БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет радиофизики и электроники
Кафедра системного анализа

В. В. Скакун

Системы управления базами данных

Учебное пособие

Моделирование данных

Оглавление

Минск 2006

1. МОДЕЛИРОВАНИЕ ДАННЫХ

Существует множество моделей, отражающих различные аспекты реального мира: физические, позволяющие понять физические свойства; математические, представляющие собой абстрактное описание мира с помощью математических знаков; экономические, отображающие тенденции экономики и позволяющие получить прогноз ее развития. Самой обобщенной моделью реального мира может служить естественный язык. Но так как естественный язык пока малопригоден для компьютерной обработки преимущественно вследствие его многозначности, то нам потребуется построить такую модель, которая бы наиболее адекватно отображала некоторую часть реального мира (предметную область) и была пригодна для компьютерной обработки. Так как мы будем работать с данными, то нас будут интересовать модели данных.

1.1. Понятие модели

Модель данных – это средство абстракции, позволяющее отобразить информационное содержание данных. В самом простейшем виде модель данных может быть представлена простым перечнем той информации, которая должна храниться в информационной системе. Но так как данные, представляющие эту информацию, практически всегда имеют сложную структуру и тесно взаимосвязаны, то необходимо представить каким либо образом также и их структуру. Иначе говоря, модель данных представляет собой совокупность правил, что могут быть использованы для описания структуры данных. Моделей для отображения одной и той же информации можно придумать множество. Критерием выбора будет являться максимум количества информации, извлекаемой из данных. Чтобы понять процесс построения модели данных необходимо знать ряд терминов, которые применяются при описании и представлении данных.

Объектом называется элемент информационной системы, информацию о котором мы сохраняем. Объекты могут быть реальными и абстрактными, а также могут представлять некоторую концепцию или событие. Например, клиент некой фирмы, отделы той же фирмы, записи купли-продажи и т.д. Всякий объект мыслится посредством своих свойств и находится в некоторых отношениях с другими объектами. Классом (типом) объектов называют совокупность объектов, обладающих одинаковым набором свойств.

Выделенную по некоторым признакам совокупность объектов реального или абстрактного мира, или относящихся к какой-либо области знаний, или необходимых для достижения какой-либо цели называют предметной областью.

Данные имеют три области своего представления. Первая область (смотри рис. 1) – это реальный мир. Объекты и их свойства являются понятиями реального мира, в котором эти объекты существуют и имеют некоторые свойства. Во второй области, области информации (иначе области идей, которые существуют в голове програм-миста или разработчика), говорят об атрибутах объектов. Атрибут это информационное отражение свойств объекта. Каждый объект характеризуется рядом основных атрибутов. Например, человек может быть охарактеризован такими атрибутами, как ФИО, год рождения, номер паспорта, адрес и т.д. Третья область – это область сохраненных данных. Это может быть бумажным источником или представлять собой память компьютера, в которой используются строки символов или совокупность битов для кодирования элементов информации.

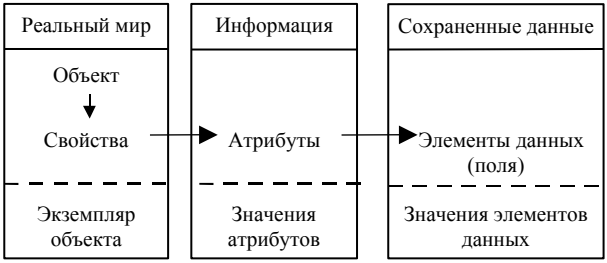


Рис. 1. Три области представления данных

Так как объектов некоторого типа может быть множество, то области представления данных в свою очередь могут быть разбиты на две подобласти. Первая дает общую характеристику объекта как некоторого типа в виде совокупности имен атрибутов (полей). Для нашего примера, объект «человек» будет иметь атрибуты ФИО, год рождения, номер паспорта и т.д. Вторая подобласть состоит из совокупностей значений атрибутов каждого экземпляра объекта в отдельности. Например, Петров (Петров Петр Петрович, 1966, V-НО №654723); Иванов (Иванов Иван Иванович, 1961, IV-ВЛ №865278) и т.д. Исходя из этого, дадим определение экземпляра объекта: экземпляром объекта называется единичный набор принимаемых элементами данных значений.

Элементы данных (атрибуты) могут быть простыми и составными, однозначными и многозначными, сохраняемыми и вычисляемыми или быть пустыми. Составные состоят (или могут быть разделены) из нескольких простых, например адрес. Обычно разложение происходит в иерархическом порядке. Такие составные элементы данных могут использоваться, если ссылки на них возможны только как на целое. Если же возможны ссылки на часть элемента данных, то их лучше разложить на простые. Многозначными называют элементы данных, которые могут принимать набор значений. Например, телефон и адрес клиента. Вычисляемыми (производными) называют элементы данных, которые могут быть вычислены на основе уже

существующих. Например, возраст человека легко может быть вычислен из текущей даты и дня рождения.

Специальное пустое значение (NULL) вводится для элементов данных которые не имеют значений или могут быть опущены. Различают следующие три случаи: мы не знаем, какое значение имеет элемент данных; элемент данных не может иметь значения в данном конкретном случае (например, ученую степень имеет смысл заполнять только для научных работников и преподавателей); мы знаем, что элемент данных имеет значение, но оно для нас безразлично или было пропущено при вводе.

Каждый атрибут связан с набором значений, называемым доменом. Домен определяет все потенциальные значения, которые могут быть присвоены атрибуту. Таким образом, домен – это набор значений, который может быть присвоен атрибуту. Например, домен «Имена» содержит все возможные имена людей, а домен «Номера телефонов» содержит все возможные номера телефонов. Для доменов могут быть приняты соглашения по написанию (формат), к примеру, для имен вначале должна быть фамилия, затем имя и отчество. Домены нельзя путать с атрибутами. Атрибуты представляют собой различные интерпретации домена. Например, атрибуты «Имя клиента», «Имя поставщика» берут свои значения из одного и того же домена «Имена». В принципе, домен это не что иное как тип данных. Его можно сравнить с такими типами данных в программировании, как перечисление *enum* в C и *type* в Pascal. Т.е. мы можем задать тип Days и затем им пользоваться: enum Days {Mon, Tue, Wed, Thu, Fri, Sat, Sun}.

Некоторые элементы данных обладают важным для построения информационной модели свойством. Если известно значение, какое принимает такой элемент данных некоторого объекта, то мы можем однозначно идентифицировать значения, принимаемые другими элементами данных того же объекта. Например, зная номер кузова автомобиля, мы вправе говорить о конкретном автомобиле, описывая его характеристики. Такой элемент данных, по которому можно однозначно идентифицировать остальные элементы данных называется ключевым элементом данных. Ключевой элемент данных может состоять из нескольких элементов данных. Тогда его называют составным или сцепленным. В общем случае элементов данных, по которым можно однозначно идентифицировать остальные элементы данных, может быть несколько. Тогда один из них выбирают в качестве первичного ключа, а остальные будут называться потенциальными или альтернативными ключами.

Минимальным фрагментов данных является элемент данных. В большинстве случаев сам по себе элемент данных информации не несет и приобретает смысл только в связи с другими элементами данных.

Совокупность значений связанных элементов данных называется записью данных. Так как некоторый объект может быть описан набором элементов данных, то между такими элементами существует естественная связь. В дальнейшем такие элементы данных мы будем всегда рассматривать в совокупности, и такой вид связи явно выделяться не будет.

Соответствующая модель объектов с составляющими их элементами данных и взаимосвязями называется концептуальной моделью. Так как такая модель позволяет наглядно отобразить структуру данных, то иначе ее называют схемой данных.

1.2. Типы связей

В общем случае связи подразделяются по количеству связываемых объектов. Можно выделить унарные, бинарные, тернарные и т.д. Унарная связь обычно смысла не имеет, кроме рекурсивной. Наибольший интерес представляют бинарные связи, так как связи более высоких порядков редко встречаются, труднореализуемы, нелегки для понимания и всегда могут быть разбиты на совокупность бинарных связей.

Первый тип связи – «один к одному» (1:1 или 1 – 1)

Эта связь означает, что одному экземпляру объекта A может соответствовать один и только один экземпляр связанного с ним объекта B. На практике такой вид связи встречается редко, так как почти всегда два связываемых объекта можно объединить в один. В качестве примера использования такой связи может послужить связь между персональной и служебной характеристикой работника некой фирмы. Поскольку доступ к служебной информации обычно ограничен, то защиту от несанкционированного доступа намного легче реализовать при раздельном хранении таких данных.

Второй тип связи – «один ко многим» (1:M или 1 – ∞)

Это означает, что одному экземпляру объекта A соответствует несколько (много) экземпляров объекта B. Например: сведения о клиентах – записи о покупках; сведения о товаре – записи наличия его на складе. Один клиент может сделать много покупок, а один и тот же товар может быть завезен многократно.

Третий тип связи – «много к одному» (M:1 или ∞ – 1)

Означает, что одному или нескольким экземплярам объекта A соответствует один экземпляр объекта B. Связи «один ко многим» и «много к одному» являются обратимыми, поэтому на практике обычно говорят о связи «один ко многим».

Четвертый тип связи – «многие ко многим» 
(M:N или ∞ – ∞)

Означает, что одному экземпляру объекта А соответствует несколько (много) экземпляров объекта В и наоборот, каждому экземпляру объекта В может соответствовать несколько экземпляров объекта А. Например, сведения об авторах и написанных ими книгах или отношение между клиентами и товарами. Книга может быть написана несколькими авторами и наоборот, один автор может написать много книг. Каждый товар может быть куплен несколькими клиентами и наоборот, один клиент может купить несколько товаров.

Наряду с взаимосвязями между объектами можно выделить зависимости между атрибутами объектов. Зависимости бывают функциональные, транзитивные и многозначные. Понятие функциональной зависимости является базовым, так как на его основе формулируются определения всех остальных видов зависимостей. Атрибут В функционально зависит от атрибута А, если каждому значению атрибута А соответствует одно и только одно значение атрибута В. Т.е. если нам известно значение атрибута А (его иначе называют детерминантом), то мы однозначно можем определить значение атрибута В. Однако данному значению атрибута В может соответствовать несколько различных значений атрибута А. Математически функциональная зависимость В от А обозначается записью А→В. Необходимо отметить, что А и В могут быть составными, т.е. состоять из двух и более атрибутов. Если же между атрибутами А и В существует функциональная зависимость вида А→В и В →А (т.е между А и В имеется взаимнооднозначное соответствие), то говорят о функциональной взаимозависимости (обозначается как А↔В или В↔А). Наличие функциональной взаимозависимости между атрибутами А и В приводит к образованию связи «один к одному» между этими атрибутами. Например, если в БД наряду с идентификационным номером клиента хранится и номер его паспорта. Функциональная зависимость может быть полной либо частичной. Частичной зависимостью (частичной функциональной зависимостью) называется зависимость атрибута В от части составного атрибута А. Полная функциональная зависимость определяется зависимостью атрибута В от всего составного атрибута А.

Атрибут С зависит от атрибута А транзитивно (существует транзитивная зависимость), если для атрибутов А, В, С выполняются условия А→В и В→С, но обратная зависимость отсутствует. Например, ФИО→Должность→Оклад.

Между атрибутами может иметь место также многозначная зависимость. Атрибут В многозначно зависит от атрибута А, если каждому значению А соответствует множество значений В. Многозначные

зависимости могут образовывать связи «один ко многим», «многие к одному» или «многие ко многим», обозначаемые соответственно A⇐B, A⇒B и A⇔B. Например, имя клиента и его идентификационный номер существуют совместно. Клиентов с одинаковыми именами может быть много, но все они имеют различные идентификационные номера (один ко многим). Или информация о клиентах и обслуживаемых их продавцах хранится совместно. Тогда несколько клиентов с одинаковыми именами могут быть обслужены разными продавцами и наоборот, несколько продавцов с одинаковыми именами могут получить заказы от нескольких клиентов (многие ко многим). Атрибуты могут быть и независимыми. Два или более атрибута будут взаимно независимыми, если ни один из этих атрибутов не является функционально зависимым от других.

Проектирование концептуальной модели можно провести на основе анализа существующих зависимостей между атрибутами. Понятно, что все атрибуты, характеризующиеся полной функциональной зависимостью от ключевого атрибута, наверняка будут атрибутами объекта одного типа. Те атрибуты, которые характеризуются транзитивной зависимостью от ключевого атрибута, необходимо выделить и сформировать отдельные объекты. Многозначные зависимости также требуют дополнительного разбиения по объектам. Практически процесс формирования объектов на основе анализа зависимостей необходимо продолжать до тех пор, пока все неключевые атрибуты будут характеризоваться только полной функциональной зависимостью от соответствующих ключевых атрибутов. На первом же шаге проектирования можно значительно сократить множество анализируемых атрибутов, исключив тривиальные зависимости, т.е. те зависимости, которые не могут не выполняться. Атрибуты, связанные тривиальной зависимостью, необходимо всегда рассматривать вместе. В качестве примера схематично представим концептуальную модель расписания вылета самолетов некоторого аэропорта (смотри рис. 2).

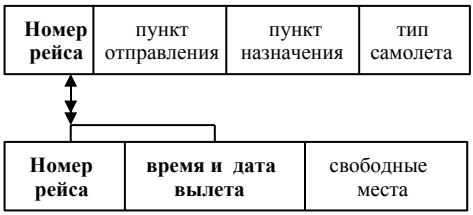


Рис. 2. Концептуальная модель расписания вылета самолетов

Существует большое количество видов концептуальных моделей, позволяющих отобразить семантику данных. Критериями выбора модели могут служить следующие характеристики:

- выразительность. Модель должна содержать достаточно средств для выражения типов, атрибутов, связей и ограничений;
- формализованность;

- простота и легкость восприятия. Модель должна восприниматься конечными пользователями;
- минимальность. Модель должна обладать минимальным (но достаточным) набором не перекрывающихся по смыслу концепций;
- представительность. Модель должна обладать выразительной диаграммной техникой представления.

Всеми этими характеристиками обладает модель сущность-связь, специально разработанная для концептуального моделирования реляционных БД. Большим достоинством ее также является возможность автоматизации процесса проектирования.

1.3. Модель сущность-связь

Модель сущность-связь (entity-relationship model или просто ER model) представляет собой высокоуровневую концептуальную модель данных, созданную Ченом (Chen) в 1976 году с целью упрощения задачи проектирования БД.

Основным объектом в ER модели является сущность, или объект (entity), которая представляет собой реальную вещь и может быть реально существующим (физическое существование) или абстрактным (концептуальное существование) объектом. Каждая сущность имеет набор атрибутов, представляющих ее свойства. Сущность имеет имя – тип, который представляет совокупность сущностей с одинаковым набором атрибутов. Таким образом, некоторая сущность определяется ее типом (именем) и набором атрибутов. В свою очередь, некоторый экземпляр сущности будет характеризоваться набором значений атрибутов. Различаются сильные и слабые типы сущностей. Слабый тип сущности определяется как тип, существование которого зависит от какого-то другого типа сущности, а сильный как тип, существование которого не зависит от всех других типов сущностей. Слабые сущности могут быть дочерними, подчиненными и зависимыми, а сильные – родительскими, владельцами или доминантными.

Между типами сущностей могут существовать связи. В общем случае связи могут быть унарными (рекурсивная), бинарными, тернарными и быть более высокого порядка (степень связи). Как и для сущностей, следует различать связи и типы связей. В свою очередь связи могут также иметь атрибуты. На связи накладываются ограничения по степени участия и по показателю кардинальности. Показатель кардинальности соответствует типу связи в ее обычной формулировке, т.е. 1:1, 1:M, M:1, M:N. Если ER модель составляется для некоторого предприятия (смотри пример на рис. 3), то показатели кардинальности будут прежде всего определяются

производственными правилами (бизнес правилами), установленными на данном предприятии.

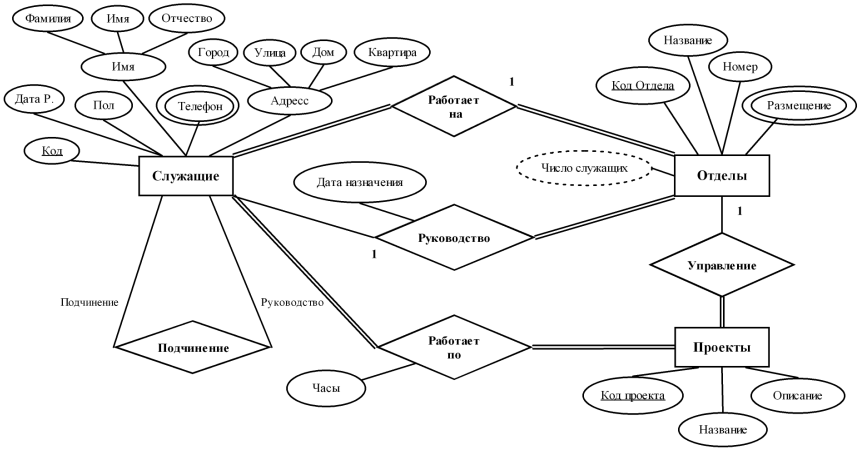


Рис. 3. Пример ER модели

Термин «бизнес-правило» получил широкое распространение, поскольку при моделировании любой предметной области важнейшей задачей является выделение и учет всех без исключения отношений между объектами (если, конечно, этого позволяет сама модель). Связям могут также присваиваться ролевые имена для однозначного определения назначения каждой связи. Если назначение каждой связи определено недвусмысленно, то ролевые имена не указываются. Степень участия может быть полной и частичной. Полной она будет в том случае, если для существования объекта, участвующего в связи, требуется существование другого объекта. Например, если правила предприятия требуют, чтобы любой служащий работал в некотором отделе, то экземпляр сущности «Служащий» будет существовать, только если он участвует в связи «Работает на» с сущностью «Отдел». Называется иначе зависимостью существования. В то же самое время только некоторые служащие могут руководить отделами. Значит участие сущности «Служащий» в связи «Руководство» будет только частичным.

Допустимо использование и альтернативного варианта обозначения структурных ограничений, накладываемых на связь в виде двух чисел (min, max), означающих минимальное и максимальное количество экземпляров сущности ($0 \le \min \le \max$ и $\max \ge 1$), участвующей в связи. Такой вид обозначения позволяет отобразить больше информации о степени участия. min = 0 означает частичное участие, min > 0 – полное, а max = N указывает на отсутствие ограничений со стороны максимума. Например, надпись (0, 1)

над связью «Руководство» означает, что любой служащий не обязательно должен руководить некоторым отделом, да и руководить одновременно двумя отделами он тоже не может.

Достоинством этой модели является наличие удобной и выразительной диаграммной техники построения модели (смотри рис. 4). На диаграмме каждый тип объекта показан в виде отдельного прямоугольника с его именем внутри, причем слабые типы объектов изображаются в двойной рамке. Атрибуты показаны в виде эллипсов с названием атрибута, соединенных сплошной линией с соответствующим объектом или типом связи. Эллипс обведен штриховой линией, если атрибут производный, и двойной линией, если атрибут многозначный. Если атрибут составной, то составляющие его атрибуты показаны в виде других эллипсов, соединенных с эллипсом составного атрибута с помощью дополнительных линий. Ключевые атрибуты, как правило, подчеркиваются, а множества значений не показываются вовсе.

Каждый тип связи показан в виде ромба с названием связи внутри. Ромб окружен двойной линией, если связь задана между слабым типом объекта и типом объекта, от существования которого находится в зависимости слабый тип объекта. Участники каждой связи присоединены к соответствующему типу связи сплошными линиями. Каждая такая линия содержит надпись "1" "M" или "N" для обозначения типа связи. Двойная линия обозначает полное участие.

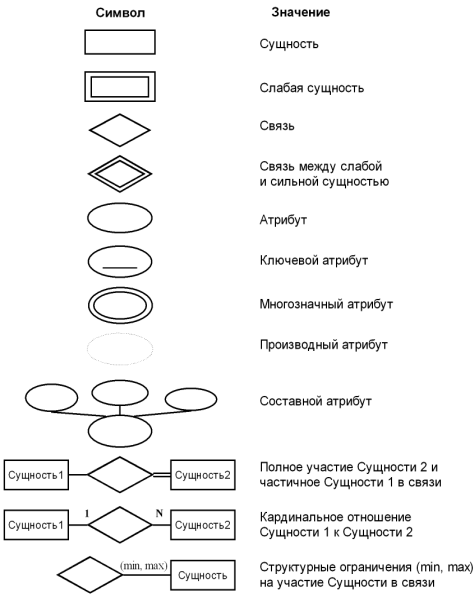


Рис. 4 Система обозначений, принятая в ER модели

К недостаткам этой модели можно отнести возможность появления «ловушек» при недостаточном определении связей между объектами. К таким ловушкам можно отнести ловушки разветвления и разрыва. Ловушка разветвления имеет место в том случае, когда модель отображает связи между типами сущностей, но путь между отдельными сущностями этих типов указан не

однозначно. Например, ловушка разветвления может возникать, когда две и более связей «один ко многим» разветвляются из одной сущности, как показано на рис. 5. В данном примере схема данных, представленная слева, не позволяет однозначно ответить на вопрос, к каким группам относятся студенты. Устранить такую ловушку можно с помощью реорганизации связей между типами сущностей – так, как показано на схеме справа.

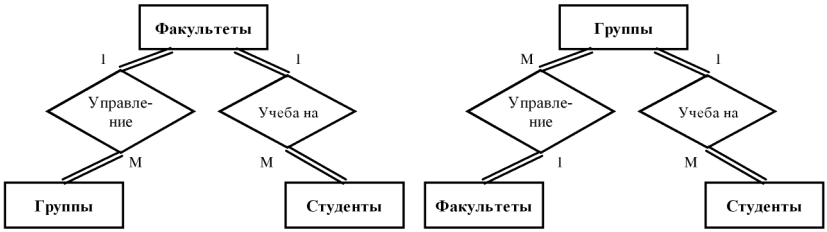


Рис. 5. Пример возникновения ловушки разветвления

Ловушка разрыва появляется в том случае, когда в модели предполагается наличие связи между типами сущностей, но не существует пути между отдельными сущностями этих типов. Ловушка разрыва может возникать между тремя и более сущностями, связанными между собой отношениями «один ко многим» при наличии связи с частичным участием. Пример представлен на рис. 6.

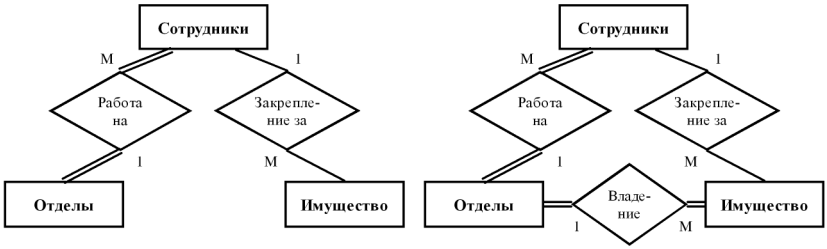


Рис. 6. Пример возникновения ловушки разрыва

Из схемы, представленной на слева следует, что в отделе работает несколько сотрудников и за ними закреплено определенное имущество. Но не за каждым сотрудником должно быть закреплено какое-то имущество и не все имущество обязательно должно быть закреплено за сотрудниками в данный момент времени. Такая схема не позволяет ответить на вопрос, какое имущество (закрепленное или не закрепленное) числится за отделом. Такая ловушка устраняется введением дополнительной связи между типами сущностей, как показано на схеме справа.

Контрольные вопросы и задания

1. Дайте определение модели данных.
2. Отобразите схематично три области представления данных и на основе рисунка дайте определения базовым понятиям моделирования данных: объекту, предметной области, атрибуту данных, домену и записи данных.
3. Перечислите виды атрибутов данных.
4. Что такое ключевой элемент данных.
5. Дайте определение концептуальной модели данных.
6. Перечислите типы связей и дайте их определения.
7. Перечислите виды зависимостей между атрибутами объекта.
8. Приведите пример частичной и полной функциональной зависимости.
9. Приведите пример транзитивной зависимости.
10. Дайте определение ER модели.
11. Назовите основные достоинства и недостатки ER модели.
12. Что такое слабый тип сущности.
13. Какие ограничения накладываются на связи между типами сущностей.
14. Разработайте ER модель расписания вылета самолетов, представленной на рис. 2.
15. Что такое ловушки разрыва и разветвления и когда они возникают. Дайте пример для каждой из ловушек.

2. МОДЕЛИ ДАННЫХ

Хранимые в базе данные имеют определенную логическую структуру, т.е описываются некоторой моделью представления данных (в дальнейшем просто моделью данных). К числу классических относятся иерархическая, сетевая и реляционная. В последние годы появились и стали внедряться на практике постреляционная, многомерная и объектно-ориентированная модели. Разрабатываются также всевозможные системы, основанные на моделях данных, расширяющих известные модели. Например, объектно-реляционная модель данных.

2.1. Иерархическая модель

Исторически первой была разработана и воплощена иерархическая модель данных. В 1960 году была разработана БД для большой ЭВМ IBM System 360. Первой коммерческой СУБД явилась СУБД IMS фирмы IBM. Среди отечественных можно назвать СУБД ОКА.

Иерархическая модель данных представляется связным графом типа дерева, вершины (типы) которого расположены на разных иерархических уровнях (рис. 7). При этом одна вершина, расположенная на самом верху дерева, называется корнем и не подчиняется ни одной вершине, а все остальные вершины связаны с одной и только одной вершиной, расположенной на более высоком уровне. Элементы, расположенные в конце ветви, т.е. не имеющие порожденных, называются листьями. Потомки, одного и того же типа называют близнецами. Элементы могут быть простыми, а также представлять из себя записи. Можно привести достаточно много примеров, когда логическая структура данных некоторой предметной области идеально описывается иерархической моделью данных, например, родословная при указании одного родителя.

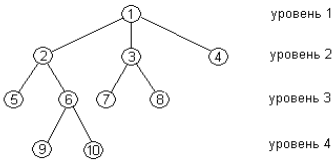


Рис. 7. Иерархическая модель данных

Иерархическая БД представляет собой упорядоченную совокупность экземпляров типа «дерево», содержащих в свою очередь совокупность экземпляров типа «запись». Обход всех элементов иерархической БД обычно производится сверху вниз и слева направо. Иерархической модели присущи связи «один к одному» и «один ко многим». Связь «многие ко многим» не может быть реализована, так как вершина не может иметь более одного родителя. Давайте представим расписание занятий на факультете с помощью иерархической модели (рис. 8). Видим, что иерархической модели

данных присуще дублирование информации. В нашем примере для каждой группы будут дублироваться записи расписания. При изменении расписания общих курсов придется менять записи расписания для всех групп. К тому же отсутствует связь расписания с преподавателями и преподавателей со студентами.

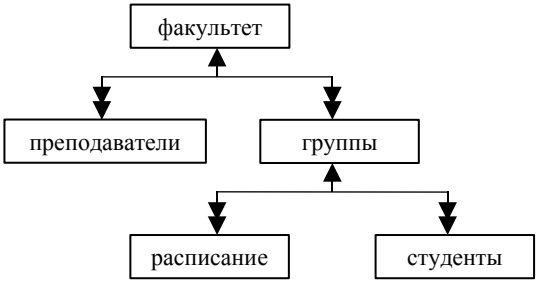


Рис. 8. Иерархическая модель расписания занятий

К достоинствам иерархической модели данных относятся эффективность использования памяти ЭВМ и неплохие показатели времени выполнения основных операций с данными. К недостаткам можно отнести:

- невозможность установления связи M:M;
- сильную логическую зависимость данных. Добавление или удаление вершины невозможно без разрушения всей структуры данных;
- избыточность информации;
- затрудненный поиск «снизу вверх». Это приводит к простому перебору всех значений в БД.

2.2. Сетевая модель данных

Впервые сетевые СУБД появились в 1970 году с появлением миникомпьютеров DEC. Это IDMS (Computer Association), VAX_DBMS (Digital), db_VistaIII и наша СЕТЬ.

Если в отношении между данными порожденный элемент имеет более одного исходного элемента, то такое отношение уже нельзя описать как древовидное. Его описывают в виде сетевой структуры. Сетевая модель данных позволяет отображать взаимосвязи элементов данных в виде произвольного графа, обобщая тем самым иерархическую модель данных (рис. 9). В сетевой модели данные представлены в виде записей и связей. Запись может иметь множество как подчиненных ей записей, так и записей, которым она подчинена. Такая сеть позволяет реализовывать отношение M:M на всех связях. С некоторой избыточностью сетевые модели всегда можно разложить на несколько древовидных, вводя дополнительные вершины. Рассмотрев обратный переход к сетевой модели легко заметить,

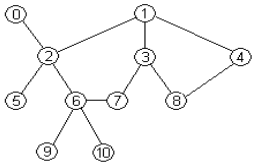


Рис. 9. Сетевая модель данных

что дублирование информации сильно уменьшается. К другим достоинствам этой модели данных можно отнести большую гибкость системы. В ней можно использовать не только предопределенные связи, а также и добавлять новые (наряду с узлами) в процессе работы. Разрешены также симметричные запросы (сверху вниз и снизу вверх), выполняющиеся по схожим алгоритмам. Сетевые модели получили достаточно широкое распространение. Результатом явился стандарт CODASYL. Основными языками программирования для создания сетевых, а также иерархических, БД можно считать PL/1 и COBOL.

К недостаткам сетевой модели данных можно отнести:

- недостаточную скорость выполнения поиска;
- некоторую избыточность информации (для каждой записи дублируются ссылки на подчиненные и родительские элементы;
- громоздкое и труднообозримое представление данных;
- ослабление контроля целостности вследствие допустимости установления произвольных связей между записями.

На рис. 10 представлена сетевая модель расписания занятий. Здесь связь типа M:M между группами и преподавателями реализовано в виде записей Пересечения «расписание», указывающее день, время и номер аудитории. Между студентами и преподавателями также можно установить связь типа M:M с помощью записей пересечения «задание», например индивидуальные задания, курсовые работы и т.п.

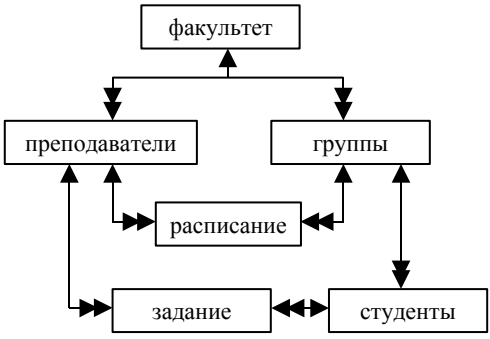


Рис. 10. Сетевая модель расписания занятий

2.3. Реляционная модель

2.3.1. Основные понятия

В реляционной модели данных информация хранится в одной или нескольких связанных таблицах. Отдельная таблица обычно представляет совокупность (группу) либо реальных объектов, либо некоторых абстрактных концепций, либо событий одного типа. Каждая запись в таблице идентифицирует один объект группы. Таблица состоит из строк и столбцов, называемых записями и полями соответственно. Таблицы обладают следующими свойствами:

1. Каждый элемент таблицы представляет собой один элемент данных, т.е. совокупность значений в одном столбце одной строки недопустима;
2. Все столбцы в таблице однородные. Это означает, что элементы столбца имеют одинаковую природу. Столбцам присвоены имена;
3. В таблице нет двух одинаковых строк;
4. Порядок размещения строк и столбцов в таблице может быть произвольным. В операциях с такой таблицей ее строки и столбцы могут просматриваться в любом порядке безотносительно к их информационному содержанию и смыслу.

Таблицы, обладающие такими свойствами, являются точным прообразом математического двумерного множества – отношения (relation). Но эти два понятия не эквивалентны. Отношение – это абстрактный математический объект, а таблица – это конкретное изображение этого абстрактного объекта. Различие проявляется в их свойствах. В отношении строки и столбцы не могут быть упорядочены, а в таблице строки упорядочены сверху вниз, столбцы слева направо. В таблице могут повторяться строки, а в отношении нет. В реляционной модели каждая строка таблицы уникальна. Это обеспечивается применением ключей, которые содержат одно или несколько полей таблицы. Ключи хранятся в упорядоченном виде, обеспечивающем прямой доступ к записям таблицы во время поиска. Связь между таблицами осуществляется посредством значений одного или нескольких совпадающих полей (преимущественно ключевых).

Приведем ряд терминов, применяющихся в реляционной модели:

- ***Отношением*** (relation) называется двумерное множество – таблица, удовлетворяющая вышеперечисленным требованиям;
- ***Атрибут*** – это свойство, характеризующие объект. В структуре таблицы каждый атрибут именуется и ему соответствует заголовок

некоторого столбца таблицы. Количество атрибутов называется ***степенью отношения***;

- ***Кортежом*** (tuple) называется строка таблицы. В общем случае кортежи представляют собой набор пар <атрибут>, <значение>. Каждое значение должно быть атомарным, т.е. не может быть многозначным или составным. Следовательно, многозначные и составные атрибуты в реляционной модели не поддерживаются. Количество кортежей называется ***кардинальным числом***;
- ***Домен*** представляет собой множество всех возможных значений определенного атрибута отношения.
- ***Первичным ключом*** называется атрибут отношения, однозначно идентифицирующий каждый из его кортежей. Ключ может быть составным (сложным), т. е. состоять из нескольких атрибутов.
- ***Потенциальный ключ*** – это подмножество атрибутов отношения, обладающего следующими свойствами:
  - Свойством уникальности. Нет одинаковых кортежей с теми же значениями потенциальных ключей;
  - Свойством неизбыточности. Никакое из подмножеств потенциального ключа не обладает свойством уникальности.

Каждое отношение обязательно имеет комбинацию атрибутов, которая может служить ключом. Его существование гарантируется тем, что отношение – это математическое множество, которое не может содержать одинаковых кортежей, т.е. по крайней мере вся совокупность атрибутов обладает свойством однозначной идентификации кортежей отношения. Возможны случаи, когда отношение имеет несколько комбинаций атрибутов, каждая из которых однозначно определяет все кортежи отношения. Все эти комбинации атрибутов являются потенциальными или возможными ключами отношения. Один потенциальный ключ выбирается в качестве первичного, остальные будут называться вторичными (альтернативными). Могут быть даже такие ситуации, когда любой из потенциальных ключей может быть выбран в качестве первичного. Примером может служить таблица Менделеева, содержащая поля *Имя*, *Символ* и *Атомное число*. Потенциальные ключи имеют очень большое значение в реляционной теории. Они служат для адресации кортежей. Указав значение потенциального ключа мы гарантированно получим не более одного кортежа. Для отношений, связанных с другими «базовыми» отношениями, существуют еще внешние ключи, использующиеся для установления связи.

- ***Внешний ключ*** – это такой атрибут подчиненного отношения, который используется для установления связи с базовым отношением. Он

содержит значения, всегда совпадающие с некоторыми значениями потенциального ключа базового отношения.

Исходя их вышеприведенных понятий, математически отношение можно описать следующим образом. Пусть даны *n* множеств *Dl, D2, D3,..., Dn*, тогда отношение *R* есть множество упорядоченных кортежей <***d1, d2, d3 ,..., dn>,*** где ***dk*** ∈ ***Dk, dk*** – атрибут***,*** а ***Dk*** – домен отношения *R.*

В середине 70-х годов инженером IBM Коддом (Codd) была предложена модель данных, основанная на математических операциях исчисления отношений и реляционной алгебре. Основной структурной единицей этой модели являлось отношение (relation). Поэтому такая модель данных получила название реляционной. Коддом был также разработан язык манипулирования данных, представленных в виде отношений. Он предложил два эквивалентных между собой по своим выразительным возможностям варианта языка манипулирования данными:

1. Реляционная алгебра. Это процедурный язык, так как отношение, являющееся результатом запроса к реляционной БД, вычисляется при выполнении последовательности реляционных операторов, применяемых к отношениям. Операторы состоят из операндов, в роли которых выступают отношения, и реляционных операций. Результатом реляционной операции является отношение. Операции можно разделить на две группы. Первую группу составляют операции над множествами, к которым относятся операции объединения, пересечения, разности, деления и декартова произведения. Вторую группу составляют специальные операции над отношениями: проекция, выборка и соединение.
2. Реляционное исчисление. Это непроцедурный язык описательного или декларативного характера, содержащий лишь информацию о желаемом результате. Процесс получения этого результата скрыт от пользователя. К языкам такого типа относятся SQL и QBE. Первый основан на реляционном исчислении кортежей, второй – на реляционном исчислении доменов.

С помощью этих языков можно извлекать подмножество столбцов и строк таблицы, создавая таблицы меньшей размерности, а также объединять связанные данные из нескольких таблиц, создавая при этом таблицы большей размерности. Результат каждой (реляционной) операции над отношениями является также отношением. Это реляционное свойство получило название **свойства замкнутости**. Следовательно, различные пользователи могут выделять в реляционной БД различные наборы данных и связей между ними. Этот способ представления данных наиболее естественен и обозрим для конечного пользователя. Реляционная модель

данных очень гибка, поскольку любое представление данных с некоторой избыточностью можно свести к двумерным таблицам.

2.3.2. Связи в реляционной БД

Отношение между объектами определяет тип связи между таблицами. Поддерживаются связи четырех типов: «один к одному», «один ко многим», «много к одному» и «многие ко многим». Рассмотрим подробнее типы связей в применении к реляционной модели данных:

- **«один к одному»**

Связь «один к одному» означает, кто каждой записи из первой таблицы соответствует одна и только одна запись из другой таблицы. Рассмотрим таблицы, содержащие персональные и служебные сведения о работниках некоторой фирмы.

Персональные сведения

Код_пс	№ паспорта	Фамилия	Имя
1	АВ 2358955	Сидоров	Сергей
2	DB 2456886	Петров	Петр
3	МА 8654212	Иванов	Иван

Служебные сведения

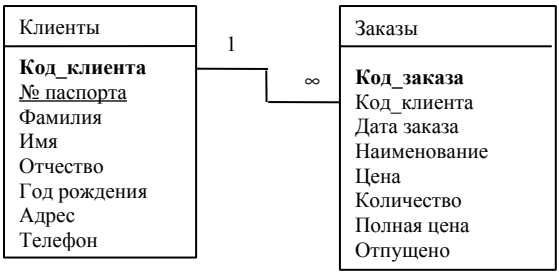
Код_сс	Место работы	Должность	Разряд
1	Бухгалтерия	Бухгалтер	3
2	Цех № 2	Токарь	7
3	Лабаратория	Начальник отдела	

Между таблицами «Персональные сведения» и «Служебные сведения» существует связь «один к одному», поскольку для одного человека, работающего в определенной фирме, может существовать только одна запись о его служебном положении. Табельные номера «Код_пс» и «Код_сс» были добавлены для однозначной идентификации записей. Эти же поля и приняты в качестве первичных ключей. Связь между этими таблицами поддерживается при помощи совпадающих значений полей

«Код_пс» и «Код_сс». Легко убедится, что между двумя ключевыми полями может существовать только связь «один к одному», поскольку любые дублирования одного и того же табельного номера исключены с обеих сторон.

- **«один ко многим» и «много к одному»**

Связь «один ко многим» означает, кто каждой записи из первой таблицы может соответствовать одна либо много записей из другой таблицы. Рассмотрим таблицы, содержащие сведения о клиентах некоторой фирмы и сделанных ими заказах.



Клиенты

Код_клиента	№ паспорта	Фамилия	Имя
1	AB 2358955	Сидоров	Сергей
2	DB 2456886	Петров	Петр
3	MA 8654212	Иванов	Иван

Заказы

Код_заказа	Код_клиента	Дата заказа	Наименование
1	1	05.03.2000	Пылесос Вихрь-150
2	2	15.04.2000	Холодильник Минск 131
3	2	21.04.2000	Телевизор Горизонт ТЦ 655
4	1	02.10.2000	Стол компьютерный
5	3	12.12.2000	Кресло «Президент»

Предполагается, что один и тот же клиент может сделать несколько заказов. Таким образом, между этими таблицами существует связь «один ко многим». Для установления связи необходимо в таблицу «Заказы» ввести поле «Код_клиента», которое будет являться внешним ключом для таблицы «Заказы». Связь между таблицами будет осуществляться на основании значений полей «Клиенты.Код_клиента» и «Заказы.Код_клиента». Причем подчеркнем, что связь устанавливается на основе значений совпадающих полей, а не их наименований. Таким образом, если связь устанавливается между ключевым полем одной таблицы и неключевым полем второй таблицы, то это будет связь типа «один ко многим».

- **«много ко многим»**

Связь «много ко многим» возникает между двумя таблицами в тех случаях, когда каждой записи из первой таблицы может соответствовать одна либо много записей из второй таблицы, и наоборот, одной записи из второй таблицы может соответствовать одна либо много записей из первой таблицы.

Типичным примером является связь между клиентами некой фирмы и покупаемыми ими товарами. Каждый товар может быть куплен несколькими клиентами и наоборот, один клиент может купить несколько товаров. Такая связь может быть реализована с помощью дополнительной таблицы, содержащей ключевые поля обеих таблиц, участвующих в связи (являющимися для нее внешними ключами).

Клиенты

Код_клиента	№ паспорта	Фамилия	Имя
1	AB 2358955	Сидоров	Сергей
2	DB 2456886	Петров	Петр
3	MA 8654212	Иванов	Иван

Товары

Код_товара	Наименование	Цена
1236	Пылесос Вихрь-150	88999
5543	Холодильник Минск 131	310056
445	Телевизор Горизонт ТЦ 655	260324

Заказы

Код_заказа	Код_клиента	Код_товара	Дата заказа
1	1	1236	05.03.2000
2	2	5543	15.04.2000
3	2	445	21.04.2000
4	1	3245	02.10.2000
5	3	1236	12.12.2000
6	3	445	30.12.2000

Связь «много ко многим» также автоматически возникает между таблицами, связанными посредством неключевых полей. Пусть первая

таблица содержит информацию о том, на каких станках могут работать рабочие некоторой бригады. Вторая таблица содержит сведения о том, кто из бригады ремонтников какие станки обслуживает.

Работа

Работает	**Станок**
Иванов А. В.	станок 1
Иванов А.В.	станок 2
Петров Н.Г.	станок 1
Петров Н.Г.	станок 3
Сидоров В. К.	станок 2

Обслуживание

Обслуживает	**Станок**
Голубев Б. С.	станок 1
Голубев Б. С.	станок 2
Зыков А.Ф.	станок 2
Зыков А.Ф.	станок 3

Первой и третьей записям таблицы «Работа» соответствует первая запись таблицы «Обслуживание». Четвертой записи таблицы «Работа» соответствуют вторая и четвертая записи таблицы «Обслуживание». Такой вид связи «много ко многим» характеризуется как слабый вид связи или даже как отсутствие связи, поскольку никакого контроля за целостностью данных в этом случае не производится.

Информация, размещенная в связанных таблицах, может быть легко объединена с помощью естественного соединения (по значениям внешних ключей). Для нашего примера будем иметь:

Заказы

Фамилия	**Имя**	**Наименование**	**Цена**	**Дата заказа**
Сидоров	Сергей	Пылесос Вихрь-150	88999	05.03.2000
Петров	Петр	Холодильник Минск 131	310056	15.04.2000
Петров	Петр	Телевизор Горизонт ТЦ 655	260324	21.04.2000
Сидоров	Сергей	Электрочайник BOSCH	50346	02.10.2000
Иванов	Иван	Пылесос Вихрь-150	88999	12.12.2000
Иванов	Иван	Телевизор Горизонт ТЦ 655	260324	30.12.2000

Кроме вышеупомянутых связей в реляционной модели возможны также рекурсивные связи. Предположим, мы должны хранить информацию о сотрудниках некоторой компании с указанием отношений подчиненности между отдельными сотрудниками. Поскольку один экземпляр объекта «Сотрудники» ссылается на другой экземпляр того же объекта, который в свою очередь может ссылаться на третий экземпляр, то связь будет унарной и рекурсивной. Данная связь легко реализуется путем введения внешнего ключа, ссылающегося на первичный ключ той же таблицы.

2.3.3. Целостность в реляционной БД

Рассмотрим подробнее ограничители целостности в применении к реляционной модели данных. Понятие ссылочной целостности неразрывно связано с понятием внешних ключей и формулируется для реляционной БД следующим образом: реляционная БД не должна содержать

несогласованных значений внешних ключей, т.е. таких значений, для которых не существует отвечающих им значений соответствующих потенциальных ключей базовых таблиц. Иначе говоря, внешние ключи могут содержать только те значения, которые принимают соответствующие им ключевые (в общем случае потенциальные) поля базовых таблиц. Если в БД уже есть клиент с номером 10 и товар с номером 25, то только тогда можно внести запись о покупке клиентом 10 товара 25 в таблицу «Заказы». С другой стороны нельзя удалить клиента с номером 25, или даже изменить его номер, если в БД хранится хотя бы одна запись о сделанном им заказе. В противном случае согласованность внешних ключей тоже нарушается, так как запись в таблице «Заказы» будет содержать значение, указывающее на несуществующего клиента. Заметим, что внешний ключ может также содержать значения первичного ключа своего же отношения для обеспечения рекурсивной связи. Ограничители ссылочной целостности могут быть схематично отображены с помощью линий со стрелками, направленных от внешних ключей к потенциальным ключам базовых отношений.

Есть два правила внешних ключей, обеспечивающих согласованное состояние БД. Первое заключается в ограничении обновления и удаления. При работе с базовой таблицей это правило запрещает удаление записей, которые содержат связанные записи в подчиненных таблицах и изменение значений их первичных ключей. При работе с подчиненной таблицей это правило запрещает установку внешнего ключа в значение, которого не существует для соответствующего потенциального ключа базовой таблицы. Второе из них заключается в каскадном обновлении внешних ключей и каскадном удалении записей подчиненных таблиц при операциях с базовой таблицей. При изменении значения базового ключа, использующегося в связи, все значения внешних ключей всех подчиненных таблиц должны быть изменены соответственно (каскадное обновление). При удалении записи из базовой таблицы все связанные с ней записи из подчиненных таблиц должны быть также удалены (каскадное удаление). Если для внешних ключей разрешены *Null* значения, то в некоторых реляционных СУБД вместо каскадного удаления может применяется операция *Nullify*, которая устанавливает внешние ключи подчиненных таблиц в *Null* при удалении соответствующего значения базового ключа.

Кроме ссылочной целостности можно сформулировать еще три ограничителя целостности для реляционной БД:

- ограничители домена (domain constraints), запрещающие ввод значений атрибутов, не принадлежащих домену. К ним можно отнести также все ограничители значений (тип, формат, задание списка и диапазона значений);

- ограничители ключей (key constraints), обеспечивающие уникальность значений потенциальных ключей (обеспечивается применением уникальных индексов);
- ограничители записи (entity constraints), запрещающие *Null* значения для первичных ключей, так как в противном случае будет нарушено требование идентифицируемости записи.

2.4. Нормализация данных в реляционной модели

Один и тот же набор данных в реляционной модели можно представить различными способами. Процесс нормализации данных позволяет решить вопрос о наиболее эффективной структуре данных, обладающей минимальной избыточностью. Это позволяет решить следующие задачи:

a) исключить ненужное повторение данных,

b) обеспечить быстрый доступ к данным,

c) обеспечить целостность данных.

Обычно говорят только об уменьшении избыточности данных и нормализацией называют процесс уменьшения избыточности информации в реляционной БД.

Теория нормализации оперирует с пятью нормальными формами и одной промежуточной, уточняющей третью. На практике же обычно руководствуются только первыми тремя нормальными формами. Процесс проектирования БД с использованием нормальных форм является итерационным и заключается в последовательном применении правил нормальных форм от низшей к высшей. Каждая следующая нормальная форма уменьшает избыточность данных и сохраняет свойства всех предыдущих форм. Перевод отношения в следующую нормальную форму осуществляется методом «декомпозиции без потерь», т.е. разбиением исходной таблицы на несколько связанных. Такая декомпозиция должна обеспечить равенство результатов выборок из исходного отношения и выборок, основанных на совокупности полученных отношений, т.е. говоря математическим языком, должно соблюдаться следующее правило:

$\bigcup_{i=1}^{m} R_i = R$, где $R = \{A_1, A_2, \ldots, A_n\}$ - исходное отношение, а $D = \{R_1, R_2, \ldots, R_m\}$ - декомпозиция, т.е. множество нормализованных отношений. При декомпозиции следует не только опасаться потери информации, но также избегать и возможности появления так называемых подложных записей, которые могут появиться вследствие проведения объединений (например, такие записи могут появляться при объединении таблиц посредством не ключевых полей). И наконец, при установлении

зависимостей между таблицами следует придерживаться связей, существующих между реальными объектами.

Рассмотрим ненормализованную таблицу «Продажи», которая будет содержать сведения о покупателях, сведения о проданных товарах и оформленных заказах:

Продажи

№	Наименование
1	Фамилия
2	Имя
3	Отчество
4	Паспорт
5	Телефон
6	Адрес
7	Кредит
8	Примечание

№	Наименование
9	Дата_заказа
10	Наим_товара
11	Категория
12	Размещение
13	Цена
14	Количество
15	Полная_цена
16	Прим_заказ

Таблицу «Продажи» можно рассматривать как однотабличную БД. Основная проблема состоит в том, что в ней содержится значительное количество повторяющейся информации. Например, сведения о каждом покупателе повторяются для каждого сделанного им заказа. Наличие повторяющейся информации ведет к возможной потере согласованности данных, т.е. к появлению аномалий обновления, неоправданному увеличению размера базы данных, и, как следствие, к снижению скорости поиска и выполнения запросов. Аномалии обновления можно условно разбить на три вида:

a) аномалии добавления. Возникают в случаях, когда информацию в таблицу нельзя поместить до тех пор, пока она не полная. Например, мы хотим добавить информацию о новом товаре, пришедшем на склад и у которого пока еще нет покупателей. Нам придется оставить незаполненными все поля, касающиеся клиентов и оформления заказа. Но если некоторые поля были объявлены обязательными, то либо в них придется ввести что-нибудь, либо вообще отказаться от записи. К тому же, после первой же покупки этого товара необходимость в этой записи вообще исчезнет;

b) аномалии удаления. Состоят в том, что при удалении некоторой информации может пропасть и другая информация, не связанная напрямую с удаляемой. Например, при удалении последней записи о продаже некоторого товара сведения об этом товаре будут утеряны, хотя это никак не будет означать, что данного товара больше нет на складе;

c) аномалии модификации. Проявляются в том, что изменение значения некоторого поля в одной записи может привести к просмотру всей

таблицы и соответствующему изменению других записей таблицы. Например, такая аномалия возникнет при обновлении не всех записей о покупках некоторого клиента при смене адреса этого клиента. Причем простая автозамена здесь не сможет помочь во всех случаях, тат как адрес может быть записан разным образом.

2.4.1. Первая нормальная форма

Первая нормальная форма требует соответствия исходной таблицы требованиям, предъявляемым к отношениям, т.е.:

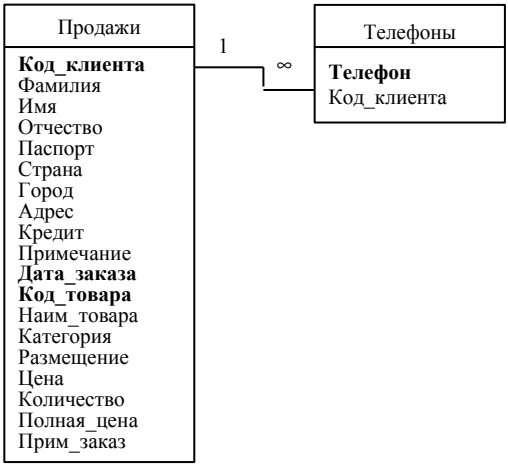
1. Таблица не должна иметь повторяющихся записей;
2. Все атрибуты должны быть простыми (скалярными);

Первое требование легко решается введением ключевого поля, однозначно определяющего остальные поля таблицы. Разумеется, можно использовать одно из полей, имеющееся в таблице. К примеру, номер паспорта. Но паспорт легко сменить, сменив фамилию. В таком случае рекомендуется ввести так называемый табельный номер, основным назначением которого будет являться идентификация некоторой записи. Если этот номер не имеет практического смысла, то его можно скрыть от пользователей. Ввод табельного номера рекомендуется в следующих случаях:

- в предметной области наблюдается синомия, т.е. ключевой атрибут не является уникальным (например, фамилия);
- если естественный атрибут может меняться со временем;
- если данный атрибут участвует во многих связях. Так как связь осуществляется посредством ключевых полей, то во всех подчиненных таблицах придется повторить значение ключевого поля. При применении не числовых и достаточно громоздких полей скорость выполнения операций доступа и поиска будет существенно падать. Лучшим решением является применение целочисленных значений.

Второе требование постулирует, чтобы в каждой ячейке было представлено одно значение, отнюдь не массив или перечисление. Составные поля должны быть разложены на простые, а многозначные – вынесены в отдельные таблицы. В нашем случае поле «Адрес» должно быть разложено на поля «Страна», «Город» и «Адрес», чтобы сохранить возможность поиска и сортировки по данным полям. Поле «Телефон» является многозначным и, следовательно, должно быть выделено в отдельную таблицу.

После ввода составного ключа «Код_клиента», «Код_товара» и «Дата_заказа» таблица «Продажи» будет находиться в первой нормальной форме. Получим:



2.4.2. Вторая нормальная форма

Таблица находится во второй нормальной форме, если:

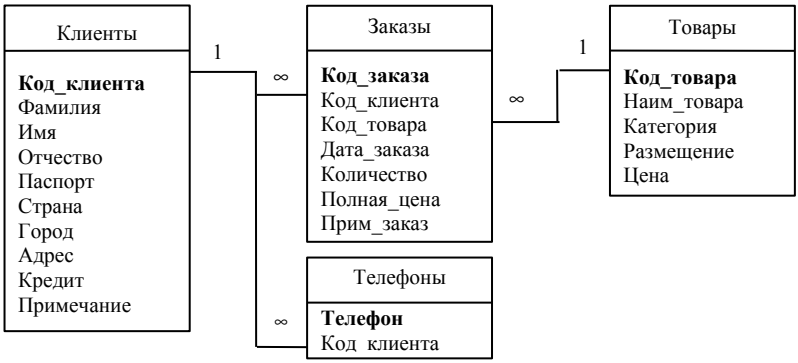
1. она удовлетворяет условиям первой нормальной формы и
2. любое поле, не входящее в ключ, должно однозначно идентифицироваться значением первичного ключа. Если первичный ключ является составным, то остальные поля должны зависеть от полного выражения ключа, а не от его части (неключевые атрибуты должны характеризоваться полной функциональной зависимостью от первичного ключа).

Легко заметить, что атрибуты «Фамилия», «Имя», «Отчество», «Паспорт», «Страна», «Город», «Адрес», «Предприятие», «Руководитель», «Кредит» и «Примечание» зависят только от части составного ключа – поля «Код_клиента», а атрибуты «Наим_товара», «Категория» и «Цена» зависят только от поля «Код_товара». Следовательно, они должны быть вынесены в две отдельные таблицы.

Иначе говоря, требование 2 также постулирует устранение повторяющейся информации для группы полей. Такие повторяющиеся группы полей обычно соответствуют некоторым реальным объектам, информацию о которых мы сохраняем. Следовательно, таблица должна содержать данные, относящиеся только к одному объекту. Поскольку здесь явно выделяются объекты «покупатели», «товары» и «заказы», нам необходимы три таблицы. Записи одной таблицы будут содержать сведения об покупателях, второй – информацию о товарах, третей – информацию о каждом из заказов. Вначале разобьем таблицу «Продажи» на две отдельные

таблицы («Клиенты» и «Заказы») и определим поле «Код_клиента» в качестве совпадающего поля для связывания таблиц.

Аналогично выделим из таблицы «Заказы» таблицу «Товары», которая будет содержать информацию о товарах каждого типа. Для связывания таблиц «Заказы» и «Товары» будем использовать поле "Код_товара". Между таблицами «Клиенты» – «Заказы» и «Товары» – «Заказы» будут отношения один-ко-многим. В таблице «Заказы» в качестве ключевого поля можно выбрать поля «Код_клиента», «Код_товара» и «Дата_заказа», а также ввести табельный номер «Код_заказа». Получим:



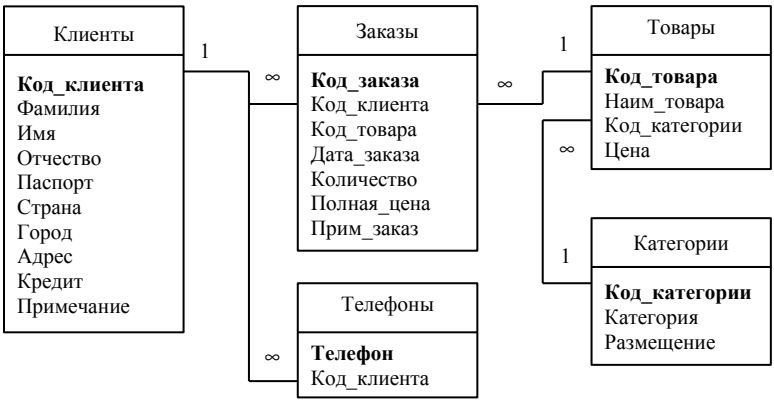
2.4.3. Третья нормальная форма

Таблица находится в третьей нормальной форме, если:

1. она удовлетворяет условиям второй нормальной формы и,
2. ни одно из полей таблицы, не входящих в ключ, не должно идентифицироваться с помощью другого поля, не входящего в ключ (иначе, отношение не должно иметь неключевых атрибутов, которые бы находились в транзитивной функциональной зависимости от первичного ключа).

Сведение таблицы к третьей нормальной форме предполагает разделение таблицы с целью помещения в отдельную таблицу (или несколько таблиц) столбцов, которые не зависят напрямую от выражения первичного ключа. В результате такого разбиения каждое из полей, не входящих в первичный ключ, должно оказаться независимым от какого-либо другого не ключевого поля. Обратимся к таблице «Товары». Поле «Размещение» этой таблицы содержит описание места размещения товара на складе, которое однозначно определяется значением поля «Категория». Поскольку поле «Размещение», не входящее в индекс, однозначно определяется другим неключевым полем «Категория», таблица «Товары» не является таблицей в третьей нормальной форме. Для приведения этой

таблицы к третьей нормальной форме создадим новую таблицу Категории. Получим:



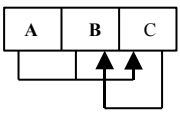
Третья нормальная форма также постулирует отсутствие полей, которые могут быть вычислены на основе других. Если полная цена может быть вычислена на основе полей «Цена» и «Количество», то его лучше исключить из таблицы и вычислять динамически. Но в нашем случае это поле содержит отпускную цену, которая может включать скидки и налоги и может не зависеть напрямую от цены товара.

На этом процесс нормализации обычно заканчивается. Если же необходима более детальная проработка данных, то дополнительно рассматривают требования нормальных форм более высокого порядка.

2.4.4. Нормальная форма Бойса – Кодда (Boyce-Codd)

Третья нормальная форма постулирует отсутствие зависимости неключевых атрибутов от других неключевых атрибутов, забывая о том, что в ряде случаев может наблюдаться зависимость части ключевого составного атрибута от некоторого неключевого. Если в таблице имеется такая зависимость, то необходимо перейти к усиленной третьей нормальной форме или форме Бойса – Кодда (BCNF). Таблица находится в форме Бойса – Кодда, если она:

1. удовлетворяет требованиям третьей нормальной формы и
2. в ней отсутствуют зависимости атрибутов составного ключа от не ключевых атрибутов.



Этот вид зависимости схематично отображен на рисунке. Не ключевой атрибут C функционально зависит от составного ключа A+B. Атрибут B, в свою очередь, функционально зависит от

атрибута C. Такая взаимозависимость требует дальнейшей декомпозиции таблицы.

2.4.5. Нормальные формы более высоких порядков

Четвертая нормальная форма устраняет нетривиальные многозначные зависимости. Если таблица содержит два и более никак не связанных между собой многозначных атрибута, то по требованиям первой нормальной формы нам потребуется записать по отдельной записи для каждого значения каждого многозначного атрибута. Общее число записей будет равно числу сочетаний значений, принимаемыми многозначными атрибутами, и может быть очень большим. Легко увидеть, что часть информации будет повторяться. Следовательно, для уменьшения избыточности информации и устранения возможности появления несогласованного состояния данных необходимо прибегнуть к дальнейшей декомпозиции отношения.

Пятая нормальная форма основывается на концепции устранения зависимостей объединения. Может быть ситуация, когда нельзя разбить таблицу на две таким образом, чтобы получить после объединения исходный результат (некоторые записи могут быть потеряны или наоборот, могут появиться подложные записи). Но при этом существует ситуация, что такую таблицу можно разбить на три и более таблиц при выполнении всех привил нормализации. На практике этот вид зависимости очень трудно обнаружить и, следовательно, устранить.

2.4.6. Правила нормализации

Резюмируя вышесказанное, сформулируем несколько правил, которых следует придерживаться при нормализации таблиц:

1. Разрабатывайте схему данных таким образом, чтобы можно было бы легко объяснить ее, т.е. не комбинируйте атрибуты независимых объектов и не создавайте сложные связи;
2. Разрабатывайте схему данных таким образом, чтобы исключить возможность появления аномалий обновления;
3. Разрабатывайте схему данных таким образом, чтобы в связях участвовали только первичные (можно допустить потенциальные) и внешние ключи. Это позволит избежать появления подложных записей;

Требование, чтобы все таблицы были нормализованы, не является обязательным. В нашем случае вполне можно было бы обойтись без таблицы «Телефоны». В заключение сформулируем определение реляционной базы данных как совокупность нормализованных отношений.

2.4.7. Выводы

Резюмирую вышесказанное, выделим основные свойства реляционной модели, выделяющие ее из остальных моделей данных:

a) Основной единицей обработки данных является отношение. Данные хранятся в связанных отношениях, любая выборка из БД представляет собой отношение, пусть она даже будет единственным числом;
b) Данные представляются одним и только одним способом, а именно явными скалярными значениями. Даже связи задаются явными значениями, а не указателями;
c) Нет указателей на отдельные строки таблицы. Строка идентифицируется по явному значению ключевого поля.

Ключи играют огромную роль в реляционных БД и используются для достижения следующих целей:

1. исключения дублирования кортежей и, как следствие, адресации кортежей;
2. организации связывания отношений.

Достоинства реляционной модели заключаются в следующем:

1. Простота. Использование таблиц очень наглядно и легко постигаемо;
2. Гибкость. Один и тот же набор данных можно представить различными способами;
3. Точность. Действия с данными основаны на точных математических операциях;
4. Связность. Реляционное преставление дает ясную картину связей;
5. Логическая независимость данных. Можно добавлять новые таблицы, поля, и связи, не разрушая старых. Можно удалять таблицы, связи и поля, не участвующие в связи и не являющиеся ключевыми.

Среди недостатков можно выделить следующие:

1. Однородная структура данных, и как следствие, невозможность хранения объектов со сложной структурой в одном кортеже отношения. Для хранения таких объектов приходится выделять несколько связанных отношений, что приводит к потере реальности восприятия сохраненных объектов;
2. Отсутствие стандартных средств идентификации отдельных записей и, как следствие, возможность повторного добавления одного и того же объекта;
3. Запрещение многозначных и составных полей;
4. Сложность описания рекурсивных, иерархических и сетевых связей;

5. Фиксированный и ограниченный набор операций (нет возможности определения новых операций.

Примерами реляционных СУБД для ПЭВМ являются следующие: Paradox (Borland) и Access (Microsoft), DB2 (IBM), MS SQL Server (Microsoft), Ingres (ASK Computer Systems) и Oracle (Oracle). Первым коммерческим продуктом стал Oracle (Oracle) в 1978 году.

2.5. Постреляционная модель

Классическая реляционная модель предполагает неделимость данных, хранящихся в полях записей таблиц. Существует ряд случаев, когда это ограничение мешает эффективной реализации приложений.

Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в ячейках таблицы. Поскольку атомарность данных постулируется первой нормальной формой, то исходя из этого определения, ее называют также Non-First Normal Form (NFNF или NF^2) реляционной моделью. Постреляционная модель данных допускает многозначные и составные поля. Набор значений многозначных полей считается самостоятельной таблицей, встроенной в основную таблицу. Вследствие этого ее еще часто называют вложенной (Nested) реляционной моделью. Все атрибуты в постреляционной модели рассматриваются как составные, имеющие иерархическую структуру, причем на количество ветвей не накладывается каких либо ограничений. Это позволяет на одном уровне иерархии хранить целые массивы данных, как показано ниже в таблице.

ID	Имя			Адрес					Тел	
	Фамилия	Имя	Отчество	Страна	Город	Доп. адрес			Номер	Тип
						Улица	Д.	Кв.		
1	Сидоров	Петр	Петрович	Беларусь	Минск	Ландера	6	67	2786544	Дом.
						Коржа	33	23	2367678	Раб.
									6257533	Моб.
2	Иванов	Иван	Иванович	Беларусь	Гродно	Ленина	23	12	244466	Дом.
				Литва	Вильнюс	Лесная	43	65	567543	Дом.

По сравнению с реляционной в постреляционной модели данные хранятся более эффективно, а при обработке не требуется выполнять операцию соединения данных из двух и более таблиц. Помимо обеспечения вложенности полей постреляционная модель поддерживает ассоциированные многозначные поля (множественные группы). Совокупность ассоциированных полей называется ассоциацией. При этом в некоторой строке первое значение одного столбца ассоциации соответствует первым значениям всех остальных столбцов ассоциации. Аналогичным образом связаны все вторые значения столбцов и т. д. На

длину полей и количество полей в записях таблицы не накладывается требование постоянства. Это означает, что структура данных и таблиц имеют большую гибкость. Но поскольку постреляционная модель допускает хранение в таблицах ненормализованных данных, возникает проблема обеспечения целостности и непротиворечивости данных. Эта проблема решается с помощью процедур, автоматически выполняемых до или после обращения к данным.

Достоинством постреляционной модели является возможность представления совокупности связанных реляционных таблиц одной постреляционной таблицей. Это обеспечивает высокую наглядность представления информации и повышение эффективности ее обработки. Недостатком постреляционной модели является сложность решения проблемы обеспечения целостности и непротиворечивости хранимых данных. Эта модель пока не реализована в полном объеме в коммерческих СУБД. Примером использования некоторых концепций этой модели может служить Oracle 8. В этой СУБД реализована поддержка составных и многозначных атрибутов, а также вложенных таблиц (Nested Tables). Для хранения составных атрибутов в Oracle 8 используется тип данных OBJECT. Например, составной атрибут «Телефон» описывается так:

CREATE TYPE Tel_Type AS OBJECT(Tel_Number CHAR(10), Tel_Descr CHAR(5));

Для хранения многозначных атрибутов в Oracle 8 используется тип данных VARRAY, представляющий собой массив некоторых объектов переменной длинны, имеющий два свойства: (1) Count – число элементов в массиве и (2) Limit – максимальное число элементов в массиве (задается при определении этого типа). Для формирования массива объектов вначале требуется определить тип объектов, которые будут храниться в массиве, например:

CREATE TYPE Tel_Type AS OBJECT(Tel_Number CHAR(15), Tel_Descr CHAR(5));

Затем необходимо определить тип массива, указав его максимальную размерность: CREATE TYPE Tel_List_Type AS VARRAY(5) OF Tel_Type;

Этот же набор данных можно определить и как вложенную таблицу:

CREATE TYPE Tel_List_Type AS TABLE OF Tel_Type;

Оба объекта VARRAY и TABLE имеют похожие свойства, но имеют и существенные различия. Во первых, тип VARRAY имеет верхний предел число элементов, а TABLE – не имеет. Во вторых, во вложенных таблицах можно обращаться к отдельным ее элементам и дополнительно определять индексы, что не допустимо в VARRAY.

2.6. Многомерная модель

Многомерный подход к представлению данных в базе появился практически одновременно с реляционным, но реально работающих многомерных СУБД (МСУБД) до недавнего времени было очень мало. С середины 90-х годов с развитием систем OLAP (On Line Analytical Processing — оперативная аналитическая обработка данных), Data Warehousing (хранилища данных) и Data Mining (добыча данных), применяющихся для сложного анализа данных, принятия решений, менеджмента и получения новых знаний, интерес к ним стал приобретать массовый характер. Основное назначение OLAP систем заключается в динамическом синтезе, анализе и консолидации больших объемов многомерных данных, а это, в свою очередь, связано с необходимостью извлекать большое количество записей из больших БД и производить их оперативную обработку (например, быстро вычислять итоговые значения). Если дать самую краткую характеристику Data Warehousing, то ее можно определить как предметно (проблемно) ориентированную, интегрированную, постоянную в структуре, но зависящую от времени совокупность данных, предназначенных для принятия некоторых решений. Главное отличие от стандартных баз данных состоит в том, что оно представляется собранием данных из многих баз данных, объединенных вместе для решения какой-либо частной задачи, либо является построенной по определенным правилам выдержкой из очень большой базы данных. Существенным отличием можно считать также отсутствие операций обновления данных.

Data Mining также является системой, хотя и основанной на данных, до весьма далекую от целей хранения данных. Ее главная цель состоит в исследовании данных с целью получения новых знаний, т.е. для сложного анализа данных, как например, поиск цепочек ДНК и т. д. Data Mining преследует следующие цели:

- предсказание (например, прогноз погоды);
- идентификация (по образцу, …);
- классификация;
- оптимизация (создание более оптимально действующих систем на основе анализа уже существующих).

Многомерные СУБД являются узкоспециализированными СУБД, предназначенными для интерактивной аналитической обработки информации. Многомерность модели данных в них означает не многомерность визуализации цифровых данных, а многомерное логическое представление структуры информации при описании и в операциях манипулирования данными. При этом зачастую создается только

многомерное логическое представление данных, а не их хранение. Т.е. сами данные могут храниться в больших БД, например, реляционных, а в требуемый момент времени извлекаться оттуда. Обычно в этом случае доступ производится с помощью многомерных индексов. Поскольку многомерные модели предназначены в основном для аналитической обработки данных, а не для создания удобных хранилищ информации, основной задачей в них является операциями консолидации, т.е. расчет всех промежуточных и основных итоговых значений, причем по всем размерностям. Это позволяет в много раз уменьшить объем представляемой информации и значительно ускорить доступ к данным и их последующую обработку. Такое предварительное обобщение является особенно ценным при работе с иерархически связанной информацией, например, при работе с данными, зависящими от времени.

По сравнению с реляционной моделью многомерная модель данных обладает более высокой скоростью доступа к данным, наглядностью и информативностью. Для сравнения ниже приведены реляционное (таблица 1) и многомерное (таблица 2 и рис. 11) представления одних и тех же данных о продаже товаров:

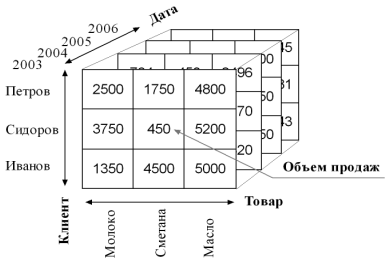


Рис 11. Пример трехмерной модели

Таблица 1

Клиент	Товар	Цена	Количество	Дата
Иванов	Масло	1000,00	1	02.02.2005
Петров	Молоко	400,00	2	02.02.2005
Иванов	Сметана	750,00	2	06.02.2005
Иванов	Масло	1000,00	1	22.02.2005
Петров	Молоко	400,00	3	03.03.2005
Сидоров	Масло	1000,00	2	02.03.2005
Иванов	Сметана	1500,00	2	12.03.2005
Иванов	Молоко	500,00	1	05.04.2005
Сидоров	Сметана	800,00	3	14.04.2005
Петров	Масло	1200,00	2	28.04.2005

Таблица 2

Дата / Товар	Июнь 2005	Июль 2005	Август 2005
Масло	2000	2000	2400
Молоко	800	1200	500
Сметана	1500	3000	2400

Если размерностей всего две (таблица 2), то такое представление называется перекрестной или сводной таблицей. Объем продаж вычисляется на основе группированного представления продаж по товарам, клиентам и месяцам. На рис. 11 показан пример трехмерной модели данных, где введена еще одна размерность *Клиент*. Если число размерностей велико, то информация не обязательно должна представляется визуально в виде многомерных объектов (n-мерных гиперкубов), тем более, что это не представляется возможным. Пользователь привык иметь дело с двумерными таблицами, диаграммами, графиками. Многомерная модель как раз предоставляет расширенный набор средств для выборки данных из многомерного хранилища данных, выполненных с разной степенью детализации.

Основными понятиями многомерных моделей являются измерение и ячейка, а операциями – консолидация, срез, вращение, агрегация и детализация:

Измерение (Dimension) – это множество однотипных данных, образующих одну из граней гиперкуба. В многомерной модели данных измерения играют роль индексов, служащих для идентификации конкретных значений в ячейках гиперкуба;

Ячейка (Cell) или показатель – это поле, значение которого однозначно определяется фиксированным набором измерений. В зависимости от того, как формируются значения некоторой ячейки, обычно она может быть переменной (значения изменяются и могут быть загружены из внешнего источника данных или сформированы программно) либо формулой (значения, подобно формульным ячейкам электронных таблиц, вычисляются по заранее заданным формулам);

Консолидация включает в себя простые обобщающие операции типа расчета итоговых значений, матричную арифметику, а также расчет сложных выражений;

Срез (Slice) представляет собой подмножество гиперкуба, полученное в результате фиксации одного или нескольких измерений. Формирование среза выполняется для ограничения выборки данных, поскольку все значения гиперкуба очень трудно представить визуально.

Вращение (Rotate) заключается в изменении порядка измерений при визуальном представлении данных.

Агрегация (Drill Up) и Детализация (Drill Down) означают соответственно переход к более общему и к более детальному представлению информации пользователю из гиперкуба. Например, в БД хранится информация о дате продажи товара. Тогда с помощью операции детализации можно уточнить момент продажи, а с помощью операции агрегации представить суммарную информацию по месяцам, кварталам и годам.

Основным достоинством многомерной модели данных является удобство и эффективность аналитической обработки больших объемов данных, связанных со временем. При организации обработки аналогичных данных на основе реляционной модели происходит нелинейный рост трудоемкости операций в зависимости от размерности БД и существенное увеличение затрат оперативной памяти на индексацию. Недостатком многомерной модели данных является ее громоздкость для простейших задач обычной оперативной обработки информации.

Примерами систем, поддерживающими многомерные модели данных, являются Essbase (Arbor Software), Media Multi-matrix (Speedware), Oracle Express Server (Oracle), Cache (Inter Systems). В последней СУБД, основанной на многомерной модели данных, реализованы три способа доступа к данным: прямой (на уровне узлов многомерных массивов), объектный и реляционный.

2.7. Объектно-ориентированная модель

Основное назначение объектно-ориентированных БД (ООБД) состоит в длительном хранении структур данных и объектов программ. Эта задача появилась очень давно, так как для достаточно большой программы фактически всегда актуальной задачей являлось сохранение и восстановление промежуточных результатов работы. Обычно для этой цели использовались файлы и потоковый ввод-вывод. Но в этом случае главный недостаток файловой системы хранения данных, заключающийся в невозможности использования сохраненных данных другими программами, являлся серьезным препятствием при разработке больших программных комплексов, состоящих из многих взаимодействующих приложений. Приходилось для каждой программы писать специальные конверторы объектов в формат, пригодный для совместного хранения и общего использования. И наоборот, чтение данных в программу подразумевало написание конвертора сохраненных в файле объектов обратно в переменные оперативной памяти компьютера. С появлением баз данных для хранения общих данных стали использовать технологию баз данных. Но это не решило проблемы, так как приходилось конвертировать объекты в формат БД и наоборот. По разным оценкам, на это уходило до 30% времени выполнения программы.

Успех объектно-ориентированного стиля программирования (ООП), наращивание объема программ, необходимость создания очень сложных программных комплексов, состоящих из многих компонентов, привело к созданию стандартов совместного хранения и использования объектов, что в свою очередь, привело к созданию объектно-ориентированной модели данных (ООМД). С другой стороны, оказалось очень удобным использовать

классы как определенные пользователем инкапсулированные типы объектов произвольной внутренней сложности для хранения данных. При этом подходе появилась возможность представления и реализации сложных моделей данных, так как не было необходимости учитывать многочисленные ограничения, накладываемые моделью данных, например, требование атомарности атрибутов в реляционной модели. Несомненным достоинством ООМД является также способность к хранению объектов произвольной сложности, отражающих сложность реальных объектов. Для хранения подобного объекта в реляционной БД потребовалось бы множество записей в нескольких таблицах, что привело бы к потере соответствия между реальным объектом и сохраненным.

ООБД преимущественно используются вместе с объектно-ориентированными программами и служат для длительного хранения их объектов. Такие программы специально разрабатываются для совместной работы с ООБД. В обычных объектно-ориентированных программах объекты существуют в оперативной памяти компьютера в течение времени выполнения программы и соответственно называются временными. ООБД расширяют время существования объектов так, чтобы они были доступными длительное время и после завершения работы программы. Другими словами, ООБД позволяет сохранять области оперативной памяти, представляющие собой копии объектов, во внешней памяти. С другой стороны ООБД обеспечивает возможность восстановления сохраненных объектов в оперативной памяти и обеспечение доступа к ним другим программам. Такая возможность длительного хранения объектов требует включения механизмов поиска, индексации, обеспечения целостности и безопасности информации, принятые в базах данных.

Структура ООМД графически представима в виде дерева, узлами которого являются объекты, а соединительные линии предоставляют связи объектов. Говорить о строгой иерархической структуре нельзя, так как объекты могут наследовать свойства нескольких базовых объектов. Несмотря на внешнюю схожесть логической структуры, ООМД коренным образом отличается от иерархической модели. Основное отличие состоит в методах обработки данных.

Для моделирования объектно-ориентированных систем разработан язык моделирования UML (Universal Modeling Language), который можно определить как промышленный объектно-ориентированный стандарт моделирования. С помощью построителя диаграмм классов этого языка можно визуально разрабатывать (и представлять) объектно-ориентированные приложения, в том числе БД.

2.7.1. Понятие класса

Понятие класса является центральным понятием ООП. Класс может быть определен как определенный пользователем инкапсулированный тип объектов произвольной внутренней сложности. Классы играют роль некоторого шаблона для определения набора подобных объектов. Каждый объект обладает своими собственными характеристиками, однако объекты, имеющие одинаковые свойства и ведущие себя сходным образом, могут быть сгруппированы вместе с образованием класса. Атрибуты и связанные с ними методы (операции над атрибутами) определяются один раз для всего класса, а не для каждого объекта в отдельности. Объекты некоторого класса называются его экземплярами.

2.7.2. Инкапсуляция данных

Понятие инкапсуляции означает, что объект содержит как структуру данных, так и набор операций, с помощью которым этой структурой можно манипулировать. Инкапсуляция ограничивает область видимости данных, принадлежащих объекту и, следовательно, доступ к ним. В противоположность реляционным БД, где операции над данными (вставка, удаление, модификация) являются общими и могут быть применены к любым объектам (таблицам) БД, в ООБД инкапсулированные данные одного объекта недоступны для методов другого объекта. Доступ к таким данным предоставляется только операциям этого объекта. В идеале для каждого атрибута объекта должна быть определена операция доступа get. Но требования полной инкапсуляции приводят к невозможности выполнения параллельных транзакций и быстрого поиска на основе индексов. Инкапсуляция также приводит к тому, что обеспечение целостности становится задачей каждого объекта в отдельности. Лишь ссылочная целостность может контролироваться системой снаружи. Поэтому требования инкапсуляции объектов в ООБД выполняются только на уровне пользователей, а не системы в целом.

Инкапсулированными являются не только данные, но и операции. Только часть операций является доступной снаружи и предоставляет интерфейс объекта. Операции в ООБД состоят из двух частей: подписи (интерфейса), содержащего имя операции и список аргументов, и тела (метода), содержащего собственно имплементацию операции. Такой подход реализует требование независимости программ и операций, поскольку позволяет модифицировать операции объектов, не затрагивая внешние программы, имеющие доступ к этим операциям.

2.7.3. Наследование

Некоторые объекты могут иметь подобные, но не идентичные атрибуты и методы. Наследование позволяет определять один класс на основе другого

класса, если имеет смысл совместно использовать подобные атрибуты и методы. Класс, от которого производится наследование, называется базовым классом или суперклассом. Производный класс называется подклассом. Подкласс наследует все основные свойства суперкласса и дополнительно определяет свои собственные. Процесс образования суперкласса называется обобщением, а процесс образования подкласса – специализацией. Такая способность к порождению классов от уже существующих делает возможным поступательное наращивание сложности программы и определяет легкость модификации. Наследование существенно сокращает избыточность данных, т.к. общие свойства могут быть легко перенесены в суперклассы. В свою очередь подклассы могут переопределять свойства суперклассов. Возможность переопределения является важной характеристикой наследования, поскольку позволяет легко управлять отдельными классами с минимальным влиянием на остальную часть системы. Переопределение позволяет повторно использовать имя операции в нескольких классах, что дает возможность определять одно и тоже имя для одной и той же операции независима от ее типа. Конкретный тип операции определяется из контекста при выполнении программы. Такое переопределение имени операции называется перегрузкой.

2.7.4. Полиморфизм

Переопределение (перегрузка) является частным случаем полиморфизма. Полиморфизм означает способность одного и того же программного кода работать с разнотипными данными. Иначе говоря, для разных классов можно определить функции с одинаковыми именами, а вызов конкретной функции будет определяться типом данных либо из контекста. Полиморфизм позволяет обеспечить динамическое (позднее) связывание объектов, когда тип объекта становиться известным не в процессе написания программы, а во время ее выполнения.

2.7.5. Идентификация объектов в ООБД

Для идентификации объектов в объектно-ориентированных моделях данных применяется автоматически генерируемый уникальный идентификатор (OID – Object Identifier), который назначается каждому объекту в момент его создания. Объекты в ООБД могут и не иметь OID, если они не представляют собой некоторой сущности и могут использоваться разными объектами, например числа и другие простые типы данных. Кроме OID для идентификации объектов может также применяться традиционный подход с использованием ключевых атрибутов. OID обладает следующими свойствами:

- генерируется СУБД;

- уникально обозначает реальный объект и не зависит от значений его атрибутов;
- инвариантен. Его нельзя изменить во время всего жизненного цикла программы. После создания объекта его OID не может быть использован повторно ни для какого другого объекта, даже после его удаления.

OID также применяется для связывания объектов и обеспечения ссылочной целостности. Бинарные связи чаще всего реализуются на основе инверсных ссылок, т.е. включения OID объектов друг в друга. Инвариантность OID упрощает контроль целостности, исключая каскадное обновление связанных объектов.

Применение OID дает неоспоримые преимущества над реляционной системой, где идентификация объекта производится по значению ключа, зачастую надуманного, и который, к тому же, может меняться. Например, в любой момент времени можно удалить некоторый объект и значение его ключа использовать для другого объекта. Также можно отметить следующие достоинства применения OID:

- эффективность и быстрота. OID обычно представляет собой указатель на фактический адрес объекта в памяти и поэтому доступ к нему занимает очень мало времени;
- независимость от данных. Объект может неоднократно менять свое содержание, но при этом он останется одним и тем же объектом;
- невозможность изменения пользователем.

Но и в ООМД все еще остается возможность создания двух идентичных копий некоторого объекта, различающихся только OID, и поэтому применение ключевых атрибутов для дополнительной идентификации объектов оправдано.

2.7.6. Обеспечение доступности и перманентности объектов

Обеспечение доступности объектов в ООБД производится на основе присвоения уникальных имен (ключей) объектам, поскольку OID объектов скрыты для пользователей. Прямое присвоение имени каждому объекту в отдельности не производится, так как БД может состоять из сотен и тысяч объектов, и наличие большого числа сложных имен может только привести к усложнению доступа. Но так как обычно объектно-ориентированная система построена на основе иерархии контейнеров, где одни объекты по крайней мере концептуально содержатся в других (обычно содержатся не сами объекты, а их глобальные уникальные идентификаторы), то имена назначают только базовым объектам либо даже коллекциям объектов. Тогда по ссылкам (OID) можно легко добраться до необходимого объекта. Такой механизм доступа сильно отличается от применяемого в реляционных

системах, где все объекты предполагаются постоянными и доступными через значения их потенциальных ключей.

Перманентность объектов, т.е. возможность длительного хранения объектов во внешней памяти, может быть реализована тремя различными способами: созданием контрольных точек, сериализацией и явной подкачкой страниц. В первом способе на диск копируется все адресное пространство программы (или его некоторая часть). В тех случаях, когда копируется все адресное пространство, программу можно вновь запустить с некоторой контрольной точки. Этот подход имеет два серьезных недостатка. Во-первых, контрольная точка может быть использована только той программой, которая ее создала. Во-вторых, на диск копируется множество совершенно ненужной информации.

Во втором способе на диск копируется отдельная иерархия контейнеров объектов. Вначале производится обход связанного графа объектов, доступных из выбранного объекта, затем вся эта структура пишется на диск. Но такой подход не позволяет восстанавливать связанные структуры данных, которые совместно используют одну и туже подчиненную структуру, при их раздельной записи, так как подчиненная структура после восстановления уже не будет совместной. К тому же сериализация производится сразу большими блоками, что приводит к неэффективному расходу памяти компьютера при сохранении небольших изменений.

Последний подход связан с явным обменом областей оперативной памяти и внешнего устройства. Этот процесс напрямую связан с преобразованием указателей объектов, существующих в оперативной памяти, и указателей на те же объекты в адресном пространстве внешней памяти. Существуют две реализации этого подхода: в первой объект считается перманентным (т.е. сохраняемым) тогда, когда он достижим для перманентного корневого объекта, а во второй он будет перманентным только в том случае, если он явно объявлен перманентным в прикладной программе. В обоих случаях перманентными будут только те объекты, в которых еще на этапе программирования заложена эта возможность. Недостатками такого подхода являются необходимость сборки «мусора» после выполнения программы, так как объект будет перманентным до тех пор, пока не будет явно удален приложением, и необходимость постоянной обработки двух систем указателей. Последний метод можно считать наиболее перспективным, если напрямую включить схемы обеспечения перманентности в базовый язык программирования.

2.7.7. Стандарт ODMG

Совместное использование внешними программами объектов, хранящихся в ООБД, может быть осуществлено только при наличии единого стандарта создания, хранения и использования объектов.

Фактически даже отсутствие языка запросов типа SQL, являющимся стандартным языком баз данных, может являться серьезной проблемой при использовании ООБД. Растущий интерес к ООБД предопределил появление такого стандарта. Группа ODMG (Object Database Management Group), включающая ряд ведущих производителей программного обеспечения, разработала стандартную объектно-ориентированную модель данных, которая может быть использована в ООБД. Вначале был создан стандарт ODMG-93 или ODMG-1. Затем он перерос в ODMG-2.

ODMG-2 состоит из следующих частей:

- объектной модели;
- языка определения объектов (object definition language – ODL);
- объектного языка запросов (object query language – OQL);
- расширения ОО языков программирования (C++, Java, Smalltalk, …).

Объектная модель и язык определения объектов стандартизирует ключевые слова, типы данных, их конструкторы и т.д., наряду с механизмами идентификации, связывания, именования, обеспечения доступности и перманентности объектов (смотри систему принятых обозначений на рис. 12). К примеру этот стандарт рекомендует применение только инверсных связей на основе OID и вводит такие ключевые слова как attribute, key, relationship, extent и т.д. Объектный язык запросов представляет собой расширение существующих ООЯП. Синтаксис его похож на SQL и дополнен концепциями наследования, инкапсуляции и полиморфизма. Например, представим определение двух классов, лежащих в основе структуры данных ООБД некоторого университета:

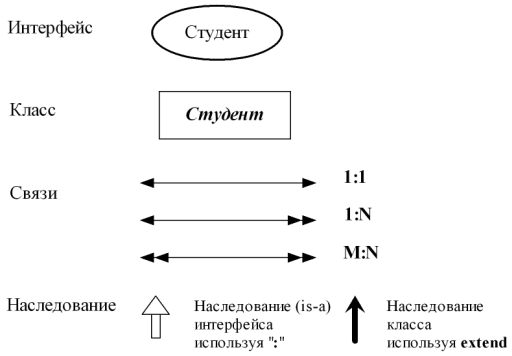


Рис. 12 Система принятых обозначений

```
class Person (extent persons, key ssn) {
  attribute struct Pname {string fname, string mname, string (name} name;
  attribute string ssn;
  attribute date birthdate;
  attribute enum Gender{M, F) sex;
```

attribute struct Address {**short** no, **string** street, **short** aptno, **string** city, **string** state, **short** zip} address;
short age(); };

class Student **extends** Person(**extent** students) {
attribute string class;
attribute Department minors_in;
relationship Department majors_in **inverse** Department::has_majors;
relationship set<Grade> completed_sections **inverse** Grade::student;
void change_major(**in string** dname) **raises**(dname_not_valid);
float gpa();
void register(**in short** secno) **raises**(section_not_valid);
void assign_grade(**in short** secno; **in** GradeValue grade)
raises(section_not_valid, grade_not_valid); };

и саму структуру (смотри рис. 13)

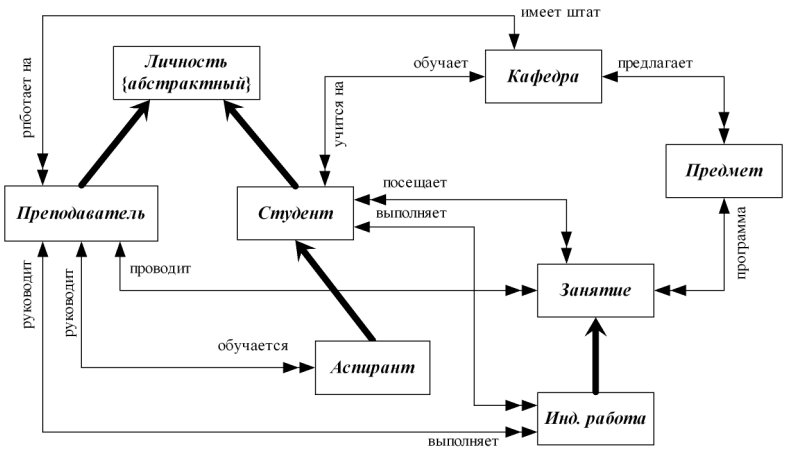


Рис. 13. Структура данных ООБД университета

2.7.8. Выводы

ООБД обладают следующими преимуществами:

1. Однозначная идентификация объектов;
2. Расширенные возможности моделирования. Позволяет более точно моделировать реальный мир;
3. Расширяемость;
4. Устранение проблем несоответствия. Программы могут читать объекты напрямую, без промежуточной конвертации;
5. Поддержка эволюции схемы;

6. Применимость для разработки сложных специализированных БД.

Также можно сформулировать и недостатки, присущие ООБД:

1. Отсутствие универсальной модели данных;
2. Недостаточный опыт эксплуатации и отсутствие стандартов;
3. Сложность;
4. Отсутствие поддержки стандартных видов представлений;
5. Недостаточность средств обеспечения целостности и безопасности информации.

Концептуальное различие объектно-ориентированной и реляционной модели заключается в следующем:

1. В типах данных. Простые в РБД и определенные пользователем классы в ООБД;
2. В идентификации объектов. OID в ООБД и потенциальные ключи в РБД;
3. В установлении связей. Инверсные включения OID в ООБД и первичные и внешние ключи в РБД;
4. В возможности наследования в ООБД;
5. В применении и разработке операций. В ООБД операции принадлежат объектам и разрабатываются вместе с ними. В РБД операции являются общедоступными и разрабатываются после определения структуры данных.

Примером ООБД можно считать СУБД OpenODB (Hewlett Packard), O2 (Ardent Software) и Object Store (Object Design).

2.8. Объектно-реляционная модель

Объектно-реляционные системы получили свое развитие на основе концептуального совпадения назначения и свойств двух фундаментальных понятий в реляционных и объектно-ориентированных баз данных: доменов и классов. Если определить домен немного шире, чем в реляционной модели, а именно как определенный пользователем инкапсулированный тип данных с произвольной внутренней сложностью, то увидим, что домен и класс – это одно и тоже. Следовательно, в ОРБД домены, и соответственно, реляционные атрибуты, могут содержать абсолютно все: массивы, списки, объекты вместе с методами обработки своих данных. Таким образом, реляционная модель данных, где в качестве доменов могут выступать классы, в принципе могла бы решать все задачи, выполняемые в ООБД и невыполняемые в реляционной БД.

Для обеспечения функциональности таких систем со стороны объектно-ориентированного подхода требуется поддержка определенных

пользователем типов данных произвольной сложности вместе с методами обработки своих данных, а со стороны реляционного подхода требуется, чтобы отношения содержали сами объекты (по крайней мере концептуально), а не ссылки или указатели на них.

Примером ОРБД можно считать DB2, Informix, и продукты Oracle 8.x.

Контрольные вопросы и задания

1. Назовите основные модели данных.
2. Укажите достоинства и недостатки иерархической модели данных.
3. Укажите достоинства и недостатки сетевой модели данных.
4. Какой вид связи невозможно реализовать в иерархической модели данных.
5. Назовите условия, при соблюдении которых таблицу (реляционная модель данных) можно считать отношением.
6. Дайте определения каждому из следующих понятий реляционной модели данных: отношению, домену, кортежу, степени отношения, кардинальному числу, первичному и внешнему ключу.
7. Укажите различия между потенциальными ключами и первичным ключом отношения.
8. Какие из нижеперечисленных терминов означают одно и тоже понятие в реляционной модели данных: домен, кортеж, отношение, строка, столбец, поле, запись, атрибут.
9. Покажите на простых примерах виды связей между отношениями в реляционной модели данных. Укажите только название отношения, первичные и внешние ключи и по нескольким неключевым атрибутам.
10. Перечислите средства обеспечения ссылочной целостности в реляционной модели данных.
11. Что дает каскадное обновление записей.
12. Что дает каскадное удаление записей.
13. Перечислите отличительные свойства реляционной модели данных.
14. Сформулируйте назначение ключей в реляционной модели данных.
15. Какие задачи решает теория нормализации данных.
16. Опишите проблемы, связанные с наличием повторяющихся данных.
17. Сформулируйте правила нормализации.
18. Дайте определение первой нормальной формы.
19. Дайте определение второй нормальной формы.

20. Дайте определение третьей нормальной формы.
21. Зачем необходимо разбивать информацию, хранящуюся в одной таблице, на несколько связанных таблиц в реляционной модели данных.
22. Перечислите основные достоинства и недостатки реляционной модели данных.
23. Чем отличается постреляционная модель данных от реляционной.
24. Сформулируйте назначение и основные принципы построения объектно-ориентированных БД.
25. Перечислите основные достоинства и недостатки многомерной модели данных.
26. На базе относительного сходства какого понятия в реляционной и объектно-ориентированной модели данных получили развитие объектно-реляционные БД.
27. Сформулируйте принципиальные отличия реляционной, объектно-ориентированной и объектно-реляционной модели данных.
28. Чем отличается расширенная ER модель от обычной ER модели.
29. Дайте определение многомерной модели данных. В чем состоит основное отличие многомерной модели от других моделей.
30. Перечислите основные достоинства и недостатки многомерной модели данных.

3. ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

Все тонкости построения информационной модели некоторой предметной области деятельности человека преследуют одну цель – получить хорошую БД. Давайте поясним термин – хорошая БД и сформулируем требования, которым должна удовлетворять БД:

1. БД должна удовлетворять информационным потребностям пользователей (организаций) и по структуре и содержанию соответствовать решаемым задачам;
2. БД должна обеспечивать получение требуемых данных за приемлемое время, т.е. отвечать требованиям производительности;
3. БД должна легко расширяться при реорганизации предметной области;
4. БД должна легко изменяться при изменении программной и аппаратной среды;
5. Корректные данные, загруженные в БД, должны оставаться корректными (данные должны проверяться на корректность при их вводе).

3.1. Этапы проектирования

Основные этапы проектирования можно отобразить с помощью схемы, представленной на рис. 14. Рассмотрим каждый из этапов подробнее:

Первый этап. Планирование разработки базы данных. На этом этапе выделятся наиболее эффективный способ реализации этапов жизненного цикла системы.

Второй этап. Определение требований к системе. Производится определение диапазона действий и границ приложения базы данных, а также производится сбор и анализ требований пользователей.

Третий этап. Проектирование концептуальной модели БД. Процесс создания БД начинается с определения концептуальной модели, представляющей объекты и их взаимосвязи без указания способов их физического хранения. Усилия на этом этапе должны быть направлены на структуризацию данных и выявление взаимосвязей между ними. Этот процесс можно разбить еще на несколько подэтапов:

а) Уточнение задачи. Перед началом работы над конкретным приложением у разработчика наверняка имеются некоторые представления о том, что он будет разрабатывать. В случаях, когда разрабатывается небольшая персональная БД, такие представления могут быть достаточно полными. В других случаях, когда разрабатывается большая БД под заказ, таких представлений может

быть очень мало, или они наверняка будут поверхностными. Сразу начинать разработку с определения таблиц, полей и связей между ними будет явно рановато. Такой подход наверняка приведет к полной переделке большей части приложения. Поэтому следует затратить некоторое время на составление списка всех основных задач, которые должны решаться этим приложением, включая и те, которые могут возникнуть в будущем.

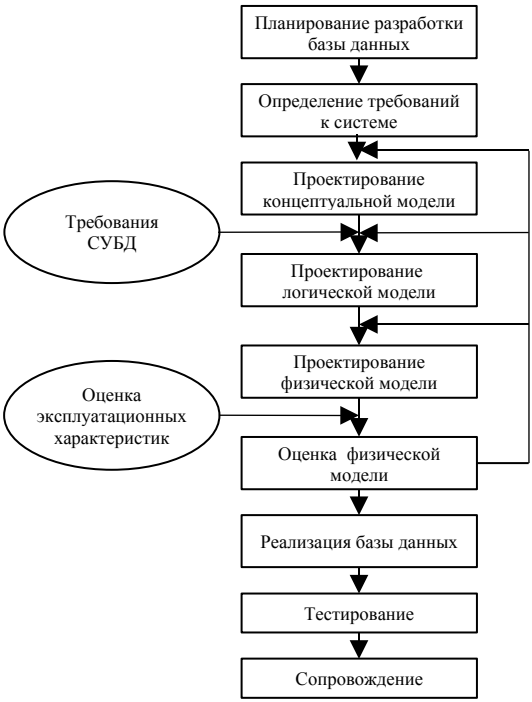


Рис. 14. Схема проектирования БД

b) Уточнение последовательности выполнения задач. Чтобы приложение работало логично и удобно, лучше всего объединить основные задачи в группы и затем упорядочить задачи каждой группы так, чтобы они располагались в порядке их выполнения. Группировка и графическое представление последовательности их выполнения поможет определить естественный порядок выполнения задач.

c) Анализ данных. После определения списка задач необходимо для каждой задачи составить полный перечень данных, требуемых для ее решения. После анализа данных можно приступать к разработке

концептуальной модели, т.е. к выделению объектов, атрибутов и связей.

Четвертый этап. Построение логической модели. Построение логической модели начинается с выбора модели данных. При выборе модели важную роль играет ее простота, наглядность и сравнение естественной структуры данных с моделью, ее представляющей. Но зачастую этот выбор определяется успехом (или наличием) той или иной СУБД. Т.е. разработчик выбирает СУБД, а не модель данных. Таким образом, на этом этапе концептуальная модель транслируется в модель данных, совместимую с выбранной СУБД. Возможно, что отображенные в концептуальной модели взаимосвязи между объектами либо некоторые атрибуты объектов окажутся впоследствии нереализуемыми средствами выбранной СУБД. Это потребует изменение концептуальной модели. Версия концептуальной модели, которая может быть обеспечена конкретной СУБД, называется **логической моделью**. Иногда процесс определения концептуальной и логической моделей называется определением структуры данных.

Пятый этап. Построение физической модели. Физическая модель определяет размещение данных, методы доступа и технику индексирования. На этапе физического проектирования мы привязываемся к конкретной СУБД и расписываем схему данных более детально, с указанием типов, размеров полей и ограничений. Кроме разработки таблиц и индексов, на этом этапе производится также определение основных запросов.

При построении физической модели приходится решать две взаимно противоположные по своей сути задачи. Первой из них является минимизация места хранения данных, а второй – достижение максимальной производительности, целостности и безопасности данных. Например, для обеспечения высокой скорости поиска необходимо создание индексов, причем их число будет определяться всеми возможными комбинациями полей, участвующими в поиске; для восстановления данных требуется ведения журнала всех изменений и создание резервных копий БД; для эффективной работы транзакций требуется резервирование места на диске под временные объекты и т.д., что приводит к увеличению (иногда значительному) размера БД.

Шестой этап. Оценка физической модели. На этом этапе проводится оценка эксплуатационных характеристик. Здесь можно проверить эффективность выполнения запросов, скорость поиска, правильность и удобство выполнения операций с БД, целостность данных и эффективность расхода ресурсов компьютера. При неудовлетворительных эксплуатационных характеристиках возможен возврат к пересмотру физической и логической моделей данных, выбору СУБД и типа компьютера.

Седьмой этап. Реализация БД. При удовлетворительных эксплуатационных характеристиках можно перейти к созданию макета приложения, т.е. набору основных таблиц, запросов, форм и отчетов. Этот предварительный макет можно продемонстрировать перед заказчиком и получить его одобрение перед детальной реализации приложения.

Восьмой этап. Тестирование и оптимизация. Обязательным этапом является тестирование и оптимизация разработанного приложения.

Этап девятый, заключительный. Сопровождение и эксплуатация. Так как выявить и устранить все ошибки на этапе тестирования не получается, то этап сопровождения является обычным для баз данных.

Существует два основных подхода к проектированию схемы данных: нисходящий и восходящий. При восходящем подходе работа начинается с нижнего уровня – уровня определения атрибутов, которые на основе анализа существующих между ними связей группируются в отношения, представляющие объекты, и связи между ними. Процесс нормализации таблиц для реляционной модели данных является типичным примером этого подхода. Этот подход хорошо подходит для проектирования относительно небольших БД. При увеличении числа атрибутов до нескольких сотен и даже тысяч более подходящей стратегией проектирования является нисходящий подход. Начинается этот подход с определения нескольких высокоуровневых сущностей и связей между ними. Затем эти объекты детализируются до необходимого уровня. Примером такого подхода проектирования является использование модели сущность-связь. На практике эти подходы обычно комбинируются. В этом случае можно говорить о смешанном подходе проектирования.

3.2. Средства автоматизированной разработки приложений

Проектирование БД можно проводить с помощью автоматизированных систем разработки приложений, так называемых CASE (Computer Aided Software Engineering) систем. Автоматизированные системы разработки приложений представляют собой программные средства, поддерживающие процессы создания и сопровождения информационных систем, такие как анализ и формулировка требований, проектирование приложений, генерация кода, тестирование, управление конфигурацией и проектом. Основная цель CASE систем состоит в том, чтобы отделить процесс проектирования программного обеспечения от его кодирования и последующих этапов разработки (тестирование, документирование и т.д.), а также автоматизировать весь процесс создания программных систем.

Процесс разработки баз данных с помощью CASE систем на этапе концептуального проектирования обычно проводится с помощью ER

модели. ER модель необходимо рассматривать как способ концептуального проектирования данных, а не как самостоятельную модель данных. Результатом проектирования практически всегда (или до недавнего времени) являлась реляционная модель данных. В последнее время с появлением объектно-ориентированной модели данных, конечным результатом проектирования все чаще стали являться объектно-ориентированная и объектно-реляционная модели данных.

Для автоматизации проектирования объектно-ориентированных баз данных CASE системы предоставляют специальный язык Unified Modeling Language (UML), который можно определить как промышленный объектно-ориентированный стандарт моделирования. Его составляющими можно назвать языки OMT (Object Modeling Technique) и OOSE (Object-Oriented Software Engineering). Большую роль в создании этого языка сыграл консорциум OMG (Object Management Group), включающий ряд ведущих производителей программного обеспечения.

CASE системы различаются по ориентации, по функциональной полноте и по типу используемой модели. По ориентации можно выделить CASE системы, предназначенные для анализа предметной области, для проектирования баз данных и для полной разработки приложений. К числу последних можно отнести MS Visual Studio. По функциональной полноте разделяются на системы, предназначенные для решения отдельных задач проектирования и на интегрированные системы, поддерживающие весь цикл разработки. По типу используемой модели разделяются на структурные ,объектно-ориентированные и комбинированные.

Примерами CASE систем могут служить: ERWin (Logic Works), Rational Rose (Rational Software) (OODBMS), S-Designer (SPD), DataBase Designer, Developer/Designer 2000 (Oracle Corp.).

Контрольные вопросы и задания

1. Сформулируйте требования, которым должна удовлетворять БД.
2. На что должны быть направлены усилия разработчиков при построении концептуальной модели.
3. Отобразите схематично основные этапы проектирования БД и дайте их характеристику.
4. Чем отличается логическая модель данных от концептуальной.
5. Какие две противоположные по сути задачи приходится решать при построении физической модели.
6. Что такое CASE системы.
7. Дайте классификацию CASE систем.