# КОНСПЕКТ ЗАНЯТИЯ № 2 ШЕСТОЙ НЕДЕЛИ КУРСА «БАЗЫ ДАННЫХ»

## 1. Проектирование баз данных с использованием методологии IDEF1X

В данном разделе будет рассмотрена методология проектирования и нотация (правила изображения) диаграмм IDEF1X, а приводимые примеры будут иллюстрироваться с использованием программного продукта ERwin Data Modeler v.9 в версии Community Edition (нам предстоит познакомиться с данным ПО при работе с лабораторными заданиями). Данная версия продукта свободно распространяется, ее можно получить через web-сайт www.erwin.com. Наиболее существенным ограничением версии ERwin Community Edition является небольшое количество объектов в модели – не более 25, но для учебных целей это не является критичным.

Как уже говорилось, модель IDEF1X (от англ. Integration Definition for Information Modeling) представляющая собой модификацию модели «сущность—связь», была объявлена национальным стандартом США в 1993 году. Ее основу составляют работы, выполнявшиеся в середине 1980-х годов для нужд военного ведомства.

1) Стандарт IDEF1X расширяет модель «сущность—связь» в ряде аспектов, благодаря чему с его помощью становится возможным создавать как концептуальные, так и внутренние схемы.
2) Стандарт IDEF1X предполагает, что создаваемая база данных является реляционной.
3) Стандарт IDEF1X включает сущности, связи и атрибуты, но значения этих понятий в нем сужены, а для их определения применяется более точная терминология.
4) В IDEF1X введено понятие *домена* (domain), отсутствующее в расширенной модели «сущность—связь».

Поскольку средства моделирования IDEF1X специально разработаны для построения реляционных информационных систем, то в случае если существует необходимость проектирования другой системы, скажем объектно-ориентированной, то лучше избрать другие методы моделирования.

Существует несколько очевидных причин, по которым IDEF1X не следует применять в случае построения нереляционных систем. К примеру, IDEF1X требует от проектировщика определить ключевые атрибуты, для того чтобы отличить одну сущность от другой, в то время как объектно-ориентированные системы не требуют задания ключевых ключей, в целях идентифицирования объектов.

Терминология IDEF1X очень близка к терминологии, используемой в других методиках моделирования БД. Напомним, главные элементы ER-диаграммы:

- *Сущность* является представлением множества реальных или абстрактных объектов (явлений, событий и т. п.) относимых к одному типу: они обладают одинаковым набором характеристик и могут участвовать в однотипных связях с другими сущностями. Конкретный объект из подобного множества будет называться экземпляром. Экземпляры должны отличаться другу от друга. Сущности именуются существительным в единственном числе, например, «Компьютер».
- *Атрибут* соответствует свойству или характеристике, имеющейся у всех или некоторых экземпляров сущности. Множество возможных значений атрибута ограничивается доменом, на котором он определяется.
- *Связь* описывает логическое отношение между двумя сущностями или между экземплярами одной сущности. При моделировании БД связь принято именовать глаголом или глагольной фразой.

Сущности изображаются в виде прямоугольников с прямыми или закругленными углами. Сущности могут изображаться без атрибутов (рис. 2.16, а), с указанием только тех атрибутов, которые составляют первичный ключ (рис 2.16, б), или с указанием всех имеющихся атрибутов (рис. 2.17).

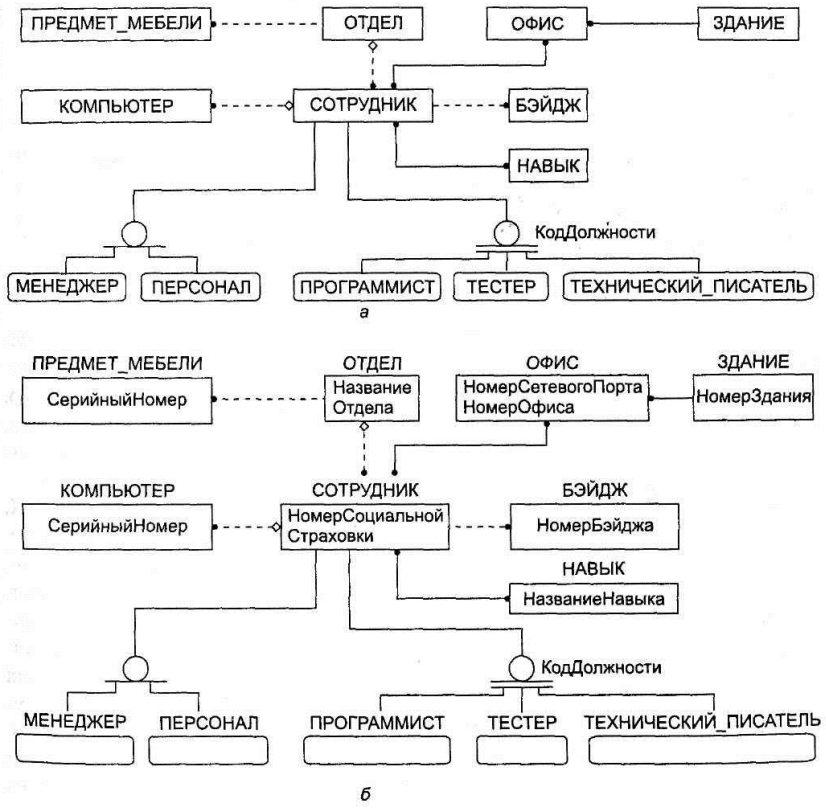


**Рис. 2.16.** Уровни детализации моделей стандарта IDEF1X: *а* — только сущности; *б* — сущности и первичные ключи

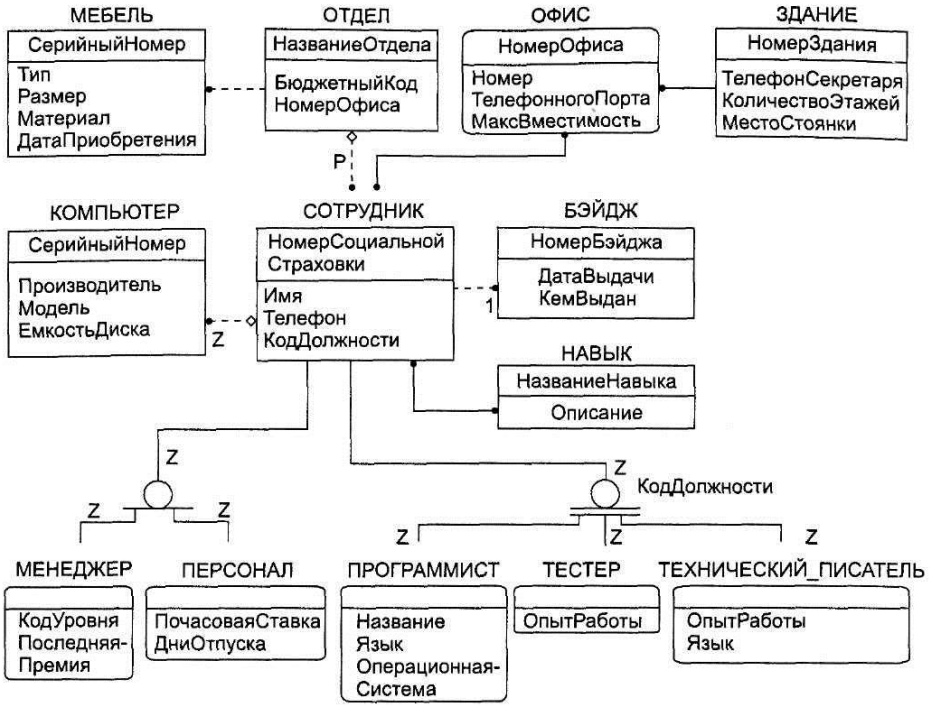


**Рис. 2.17.** Уровни детализации моделей стандарта IDEF1X — сущности и атрибуты

Напомню, что ключ или возможный ключ – это минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся. Каждая сущность обладает хотя бы одним возможным ключом. Один из них принимается за первичный ключ.

В случае, когда сущность изображается со всеми атрибутами, символизирующий ее прямоугольник делится на две части: в верхней части указываются атрибуты, входящие в первичный ключ, а в нижней — все прочие атрибуты.

Поскольку средства моделирования данных, отвечающие стандарту IDEF1X, могут использоваться для создания как концептуальных, так и внутренних схем, они позволяют изобразить на схеме размещение внешних ключей (связи в реляционной модели создаются путем помещения ключа одной таблицы в другую таблицу. По отношению ко второй таблице такой ключ называется внешним ключом - foreign key).

Рассмотрим пример, иллюстрирующий использование различных типов сущностей и связей, определяемых нотацией IDEF1X. Пусть требуется разработать БД для хранения информации о компьютерной системе предприятия. С помощью программы ERwin Data Modeler начнем создавать логическую модель. Компьютеры имеют уникальные инвентарные номера и имена, причем для одного компьютера может быть использовано несколько дополнительных имен (псевдонимов). Например, компьютер `serv1.corp.ru` может иметь псевдонимы `www.mycorp.net` и `ftp.mycorp.net`. Компьютеры размещены в помещениях организации, на них

установлено программное обеспечение. Фрагмент логической модели БД представлен на рис. 6.4.

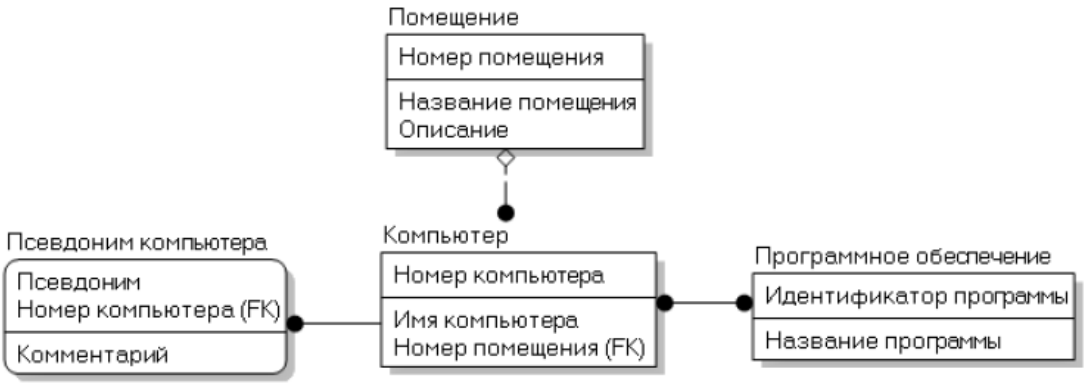


Рис. 6.4. Фрагмент логической модели базы данных в нотации IDEF1X

Сущности в IDEF1X изображаются с помощью прямоугольника, над которым указывается его название, а в верхней части над чертой перечисляются атрибуты, входящие в первичный ключ. Сущности могут быть независимыми (англ. Identifier-Independent Entity) и зависимыми (англ. Identifier-Dependent Entity). Отличие заключается в том, что для существования экземпляра зависимой сущности необходима его связь с экземпляром сущности, от которой он зависит. Зависимые сущности изображаются прямоугольником со скругленными краями. В примере, приведенном на рис. 6.4, имеется независимая сущность «Компьютер» и зависящая от нее сущность «Псевдоним компьютера». Это означает, что псевдоним не может существовать без указания компьютера, для которого он задан.

Между независимой сущностью и подчиненной ей зависимой устанавливается *идентифицирующая связь* типа «один-ко-многим». Связь изображается непрерывной линией с черным кружком на стороне подчиненной сущности. При создании модели БД в ERwin, сущности сначала создаются как независимые, а после установки идентифицирующей связи, дочерняя сущность автоматически преобразуется в зависимую. При этом в ее первичный ключ автоматически добавляются атрибуты первичного ключа родительской сущности, которые также сформируют внешний ключ: на рис. 6.4 это атрибут «Номер компьютера», который входит в состав первичного ключа сущности «Псевдоним компьютера». Внешний ключ отмечается на диаграмме символами «(FK)».

Операция автоматического добавления (при создании связи) ключевых атрибутов родительской сущности в дочернюю сущность в качестве внешнего ключа называется *миграцией ключей*.

Нотация IDEF1X предусматривает дополнительные обозначения (см. рис. 2.19):

1. Для связи 1:N следует следующее обозначение: если потомок является обязательным, рядом с кружком на соответствующем конце линии связи помещается символ P (от positive — положительный; имеется в виду

положительное число). Индекс указывает на любое количество экземпляров потомка большее или равное 1.

2. Если родитель является необязательным, то на родительском конце линии связи рисуется ромб.
3. Для связи 1:1 обозначаются при помощи индексов рядом с кружком на соответствующей стороне связи. Число 1 показывает, что требуется ровно один потомок, не больше и не меньше, а буква Z означает, что потомок может быть один, либо его может не быть вовсе.

В случае идентифицирующей связи на родительской стороне не может быть ромба, поскольку сущности-родители в таких связях всегда являются обязательными.

Рассмотрим вторую связь типа «один-ко-многим» представленную на диаграмме (рис.6.4). Это связь между двумя независимыми сущностями «Помещение» и «Компьютер». В терминологии IDEF1X такая связь будет являться *неидентифицирующей*. Она изображается на диаграмме пунктирной линией с черным кружком со стороны дочерней сущности. В данном случае, мы считаем, что для компьютера может быть определено не более одного помещения, и могут существовать компьютеры, для которых помещение не задано. С точки зрения проектирования реляционной базы данных, это означает, что в таблице «Компьютер» для внешнего ключа «Номер помещения» допустимо значение NULL. На диаграмме это изображается пустым ромбом со стороны родительской сущности (рис.6.4). Такая связь называется *опциональной* или *необязательной*. Если ромба нет, это означает обязательное участие подчиненной сущности в данной связи, что реализуется ограничением NOT NULL для соответствующего внешнего ключа. В IDEF1X такая связь называется *обязательной неидентифицирующей связью*.

При добавлении в модель неидентифицирующей связи, также осуществляется миграция ключей: в дочернюю сущность автоматически добавляется внешний ключ, формируемый на основе первичного ключа родительской сущности. Но в отличие от идентифицирующей связи, в данном случае внешний ключ не становится частью первичного ключа дочерней сущности.

Таким образом, *неидентифицирующие связи* – это связи 1:1 и 1:N между двумя независимыми сущностями. В случае связи 1:N родителем является сущность на той стороне связи, которая обозначена цифрой 1. В случае связи 1:1 любая из сущностей может считаться родителем, но стандарт IDEF1X предписывает выбрать на эту роль какую-то одну из сущностей.

Рассмотрим еще один пример неидентифицирующей связи с дополнительными обозначениями (рис. 2.19):

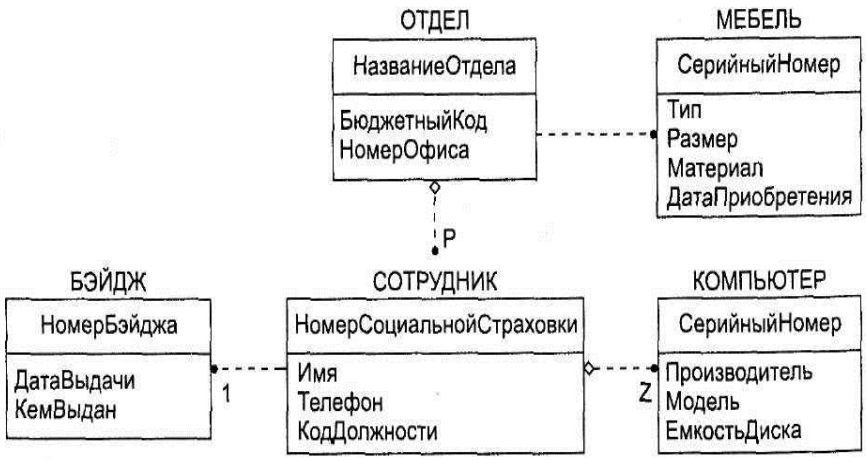


**Рис. 2.19.** Неидентифицирующие связи принадлежности

На *рис. 2.19* связь между сущностями ОТДЕЛ и СОТРУДНИК имеет вид 1:N, причем любой отдел должен иметь по крайней мере одного сотрудника, но отдельно взятый сотрудник не обязан числиться в каком-либо отделе (это определение достигается путем использования идентификаторов – P на стороне потомка (1 потомок или более) и пустого ромба на стороне родителя).

На *рис. 2.19* сущность СОТРУДНИК имеет связь с одной и только одной сущностью БЭЙДЖ (достигается путем использования символа 1), а также с одной или нулевым количеством сущностей КОМПЬЮТЕР (обозначение Z дает нам эту информацию). Ромб показывает, что компьютер не обязательно должен быть связан с каким-либо сотрудником. Поскольку на линии связи БЭЙДЖ-СОТРУДНИК со стороны сущности СОТРУДНИК ромба нет, каждый значок должен принадлежать какому-то сотруднику.

Третий тип связи, определяемый методологией IDEF1X – связь «многие-ко-многим», ее также именуют *неспецифической связью*. При проектировании реляционной базы данных подобная связь допустима только в логической модели. В приведенном на рис. 6.4 примере, это связь между сущностями «Компьютер» и «Программное обеспечение». Связь «многие-ко-многим» изображается сплошной линией с двумя черными кружками на концах.

Реляционная модель в явном виде не поддерживает связи типа «многие-ко-многим». При проектировании БД в среде ERwin, при переходе от логической модели БД к физической, связь «многие-ко- многим» автоматически преобразуется: создается зависимая сущность, для которой определяются две связи «один-ко-многим». На рис. 6.5 изображен фрагмент физической модели БД, построенной для логической модели, представленной на рис. 6.4.

Автоматически созданная таблица «Computer_Software», имя которой было сгенерировано на основе имен родительских таблиц, получила составной первичный

ключ, включающий первичные ключи исходных таблиц. Таким образом, миграция ключей в данном случае произошла на уровне физической модели.

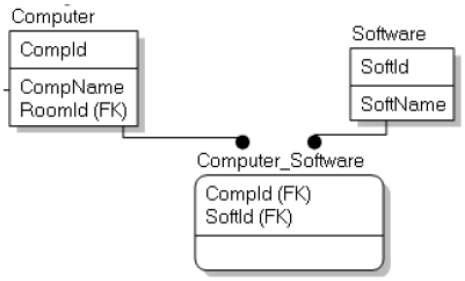


Рис. 6.5. Преобразование связи «многие-ко-многим» в физической модели

Четвертая связь в IDEF1X *именуется категориальной связью*.

*Категориальная связь* — это связь между *порождающей сущностью* и другой сущностью, которая называется *сущностью-категорией*. Вспомните в расширенной версии «сущность-связь» мы также говорили об этой связи, но там использовались термины: надтип и подтип.

Сущности-категории группируются в *категориальные кластеры*.

Например, на *рис. 2.22* сущность СОТРУДНИК является порождающей сущностью, сущности ПРОГРАММИСТ, ТЕСТЕР и ТЕХНИЧЕСКИЙ ПИСАТЕЛЬ являются сущностями-категориями.

Первичным ключом сущностей-категорий служит ключ порождающей сущности. В данном примере первичным ключом сущностей ПРОГРАММИСТ, ТЕСТЕР и ТЕХНИЧЕСКИЙ_ПИСАТЕЛЬ является атрибут НомерСоциальнойСтраховки. В связи с этим обстоятельством сущности-категории показываются на схеме без первичных ключей.

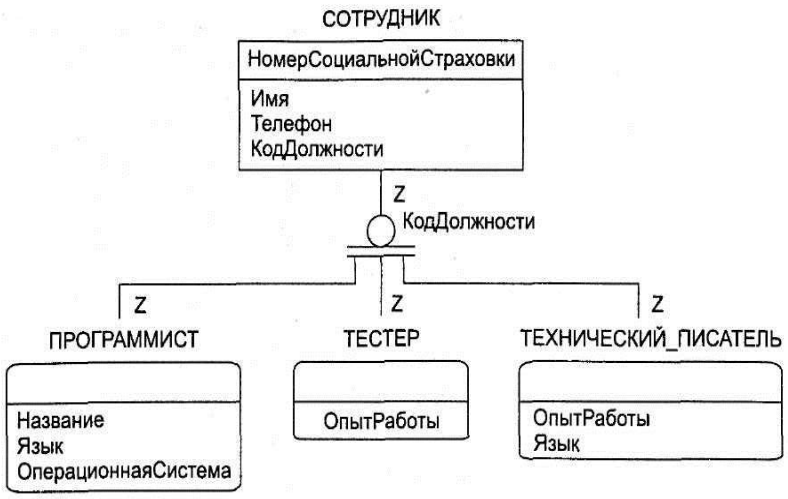


**Рис. 2.22.** Категориальные связи


**Рис. 2.23.** Неполные и полные категориальные кластеры

В стандарте IDEF1X сущности, входящие в категориальный кластер (на диаграмме представлен в виде кружка), являются взаимоисключающими. На *рис. 2.22* сотрудник может быть либо программистом, либо тестером, либо техническим писателем. Состоять на двух или более должностях одновременно сотрудник не может.

Категориальные кластеры могут иметь *дискриминатор* (discriminator) — атрибут порождающей сущности, указывающий на тип сотрудника. На рис. 2.22 дискриминатором является атрибут КодДолжности. Таким образом, по значению данного атрибута можно определить, какую должность занимает сотрудник — программист, тестер или технический писатель. Способ, при помощи которого это делается, остается за рамками концептуальной модели. Некоторые категориальные кластеры не имеют дискриминатора, и конкретная сущность-категория остается в них неопределенной.

Есть два типа категориальных кластеров: полные и неполные. В *полном категориальном кластере* показана каждая из возможных для этого кластера категорий. Полные категориальные кластеры обозначаются двумя горизонтальными линиями с зазором между ними. Категориальный кластер, изображенный на рис. 2.22, является полным – то есть на схеме изображены все без исключения категории сотрудников, входящие в данный кластер. Соответственно, каждый сотрудник может быть классифицирован как либо программист, либо тестер либо технический писатель.

Все сущности-категории являются жизненно зависимыми от порождающей сущности, поэтому минимальное кардинальное число связи со стороны порождающей сущности равно 1. Так как это справедливо во всех случаях, данное кардинальное число не показывается на диаграмме.

Согласно стандарту IDEF1X, кардинальное число категориальной связи на стороне сущности-категории всегда равно 0 или 1. Индекс Z на линии, соединяющей

порождающую сущность с символом кластера, показывает, что порождающая сущность не обязана (хотя и может) иметь категории. Индекс Z на линии, соединяющей символ кластера с сущностью-категорией, показывает, что порождающая сущность может принадлежать одному из указанных типов.

На рис. 2.23 показан еще один категориальный кластер, состоящий из сущностей-категорий МЕНЕДЖЕР и ПЕРСОНАЛ. Этот кластер является неполным, на что указывает его символ, состоящий из одной горизонтальной линии вместо двух. Это означает, что по меньшей мере одна категория отсутствует. Опять-таки все кардинальности на схеме обозначены как Z.

Мы говорили, что сущности-категории, входящие в категориальный кластер, являются взаимоисключающими. Но это вовсе не означает, что сущность не может быть связана с двумя или более сущностями-категориями, принадлежащими к разным кластерам. Так, например, на *рис. 2.23* сотрудник может быть менеджером и одновременно программистом. При этом сотрудник не может в одно и то же время принадлежать и к менеджерам, и к персоналу.

Рассмотрим уровни представления модели БД, предлагаемые ERwin Data Modeler. Надо отметить, что данное разделение уровней не определяется в IDEF1X, но в целом характерно для CASE-средств проектирования БД, поддерживающих эту методологию.

В соответствии с идеологией нисходящего проектирования разработчики ERwin ввели несколько промежуточных уровней моделирования. Для логической модели определяются следующие формы представления.

- *Диаграмма сущность-связь* (*англ.* Entity-Relationship Diagram, ERD) включает основные сущности, выявленные в предметной области, и связи между ними. Атрибуты не указываются. Диаграммы не перегружены деталями и могут использоваться для обсуждения структуры проектируемой базы «в целом». Пример подобной диаграммы представлен на рис. 6.6.

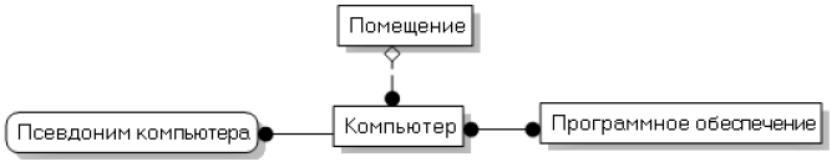


Рис. 6.6. Диаграмма сущность-связь

- *Модель базы данных, основанная на ключах* (*англ.* Key-Based) является более подробной, включает все сущности, их ключи, а также связи. В интерфейсе ERwin Data Modeler при работе с моделью можно переключиться на представление, использующее только первичные ключи (*англ.* Primary Key Display Level, рис. 6.7-а) или первичные и внешние ключи (*англ.* Keys Display Level, рис. 6.7-б).

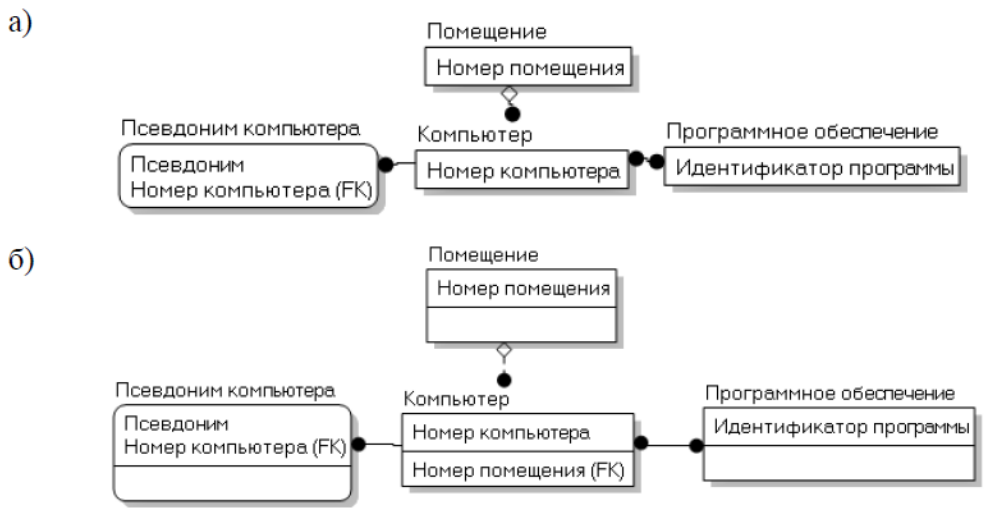


Рис. 6.7. Представления модели базы данных, основанной на ключах

- Нижний уровень для логической модели – *полная атрибутивная модель* (*англ.* Fully-Attributed, FA), включающая все сущности, их атрибуты и связи.

Структура проектируемой БД должна быть приведена в соответствие третьей нормальной форме.

***Домены***

Стандарт IDEF1X дополнил расширенную модель «сущность—связь» понятием домена. *Домен* — это именованный набор значений, которые может принимать атрибут. Домен может представлять собой фиксированный список значений, а может быть определен и более общим образом — например, как набор строк с заданной максимальной длиной.

В качестве примера первого подхода можно привести домен НАЗВАНИЯ_КАФЕДР, включающий названия всех официальных кафедр в неком университете. Определение этого домена выглядит как простое перечисление названий кафедр: {'Вычислительная техника', 'Информационные системы', 'Физика твердого тела', 'Неорганическая химия', 'Общая психология', 'Менеджмент'}. Примером второго подхода может служить домен ИМЕНА_СТУДЕНТОВ, который можно определить как строку длиной до 75 символов.

*Домены устраняют неоднозначность*

Например, обратите внимание, что на *рис. 2.23* как у сущности ПРОГРАММИСТ, так и у сущности ТЕХНИЧЕСКИЙ_ПИСАТЕЛЬ имеется атрибут под названием Язык. Без использования доменов имена этих атрибутов могут толковаться неоднозначно. Описывают ли они одно и то же, или нет? Возможно, для программиста Язык обозначает язык программирования, а для технического писателя — естественный язык. А может быть, в обоих случаях речь идет о языке

программирования. Эту неоднозначность можно устранить, указав домен, к которому должен принадлежать каждый из атрибутов.

Домен ЯЗЫК_ПРОГРАММИРОВАНИЯ можно определить списком {'C#', 'C++', 'Java', 'VisualBasic', 'VisualBasic.Net'}, а домен ЕСТЕСТВЕННЫЙ_ЯЗЫК - списком {'Русский', 'Французский', 'Испанский', 'Британский английский', 'Американский английский'}. Теперь, если постановить, что атрибут Язык сущности ПРОГРАММИСТ принадлежит к домену ЯЗЫК_ПРОГРАММИРОВАНИЯ, а атрибут Язык сущности ТЕХНИЧЕСКИЙ, ПИСАТЕЛЬ принадлежит к домену ЕСТЕСТВЕННЫЙ_ЯЗЫК, эти два атрибута невозможно будет перепутать.

*Домены полезны на практике*

Предположим, что в модели университетской базы данных у нас есть атрибут под названием МестныйАдрес, используемый во множестве различных сущностей. Это могут быть, например, сущности СТУДЕНТ, ПРЕПОДАВАТЕЛЬ, КАФЕДРА, ЛАБОРАТОРИЯ и так далее. Предположим также, что все атрибуты МестныйАдрес отнесены к одному и тому же домену МЕСТНЫЙ_АДРЕС, который определен как множество всех возможных значений вида BBB-NNN, где BBB — это список номеров корпусов, а NNN — список номеров комнат. Тогда все атрибуты МестныйАдрес унаследуют это определение.

Теперь допустим, что, построив нашу модель, мы обнаружили, что некоторые комнаты имеют четырехзначные номера. Если бы у нас не был определен соответствующий домен, нам бы пришлось найти в модели все атрибуты, использующие местный адрес, и поменять в них трехзначные номера на четырехзначные. Когда же есть домен, задача становится гораздо проще: стоит изменить определение домена, и все атрибуты, принадлежащие к этому домену, унаследуют данное изменение.

Есть еще одно полезное применение для доменов: они помогают определить, не описывают ли два атрибута с различными названиями одно и то же.

*Базовые и типовые домены*

В стандарте IDEF1X определены два типа доменов.

*Базовый домен* (base domain) — это домен, которому присвоен тип данных и, возможно, перечень значений или определение диапазона.

Стандартными типами данных являются Character (символьный тип), Numeric (числовой тип) и Boolean (логический тип). В спецификации доменов можно использовать дополнительные типы данных, такие как Date (дата), Time (время), Currency (валюта) и так далее. Список значений — это список, подобный тому, что мы задавали для домена ЯЗЫК_ПРОГРАММИРОВАНИЯ, а примером определения диапазона может служить определение домена ВЫСЛУГА.

*Типовой домен* (type domain) представляет собой подмножество значений базового домена или другого типового домена.

На *рис. 2.26* изображены типовые домены, основанные на базовом домене НАЗВАНИЯ_КАФЕДР. Типовые домены можно организовывать в иерархии, что позволяет достичь большей точности при определении атрибутов.

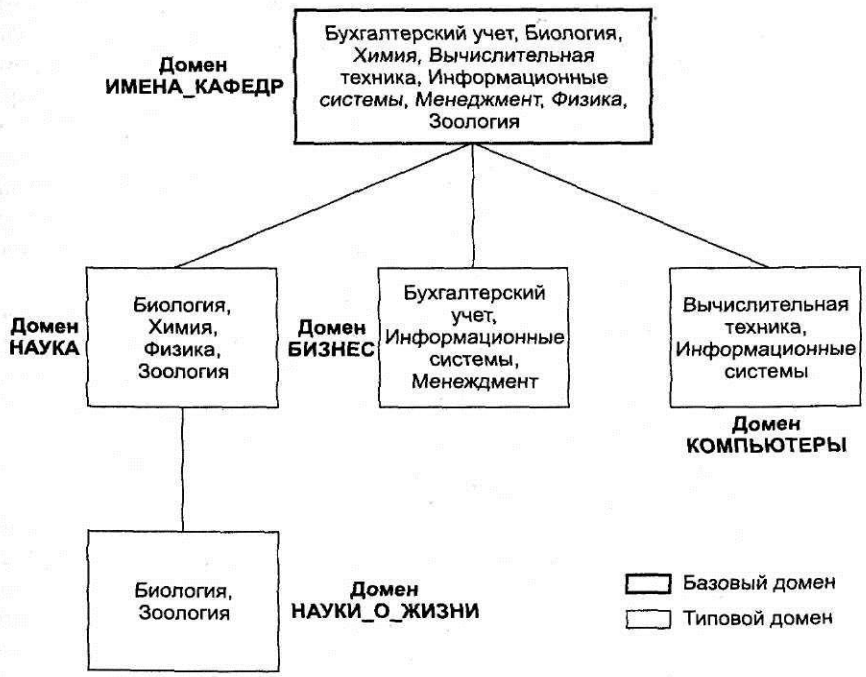


**Рис. 2.26.** Пример иерархии доменов

## 2. Этапы разработки базы данных
(дополнительный материал)

Целью разработки любой базы данных является хранение и использование информации о какой-либо предметной области. Для реализации этой цели имеются следующие инструменты:

1. Реляционная модель данных - удобный способ представления данных предметной области.
2. Язык SQL - универсальный способ манипулирования такими данными.

Однако очевидно, что для одной и той же предметной области реляционные отношения можно спроектировать множеством различных способов.

При разработке базы данных обычно выделяется несколько уровней моделирования, при помощи которых происходит переход от предметной области к конкретной реализации базы данных средствами конкретной СУБД. Можно выделить следующие уровни:

- Сама предметная область
- Модель предметной области
- Логическая модель данных

- Физическая модель данных
- Собственно база данных и приложения

***Предметная область*** - это часть реального мира, данные о которой мы хотим отразить в базе данных. Например, в качестве предметной области можно выбрать бухгалтерию какого-либо предприятия, отдел кадров, банк, магазин и т.д. Предметная область бесконечна и содержит как существенно важные понятия и данные, так и малозначащие или вообще не значащие данные. Так, если в качестве предметной области выбрать учет товаров на складе, то понятия "накладная" и "счет-фактура" являются существенно важными понятиями, а то, что сотрудница, принимающая накладные, имеет двоих детей - это для учета товаров неважно. Однако, с точки зрения отдела кадров данные о наличии детей являются существенно важными. Таким образом, важность данных зависит от выбора предметной области.

***Модель предметной области***. Модель предметной области - это наши знания о предметной области. Знания могут быть как в виде неформальных знаний в мозгу эксперта, так и выражены формально при помощи каких-либо средств. В качестве таких средств могут выступать текстовые описания предметной области, наборы должностных инструкций, правила ведения дел в компании и т.п. Опыт показывает, что текстовый способ представления модели предметной области крайне неэффективен. Гораздо более информативными и полезными при разработке баз данных являются описания предметной области, выполненные при помощи специализированных графических нотаций. Имеется большое количество методик описания предметной области. Из наиболее известных можно назвать методику структурного анализа SADT и основанную на нем IDEF0, диаграммы потоков данных Гейна-Сарсона, методику объектно-ориентированного анализа UML, и др. Модель предметной области описывает скорее процессы, происходящие в предметной области и данные, используемые этими процессами. От того, насколько правильно смоделирована предметная область, зависит успех дальнейшей разработки приложений.

***Логическая модель данных***. На следующем, более низком уровне находится логическая модель данных предметной области. Логическая модель описывает понятия предметной области, их взаимосвязь, а также ограничения на данные, налагаемые предметной областью. Примеры понятий - "сотрудник", "отдел", "проект", "зарплата". Примеры взаимосвязей между понятиями - "сотрудник числится ровно в одном отделе", "сотрудник может выполнять несколько проектов", "над одним проектом может работать несколько сотрудников". Примеры ограничений - "возраст сотрудника не менее 16 и не более 60 лет".

Логическая модель данных является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но *без*

*привязки к конкретной СУБД*. Более того, логическая модель данных необязательно должна быть выражена средствами именно *реляционной* модели данных. Основным средством разработки логической модели данных в настоящий момент являются различные варианты ***ER-диаграмм*** (***Entity-Relationship***, ***диаграммы сущность-связь***). Одну и ту же ER-модель можно преобразовать как в реляционную модель данных, так и в модель данных для иерархических и сетевых СУБД, или в постреляционную модель данных.

Решения, принятые на предыдущем уровне, при разработке модели предметной области, определяют некоторые границы, в пределах которых можно развивать логическую модель данных, в пределах же этих границ можно принимать различные решения. Например, модель предметной области складского учета содержит понятия "склад", "накладная", "товар". При разработке соответствующей реляционной модели эти термины обязательно должны быть использованы, но различных способов реализации тут много - можно создать одно отношение, в котором будут присутствовать в качестве атрибутов "склад", "накладная", "товар", а можно создать три отдельных отношения, по одному на каждое понятие.

При разработке логической модели данных возникают вопросы: хорошо ли спроектированы отношения? Правильно ли они отражают модель предметной области, а следовательно и саму предметную область?

***Физическая модель данных***. На еще более низком уровне находится физическая модель данных. Физическая модель данных описывает данные средствами конкретной СУБД. Мы будем считать, что физическая модель данных реализована средствами именно *реляционной* СУБД, хотя, как уже сказано выше, это необязательно. Отношения, разработанные на стадии формирования логической модели данных, преобразуются в таблицы, атрибуты становятся столбцами таблиц, для ключевых атрибутов создаются уникальные индексы, домены преображаются в типы данных, принятые в конкретной СУБД.

Ограничения, имеющиеся в логической модели данных, реализуются различными средствами СУБД, например, при помощи индексов, декларативных ограничений целостности, триггеров, хранимых процедур. При этом опять-таки решения, принятые на уровне логического моделирования определяют некоторые границы, в пределах которых можно развивать физическую модель данных. Точно также, в пределах этих границ можно принимать различные решения. Например, отношения, содержащиеся в логической модели данных, должны быть преобразованы в таблицы, но для каждой таблицы можно дополнительно объявить различные индексы, повышающие скорость обращения к данным. Многое тут зависит от конкретной СУБД.

При разработке физической модели данных возникают вопросы: хорошо ли спроектированы таблицы? Правильно ли выбраны индексы? Насколько много

программного кода в виде триггеров и хранимых процедур необходимо разработать для поддержания целостности данных?

***Собственно база данных и приложения***. И, наконец, как результат предыдущих этапов появляется собственно сама база данных. База данных реализована на конкретной программно-аппаратной основе, и выбор этой основы позволяет существенно повысить скорость работы с базой данных. Например, можно выбирать различные типы компьютеров, менять количество процессоров, объем оперативной памяти, дисковые подсистемы и т.п. Очень большое значение имеет также настройка СУБД в пределах выбранной программно-аппаратной платформы.

Но опять решения, принятые на предыдущем уровне - уровне физического проектирования, определяют границы, в пределах которых можно принимать решения по выбору программно-аппаратной платформы и настройки СУБД.

Таким образом ясно, что решения, принятые на каждом этапе моделирования и разработки базы данных, будут сказываться на дальнейших этапах. Поэтому особую роль играет принятие правильных решений *на ранних этапах моделирования*.

**Критерии оценки качества логической модели данных**

Цель данной главы - описать некоторые принципы построения *хороших логических моделей данных*. Хороших в том смысле, что решения, принятые в процессе логического проектирования приводили бы к хорошим физическим моделям и в конечном итоге к хорошей работе базы данных.

Для того чтобы оценить качество принимаемых решений на уровне логической модели данных, необходимо сформулировать некоторые критерии качества в терминах физической модели и конкретной реализации и посмотреть, как различные решения, принятые в процессе *логического* моделирования, влияют на качество *физической* модели и на скорость работы базы данных.

Конечно, таких критериев может быть очень много и выбор их в достаточной степени произволен. Мы рассмотрим некоторые из таких критериев, которые являются безусловно важными с точки зрения получения качественной базы данных:

- Адекватность базы данных предметной области
- Легкость разработки и сопровождения базы данных
- Скорость выполнения операций обновления данных (вставка, обновление, удаление кортежей)
- Скорость выполнения операций выборки данных

*Адекватность базы данных предметной области*

База данных должна адекватно отражать предметную область. Это означает, что должны выполняться следующие условия:

1. Состояние базы данных в каждый момент времени должно соответствовать состоянию предметной области.

2. Изменение состояния предметной области должно приводить к соответствующему изменению состояния базы данных
3. Ограничения предметной области, отраженные в модели предметной области, должны некоторым образом отражаться и учитываться базе данных.

*Легкость разработки и сопровождения базы данных*

Практически любая база данных, за исключением совершенно элементарных, содержит некоторое количество программного кода в виде триггеров и хранимых процедур.

***Хранимые процедуры*** - это процедуры и функции, хранящиеся непосредственно в базе данных в откомпилированном виде и которые могут запускаться пользователями или приложениями, работающими с базой данных. Хранимые процедуры обычно пишутся либо на специальном процедурном расширении языка SQL (например, PL/SQL для ORACLE или Transact-SQL для MS SQL Server), или на некотором универсальном языке программирования, например, C++, с включением в код операторов SQL в соответствии со специальными правилами такого включения. Основное назначение хранимых процедур - реализация бизнес-процессов предметной области.

***Триггеры*** - это хранимые процедуры, связанные с некоторыми событиями, происходящими во время работы базы данных. В качестве таких событий выступают операции вставки, обновления и удаления строк таблиц. Если в базе данных определен некоторый триггер, то он запускается *автоматически* всегда при возникновении события, с которым этот триггер связан. Очень важным является то, что пользователь не может обойти триггер. Триггер срабатывает независимо от того, кто из пользователей и каким способом инициировал событие, вызвавшее запуск триггера. Таким образом, основное назначение триггеров - автоматическая поддержка целостности базы данных. Триггеры могут быть как достаточно простыми, например, поддерживающими ссылочную целостность, так и довольно сложными, реализующими какие-либо сложные ограничения предметной области или сложные действия, которые должны произойти при наступлении некоторых событий. Например, с операцией вставки нового товара в накладную может быть связан триггер, который выполняет следующие действия - проверяет, есть ли необходимое количество товара, при наличии товара добавляет его в накладную и уменьшает данные о наличии товара на складе, при отсутствии товара формирует заказ на поставку недостающего товара и тут же посылает заказ по электронной почте поставщику.

Очевидно, что чем больше программного кода в виде триггеров и хранимых процедур содержит база данных, тем сложнее ее разработка и дальнейшее сопровождение.

*Скорость операций обновления данных (вставка, обновление, удаление)*

На уровне логического моделирования мы определяем реляционные отношения и атрибуты этих отношений. На этом уровне мы не можем определять какие-либо физические структуры хранения (индексы, хеширование и т.п.). Единственное, чем мы можем управлять - это распределением атрибутов по различным отношениям. Можно описать мало отношений с большим количеством атрибутов, или много отношений, каждое из которых содержит мало атрибутов. Таким образом, необходимо попытаться ответить на вопрос - влияет ли количество отношений и количество атрибутов в отношениях на скорость выполнения операций обновления данных. Такой вопрос, конечно, не является достаточно корректным, т.к. скорость выполнения операций с базой данных сильно зависит от физической реализации базы данных. Тем не менее, попытаемся *качественно* оценить это влияние при *одинаковых подходах к физическому моделированию*.

Основными операциями, изменяющими состояние базы данных, являются операции вставки, обновления и удаления записей. В базах данных, требующих постоянных изменений (складской учет, системы продаж билетов и т.п.) производительность определяется скоростью выполнения большого количества небольших операций вставки, обновления и удаления.

Рассмотрим операцию вставки записи в таблицу. Вставка записи производится в одну из свободных страниц памяти, выделенной для данной таблицы. СУБД постоянно хранит информацию о наличии и расположении свободных страниц. Если для таблицы не созданы индексы, то операция вставки выполняется фактически с одинаковой скоростью независимо от размера таблицы и от количества атрибутов в таблице. Если в таблице имеются индексы, то при выполнении операции вставки записи индексы должны быть перестроены. Таким образом, скорость выполнения операции вставки *уменьшается при увеличении количества индексов* у таблицы и *мало зависит от числа строк* в таблице.

Рассмотрим операции обновления и удаления записей из таблицы. Прежде, чем обновить или удалить запись, ее необходимо найти. Если таблица не индексирована, то единственным способом поиска является последовательное сканирование таблицы в поиске нужной записи. В этом случае, скорость операций обновления и удаления существенно увеличивается с увеличением количества записей в таблице и не зависит от количества атрибутов. Но на самом деле неиндексированные таблицы практически никогда не используются. Для каждой таблицы обычно объявляется один или несколько индексов, соответствующий потенциальным ключам. При помощи этих индексов поиск записи производится очень быстро и практически не зависит от количества строк и атрибутов в таблице (хотя, конечно, некоторая зависимость имеется). Если для таблицы объявлено несколько индексов, то при выполнении операций обновления и удаления эти индексы должны быть перестроены, на что тратится дополнительное время. Таким образом, скорость

выполнения операций обновления и удаления также *уменьшается при увеличении количества индексов* у таблицы и *мало зависит от числа строк* в таблице.

Можно предположить, что чем больше атрибутов имеет таблица, тем больше для нее будет объявлено индексов. Эта зависимость, конечно, не прямая, но *при одинаковых подходах* к физическому моделированию обычно так и происходит. Таким образом, можно принять допущение, что *чем больше атрибутов имеют отношения*, разработанные в ходе логического моделирования, *тем медленнее будут выполняться операции обновления данных*, за счет затраты времени на перестройку большего количества индексов.

*Скорость операций выборки данных*

Одно из назначений базы данных - предоставление информации пользователям. Информация извлекается из реляционной базы данных при помощи оператора SQL - SELECT. Одной из наиболее дорогостоящих операций при выполнении оператора SELECT является операция соединение таблиц. Таким образом, чем больше взаимосвязанных отношений было создано в ходе логического моделирования, тем больше вероятность того, что при выполнении запросов эти отношения будут соединяться, и, следовательно, тем медленнее будут выполняться запросы. Таким образом, увеличение количества отношений приводит к замедлению выполнения операций выборки данных, особенно, если запросы заранее неизвестны.