображения показан на рис. 8.5. Мы ввели два ОБД-интерфейса — `PersonShortName` (Сокращенное имя лица) и `PersonLongName` (Полное имя лица). Первый интерфейс определяет тип данных для "контактного имени" `contact_name`. Мы задаем тип для `contact_name` явно внутри класса `Contact` и дополнительно через отношение `ISA`. Строго говоря, отношение `ISA` является здесь избыточным. Классу `Contact` не требуется наследовать два отдельных атрибута класса `PersonShortName`. Он использует `PersonShortName` только в качестве типа своего собственного атрибута `contact_name`.

Класс `PersonLongName` наследует два атрибута класса через отношение `ISA` и вместе со своим собственным атрибутом образует тип для атрибута `employee_name`. Как и в предыдущем случае, отношение `ISA` между классами `Employee` и `PersonLongName` дает только визуальную связь, в остальном же оно является избыточным.

Возможность контактного лица (`Contact`) обладать несколькими телефонами и факсами, а также адресами электронной почты реализуется с помощью типа `set` — одного из типов коллекций, поддерживаемых ОБД-системами.

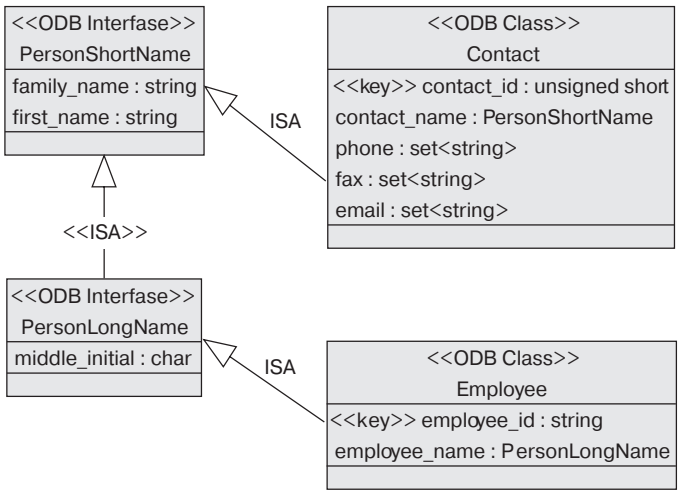


Рис. 8.5. Отображение классов-сущностей на проект ОБД (Управление контактами с клиентами)

8.2.2.2. Отображение ассоциаций

В UML-моделях ассоциации между классами позволяют осуществлять навигацию по объектам этих классов. Это именно то, что составляет сильную сторону объектной базы данных — навигация по объектам, связанным с помощью идентификаторов постоянных объектов.

Поэтому отображение ассоциаций в ОБД — не вызывающая особых затруднений деятельность, что, собственно, и было продемонстрировано на примере ассоциации между классами `Student` и `CourseOffering` в разд. 8.2.1.2 (рис. 8.3). Свойство отношения для класса ОБД обозначается именем класса (или коллекции классов), с которым он связан ассоциативной связью.

В свете сказанного может потребоваться в ходе спецификации отображения оптимизировать проект. В частности, можно принять решение о том, что некоторые атрибуты UML (или классы UML) следует моделировать как интерфейсы ОБД, преследуя при этом цель использования данных интерфейсов как типов для свойств классов ОБД (как показано на рис. 8.5).

Пример 8.2. Управление контактами с клиентами

Обратитесь к спецификациям ассоциаций для приложения *Управление контактами с клиентами* в примере 4.8, рис. 4.5 (разд. 4.2.2.3). Обратитесь также к предыдущему примеру отображения классов-сущностей (рис. 8.5).

Наша задача заключается в отображении модели, представленной на рис. 4.5, в проект ОБД. В процессе отображения может потребоваться выяснить, какие UML-атрибуты или классы наилучшим образом подходят на роль интерфейсов ОБД. Затем можно использовать (повторно использовать) эти интерфейсы для релевантных классов ОБД.

Одно из возможных решений для нашего примера показано на рис. 8.6. Помимо введенных в предыдущем примере *ОБД-интерфейсов* (`PersonShortName` и `PersonLongName`) мы создали *иерархию наследования* (*inheritance hierarchy*) для ОБД-интерфейса `Address`. Затем ввели *атрибуты объектов* `postal_address` и `courier_address` внутрь ОБД-классов

`Organization` и `Contact` в качестве *вложенных атрибутов* (*nested attribute*). Ассоциативные связи на рисунке не показаны, поскольку отношения представлены как свойства классов.

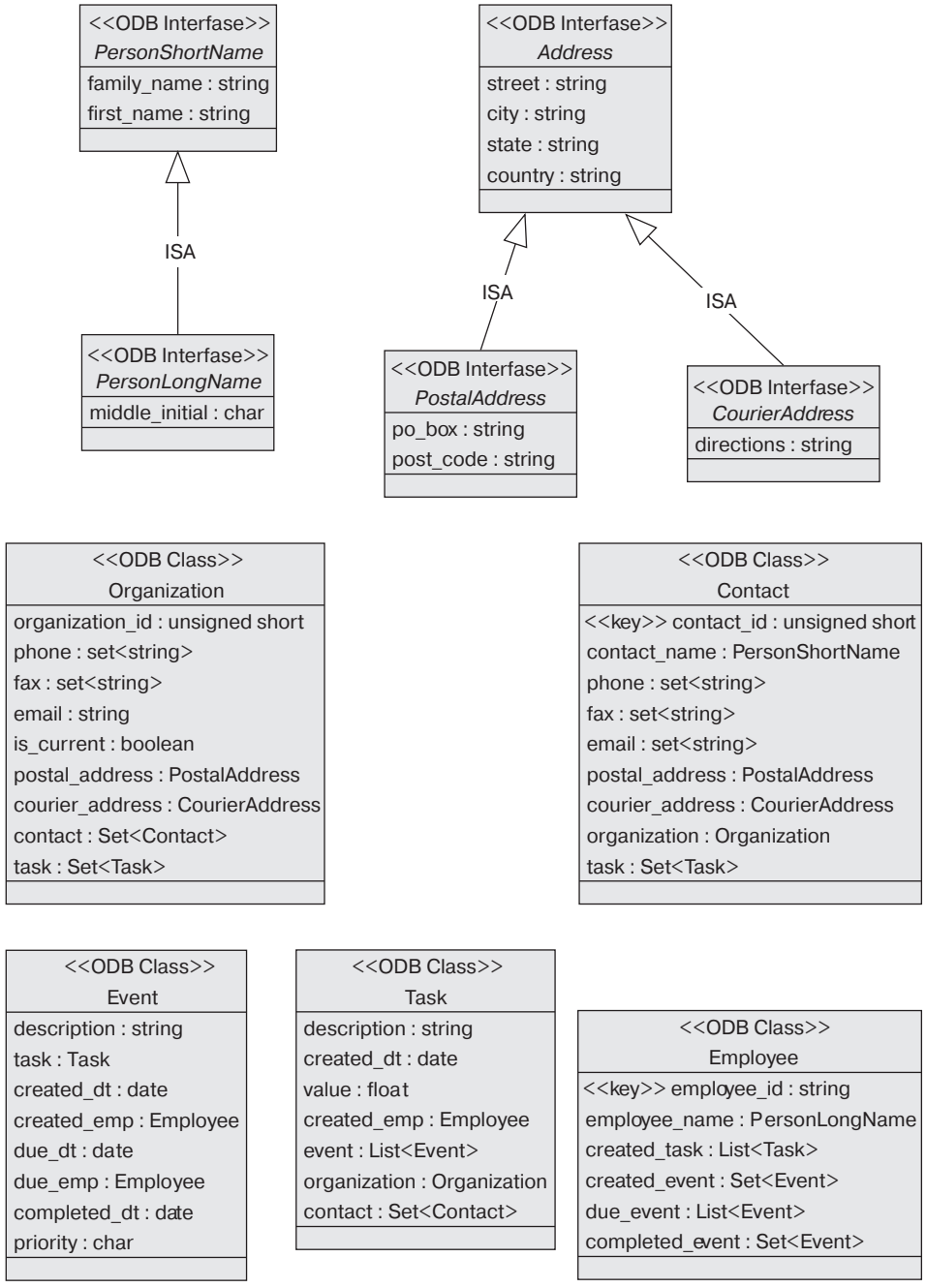


Рис. 8.6. Отображение ассоциаций на проект ОБД (Управление контактами с клиентами)

8.2.2.3. Отображение агрегаций

Как было объяснено ранее (разд. 2.1.4, 4.2.3 и 5.4), UML распознает только две семантики в отношении агрегации — собственно *агрегацию* с семантикой "по ссылке" и *композицию* с семантикой "по значению". Ввиду этих ограничений нотация UML для агрегации конечно не поддерживается напрямую реализацией ОБД (и в этом отношении — любой другой базой данных).

В базе данных отношения агрегации можно моделировать как ассоциации или вложенные атрибуты. Если вводить специальную семантику агрегации, то этого можно добиться скорее с помощью процедурных средств (т.е. ввести семантику прямо в программу), чем декларативных (за счет введения семантики структуры данных).

Принцип отображения агрегаций прост. *Агрегация UML* отображается в модель ОБД так, как будто это ассоциация. *Композиция UML* превращается в составной класс ОБД, который содержит вложенные атрибуты, представляющие компонентный класс. Поскольку компонентный класс обладает внутренней структурой (т.е. не является атомическим), ОБД-интерфейс должен сначала определяться как структурированный тип объекта. Затем вложенные атрибуты принимают значения объектного типа. В целях достижения определенного уровня эффективности, повторного использования, масштабируемости и приспособленности к сопровождению и т.д. возможно применение некоторых вариантов описанной выше стратегии отображения.

Пример 8.3. Запись на университетские курсы

Обратитесь к спецификациям агрегаций для приложения *Запись на университетские курсы* в примере 4.9, рис. 4.6 (разд. 4.2.3.3).

Наша задача заключается в отображении модели, представленной на рис. 4.6, в проект ОБД. Чтобы ввести вложенные атрибуты и упростить модель, может потребоваться определить и использовать некоторые ОБД-интерфейсы как типы атрибутов классов.

На рис. 8.7 показана предлагаемая модель. Мы определили два интерфейса: `YearSemester` и `AcademicRecord`. Первый носит лишь "косметический" характер — модель может обойтись без него. Второй необходим для определения UML-композиции как вложенного атрибута `academic_record` класса `Student`.

UML-агрегация между классами `Course` и `CourseOffering` моделируется как обычная ассоциация. Семантику агрегации может потребоваться ввести процедурно. Свойства отношения определяют отношение агрегации и ассоциации, представленное на рис. 4.6.

8.2.2.4. Отображение обобщений

Отображение отношения обобщения UML на отношения *ISA* и *EXTENDS* ОБД является по существу отношением "один к одному". Отношение *ISA* является результатом наследования от ОБД-интерфейса. Наследование от класса ОБД моделируется с помощью отношения *EXTENDS*.

Преобразование модели обобщения UML в модель ОБД показано на рис. 8.8. Поскольку иерархия обобщения на рис. 4.7 представляла наследование интерфейса, мы использовали отношение *ISA* OMG.

Класс UML `RentalConditions` преобразуется в ОБД-интерфейс. Этот ОБД-интерфейс используется как структурированный объектный тип для атрибута `rental_cond`, встроенного в ОБД-интерфейс `VideoMedium`. Мы принимаем, что класс `RentalConditions` является свойством `VideoMedium` и, следовательно, должен быть встроен в конкретный класс, наследуемый из интерфейса `VideoMedium` (т.е. `BetaTape`, `VHSTape` и `DVDDisk`).

Пример 8.4. Магазин видеопроката

Обратитесь к спецификациям обобщения для приложения *Магазин видеопроката* в примере 4.10, рис. 4.7 (разд. 4.2.4.3).

Наша задача заключается в отображении модели, представленной на рис. 4.7, в проект ОБД. Хотя подклассы UML-модели не содержат своих собственных атрибутов (пока что), мы предполагаем, что они отличаются по состоянию и поведению и в свое время к этим классам будут добавлены свойства.

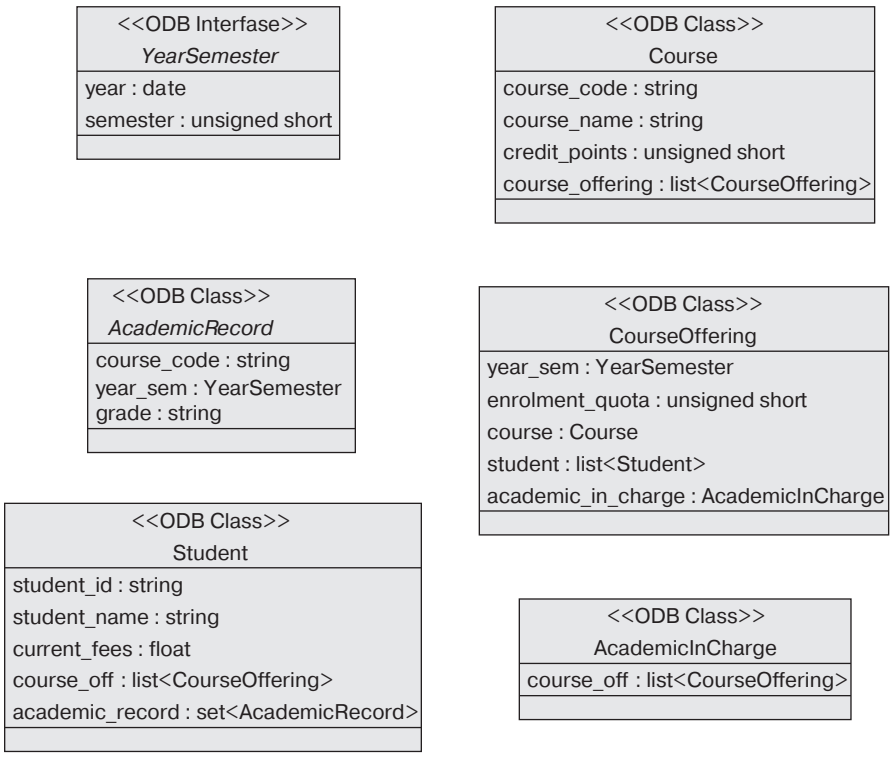


Рис. 8.7. Отображение агрегаций на проект ОБД (Запись на университетские курсы)

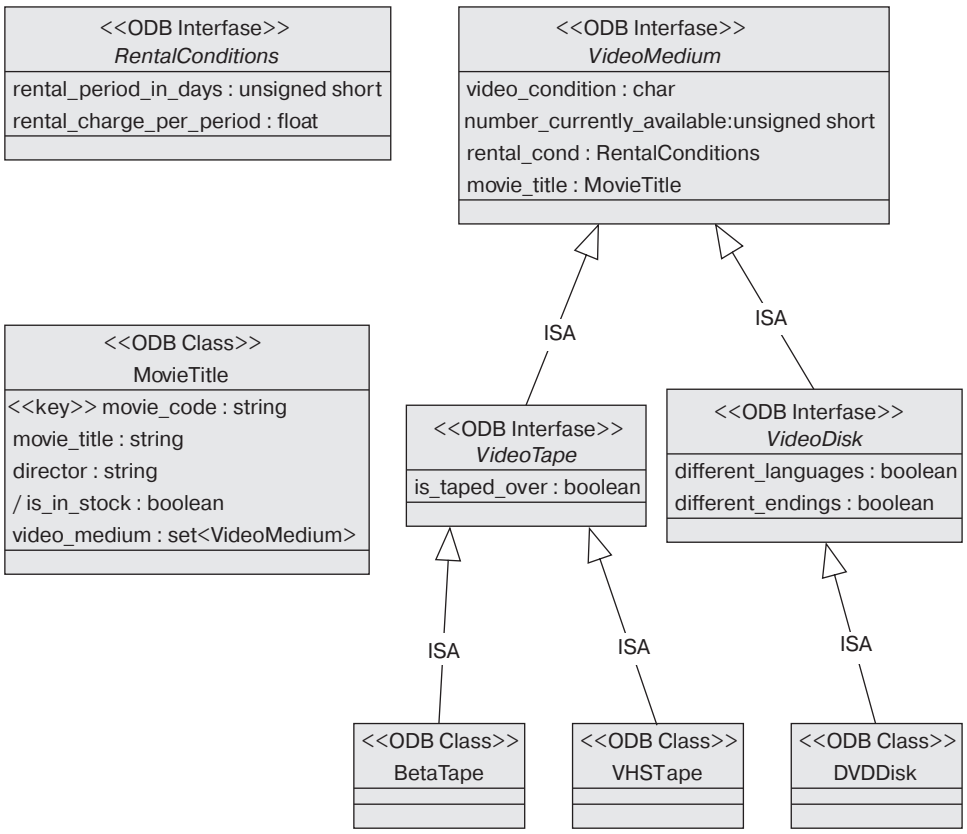


Рис. 8.8. Отображение обобщений на проект ОБД (Магазин видеопроката)

8.3. Объектно-реляционная модель базы данных

Следующей "большой волной" в области технологий баз данных является объектно-реляционная модель [87]. Как ясно из названия, объектно-реляционная база данных (ОРБД) сочетает в себе старомодную реляционную модель и новомодную объектную модель. Одна и та же объектно-реляционная система управления базами данных (СУОРБД) способна обрабатывать реляционные структуры данных (*реляционные таблицы*) и объектные структуры данных (*объектные таблицы*).

Стандарт для модели ОРБД был согласован в 1999 году, после более чем шестилетней разработки (этот стандарт известен под неофициальным именем SQL3). Данный стандарт является результатом труда Американского национального института стандартов (American National Standards Institute – ANSI) и Международной организации по стандартизации (International Organization for Standardization – ISO). Официальное название этого стандарта SQL:1999 [21]. Стандарт оставил многие вопросы, касающиеся ОРБД, нерешенными и должен по положению пересматриваться примерно раз в три года.

Модель ОРБД совместима "снизу вверх" с последним стандартом для реляционных баз данных — так называемым стандартом SQL92. Модель расширяет традиционные возможности реляционных таблиц за счет нового механизма, позволяющего хранить объекты в SQL таблицах. Модель также расширяет ограниченную реляционную поддержку для определяемых пользователем типов за счет введения произвольных сложных *структурированных типов* (чтобы инкапсулировать атрибуты и операции в одном объектном типе — классе).

Хотя стандарт развивался (и продолжает развиваться), большинство поставщиков реляционных баз данных (Oracle, IBM, Informix) взялись за задачу поставки продуктов для СУОРБД, обеспечивающих по меньшей мере частичную поддержку модели ОРБД. Одной из основных проблем для поставщиков остается интеграция ранее существовавших реляционных возможностей с новыми объектно-ориентированными, чтобы сделать возможным беспрепятственный перенос реляционных систем в решения, основанные на использовании ОРБД. Стандарт SQL: 1999 фактически не касается этой проблемы.

В последующем изложении мы стремимся придерживаться спецификаций SQL:1999 [21], однако, время от времени обращаемся к рассмотрению реальных продуктов СУОРБД (в частности, для Oracle 8), чтобы подчеркнуть различия между теорией и практикой.

8.3.1. Элементарные типы модели ОРБД

Элементарные типы модели ОРБД включают новые объектные элементарные типы и прежние реляционные элементарные типы. Принципиально новым объектным элементарным типом является определяемый пользователем *структурированный тип*, соответствующий нотации ОБД для *интерфейса* и нотации UML для *класса*.

Структурированный тип определяется заданием его атрибутов и операций. Его можно также определить как подтип другого структурированного типа. На самом деле модель ОРБД поддерживает *множественное наследование интерфейса* (разд. 5.3.3).

В качестве механизма хранения данных используется *таблица*. Столбцы таблицы могут принимать значения структурированных типов, определяемых пользователем. В некоторых реализациях СУОРБД (например, в Oracle8) подобные таблицы называются *объектными таблицами*, чтобы отличить их от традиционных *реляционных таблиц*.

В ОРБД также поддерживаются структурированные типы и типы коллекций, введенных для ОБД. Специальный вид структурированного типа — называемого *строчным типом* (*row type*) — позволяет задать вложенные структуры данных в рамках объектных таблиц. Строчный тип можно также использовать для определения ссылочного типа (*reference type*). Ссылки обеспечивают возможности навигации по объектам.

8.3.1.1. Особые и структурированные типы

Поля *столбцов* в таблице ОРБД могут принимать значения *встроенных типов* или *типов, определяемых пользователями*. Что касается встроенных типов, то возможности ОРБД аналогичны возможностям, которые обычно реализуются ОБД (разд. 8.2.1.1). То же самое можно сказать в отношении встроенных операций.

Типы данных, определяемые пользователями, можно разделить на две категории:

- *особый тип* (*distinct type*) — выражается в виде единственного предопределенного типа, называемого исходным типом (source type) (особый тип соответствует *атомическому объектному типу* стандарта ODMG);

- *структурированный тип* (*structured type*) — выражается в виде списка определений атрибутов и операций (структурированный тип соответствует *структурированному объектному типу* стандарта ODMG).

Структурированный тип позволяет пользователям вводить свои собственные именованные типы. Определение структурированного типа состоит из объявления следующих компонент.

- *Атрибуты*, представляющие состояние объектов структурированного типа.
- *Операции*, определяющие поведение объектов структурированного типа.
- Операции, определяющие *эквивалентность/неэквивалентность*, *упорядоченность* и *преобразование* объектов структурированного типа (необходимы, если нам требуется сравнивать два объекта структурированного типа на эквивалентность, некоторым образом сортировать их или преобразовывать из одного структурированного типа в другой).

Структурированный тип *атрибута* может определяться с помощью *исходного типа*, *особого типа*, *типа коллекции* или другого структурированного типа. Стандарт SQL:1999 определяет следующие типы коллекций (продукты ОРБД могут определять больше типов коллекций):

- множество (set);
- список (list);
- мультимножество (multiset) (тип, аналогичный типу *bag* ОБД);
- массив (array).

На рис. 8.9 показано определение структурированного типа для класса `Employee`, соответствующее определению для ОБД, приведенному на рис. 8.2. При сравнении определений видно, что их различия незначительны и связаны со встроенными исходными типами (если речь не идет о конкретной СУОРБД, их выбор довольно произволен). Атрибут `gender` (род) введен как `char`, поскольку типичная СУОРБД, как правило, не поддерживает тип `enum`. Аналогичного результата для ОРБД можно достичь, задав *проверку ограничения* (*check constraint*) на типе `gender` (разд. 8.4).

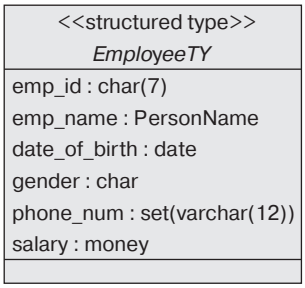


Рис. 8.9. Объявление типов в объектно-реляционной модели базы данных

8.3.1.2. Объектные таблицы

Таблица (объектная таблица) представляет собой множество строк и столбцов. При этом строка может состоять из стольких полей, сколько столбцов содержит таб-

лица объектов. *Строка таблицы* (*table row*) — это объект (экземпляр) *строчного типа* (см. разд. 8.3.1.3 ниже). Каждая строка в таблице объектов — объект, уникальным идентификатором которого служит OID. Соответственно, строка таблицы — это наименьшая единица данных, которую можно поместить в таблицу или удалить из нее.

Для того, чтобы быстро отличить *объектный тип* от *объектной таблицы*, на практике в имени структурированного типа рекомендуется использовать суффикс `TY`. Для постоянного хранения экземпляров типа `EmployeeTY` в ОРБД необходимо создать таблицу `EmployeeTY`. Удобно назвать подобную таблицу по имени типа, но без суффикса `TY`. В соответствии со стандартом SQL:1999 объявление могло бы выглядеть следующим образом

```
create table Employee of EmployeeTY;
```

В стандарте SQL:1999 не определена *инкапсуляция* атрибутов структурированного типа с помощью операций (например, разд. 2.1.2.1.2 и 5.1.4) [21]. Однако, стандарт SQL:1999 предполагает, что ОРБД должна генерировать для каждого атрибута *операции наблюдателя* (*observer* (`get`)) и *мутатора* (*mutator* (`set`)). Они, соответственно, позволяют зачитывать и модифицировать каждый атрибут.

8.3.1.3. Строчные типы

Строчный тип (*row type*) позволяет строить относительно сложные внутренние структуры даже без необходимости использования структурированных типов или коллекций. Строчный тип представляет собой последовательность *полей* (пар <имя поля><тип данных>). На самом деле строчный тип дает возможность поместить одну таблицу внутри другой. Столбец таблицы может содержать строчные значения.

Следующий пример объясняет, каким образом использовать строчные типы для определения таблицы со сложной внутренней структурой. Источником для этого примера послужили некоторые классы приложения *Управление контактами с клиентами* (см. рис. 8.6).

```
create table Contact
      (contact_id  integer,
      contact_name row
          (family_name varchar(30),
          first_name   varchar(20)),
      postal_address row
          (po_box  varchar(10),
          post_code varchar(10),
          address  row
              (street  varchar(30),
              cityvarchar(20),
              state    varchar(20),
              country varchar(25))));
```

С точки зрения программирования баз данных строчные типы дают возможность хранить полные строки таблицы в переменных, передавая их в качестве входных аргументов операций и возвращая их как выходные аргументы операций или возвращаемых значений.

8.3.1.4. Ссылочные типы

Структурный тип можно использовать для определения *ссылочного типа* (*reference type*). Для определения ссылки используется ключевое слово `ref`. Например, `emp ref(EmployeeTY)` представляет собой ссылку в объектной таблице на структурированный тип. В SQL:1999 ссылочные типы *обладают областью действия* — таблица, на которую они ссылаются, известна во время компиляции (т.е. динамическая классификация не поддерживается (см. разд. 2.1.5.2.3)).

Как и следовало ожидать, значением ссылочного типа является OID, и оно уникально в пределах базы данных; указывает на строку в *таблице, которая допускает ссылки на нее* (*referenceable table*). Эта подходящая для ссылок таблица должна быть *типизированной таблицей* (*typed table*), т.е. таблицей с ассоциированным *структурированным типом*. Ссылочные типы можно использовать в ОРБД для реализации *ассоциаций "один к одному"*.

Для реализации *ассоциаций "многие ко многим"* можно использовать *коллекции* (разд. 8.3.1.1) *ссылок* — при их наличии. На рис. 8.10 использован пример, представленный на рис. 8.3, чтобы продемонстрировать, каким образом можно представить в ОРБД ассоциацию "многие ко многим" между объектами `Student` и `CourseOffering`.

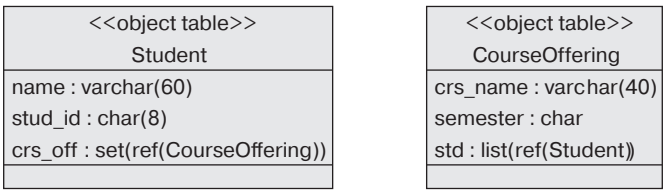


Рис. 8.10. Ассоциация в объектно-реляционной модели базы данных

8.3.1.5. Столбцы, поля и атрибуты

Стандарт SQL:1999 проводит четкое различие между понятием столбцов, полей и атрибутов [83]. Это важное терминологическое уточнение. Ниже приводится смысл различий.

- *Столбец* (*column*) — это структурный компонент *таблицы*.
- *Поле* (*field*) — это структурный компонент *строчного типа*.
- *Атрибут* (*attribute*) — это структурный компонент *структурированного типа*.

Столбец *доступен для пустых значений* (*nullable*) и может принимать значения NULL. Он может быть также *идентифицирующим* столбцом (принимающим значения OID). Типом данных столбца, поля и атрибута может быть *ссылочный тип*.

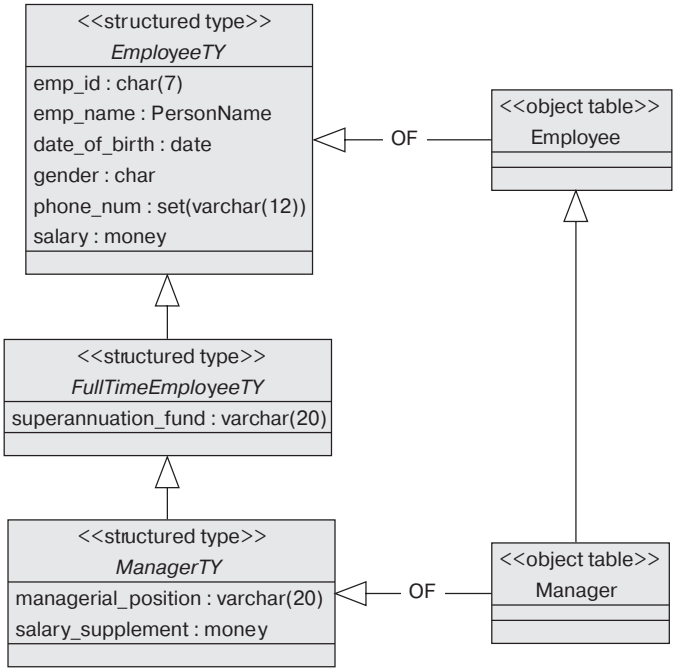
8.3.1.6. Наследование типа OF и UNDER

Стандарт SQL:1999 допускает специализацию существующих типов. В настоящее время допустимо только *одинарное наследование*. Иерархию таблиц можно создать в соответствии с иерархией типов, т.е. "супертаблица" должна быть объявлена как "принадлежащая" (англ. "of") супертипу, а "подтаблица" как "принадлежащая" подтипу. Однако, в иерархии таблицы тип может быть "пропущен", как показано на рис. 8.11.

Программа на языке SQL:1999, приведенная ниже диаграммы на рис. 8.11, относится к типу с наиболее сильной специализацией (`ManagerTY`) и к таблице (`Manager`). Ключевое слово `under` используется в SQL:1999 для установления иерархии типов и иерархии таблиц. Ключевое слово `of` устанавливает структурированный тип таблицы.

Мы также используем ключевое слово OF как имя отношений обобщения между таблицами и их типами на диаграмме.

Мы объявили тип ManagerTY как instantiable — это значит, что мы можем создавать объекты этого типа. Мы также объявили его как специализированный тип final — это значит, что подтипы для него больше недопустимы.




```
create type ManagerTY
       under FullTimeEmployeeTY
       as (managerial position varchar (20),
          salary_supplement money)
       instantiable
       final;
   create table Manager of ManagerTY
       under Employee;
```

Рис. 8.11. Наследование в объектно-реляционной модели базы данных

Заметим, что SQL:1999 не содержит какого-либо явного понятия *интерфейса*. *Инкапсуляция* (закрытый, защищенный, открытый) не определена. Принятое в SQL:1999 наследование является *одинарным наследованием реализации*.

8.3.2. Отображение в ОРБД

Так же, как в случае модели ОБД, отображение классов UML, содержащихся в пакете сущностей, в проект схемы ОРБД может быть осуществлено средствами самого UML. Стереотипов UML и других механизмов расширяемости должно быть достаточно для выражения понятий ОРБД.

Заметим, однако, что на практике отображение осуществляется не по отношению к "абстрактному" стандарту SQL:1999, а к реальному продукту ОРБД. Реальный продукт может не поддерживать некоторых возможностей SQL:1999 и, в то же время, может обладать возможностями, не представленными в SQL:1999.

Поэтому, например, текущая версия Oracle8 (на момент написания книги – редакция 8.1.5.0.0 Oracle8i) не поддерживала ни наследования, ни строчного типа. В качестве типа коллекции она включала только массив и не включала каких-либо других типов коллекций, а также включала понятие вложенной таблицы, которая допускала вложенную таблицу ссылок. Она включала понятие представления объекта для "преобразования" реляционных таблиц в объекты и т.д.

8.3.2.1. Отображение классов-сущностей

Пример 8.5. Управление контактами с клиентами

Обратитесь к спецификациям классов для приложения *Управление контактами с клиентами* в примере 4.6, рис. 4.3 (разд. 4.2.1.2.3). Обратитесь также к примеру 8.1 (разд. 8.2.2.1).

Наша задача заключается в разработке проекта модели ОРБД, семантически соответствующей проекту ОБД, представленному на рис. 8.5.

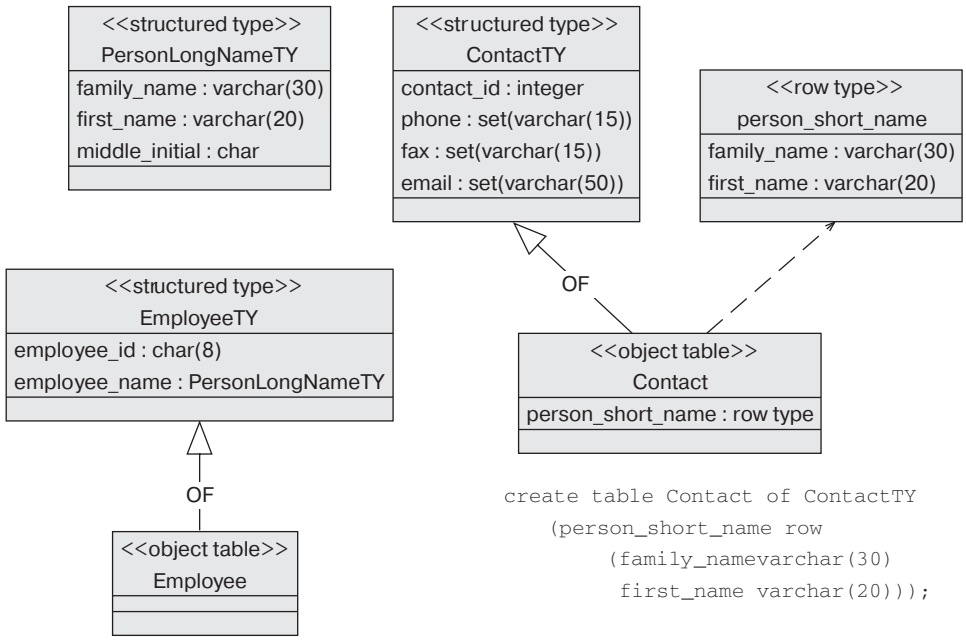


Рис. 8.12. Отображение классов-сущностей в проект схемы объектно-реляционной модели базы данных (Управление контактами с клиентами)

Этот пример требует принять решение по поводу того, каким образом отобразить ОБД-интерфейсы (рис. 8.5). В SQL:1999 интерфейсы не поддерживаются, но нам по-прежнему необходимо некоторым образом моделировать классы `PersonShortName` и `PersonLongName`.

Рис. 8.12 демонстрирует два возможных решения. В случае класса `PersonLongName` для него создается структурированный тип. Затем он используется в качестве типа атрибута `EmployeeTY`. В случае класса `PersonShortName` мы создаем класс (`person_short_name`) для "имитации" строчного типа (строчный тип — это не объектно-ориентированное понятие!). Таблица `Contact` принадлежит типу `ContactTY`, однако, она включает дополнительный столбец `person_short_name`, принадлежащий строчному типу. Отношение зависимости по отношению к классу `person_short_name` используется для отображения структуры строчного типа.

8.3.2.2. Отображение ассоциаций

Пример 8.6. Управление контактами с клиентами

Обратитесь к спецификациям ассоциаций для приложения *Управление контактами с клиентами* в примере 4.8, рис. 4.5 (разд. 4.2.2.3). Обратитесь также к предыдущему примеру отображения для приложения "Управление контактами с клиентами", приведенному в разд. 8.2.2.2 и 8.3.2.1.

Задача в этом примере заключается в разработке проекта модели ОРБД, семантически соответствующей проекту ОБД, представленному на рис. 8.6 (разд. 8.2.2.2). Чтобы упростить решение и в то же время сконцентрироваться на спецификации ассоциации, предположим, что все ОБД-интерфейсы, приведенные на рис. 8.6, преобразованы к строчному типу нашего проекта ОРБД (в решении нет необходимости показывать определения строчных типов).

Наше решение для примера показано на рис. 8.13. Объектные таблицы содержат столбцы, типизированные как строчные типы, и столбцы, типизированные как ссылочные типы. Структурные типы содержат атрибуты, типизированные как исходные типы или как коллекции исходных типов.

8.3.2.3. Отображение агрегаций

Пример 8.7. Запись на университетские курсы

Обратитесь к спецификациям агрегаций для приложения *Запись на университетские курсы* в примере 4.9, рис. 4.6 (разд. 4.2.3.3).

Наша задача в этом примере заключается в отображении UML-модели, показанной на рис. 4.6, в проект схемы ОРБД. Хотя мы разработали проект ОБД для этой проблемы в примере 8.3, мы не желаем оказаться под его влиянием при решении данного примера. Мы предполагаем, что коллекции структурированных типов (не только коллекции элементарных исходных типов) поддерживаются как типы атрибутов в структурированных типах.

При разборе этого примера требуется принять решение по поводу того, каким образом моделировать в терминах ОРБД UML-агрегацию и UML-композицию. Предположение, изложенное в определении примера, дает нам четкие указания. Композицию между классами `Student` и `AcademicRecord` можно моделировать в классе `Student` с помощью атрибута, типизированного как коллекция классов `AcademicRecord`. Это показано на рис. 8.14. Коллекция представляет собой множество `set` (`AcademicRecordTY`).

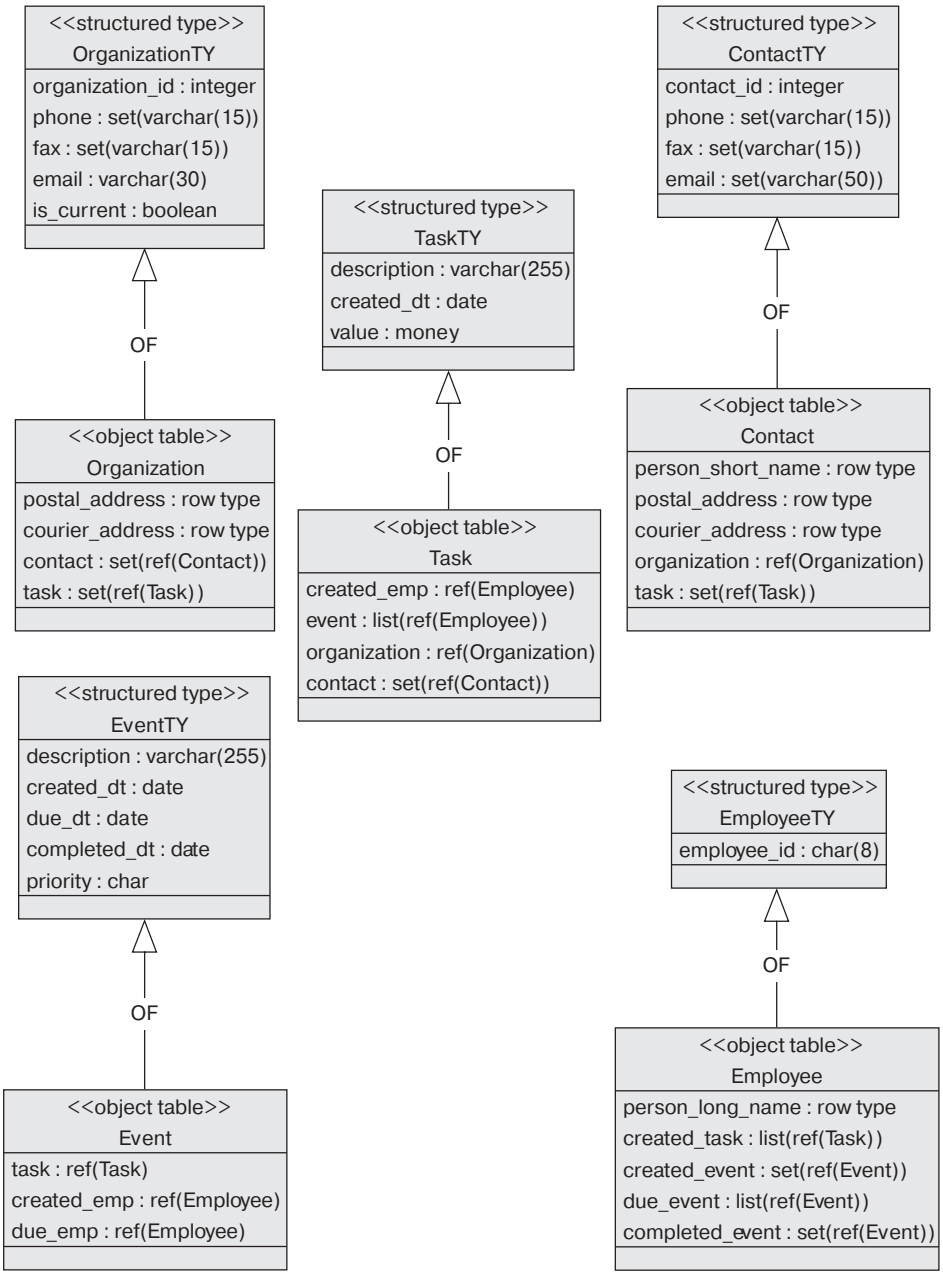


Рис. 8.13. Отображение ассоциаций в проект схемы объектно-реляционной модели базы данных (Управление контактами с клиентами)

UML-агрегацию между классами `Course` и `CourseOffering` можно моделировать как обычную ассоциацию — с помощью ссылок. Для каждой таблицы объектов необходимо задать соответствующий структурированный тип. Это необходимо, помимо

прочего, с точки зрения того требования стандарта языка SQL:1999, что значение типа `ref` должно идентифицировать строку в типизированной таблице (т.е. таблице заданного структурированного типа).

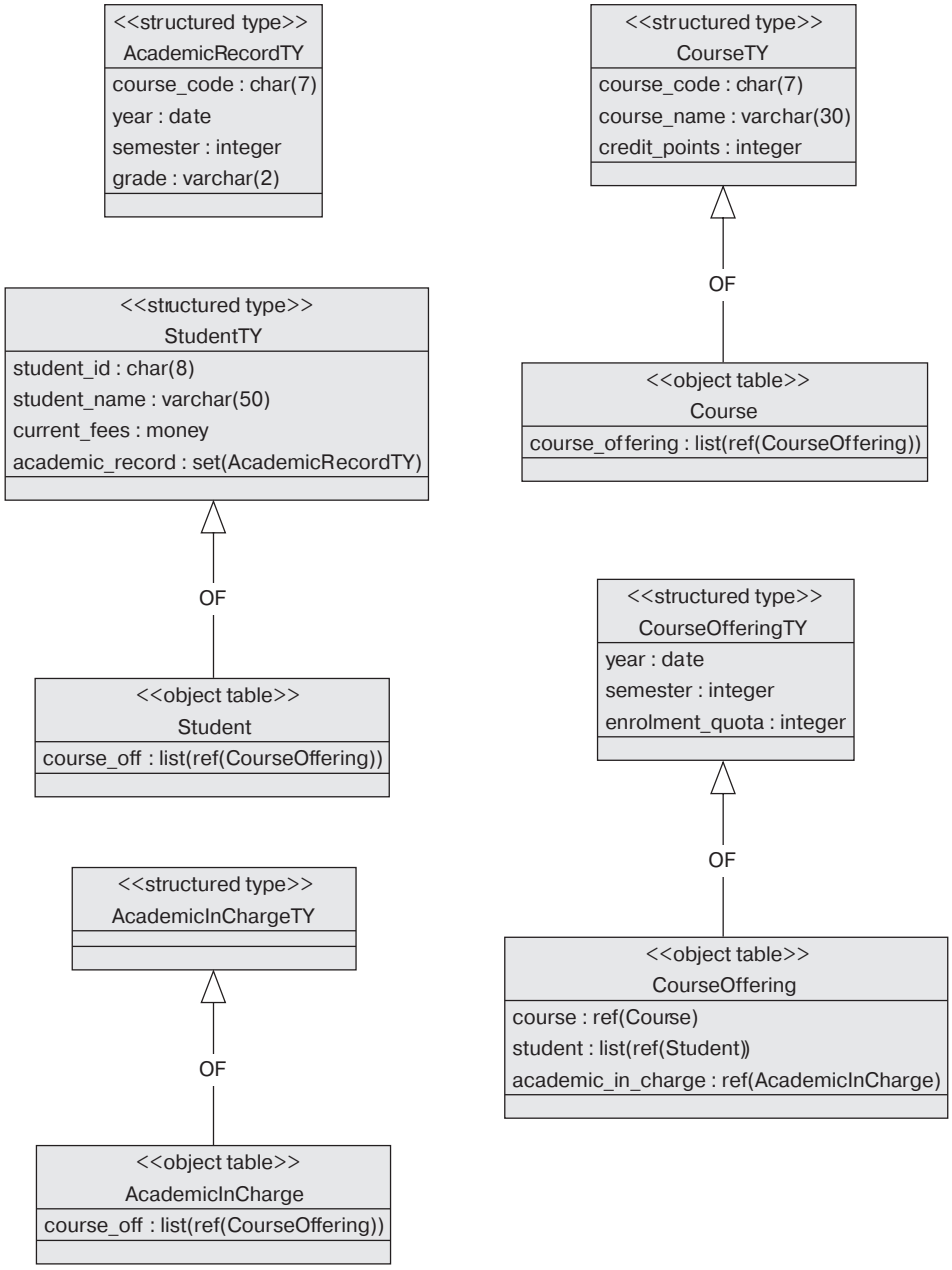


Рис. 8.14. Отображение ассоциаций в проект схемы объектно-реляционной модели базы данных (Управление контактами с клиентами)

8.3.2.4. Отображение обобщений

Пример 8.8. Магазин видеопроката

Обратитесь к спецификациям обобщения для приложения *Магазин видеопроката* в примере 4.10, рис. 4.7 (разд. 4.2.4.3). Обратитесь также к предыдущему примеру отображения обобщений в проект ОБД в разд. 8.2.2.4.

Наша задача заключается в разработке проекта модели ОРБД, семантически соответствующей проекту ОБД, представленному на рис. 8.8. (разд. 8.2.2.4).

Поскольку у нас нет уверенности в том, каким образом модель ОРБД должна поддерживать выводимые и статические атрибуты, мы предполагаем, что они вычисляются процедурно и моделировать их в качестве структур данных ОРБД нет необходимости. Рассмотрение касается двух атрибутов: `is_in_stock` (в запасе) и `number_currently_available` (в наличии).

Основная трудность при решении этого примера имеет только косвенное отношение к отображению обобщения. Основная проблема связана с ограничением стандарта SQL:1999, которое выражается в том, что значение ссылочного типа `ref` "заключено" в пределах области действия. В области действия типа `ref` может находиться только одна таблица, и эта область действия должна быть известна статически — во время компиляции.

Перед этим мы связали класс `MovieTitle` с интерфейсом `VideoMedium` в предположении, что любой объект `VideoMedium` (т.е. `BetaTape`, `VHSTape` или `DVDDisk`) связан с `MovieTitle`. Теперь же, чтобы справиться с проблемой, мы вынуждены создать в `MovieTitle` три ассоциации. Полностью решение приведено на рис. 8.15.

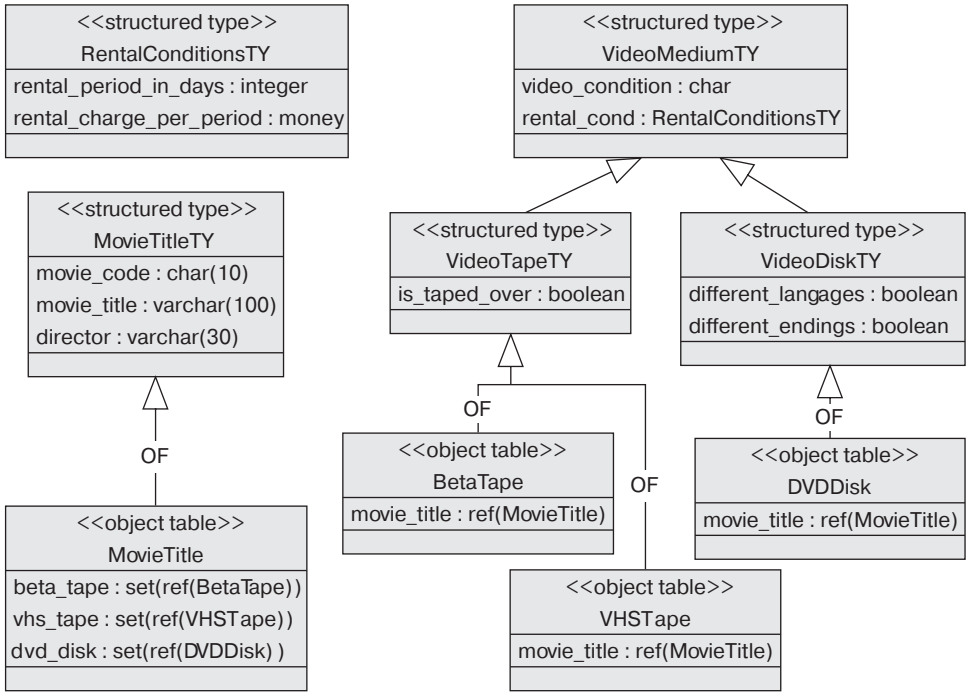


Рис. 8.15. Отображение обобщения в проект схемы объектно-реляционной модели базы данных (Магазин видеопроката)

8.4. Модель реляционной базы данных

В течение последних двадцати лет реляционная модель господствовала на рынке ПО баз данных. Модель реляционной базы данных (РБД) пришла на смену иерархической и сетевой моделям баз данных. Во второй половине 1990-х годов поставщики реляционных систем управления базами данных (СУРБД) постоянно увеличивали свое внимание к модели объектной базы данных, стандартам ODMG и различным продуктам СУОБД.

Как следствие возник целый ряд продуктов СУОБД, которым сегодня прочат ведущую роль в будущем. Поставщики традиционных СУРБД, такие как Oracle, IBM или Informix, предлагают сегодня продукты, которые оказывают наибольшее влияние на этот сегмент рынка. (Компания Informix вместе с соответствующими продуктами была приобретена в 2001 году компанией IBM. *Прим. ред.*). Между тем слабые продукты СУОБД не увеличили своей рыночной доли, более того, произошел сдвиг в самом их назначении — они превратились в API-интерфейсы хранилищ объектов, предназначенные для поддержки интероперабельности клиентских приложений и различных серверных источников данных, в частности, реляционных баз данных.

Хотя будущее более и не принадлежит моделям РБД, инерция развития бизнеса такова, что должно пройти десятилетие, если не больше, прежде чем станет заметным переход больших систем на технологию ОРБД или ОБД. Кроме того, в ближайшем будущем появится огромное количество приложений, разработанных в технологии РБД, просто потому, что предприятия и организации стремятся избегать сложных и трудных в освоении объектных решений.

Ныне действующий (и последний из разработанных) стандарт для модели РБД известен как стандарт SQL92. Он был принят ANSI и ISO в 1992 году. Подавляющее большинство продуктов СУРБД на рынке (Oracle, DB2, Sybase, Informix, Microsoft SQL Server и др.) соответствуют этому стандарту, хотя каждый в своей собственной манере. Фактически, некоторые концепции РБД (например, *триггер* (*trigger*)) уже были широко реализованы в продуктах СУРБД, прежде чем получили официальное признание в рамках стандарта SQL:1999.

8.4.1. Элементарные типы модели РБД

Элементарные типы модели РБД, конечно же, элементарны по определению. Простота модели РБД, которая вытекает из математического понятия *множества*, составляет как сильную, так и слабую стороны этой модели. Математический фундамент, на котором зиждется реляционная модель, определил ее *декларативный* характер (в противоположность *процедурному*). Пользователь просто объявляет, *что* ему требуется получить от базы данных, а не инструктирует систему о том, *как* именно отыскать нужную информацию (СУРБД знает, как искать данные в своей базе данных).

Однако то, что кажется простым поначалу, становится довольно сложным вместе с ростом сложности решаемой проблемы. Для сложных проблем нет простых решений. Чтобы решить сложную проблему, требуются тонкие механизмы. Чтобы приступить к моделированию, потребуются развитые элементарные типы данных.

Наверное лучший путь охарактеризовать модель РБД — сформулировать, какие из элементарных типов она не поддерживает. Из основных элементарных типов моделирования, применяемых в моделях ОБД и ОРБД, реляционная модель не поддерживает следующих.

- Объектные типы и связанные с ними понятия (такие, как наследование или методы).
- Структурированные типы.
- Коллекции.
- Ссылки.

Основным элементарным типом модели РБД является *реляционная таблица* (*relational table*), которая состоит из столбцов. *Столбцы* таблицы могут принимать только атомические значения — структурированные значения или значения коллекций не допускаются.

Модель РБД ведет себя весьма жестко в отношении любых видимых пользователю *навигационных связей* между таблицами — они исключаются. Отношения между таблицами поддерживаются с помощью сравнения значений в столбцах. Постоянные связи отсутствуют. Полезное свойство ОРБД по поддержанию предопределенных отношений между таблицами называется *ссылочной целостностью* (*referential integrity*).

На рис. 8.16 показаны элементарные типы модели РБД и зависимости между ними. Все понятия выражены с помощью существительных в единственном числе, однако, некоторые зависимости применимы, более чем к одному экземпляру понятия. Например, ссылочная целостность определяется на одной или более таблицах. Большая часть понятий, приведенных на рис. 8.16, рассматривается в последующих разделах этой главы. Другие понятия более подробно рассмотрены в главе 9.

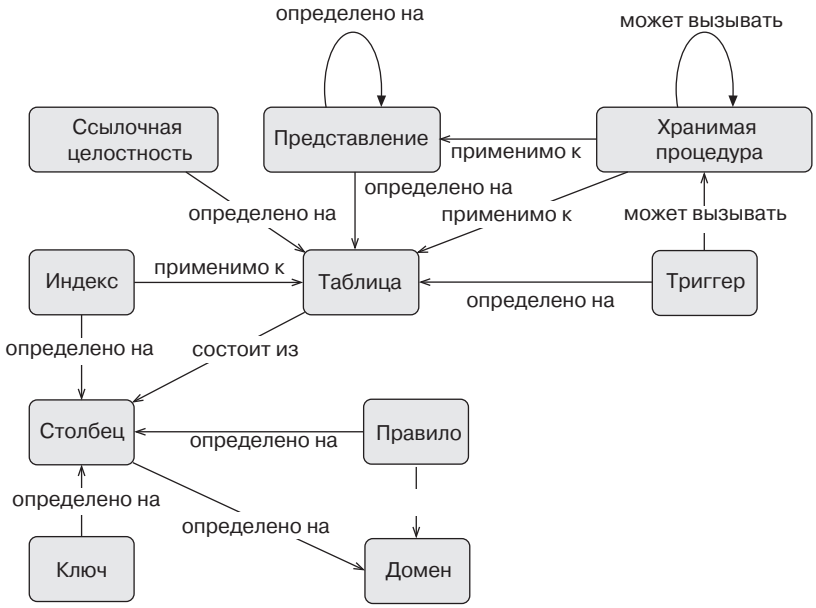


Рис. 8.16. Зависимости между элементарными типами модели РБД

8.4.1.1. Столбцы, домены и правила

Реляционные базы данных определяют данные в столбцах и строках. Значение данных, хранимых на пересечении любого столбца и строки, должно быть простым (неделимым) и однократным (не повторяющимся) значением. Мы говорим, что *столбцы* образуют *атомические домены* (*atomic domains*) (типы данных).

Домен (*domain*) определяет допустимое множество значений, которое может принимать столбец. Домен может быть безымянным (например, `gender char(1)`) или может быть поименован (например, `gender Gender`). В последнем случае домен `Gender` был определен ранее и используется в определении столбца. Ниже приводится возможный синтаксис определения домена.

```
create domain Gender char(1);
```

Именованный домен можно использовать для определения многих столбцов в различных таблицах. Это способствует достижению непротиворечивости между этими определениями. Изменения, вносимые в определение домена, автоматически отражаются на определении столбца. Хотя подобная возможность и выглядит на первый взгляд довольно привлекательно, ее использование тут же становится помехой, как только база данных заполняется, т.е. загружается данными.

Со столбцами и доменами могут быть связаны некоторые бизнес-правила, накладывающие на них ограничения. Бизнес-правила могут определять следующие характеристики данных.

- *Значение, используемое по умолчанию* (например, если для города (`city`) не задано никакого значения, предполагается значение "Сидней").
- *Диапазон значений* (например, допустимое значение для возраста (`age`) лежит в пределах от 18 до 80).
- *Список значений* (например, допустимыми цветами (`color`) могут быть "зеленый", "желтый" или "красный").
- *Регистр, используемый для представления значения* (например, значение должно состоять из символов верхнего или нижнего регистра клавиатуры).
- *Формат значения* (например, значение должно начинаться с буквы "`K`").

С помощью средств задания *правил* можно определить только простые правила, относящиеся к единственному столбцу или домену. Более сложные правила, охватывающие таблицы, можно определить с помощью *правил ограничения целостности*. Основным механизмом определения бизнес-правил являются *триггеры*.

8.4.1.2. Реляционные таблицы

Реляционная таблица (*relational table*) определяется фиксированным множеством своих столбцов. Типы данных, которые хранятся в столбцах, относятся к встроенным или определяемым пользователем типам (т.е. доменам). Таблица может содержать произвольное количество *строк* (или записей). Поскольку таблица есть не что иное, как математическое множество, повторяющиеся строки в таблице отсутствуют.

Для некоторых столбцов допускается, что значение столбца в определенной строке может принимать значение `NULL`. Значение `NULL` означает одну из двух возможностей: "не определенное в данный момент значение" или "неприменимое значение" (разд. 8.2.1.1).

Одним из следствий требования к модели РБД, которое состоит в недопустимости дублирования строк, является наличие у каждой таблицы *первичного ключа* (*primary key*). *Ключ* — это *минимальное* множество столбцов (возможно один) таких, что значения в этих столбцах *единственным* способом идентифицируют одну строку в таблице. Таблица может содержать много подобных ключей. Один из этих ключей произвольным образом выбирается как наиболее важный — это *первичный ключ* (*primary key*). Другие ключи называются *потенциальными* (*candidate*) или *альтернативным* (*alternative*) *ключами*.

На практике таблица СУРБД не обязана иметь ключ. Это значит, что таблица (без уникального ключа) может содержать идентичные строки — чудное бесполезное свойство реляционной базы данных, когда две строки, содержащие одинаковые значения во всех своих столбцах, никак нельзя отличить. В системах ОБД и ОРБД подобное различение обеспечивается за счет наличия OID (два объекта могут быть равными, но не идентичными, например, как две копии одной книги).

Хотя существует возможность ввести в UML стереотипы для моделирования реляционных баз данных, более удобно использовать специально предназначенные для этого диаграммные методы, позволяющие осуществить логическое моделирование реляционных баз данных. На рис. 8.17 показана одна из подобных систем нотации. В качестве целевой базы данных здесь использовалась база данных DB2 копании IBM.

Employee			
emp_id	CHAR(7)	<pk>	not null
family_name	VARCHAR(30)	<ak>	not null
first_name	VARCHAR(20)		not null
date_of_birth	DATE	<ak>	not null
gender	Gender		not null
phone_num1	VARCHAR(12)		null
phone_num2	VARCHAR(12)		null
salary	DEC(8,2)		null

Рис. 8.17. Определение таблиц реляционной базы данных

Таблица `Employee` состоит из восьми столбцов. Последние три столбца допускают в качестве значений использование значения `NULL`. Столбец `emp_id` представляет собой первичный ключ. Столбцы `{family_name, date_of_birth}` определяют потенциальные (альтернативные) ключи. Столбец `gender` определен на домене `Gender`.

Поскольку на РБД накладывается ограничение, которое заключается в том, что столбцы могут принимать только атомические однократные значения, моделирование имени и телефонного номера работника вызывает у нас определенные затруднения. В предыдущем случае мы использовали два столбца : `family_name` и `first_name`. Столбцы не сгруппированы и никак не связаны в модели. В последнем случае мы предпочли решение с двумя столбцами (`phone_num1`, `phone_num2`), допускающими наличие максимум двух телефонных номеров у каждого работника.

После того, как таблица определена с помощью CASE-средств, можно автоматически сгенерировать программный код для создания таблицы, как показано ниже. Сгенерированный текст программы включает определение домена `Gender` и определение бизнес-правила, определенного на этом домене.

```
============================================================
-- Domain: Gender
============================================================
create distinct type Gender as CHAR(1) %WITHCOMPAR%;
============================================================
-- Table: Employee
============================================================
create table Employee (
   emp_id          CHAR(7)               not null,
   family_name     VARCHAR(30)           not null,
   first_name      VARCHAR(20)           not null,
   date_of_birth   DATE                  not null,
   gender          Gender                 not null
  constraint  C_gender  check (gender in ('F','M','f','m')),
   phone_num1      VARCHAR(12),
   phone_num2      VARCHAR(12),
   salary          DEC(8,2),
primary key (emp_id),
unique (date_of_birth, family_name)
);
```

8.4.1.3. Ссылочная целостность

Модель РБД поддерживает отношения между таблицами посредством ограничений ссылочной целостности. Отношения не привязаны к связям отдельных строк между собой. Вместо этого РБД "отыскивает" связи между строками всякий раз, когда пользователь просит систему отыскать соответствующие отношения. Это "нахождение" осуществляется с помощью сравнения значений первичных ключей в одной таблице со значениями внешних ключей в той же самой или другой таблице.

Внешний ключ (*foreign key*) определяется как множество столбцов в одной или более таблицах, значения которых либо равны `NULL`, либо должны совпадать со значениями первичного ключа в той же самой или другой таблице. Это соответствие между первичным и внешним ключом называется *ссылочной целостностью*. Первичный и внешний ключи, участвующие в определении ссылочной целостности, должны быть определены на одном и том же домене, однако, они не обязательно должны иметь одинаковые имена.

На рис. 8.18 показано графическое представление ссылочной целостности. В результате изображения отношения между таблицами `Employee` и `Department` к таблице `Employee` был добавлен внешний ключ `dept_id`. Для каждой строки таблицы `Employee` значение внешнего ключа должно быть либо равно `NULL`, либо совпадать с одним из значений `dept_id` в таблице `Department` (в противном случае сотрудник может работать в несуществующем подразделении).

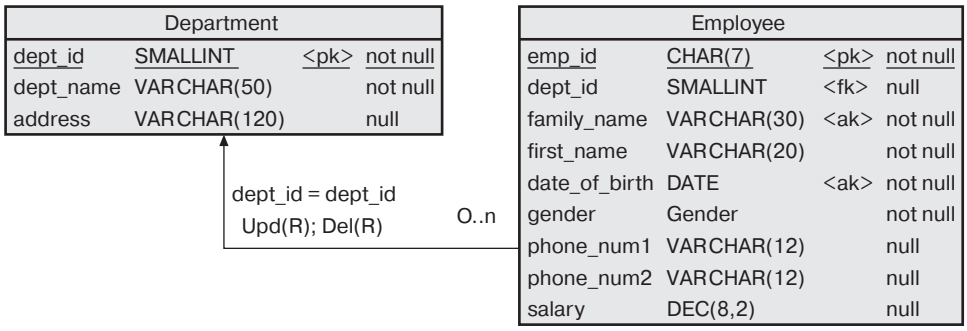


Рис. 8.18. Ссылочная целостность

Дополнительное описание, сопровождающее линию отношения, *декларативно* определяет поведение, связанное со ссылочной целостностью. Существует четыре возможных декларативных ограничения ссылочной целостности, связанных с операциями *delete* и *update*. А именно, вопрос состоит в том, что делать со строками таблицы `Employee` при удалении или обновлении строк таблицы `Department` (т.е. обновляется столбец `dept_id`). На этот вопрос существует четыре варианта ответа.

1. `Upd(R); Del(R)` — *ограничить* операции update или delete (т.е. не давать возможности операции выполниться, если только еще в таблице `Employee` существуют строки, связанные с данным подразделением — `Department`).
2. `Upd(C); Del(C)` — *каскадировать* операцию (т.е. удалить все строки таблицы `Employee`, связанные с данным подразделением — `Department`).
3. `Upd(N); Del(N)` — *установить* значение ключа равным *null* (т.е. обновить или удалить строку таблицы `Department` и установить ключ `dept_id` для связанной строки таблицы `Employee` в `NULL`).
4. `Upd(D); Del(D)` — *установить* значение ключа равным значению, *принятому по умолчанию* (т.е. обновить или удалить строку таблицы `Department` и установить ключ `dept_id` для связанной строки таблицы `Employee` равным значению по умолчанию).

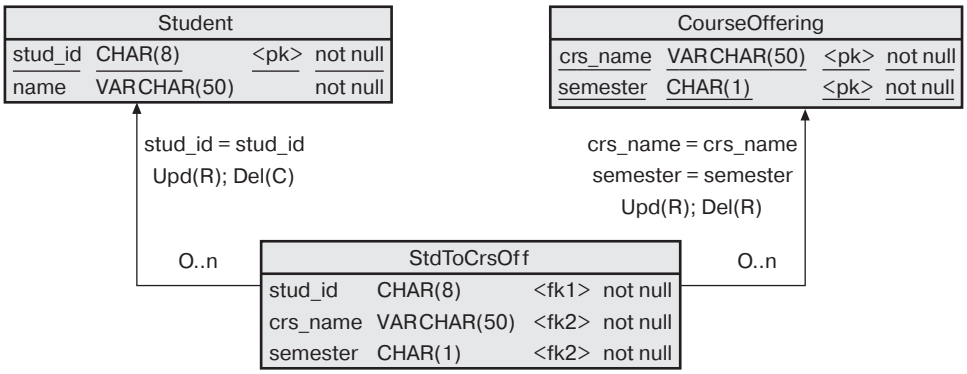


Рис. 8.19. Ссылочная целостность для отношений "многие ко многим"

Моделирование ссылочной целостности усложняется, когда отношение между таблицами принимает характер "многие ко многим", как в случае отношения между классами `Student` и `CourseOffering` (рис. 8.3 в разд. 8.2.1.2). Чтобы справиться с проблемой, вызванной ограничением на РБД, которое выражается в недопустимости принятия столбцом множественных значений, нам требуется ввести *перекрестную таблицу* (*intersection table*) наподобие таблицы `StdToCrsOff`, показанной на рис. 8.19. Единственным назначением этой таблицы является моделирование отношений "многие ко многим" и задание декларативных ограничений ссылочной целостности.

8.4.1.4. Триггеры

Правила и *декларативные ограничения ссылочной целостности* позволяют определять простые бизнес-правила на базе данных. Их недостаточно для определения более сложных правил или каких-либо исключений из правил. Применительно к РБД решением данной проблемы (в соответствии со стандартом SQL:1999) является *триггер* (*trigger*).

Триггер — это небольшая программа, написанная в расширенном языке SQL, которая выполняется автоматически (запускается) в результате операции модификации таблицы, над которой определен триггер. В качестве операции модификации может выступать любой оператор модификации языка SQL: `insert`, `update` или `delete`.

Триггер можно использовать для реализации *бизнес-правил*, которые выходят за рамки возможностей оператора `rule` правил SQL (разд. 8.4.1.1). Например, бизнес-правило, которое запрещает вносить изменения в таблицу `Employee` во время выходных, можно запрограммировать в виде триггера. Любая попытка применить к таблице оператор SQL `insert`, `update` или `delete` в выходные дни приводит к срабатыванию триггера, и база данных отказывается выполнять операцию.

Триггер можно также использовать для приведения в действие более сложных ограничений ссылочной целостности. Например, наше бизнес-правило может гласить, что при удалении строки таблицы `Department` (подразделение) строка таблицы `Employee`, которая представляет информацию о работнике, который является менеджером этого подразделения, также должна быть удалена. При этом для всех остальных работников (не менеджеров) значения `dept_id` должны быть установлены в `NULL`. Подобное бизнес-правило невозможно ввести декларативно. Чтобы привести его в исполнение, необходимо задействовать процедурный триггер.

Если для введения ссылочной целостности над базой данных используются триггеры, декларативные ограничения ссылочной целостности обычно ликвидируются. Смешение процедурных и декларативных ограничений — плохая идея, поскольку значительно затрудняет их взаимодействие. Как следствие, сегодня преобладает практика программирования ограничений ссылочной целостности только на основе триггеров. Проблема на самом деле не так страшна, как может показаться на первый взгляд, поскольку мощные CASE-средства способны генерировать большую часть программного кода автоматически.

Например, ниже приведена программа для триггера, сгенерированная с помощью CASE-средств для СУРБД Sybase. Триггер реализует декларативное ограничение `Del(r)` — т.е. он не позволяет удалить строку таблицы `Department` до тех пор, пока существует хоть одна строка таблицы `Employee`, связанная с ней.

```
create trigger keepdpt
        on Department
        for delete
        as
        if @@rowcount = 0
                return /* avoid firing trigger if no rows affected
*/
        if exists
                (select * from Employee, deleted
                where Employee.dept_id =
                    deleted.dept_id)
        begin
                print 'Test for RESTRICT DELETE failed. No deletion'
                rollback transaction
                return
        end
        return
go
```

Оператор проверяет, осуществляется ли операция SQL `delete` (которая запускает триггер) по удалению вообще какой-либо строки. Если это не так, триггер прекращает свои действия — никакого вреда нанесено быть не может. Если строка таблицы `Department` может быть удалена, СУБД Sybase запоминает эти (подлежащие удалению) строки в промежуточной таблице с именем `deleted`. Затем триггер выполняет операцию *логического соединения* по столбцу `dept_id` на таблицах `Employee` и `deleted`, чтобы определить, существует ли хоть один сотрудник, работающий на подразделение, информация о котором подлежит удалению. Если это так, то триггер отклоняет действие по удалению, отображает сообщение и осуществляет откат транзакции. В противном случае строку таблицы `Department` разрешается удалить.

8.4.1.5. Реляционные представления

Реляционное представление (*relational view*) (иногда называемое сленговым термином *взгляд. Прим. ред.*) — хранимый и поименованный SQL-запрос. Поскольку результатом любого SQL-запроса является переходная таблица, представление может использоваться вместо таблицы в другом SQL-операторе, выводиться на основе одной или более таблиц и/или одного или более других представлений (рис. 8.16).

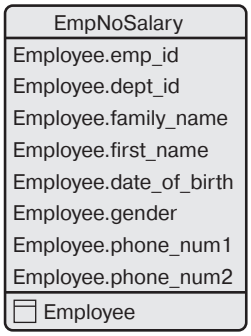


Рис. 8.20. Реляционное представление

На рис. 8.20 показан графический вид представления EmpNoSalary — представление отображает всю информацию из таблицы Employee за исключением столбца salary. Оператор создания представления create view демонстрирует, что представление фактически является именованным запросом, который выполняется всякий раз при выдаче SQL-запроса или оператора обновления применительно к представлению.

```
=====================================================
-- View: EmpNoSalary  --
=====================================================
create view EmpNoSalary () as
    select Employee.emp_id, Employee.dept_id,
           Employee.family_name, Employee.first_name,
           Employee.date_of_birth, Employee.gender,
           Employee.phone_num1, Employee.phone_num2
    from Employee;
```

Теоретически представление являет собой очень мощный механизм, который находит множество применений; использоваться для поддержки *безопасности базы данных* за счет ограничения круга пользователей, имеющих возможность просмотра таблицы; представить данные пользователю в различных разрезах. Оно может *изолировать приложение от изменения определений таблицы*, если изменяемое определение не составляет части представления. Оно позволяет упростить выражения для *сложных запросов* — запрос может быть встроен в манере "разделяй и властвуй" за счет использования нескольких уровней представлений.

На практике использование концепции представления в модели РБД жестко ограничено из-за невозможности обновления представления. Обновление представления эквивалентно возможности отправки операции модификации (SQL-операторы insert, update или delete) представлению и изменению в результате базовых таблиц базы данных. Поддержка обновления представления в рамках стандарта SQL92 весьма ограничена — она практически отсутствует.

8.4.1.6. Нормальные формы

Можно с уверенностью утверждать, что одной из наиболее важных, но в то же время недооцениваемых концепций проектирования РБД является *нормализация* (*normalization*). Реляционная таблица должна быть представлена в *нормальной форме* (НФ). Существует шесть типов нормальных форм.

- 1-я НФ
- 2-я НФ
- 3-я НФ
- НФБК (Нормальная форма Бойса-Кодда (Boyce-Codd))
- 4-я НФ
- 5-я НФ

Таблица, представленная в младшей нормальной форме, также представлена и в более старшей НФ. Она должна быть приведена, по меньшей, мере в 1-й НФ. Таблица, в которой отсутствуют структурированные или многозначные столбцы, выражена в 1-й НФ (и это составляет фундаментальное требование модели РБД).

Таблица в младшей НФ может проявлять так называемые аномалии обновления. *Аномалия обновления* — это нежелательный побочный эффект, возникающий в результате выполнения операции модификации (`insert`, `update`, `delete`) над таблицей. Например, если одна и та же информация многократно повторяется в одном и том же столбце таблицы, обновление этой информации должно осуществляться в каждом месте ее хранения, иначе база данных останется в некорректном состоянии. Можно показать, что аномалии обновления постепенно устраняются по мере перехода к старшим НФ.

Итак, каким образом можно нормализовать таблицу к старшей НФ? Таблицу можно привести к старшей НФ, разделив ее по вертикали вдоль столбцов на две или более таблицы меньшего размера. Эти меньшие таблицы, скорее всего, окажутся в старшей НФ и заменят исходную таблицу в проекте модели РБД. Исходная таблица, однако, может всегда быть восстановлена с помощью объединения меньших таблиц с использованием SQL-оператора `join`.

Рамки этой книги не позволяют нам рассмотреть теорию нормализации сколько-нибудь подробно. За подробностями читатель может обратиться к книгам Дейта (Date) [17], Мацяшека (Maciaszek) [51], Зильберкатца (Silberschatz) [79], Рамакришнана (Ramakrishnan) и Герке (Gehrke) [67]. Единственный момент, который нам хотелось бы подчеркнуть, это то, что надлежащее проектирование РБД проводится на надлежащем уровне нормализации.

Что мы подразумеваем под надлежащим проектом в контексте нормализации? *Надлежащий проект* означает, что у нас есть понимание того, каков будет характер использования РБД с точки зрения смеси операций обновления и получения информации. Если база данных отличается динамизмом, т.е. подвергается частым операциям обновления, то мы, естественно, должны стремиться создавать небольшие по размерам таблицы для лучшей локализации и осуществления этих действий по обновлению. Таблицы создаются в старшей НФ, и аномалии обновления будут устранены или уменьшены.

С другой стороны, если база данных отличается относительно статичным характером, т.е. над ней часто выполняются операции поиска информации, а ее содержимое обновляется время от времени, приобретает смысл *денормализованный проект* базы данных. Это является следствием того, что поиск в одной таблице большого размера оказывается намного более эффективным, чем аналогичный поиск в нескольких таблицах, которые требуется объединить перед началом поиска.

8.4.2. Отображение в РБД

При отображении модели классов UML в проект схемы РБД необходимо учитывать ограничения на модель РБД. Речь идет о том, чтобы поступиться некоторой *декларативной семантикой* диаграмм классов в счет *процедурных решений* проекта логической схемы базы данных. Другими словами, может оказаться невозможным выразить некоторую встроенную декларативную семантику классов в реляционной схеме. Подобная семантика должна получить процедурное разрешение в программах базы данных, т.е. в хранимых процедурах (разд. 9.1.2.1).

Проблемы отображения в модель РБД широко изучались в контексте ER-моделирования и расширенного ER-моделирования (например, [51], [22]). В ходе этих исследований использовались те же принципы и были выявлены все основные вопросы. Так же как в случае моделей ОРБД и ОБД, отображение должно не просто соответствовать некоторым стандартам (в случае РБД это SQL92), но также учитывать особенности целевой СУРБД.

8.4.2.1. Отображение классов-сущностей

Отображение классов-сущностей в реляционные таблицы должно удовлетворять 1-й НФ таблиц. Столбцы должны быть атомическими типами. Однако, хотя об этом и неудобно напоминать, как мы уже отмечали в разд. 8.2.2.1, атрибуты классов (в противоположность атрибутам отношений) в модели анализа UML уже относятся к атомическому типу. Это упрощает отображение.

Пример 8.9. Управление контактами с клиентами

Обратитесь к спецификациям классов для приложения *Управление контактами с клиентами* в примере 4.6, рис. 4.3 (разд. 4.2.1.2.3). Рассмотрите также обсуждение, касающееся не атомических атрибутов в примере 8.1.

Отобразите классы `Contact` и `Employee` в проект РБД таким образом, чтобы продемонстрировать несколько альтернативных стратегий отображения.

Наше решение для данного примера приведено на рис. 8.21. В решении предполагается использование СУРБД Oracle. Мы моделируем `contact_name` как атомический тип данных в таблице `Contact`. Для каждого контактного лица (`Contact`) предполагается наличие только одного факса и адреса электронной почты. Однако, мы допускаем произвольное количество телефонных номеров. Этой цели служит таблица `ContactPhone`.

В таблице `Employee` мы держим три отдельных атрибута для имен работника: `family_name`, `first_name` и `middle_initial`. Однако, в базе данных не содержится никаких сведений об имени работника `employee_name` как некоторой комбинации этих трех атрибутов.

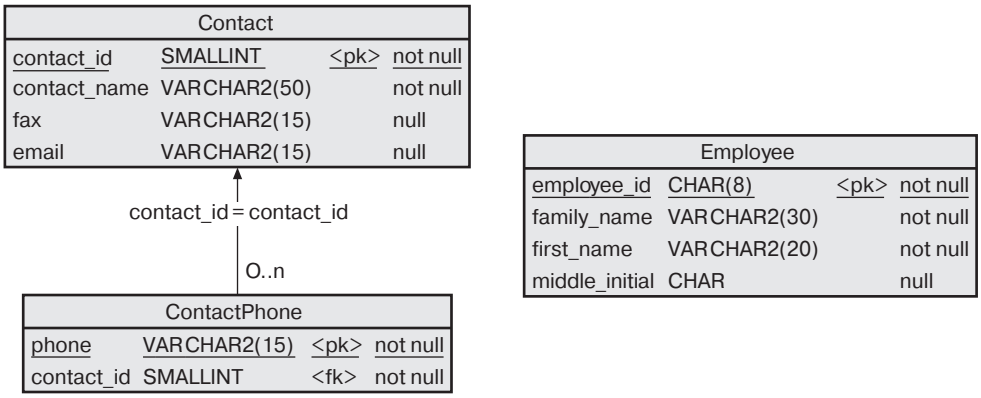


Рис. 8.21. Отображение классов-сущностей в проект РБД (Управление контактами с клиентами)

8.4.2.2. Отображение ассоциаций

Отображение ассоциаций в РБД связано с использованием ограничений *ссылочной целостности* между таблицами. Любую ассоциацию кратности "один к одному" или "один ко многим" можно непосредственно выразить с помощью помещения *внешнего ключа* в одну из таблиц для установления соответствия первичному ключу другой таблицы.

В случае ассоциации "*один к одному*" внешний ключ может быть добавлен к любой из таблиц (решение здесь принимается на основе использования шаблонов ассоциа-

ций), а также может быть желательно составить комбинацию из двух классов-сущностей в одной таблице (в зависимости от требуемого уровня нормализации).

Для *рекурсивных* ассоциаций "один к одному" и "один ко многим" внешний ключ и первичный ключ помещаются в одной таблице. Каждая ассоциация "*многие ко многим*" (независимо от того, рекурсивна она или нет) требует введения перекрестной таблицы, как показано на рис. 8.19.

Пример 8.10. Управление контактами с клиентами

Обратитесь к спецификациям ассоциаций для приложения *Управление контактами с клиентами* в примере 4.8, рис. 4.5 (разд. 4.2.2.3).

Отобразите диаграмму, представленную на рис. 4.5, в модель РБД.

Этот пример оказывается довольно простым из-за отсутствия в спецификации ассоциаций UML-ассоциаций типа "многие ко многим". Диаграмма РБД (для СУРБД DB2) показана на рис. 8.22. В соответствии с принципами построения РБД мы создали несколько новых столбцов в качестве первичных ключей; решили сохранить модель, приведенную на рис. 8.21, как частичное решение этого примера. Ради экономии места отказались от отображения способности столбцов принимать значение `NULL` и индикаторов ключей.

Ограничения ссылочной целостности между таблицами `PostalAddress` и `CourierAddress` с одной стороны, и таблицами `Organization` и `Contact` с другой стороны моделируются с помощью внешних ключей в таблице адресов. Это отчасти произвольное решение, и ограничения можно было бы моделировать в противоположном направлении (т.е. посредством введения внешних ключей в таблицы `Organization` и `Contact`).

8.4.2.3. Отображение агрегаций

Модель РБД не различает отношений ассоциации и агрегации за исключением случаев их процедурной реализации с помощью триггеров или хранимых процедур. Основные принципы отображения ассоциаций (разд. 8.4.2.2) применимы и к отображению агрегаций. Только в том случае, когда ассоциация может быть преобразована в несколько результирующих реляционных решений, семантика агрегации (как специфической формы ассоциации) оказывает влияние на решение.

В случае строгой формы агрегации (т.е. композиции) следует попытаться создать комбинацию подмножества и супермножества класса-сущности в одной таблице. Это возможно в случае агрегации "один к одному". Для агрегаций типа "один ко многим" класс подмножества (в сильной и слабой форме агрегации) должен моделироваться в виде отдельной таблицы (с внешним ключом, связывающим ее с ее таблицей-владелицей).

Пример 8.11. Запись на университетские курсы

Обратитесь к спецификациям агрегаций для приложения *Запись на университетские курсы* в примере 4.9, рис. 4.6 (разд. 4.2.3.3).

Отобразите диаграмму, представленную на рис. 4.6, в модель РБД.

Этот пример включает в отношения агрегации — композиция от класса `Student` к классу `AcademicRecord` и слабая агрегация от класса `Course` к классу `Course Offering`. Обе агрегации относятся к типу "один ко многим" и требуют отдельных таблиц "подмножества".

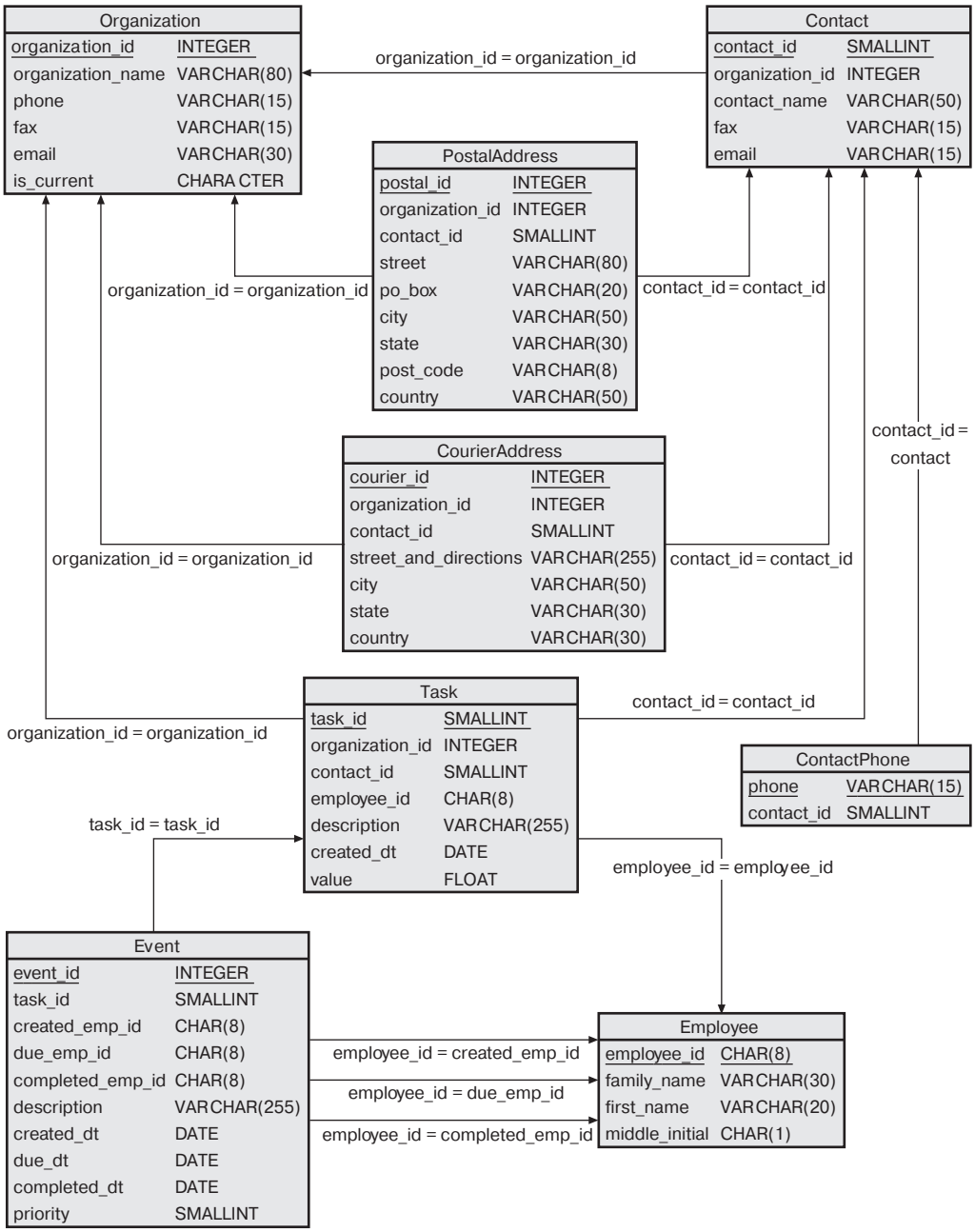


Рис. 8.22. Отображение ассоциации в проект РБД (Управление контактами с клиентами)

Для UML-модели, представленной на рис. 4.6, мы предполагаем (что довольно естественно) опосредованную навигационную связь от класса `AcademicRecord` к кассу `Course`. Что касается проекта РБД, нам может потребоваться установить непосредст-

венную ссылочную целостность между таблицами AcademicRecord и Course. Кроме того, таблица AcademicRecord обладает атрибутом course_code как частью ее первичного ключа. Аналогичный атрибут можно ввести во внешний ключ в таблице Course. Это показано на рис. 8.23 (для СУРБД Informix).

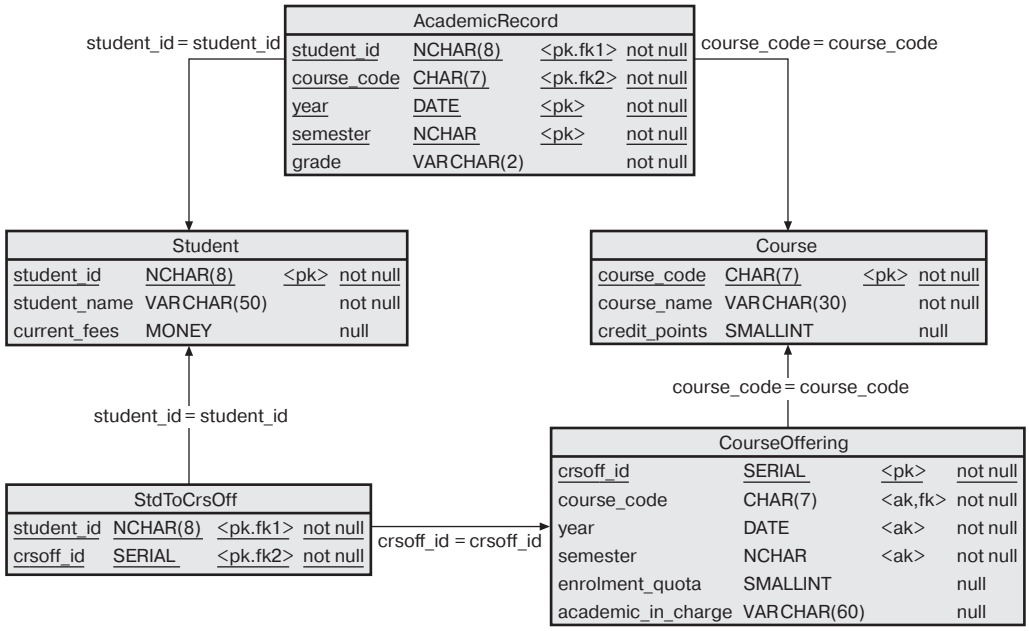


Рис. 8.23. Отображение ассоциации в проект РБД (Запись на университетские курсы)

Ассоциация "многие ко многим" между классами Student и CourseOffering ведет нас к другому замечательному наблюдению, хотя и не связанному с отображением агрегации. Результатом ассоциации является перекрестная таблица StdToCrsOff с первичным ключом, который образуется в результате конкатенации первичных ключей двух главных таблиц.

Первичный ключ для таблицы CourseOffering может выглядеть как {course_code, year, semester}. Однако, подобный ключ имеет результатом громоздкий первичный ключ для таблицы StdToCrsOff. Поэтому мы останавливаем свой выбор на первичном ключе в таблице CourseOffering, сгенерированном системой. Он называется crsoff и обладает типом SERIAL (в СУРБД Informix сгенерированные уникальные идентификаторы относятся к типу SERIAL; в других СУРБД аналогичный тип может называться иначе — например, в Sybase он называется IDENTITY, в Microsoft SQL Server — UNIQUEIDENTIFIER и в Oracle — SE_QUENCE).

8.4.2.4. Отображение обобщений

Отображение отношений обобщения можно осуществить разными способами, однако, принципы отображения менее сложны, чем можно было бы ожидать. Следует, однако, помнить, что выражение обобщения через структуры данных РБД игнорирует вопросы, которые составляют "соль" обобщения — наследование, полиморфизм, повторное использование программного кода и т.д.

Для иллюстрации стратегий отображения агрегации рассмотрим пример, показанный на рис. 8.24. Для преобразования иерархии обобщения в проект модели РБД существует четыре стратегии (хотя для некоторых из этих стратегий возможны дополнительные варианты).

1. Отобразить каждый класс в таблицу.
2. Отобразить полную иерархию класса в одну таблицу "суперкласса".
3. Отобразить каждый конкретный класс в таблицу.
4. Отобразить каждый отдельный класс в таблицу.

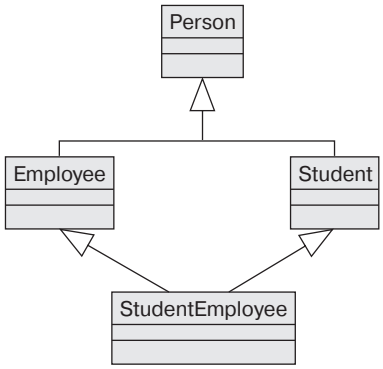


Рис. 8.24. Иерархия обобщения, иллюстрирующая отображение в модель РБД

Первая стратегия проиллюстрирована на рис. 8.25. Каждая таблица имеет свой первичный ключ. Представленное решение ничего не говорит о том, "наследует" ли таблица "подкласса" некоторые из ее столбцов от таблицы "суперкласса". Например, хранится ли `person_name` в `Person` и "наследуется" ли таблицами `Employee`, `Student` и `StudentEmployee`? "Наследование" означает в действительности операцию соединения, и плата за производительность соединения может заставить нас продублировать `person_name` во всех таблицах иерархии.

Вторая стратегия отображения проиллюстрирована на рис. 8.26 (для СУРБД Microsoft SQL Server). Таблица `Person` должна содержать комбинированное множество атрибутов во все классах иерархии обобщения. Она также содержит два столбца (`is_employee` и `is_student`) для фиксирования того, является ли лицо работником, студентом или и тем и другим.

Для иллюстрации третьей стратегии отображения мы предполагаем, что класс `Person` является абстрактным. Все атрибуты класса `Person` "наследуются" таблицами, соответствующими конкретному классу. Результат аналогичен показанному на рис. 8.27.

Последняя из стратегий проиллюстрирована на рис. 8.28 (для СУРБД Informix), при этом по-прежнему предполагается, что класс `Person` является абстрактным. В противоположность модели, показанной на рис. 8.25, мы предполагаем, что нам всегда известен тот факт, является работник также студентом и наоборот. Отсюда запрет на значения `null` для столбцов типа `BOOLEAN`.

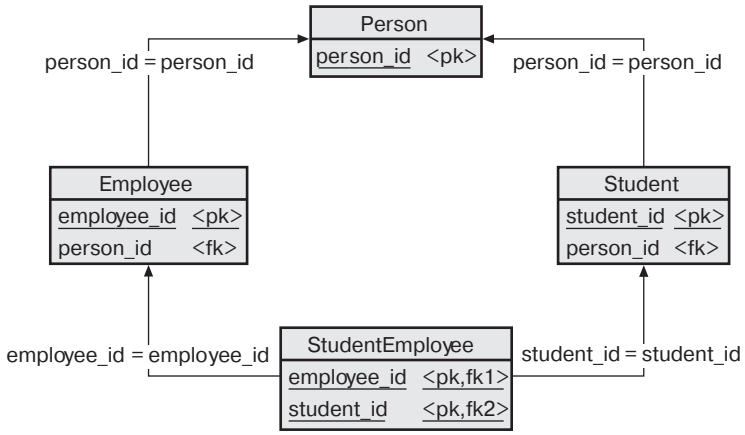


Рис. 8.25. Отображение каждого класса в таблицу

Person			
person_id	uniqueidentifier	<pk>	not null
is_employee	char(1)		null
is_student	char(1)		null

Рис. 8.26. Отображение иерархии классов в таблицу

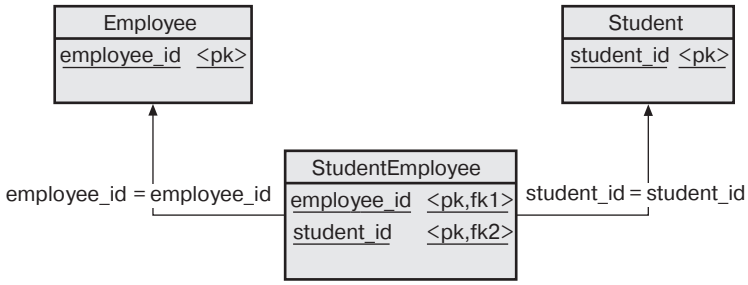


Рис. 8.27. Отображение каждого конкретного класса в таблицу

Employee			
employee_id	NCHAR(8)	<pk>	not null
is_student	BOOLEAN		not null

Student			
student_id	NCHAR(10)	<pk>	not null
is_employee	BOOLEAN		not null

Рис. 8.28. Отображение каждого отдельного конкретного класса в таблицу

Проект РБД (для СУРБД Sybase) на рис. 8.29 содержит таблицу для каждого из трех конкретных классов (`BetaTape`, `VHSTape` и `DVDDisk`). Таблицы дублируют следующую информацию: `video_condition`, `rental_period_in_days` и `rental_charge_per_period`. Для каждой ленты или диска может быть зафиксировано только по одному параметру `rental_period_in_days` и one `rental_charge_per_period`.

Пример 8.12. Магазин видеопроката

Обратитесь к спецификациям обобщения для приложения *Магазин видеопроката* в примере 4.10, рис. 4.7 (разд. 4.2.4.3). Обратитесь также к некоторым расширениям модели, введенным в примере 8.4 (разд. 8.2.2.4).

Задача заключается в том, чтобы отобразить диаграмму, представленную на рис. 4.7, в модель РБД с учетом расширений, приведенных на рис. 8.8 (пример 8.4). Мы воспользуемся третьей стратегией отображения каждого конкретного класса в таблицу.

Нам требуется рассмотреть, каким образом справиться с производным атрибутом `is_in_stock` и статическим атрибутом `number_currently_available`. Как и в предыдущих моделях, для приложения *Магазин видеопроката* мы игнорируем возможность наличия более одного условия аренды для одного и того же видеоносителя (т.е. `BetaTape`, `VHSTape` и `DVDDisk`).

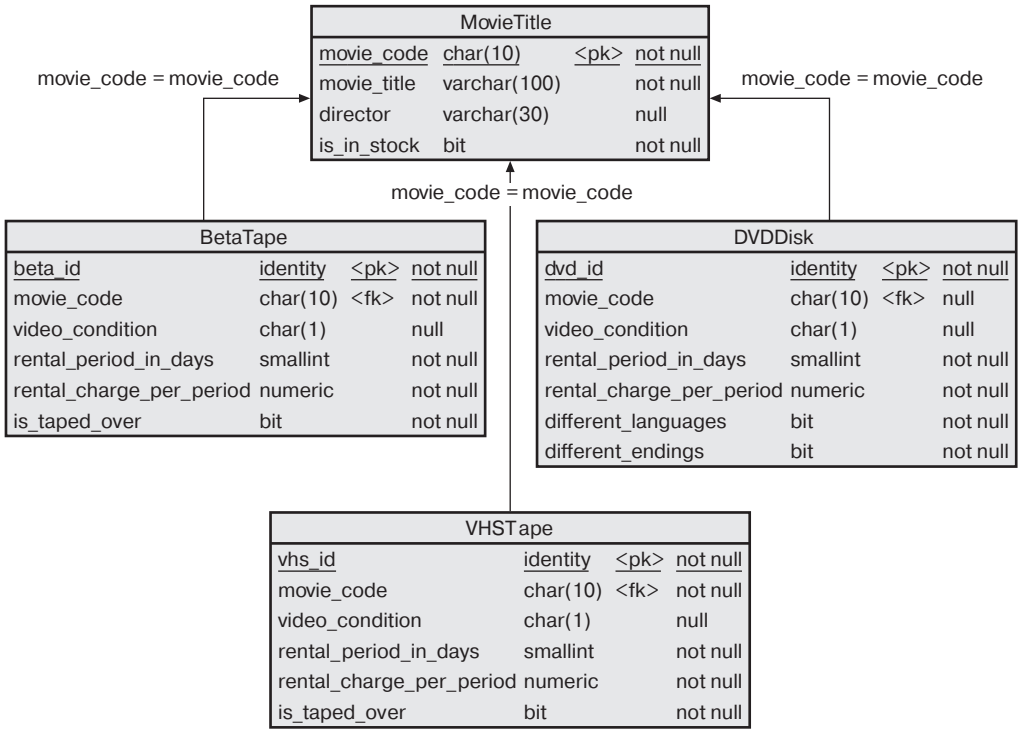


Рис. 8.29. Отображение обобщения в проект РБД (Магазин видеопроката)

Столбцы `is_taped_over`, `different_languages`, `different_endings` и `is_in_stock` введены с типом `bit`. По определению тип `bit` не допускает значений `null` (столбец типа `bit` принимает два значения: “0” и “1”).

Столбец `is_in_stock` устанавливается равным “истина” (“1”), если в запасе имеется в наличии, по меньшей мере, одна лента или диск с конкретным фильмом. Это не слишком полезная информация, если клиент интересуется одним из трех видеоносителей. Лучшим решением было бы введение трех столбцов типа `bit` или предполо-

жение о том, что эта информация никогда не хранится, т.е. является производной (вычисляемой) всякий раз, когда клиент спрашивает ленту или диск.

Статический атрибут `number_currently_available` не хранится ни в одной из таблиц, рассматриваемых в нашем проекте. Единственная таблица, в которой он мог бы постоянно храниться, — это таблица `MovieTitle`. Если принимается решение все же хранить его в этой таблице, то к нему применимы те же соображения, что для атрибута `is_in_stock`.

Резюме

Эта глава отражает крайне важную роль, которую играют базы данных при разработке ПО. Неудовлетворительный проект базы данных нельзя компенсировать другими средствами, включая качественную разработку приложений. Были рассмотрены все три основные модели баз данных — объектная, объектно-реляционная и реляционная модель.

Существует три уровня моделей баз данных — внешняя, логическая и физическая. В этой главе мы сконцентрировались на *логической модели*. Под *отображением объектов в базу данных* мы понимаем отображение классов UML-модели в логическую модель базы данных.

Отображение в *логическую модель ОБД* осуществляется наиболее просто. Модель ОБД поддерживает *объектные типы* наряду со *структурированными литералами* и *литералами-коллекциями*. Отношения моделируются с помощью *инверсии*. Для отображения обобщений UML можно использовать наследование двух видов: *ISA* и *EXTENDS*. Агрегации UML можно отображать с *помощью* коллекций.

Логическая модель ОРБД отличается большей сложностью по сравнению с моделью ОБД. Модель ОРБД проводит четкое различие между *структурированным типом* и *таблицей* как единственным механизмом хранения данных. *Строчный тип* позволяет задавать вложенные структуры данных — возможность, которая не так просто реализуется в UML. Поддерживаются также *ссылки* и *коллекции ссылок*. Для отображения обобщений UML можно использовать наследование двух видов: *OF* и *UNDER*.

Отображение в *логическую модель РБД* особенно неудобно. Модель РБД не поддерживает объектные типы, наследование, структурированные типы, коллекции или ссылки. Данные хранятся в *таблицах*, связанных *ограничениями ссылочной целостности*. Для программирования семантики, зафиксированной в бизнес-правилах, вытекающих из моделей классов UML, можно использовать *триггеры*. Кроме того, на отображение влияет *нормализация* таблиц.

Вопросы

В1. Дайте пояснения относительно трех уровней моделей баз данных: внешней, логической и физической модели.

В2. Как происходила эволюция стандартов ОБД? Каково практическое значение этой эволюции?

В3. В чем заключается различие между типом ОБД и классом ОБД?

В4. Что такое литерал коллекции ОБД? В чем он может быть полезен при отображении из модели классов UML?

В5. Что такое объектный тип коллекции ОБД? В чем он может быть полезен при отображении из модели классов UML?

В6. Дайте пояснения в отношении семантики и применимости ключевого слова inverse в схеме ОБД.

В7. В чем заключается различие между наследованием по типу ISA и по типу EXTENDS в схеме ОБД?

В8. Как главным образом используется строчный тип (row type) в ОРБД? Можно ли его использовать для определения ссылочных типов? Объясните свои соображения.

В9. В чем заключается различие между строкой ссылок и коллекцией ссылок в ОРБД?

В10. Поддерживают ли системы ОРБД динамическую классификацию? Поясните свой ответ.

В11. Объясните, в чем заключается различие между столбцом, полем и атрибутом ОРБД?

В12. В чем заключается различие между наследованием по типу OF и по типу UNDER в схеме ОРБД?

В13. Что такое ссылочная целостность РБД? В чем она может быть полезна при отображении из модели классов UML?

В14. Дайте пояснение в отношении четырех типов декларативных ограничений ссылочной целостности.

В15. Что такое триггер РБД? Как он связан со ссылочной целостностью?

В16. Что такое надлежащий уровень нормализации в РБД? Поясните свой ответ.

Упражнения

У1. *Телемаркетинг* — обратитесь к разбору примера *Телемаркетинг*, как это определено в диаграмме видов деятельности для книги, и к решению примера У1 в главе 7.

Отобразите диаграмму классов на схему ОБД. Объясните свое решение по отображению.

У2. *Телемаркетинг* — обратитесь к разбору примера *Телемаркетинг*, как это определено в диаграмме видов деятельности для книги, и к решению примера У1 в главе 7.

Отобразите диаграмму классов на схему ОРБД. Объясните свое решение по отображению.

У3. *Телемаркетинг* — обратитесь к разбору примера *Телемаркетинг*, как это определено в диаграмме видов деятельности для книги, и к решению примера У1 в главе 7.

Отобразите диаграмму классов на схему РБД. Объясните свое решение по отображению.