Содержание

Предисловие

Хотя на сегодняшний день теория логического проектирования баз данных – одна из областей информатики, развивающихся наиболее медленно, все-таки некоторые успехи в этой сфере налицо. Как раз теперь проектировщики баз данных являются свидетелями возникновения новой – объектно-ориентированной – логической модели данных, а также включения объектно-ориентированных возможностей в реляционную модель.

Пока еще рано говорить о том, за каким из этих двух подходов будущее. Но тем не менее уже есть несколько систем управления базами данных, основанных на объектно-ориентированной модели, достаточно надежных для эксплуатации в больших компаниях. В то же время производители давно признанных реляционных СУБД включают в свою продукцию поддержку объектов.

В этой книге представлены оба способа включения объектов в базы данных. Сначала изложены концепции, лежащие в основе объектно-ориентированной парадигмы, а затем показано, как эти концепции трансформируются в модель, поддерживаемую «чистыми» объектно-ориентированными СУБД.

Далее речь пойдет о том, как работают с объектами гибридные объектно-реляционные системы. Вы увидите, что рассматриваемые подходы имеют фундаментальные отличия, которые определяют возможности каждой системы.

Примечание *Споры между сторонниками объектно-ориентированных и объектно-реляционных баз, похоже, приобрели уже религиозный характер. Прочитав книги по данной теме, вы обнаружите, что у каждого подхода есть свои приверженцы. У меня, впрочем, нет особых предпочтений, поскольку существуют приложения, в которых лучше применять системы первого типа, и наоборот.*

В отличие от ситуации с реляционными базами данных, для объектно-ориентированных баз нет официального стандарта. Группа по управлению объектными базами данных (Object Database Management Group, ODMG), которая отвечает за разработку стандарта, существует вне традиционных органов стандартизации. Некоторые элементы проекта будут представлены в этой книге, но вы увидите и примеры конкретных реализаций, не следующих данному стандарту. Для реляционных систем такой разнобой довольно необычен, поскольку в среде теоретиков установилось молчаливое соглашение о том, что представляет собой реляционная база данных, а стандарт SQL в качестве языка манипулирования данными принят повсеместно.

Настоящее издание представляет собой дополнение к моей книге «Relational Database Design Clearly Explained», вышедшей в издательстве AP Professional. Поэтому не случайно, что примеры баз данных, приведенные в главах 6, 7 и 8, совпадают с рассмотренными в вышеуказанной книге. Это позволит вам сравнить проект реляционной базы данных с проектом объектной и гибридной объектно-реляционной баз и выбрать ту модель, которая в большей степени соответствует задачам, стоящим перед вашей организацией.

Что необходимо знать

Чтобы извлечь из этой книги максимум пользы, нужно иметь хорошее представление о реляционных базах данных. Необходимо понимать смысл таких терминов, как *отношение*, *домен*, *потенциальный ключ*, *первичный ключ*, *внешний ключ*, *целостность сущностей*, *ссылочная целостность*. Следует также знать язык SQL и основные конструкции, применяемые в *ER-диаграммах* (диаграммах «сущность-связь»).

Никаких знаний об объектно-ориентированной парадигме, напротив, не требуется; все необходимое объясняется по ходу изложения. Необязательно также уметь писать программы, но стоит помнить о том, что современные объектно-ориентированные СУБД тесно связаны с языками программирования, а синтаксис определения структуры базы данных основан на концепциях, применяемых в программировании. Поэтому знакомство хотя бы с одним из языков – C++ или Java – не повредит. Так, информация, содержащаяся в главах 9 и 10, где речь идет о конкретных реализациях объектных баз данных, окажется для вас намного ценнее, если вы умеете программировать.

Благодарности

Хочется поблагодарить людей, которые помогали мне при написании книги: Кена Мортона (Ken Morton) – редактора, от работы с которым я всегда получала огромное наслаждение; Габриэль Биллетер (Gabrielle Billeter) – помощника редактора, которая отвечала за все частности; Джули Шампейн (Julie Champagne) и Шона Гирсбергера (Shawn Girsberger) – выпускающих редакторов; Мэри Прескотт (Mary Prescott) – корректора. У нее острый глаз и легкая рука; Кэрола Макклендона (Carole McClendon) – моего агента, который и свел меня с издательством Morgan Kauffman (тогда оно еще называлось AP Professional).

JLH
http://www.blackgryphonltd.com

Часть I

Теория

В первой части книги рассматриваются теоретические аспекты объектно-ориентированной парадигмы, а также ее расширение для представления сущностей, хранящихся в базе данных, и связей между ними. Вы познакомитесь с проектом стандарта объектно-ориентированных баз данных (ООБД) и со способами моделирования таких баз посредством ER-диаграмм.

Примечание *Хотя назначение объектно-ориентированных ER-диаграмм такое же, как и при моделировании реляционных баз данных, сами модели все же несколько различаются.*

Глава 1. Введение

С точки зрения профессионала в области баз данных, *моделирование данных* – это искусство выявления сущностей, которые должны быть представлены в базе, и связей между ними. Объектно-ориентированная модель данных – последняя в ряду моделей, первая из которых возникла в начале шестидесятых годов.

В этой главе будет дан краткий обзор истории моделирования данных, чтобы стало понятно, почему объектно-ориентированную модель можно считать не только шагом вперед, но и шагом назад. Вы познакомитесь также с двумя способами включения объектов в модель данных. Кроме того, здесь приводится перечень известных проектов, в которых используются объектно-ориентированные базы данных.

Краткая история моделирования данных

До того как была разработана первая система управления базами данных (СУБД), доступ к данным осуществлялся прикладными программами напрямую, а сами данные были организованы в виде плоских файлов. Возникавшие при этом проблемы с логической целостностью данных, а также невозможность представить логические связи между ними в таких системах послужили причиной создания первой модели данных – иерархической.

Иерархическая модель данных

Иерархическая модель данных, реализованная, прежде всего, в системе Information Management System (IMS) компании IBM, допускает только два вида связей между сущностями: «один-к-одному» и «один-ко-многим». Любая сущность со стороны «много» может соотноситься только с одной сущностью со стороны «один».

Взгляните на рис. 1.1. На ER-диаграмме представлены две связи между сущностями `Product` (Изделие) и `Vendor` (Поставщик). Первая является связью типа «многие-ко-многим», показывающей, кто какой продукт поставляет; вторая предназначена для соотнесения изделий с поставщиками.

Связь типа «многие-ко-многим» между изделиями и поставщиками требует наличия композиционной сущности для представления цены изделия, назначенной поставщиком. Логически эта сущность – на рис. 1.1 она названа `Catalog Entry` (Элемент Каталога) – связана с обеими сущностями `Product` и `Vendor`. Однако в иерархической модели данных такого рода *множественная подчиненность* запрещена.

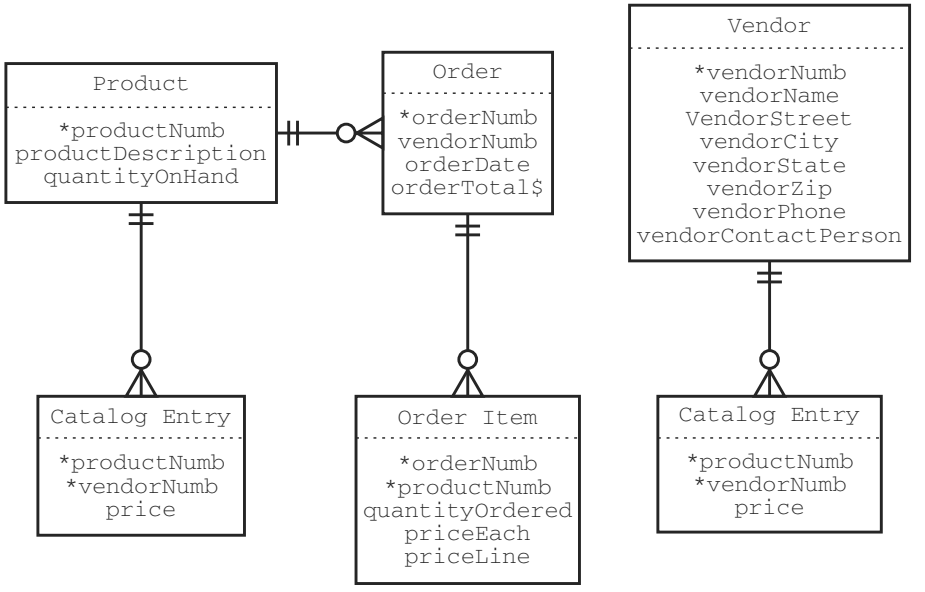

Рис. 1.1. Две иерархии данных

Примечание *Если быть точным, то в некоторых специальных случаях IMS допускает множественную подчиненность, в частности, когда типы родительских сущностей принадлежат разным иерархиям. Однако существуют жесткие ограничения на случаи, в которых можно этим пользоваться.*

Одно из возможных решений – продублировать сущность, как показано на рис. 1.1. Вместо этого можно было бы связать ее только с одним из двух родителей; так происходило со связями `Product–Order` (Изделие–Заказ) и `Order–Order Item` (Заказ–Строка Заказа).

Ни то, ни другое решение не идеально. Дублирование сущностей `Catalog Entry` создает возможность рассогласования данных. Одиночный же характер сущностей `Order` и `Order Item` накладывает ограничения на виды доступа, поддерживаемые базой данных.

Иерархические базы данных по сути своей являются *навигационными*. Это означает, что доступ возможен только с помощью заранее определенных связей. Кроме того, доступ, как правило, должен следовать иерархии и производиться начиная с ее вершины (корня). Например, чтобы добраться до экземпляра сущности `Order Item`, прикладная программа должна сначала отыскать подходящий экземпляр сущности `Product`, а затем обойти все экземпляры `Order`, связанные с найденным экземпляром `Product`, пока не будет обнаружено нужное изделие. В иерархии прямой доступ к данным крайне ограничен, а обычно и вовсе невозможен. Пользуясь иерархиями, изображенными на рис. 1.1, нереально без выполнения сложных операций ответить на вопрос: «У какого поставщика

заказано конкретное изделие?» Единственный способ удовлетворить такой запрос – пройти от `Product` к `Order` и затем к `Order Item`, чтобы найти все строки заказа, в которых фигурирует искомое изделие. После этого приложение должно было бы отыскать экземпляры `Catalog Entry` во второй иерархии, чтобы определить поставщика.

Достоинство иерархической базы данных в том, что ее навигационная природа обеспечивает очень быстрый доступ при следовании вдоль заранее определенных связей. Однако негибкость модели данных и, в частности, невозможность наличия у сущности нескольких родителей, а также отсутствие прямого доступа к данным делают ее непригодной в условиях частого выполнения запросов, не запланированных заранее.

IMS используется и сегодня в некоторых унаследованных системах на больших ЭВМ, но компания IBM больше не продает этот продукт и рекомендует своим клиентам по возможности переходить на реляционные СУБД.

Сетевые модели данных

Задолго до того, как компьютерные сети стали привычными, проектировщики баз данных создали две модели, описывающие сети связей между данными: *простую* и *сложную сетевые модели*. Они были призваны устранить ограничения, свойственные иерархической модели данных.

Простая сетевая модель данных

В простой сетевой базе данных все связи имеют тип «один-к-одному» или «один-ко-многим» – непосредственные связи «многие-ко-многим» не допускаются. Однако разрешена множественная подчиненность. Как видно из рис. 1.2, допустимость множественной подчиненности решает проблемы избыточного дублирования и ограничений на виды доступа.

Простые сетевые базы данных сохраняют навигационный характер: по большей части доступ осуществляется с помощью заранее определенных связей. В какой-то степени прямой доступ к экземплярам сущностей поддерживается за счет хэширования, но, поскольку последнее влияет на физическое размещение данных в файле, на практике быстрый доступ возможен лишь к одной сущности в иерархии. Например, в базе данных изделий и заказов из примера выше сущности `Product` и `Vendor` можно хранить с использованием хэширования для предоставления к их экземплярам прямого доступа. Но остальные сущности, скорее всего, будут доступны только через своих родителей.

Примечание *Хэшированием называется техника, при которой значение ключа, аналогичное первичному ключу отношения, с помощью некоторой процедуры (алгоритма хэширования) преобразуется в адрес экземпляра сущности в файле данных. Когда пользователь указывает ключ, СУБД может вычислить соответствующее хэшированное значение и использовать его для прямого доступа к данным, преимущество которого состоит в его скорости.*

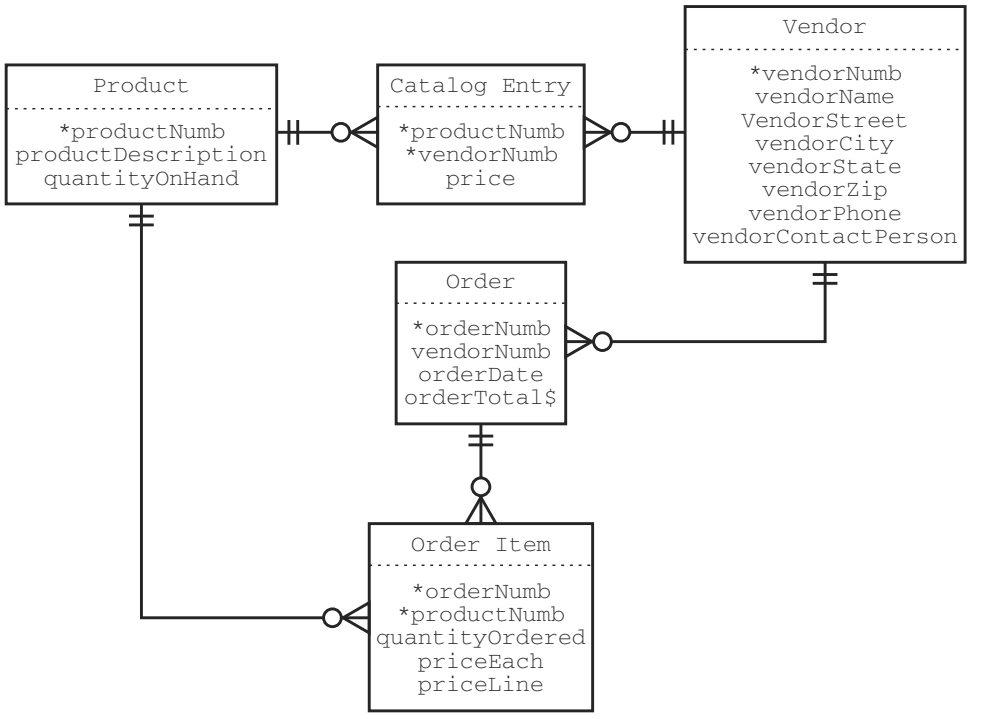

Рис. 1.2. Простая сетевая модель данных

Таким образом, простые сетевые базы данных демонстрируют хорошую производительность, если доступ осуществляется с помощью заранее определенных связей, но работают медленно, когда в программе приходится кодировать поиск, не основанный ни на таких связях, ни на хэшированных ключах. В зависимости от того, как спроектирована база данных, удовлетворить незапланированные запросы может оказаться очень сложно.

Примечание *Когда простые сетевые базы данных стали вытесняться реляционными, многие начали проектировать простые сети так, как если бы они были реляционными конструкциями, используя, в частности, и внешние ключи в связанных сущностях. В результате некоторые поставщики простых сетевых СУБД сумели встроить в свои продукты языки для формирования запросов, обеспечив тем самым выполнение незапланированных запросов так, как это делается в реляционных базах. Однако подобная техника хорошо работала лишь в случае, если модель данных была нормализована.*

Национальный стандарт простой сетевой модели данных первоначально был предложен Группой по базам данных (Database Task Group, DBTG) или Комитетом по языкам систем обработки данных (Committee on Data Systems Language,

CODASYL). Кстати, именно этот комитет разработал язык COBOL. Хотя некоторые унаследованные базы данных на основе стандарта CODASYL эксплуатируются и поныне, новые системы такого рода устанавливаются очень редко.

Сложная сетевая модель данных

Сложная сетевая модель данных аналогична простой, но допускает непосредственную реализацию связей типа «многие-ко-многим» (см. рис. 1.3).

Связь «многие-ко-многим» существует между служащими и их детьми. Вторая представленная на диаграмме связь – это связь «один-к-одному», показывающая, кто на ком женат. Поскольку не существует данных, ассоциированных со связью, прямая реализация связи «многие-ко-многим» в этом случае не составляет проблемы и избавляет от необходимости создавать композиционную сущность, нужную исключительно для представления связи. Однако если со связью ассоциированы некоторые данные, то даже в сложной сетевой модели приходится формировать композиционную сущность. Кроме того, бывают ситуации, когда непосредственная реализация связи «многие-ко-многим» приводит к потере информации в базе данных (см. главу 5).

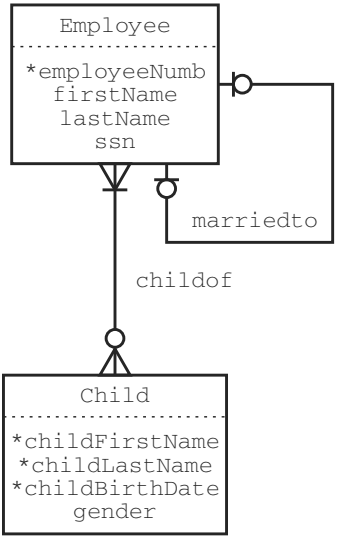

Рис. 1.3. Сложная сетевая модель данных

Коммерчески успешных реализаций сложных сетевых СУБД так и не появилось, в основном из-за того, что связи «многие-ко-многим» стали чересчур запутанными и сложными для сопровождения. Таким образом, эта модель представляет лишь теоретический интерес и продолжает обсуждаться разве что в вузовских учебниках.

Решительный прорыв: реляционная модель данных

Если вы внимательно проследите эволюцию моделей данных от иерархических до сложных сетевых, то заметите, что все они, по сути дела, похожи. Каждая следующая модель устраняла ограничения на типы допустимых связей. Напротив, реляционная модель имела принципиальные отличия. Хотя ER-диаграмма реляционной модели базы данных об изделиях и поставщиках выглядит в точности так же, как показано на рис. 1.2, реализация отличается весьма существенно.

В частности, связи, изображенные на рис. 1.2, больше не являются составной частью реляционной базы. Логические связи представлены только соответствиями между первичными и внешними ключами, которые при необходимости используются для выполнения операции соединения. Поэтому реляционные базы данных уже не являются навигационными и предоставляют поддержку незапланированных запросов, несопоставимую с тем, что было раньше. Модифицировать

схему реляционной базы данных опять же проще, поскольку в большинстве случаев это можно сделать, не приостанавливая работу с ней.

Традиционно недостатком реляционных баз данных считалась сравнительно низкая производительность. Поскольку иерархическая и сетевая модели тесно связаны с физическим размещением информации, а структуры, описывающие связи, являются неотъемлемой частью базы, то доступ к связанным данным удается выполнить гораздо быстрее. И все же, по мере того как для ведения бизнеса все чаще приходилось выполнять незапланированные запросы, реляционные базы стали вытеснять старые системы. Если принять во внимание быстродействие современных компьютеров, низкая производительность уже не является непреодолимым препятствием.

Переход к объектно-ориентированной модели

Появление объектно-ориентированной парадигмы (подробно она рассматривается в главе 2) внесло фундаментальные изменения во взгляды на данные и процедуры их обработки. Традиционно информация и процедуры хранились раздельно, данные и связи между ними – в базе данных, а процедуры – в прикладной программе. Объектная же ориентированность позволяет хранить процедуры обработки сущностей вместе с данными.

Обычно такое совместное хранение считается шагом вперед в методах управления данными. Сущности становятся замкнутыми единицами, которые сравнительно легко можно использовать повторно и перемещать в новое место. Теперь поведение сущности не привязано к конкретной прикладной программе, а является частью самой сущности, поскольку вне зависимости от того, где сущность используется, она ведет себя предсказуемым, заранее известным образом.

В главе 3 вы узнаете, что объектно-ориентированная модель к тому же непосредственно поддерживает связи типа «многие-ко-многим» – впервые в истории моделирования данных. Но это палка о двух концах. Вы увидите, что при проектировании таких связей надо соблюдать осторожность, чтобы не потерять информацию.

Ко всему прочему, объектно-ориентированные базы являются еще и навигационными: доступ к данным производится с помощью связей, хранящихся внутри самих данных. Потому это и шаг назад. Формулирование незапланированных запросов к объектно-ориентированной базе осуществляется не так просто, как к реляционной, хотя, как правило, такие запросы поддерживаются. Однако они, в силу навигационной природы объектно-ориентированной базы данных, должны следовать заранее определенным связям и не могут создавать новые связи «на лету».

По прочтении этой книги вы поймете, почему современное поколение объектно-ориентированных баз данных вряд ли сумеет полностью заменить реляционные БД во всех приложениях, как в свое время реляционные базы вытеснили все предшествующие модели. Вы также научитесь распознавать те ситуации, в которых применение объектно-ориентированных баз оправдано.

Роль объектов в базах данных

Объекты включаются в базу данных двумя способами:

- *полностью объектно-ориентированные СУБД*. В основе такой СУБД лежит исключительно объектно-ориентированная модель данных. Сегодня имеется несколько промышленных объектно-ориентированных СУБД, и некоторые из них используются в критически важных приложениях;
- *гибридные*, или *постреляционные*, *СУБД*. Такая СУБД по преимуществу является реляционной, но может хранить объекты в отношениях.

Теоретики и практики не пришли к общему мнению касательно того, какой из указанных подходов в конечном счете выживет. В отличие, скажем, от стандартов видеозаписи Beta и VHS, каждая из названных стратегий будет применяться для решения своего класса задач. Различие между полностью объектно-ориентированными и гибридными базами обсуждается более подробно в главе 3.

Примеры использования объектно-ориентированных баз данных

Сегодня объектно-ориентированные базы данных применяются в самых разных организациях для решения широкого круга задач. Вот некоторые примеры:

- F. A. Davis Company – компания, издающая медицинский словарь «Taber's Cyclopedic Medical Dictionary» и другие технические справочники, – использует объектно-ориентированную базу для управления содержанием своих изданий. Для форматирования документов применяются языки XML и SGML. Как выяснилось, иерархическая природа таких документов лучше ложится на объектно-ориентированную, а не реляционную модель;
- компания Adidas AG применяет объектно-ориентированную базу для ведения своего каталога на CD-ROM, сопровождения Web-сайта и управления работой магазинов розничной торговли. Консалтинговая компания, нанятая Adidas для разработки корпоративной информационной системы, пришла к выводу, что такая база данных лучше всего подходит для обработки огромных объемов мультимедийной информации (текста, аудиоинформации, графики и видео). Навигационная природа объектно-ориентированной модели обеспечивает высокую производительность при выборке мультимедийных файлов;
- федеральная авиационная администрация (Federal Aviation Administration, FAA) применяет систему, созданную на основе объектно-ориентированной базы данных, для моделирования пассажиро- и грузопотоков. Поскольку в такой базе процедуры хранятся вместе с данными, моделирование сложных взаимосвязей между объектами оказывается проще, чем при использовании реляционной базы;

- компания Sales Media Inc., разработчик программ для индустрии страхования, использует ООБД в качестве основы своего продукта Automated Agent. Комбинация объектно-ориентированного хранилища данных и объектно-ориентированного языка программирования позволяет компании ускорить поставку обновленных версий своих продуктов (о преимуществах объектно-ориентированного программирования речь пойдет в главе 3);
- компания Radio Computing Services предлагает программы автоматизации радиостанций. Первая выпущенная ей программа Selector поддерживала отбор, упорядочение и мониторинг песен, звучавших на радиостанции. Сегодня компания использует объектно-ориентированную базу в качестве хранилища данных для более функционального пакета программ автоматизации. Объектно-ориентированная среда сокращает время разработки, давая программистам возможность одинаково работать со всеми составными частями программы независимо от их природы;
- компания Wandel * Goltermann Technologies Inc. использует объектно-ориентированную базу как хранилище данных для своего продукта Domino WIZARD. Эта система производит измерение различных характеристик сети и сохраняет их результат, с тем чтобы базовую статистику можно было впоследствии сравнить с результатами новых измерений – для поиска неисправностей, а также в интересах фирм, продающих сетевое оборудование. В последнем случае продавец может, например, сравнить текущую производительность сети с той, которой удалось достичь после установки нового коммутатора или маршрутизатора (разумеется, в надежде, что она повысилась);
- компания Echelon применяет объектно-ориентированную базу как основу для своих программ создания управляющих сетей[1]. Программа моделирует каждый управляемый элемент, учитывая его ожидаемое поведение, и облегчает обнаружение отклонений от нормальной работы;
- компания Interface and Control Systems, Inc. использует объектно-ориентированные технологии для создания базы данных, поддерживающей систему слежения за полетом космического аппарата и управления им. Для работы данной системы требуется анализировать очень большие объемы данных. Объектно-ориентированная база способна сделать это без заметного снижения производительности;
- французский национальный центр космических исследований применяет объектно-ориентированные технологии в авиакосмической промышленности. В частности, там разработана мультимедийная база данных, помогающая

[1] Это сеть, которая следит за некоторым процессом или управляет им при минимальном участии человека. К данному классу можно отнести заводские роботизированные комплексы, системы контроля климата и управления лифтами в высотных зданиях.

моделировать интегрированные системы, необходимые в проектировании космических аппаратов. Специалисты, работающие в этой организации, пришли к выводу, что объектно-ориентированная база хорошо приспособлена к среде, в которой имеется большое число элементов, взаимодействующих между собой сложным образом;

- крупнейшая швейцарская компания медицинского страхования Swiss Social Christian располагает объектно-ориентированной базой данных, лежащей в основе системы обработки обращений о страховых выплатах. Компания пришла к решению использовать такую базу, поскольку правила принятия решений о том, какая сумма подлежит выплате, очень сложны. Система способна выдерживать большой поток обращений, предположительно до 18000 в день в 2000 году;
- компания STERIA применяет объектно-ориентированную базу данных для поддержки коммерческого приложения, предназначенного для проектирования и планирования больших сетей передачи данных. Сложность таких сетей настолько быстро возрастает, что весьма трудно следить за тем, где какое оборудование и в какой конфигурации установлено. Объектно-ориентированная база позволяет проектировщику моделировать взаимодействия различных элементов сети и постоянно быть в курсе ее структуры;
- французская электрическая компания Electricite de France пользуется объектно-ориентированной базой для управления нагрузкой на линии электропередачи. База данных способна нарисовать карту линий, расположенных в области ответственности обслуживающего предприятия. ООБД также помогает определить, какое оборудование необходимо для прокладки новой линии.

Примечание *Более подробную информацию об упомянутых приложениях можно получить на следующих сайтах: http://www.poet.com, http://www.advantasoft.com.*

После знакомства со всеми этими приложениями можно сформулировать некоторые общие наблюдения:

- многие приложения состоят из большого числа взаимодействующих частей, как, например, космические аппараты или большие вычислительные сети. Каждая из этих частей обладает своим поведением, которое зависит от поведения других;
- многие системы должны обрабатывать большие объемы данных;
- многие приложения осуществляют предсказуемый доступ к данным, поэтому навигационная природа объектно-ориентированных баз не является существенным недостатком;

- в большинстве приложений потребность в незапланированных запросах ограничена.

Сформулированные выше характеристики помогают идентифицировать приложения, в которых оправдано применение объектно-ориентированных баз данных. Однако в случаях, когда способы доступа приложения к данным нелегко определить заранее, реляционная база, возможно, станет более подходящим выбором.

Глава 2. Основы объектно-ориентированной парадигмы

Объектно-ориентированную парадигму предложил доктор Кристен Нигард (Kristen Nygarrd), норвежский ученый, который пытался написать компьютерную программу для моделирования взаимодействия судов, приливов и фьордов. Он обнаружил, что связи очень сложны, и понял, что составить программу проще, если разделить три вида моделируемых элементов и позволить каждому из них определять свое поведение относительно двух других.

Используемые в настоящее время объектно-ориентированные языки программирования (наиболее известны C++, Smalltalk и Java) являются прямым результатом ранней работы Нигарда. Объектно-ориентированная модель данных – это развитие объектно-ориентированного программирования.

Примечание *Ситуация здесь прямо противоположна истории создания реляционной модели, которая была специально разработана для моделирования связей между данными, хотя большая часть ее теоретических основ заимствована из математической теории множеств.*

Поэтому для понимания объектно-ориентированной модели данных вы должны прежде всего усвоить объектно-ориентированную парадигму в том виде, в котором она используется в программировании. В этой главе вы познакомитесь с основными понятиями указанной парадигмы. Не огорчайтесь, если не умеете программировать: для усвоения этого материала необязательно быть программистом. Если же вы хорошо знакомы с объектно-ориентированными базами данных, можете сразу перейти к главе 3, в которой объясняется, как оно применяется при моделировании данных.

Самый простой способ разобраться в том, что такое объектно-ориентированное программирование, – начать с примера, который к программированию не имеет ровным счетом никакого отношения.

Составление инструкции

Предположим, что у вас есть шестнадцатилетняя дочь (или сестра) по имени Джейн и ваша семья собралась в дальнюю поездку на автомобиле. Как все подростки ее возраста, Джейн отнюдь не горит желанием путешествовать с семьей, а уж тем более проводить время в обществе своего двенадцатилетнего брата. Чтобы брат не мешал ей читать, пока родители по очереди ведут машину, Джейн решает написать для него инструкции по раскладыванию пасьянсов.