Лекция 2. Понятие модели данных. Обзор разновидностей моделей данных

2.1. Введение

Историю технологии БД принято отсчитывать с начала 1960-х гг., когда появились первые попытки создания специальных программных средств управления базами данных. За прошедшие десятилетия возникали и использовались различные подходы к организации баз данных. Для описания и сравнения некоторых из них мы воспользуемся понятием *модели данных*, предложенным в 1969 г. Эдгаром Коддом [2.1]. Кодд ввел это понятие для описания конкретного *реляционного подхода* к организации БД. Соответственно, он говорил о *реляционной модели данных*, различным теоретическим и реализационным аспектам которой в основном посвящен этот курс. Однако понятие модели данных оказалось удобным не только для описания реляционного подхода и сравнения реализаций реляционных СУБД, но и для реализационно-независимого представления и сопоставления других подходов к организации баз данных.

2.2. Модель данных

В модели данных описывается некоторый набор родовых понятий и признаков, которыми должны обладать все конкретные СУБД и управляемые ими базы данных, если они основываются на этой модели. Наличие модели данных позволяет сравнивать конкретные реализации, используя один общий язык.

Хотя понятие модели данных было введено Коддом, наиболее распространенная трактовка модели данных, по-видимому, принадлежит Кристоферу Дейту, который воспроизводит ее (с различными уточнениями) применительно к реляционным БД практически во всех своих книгах (см., например, [1.3]). Согласно Дейту реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: структурной части, манипуляционной части и целостной части.

В структурной части модели данных фиксируются основные логические структуры данных, которые могут применяться на уровне пользователя при организации БД, соответствующих данной модели. Например, в модели данных SQL основным видом структур базы данных являются таблицы, а в объектной модели данных – объекты ранее определенных типов.

Манипуляционная часть модели данных содержит спецификацию одного или нескольких языков, предназначенных для написания запросов к БД. Эти языки могут быть абстрактными, не обладающими точно проработанным синтаксисом (что свойственно языками реляционной алгебры и реляционного исчисления, используемым в реляционной модели данных), или законченными производственными языками (как в случае модели данных SQL). Основное назначение манипуляционной части модели данных – обеспечить эталонный «модельный» язык БД, уровень выразительности которого должен поддерживаться в реализациях СУБД, соответствующих данной модели.

Наконец, в целостной части модели данных (которая явно выделяется не во всех известных моделях) специфицируются механизмы ограничений целостности, которые обязательно должны поддерживаться во всех реализациях СУБД, соответствующих данной модели. Например, в целостной части реляционной модели данных категорически требуется поддержка ограничения первичного ключа в любой переменной отношения, а аналогичное требование к таблицам в модели данных SQL отсутствует.

В этой лекции мы применим понятие модели данных для обзора как подходов, предшествовавших появлению реляционных баз данных, так и подходов, которые возникли позже. Мы не будем касаться особенностей каких-либо конкретных систем; это привело бы к изложению многих технических деталей, которые, хотя и интересны, но находятся несколько в стороне от основной цели курса.

2.3. Ранние модели данных

Начнем с рассмотрения общих подходов к организации трех типов ранних систем, а именно, систем, основанных на инвертированных списках, иерархических и сетевых систем управления базами данных. В целом ранние системы можно охарактеризовать следующим образом[1]:

- Эти системы активно использовались в течение многих лет, задолго до появления работоспособных реляционных СУБД. На самом деле некоторые из ранних систем используются даже в наше время, накоплены громадные базы данных, и одной из актуальных проблем информационных систем является использование этих систем совместно с современными.
- Все ранние системы не основывались на каких-либо абстрактных моделях. Как мы упоминали, понятие модели данных фактически вошло в обиход специалистов в области БД только вместе с реляционным подходом. Абстрактные представления ранних систем появились позже на основе анализа и выявления общих признаков у различных конкретных систем.
- В ранних системах доступ к БД производился на уровне записей. Пользователи этих систем осуществляли явную навигацию в БД, используя языки программирования, расширенные функциями СУБД. Интерактивный доступ к БД поддерживался только путем создания соответствующих прикладных программ с собственным интерфейсом.
- Можно считать, что уровень средств ранних СУБД соотносится с уровнем файловых систем примерно так же, как уровень языка Cobol соотносится с уровнем языков ассемблера. Заметим, что при таком взгляде уровень реляционных систем соответствует уровню языков Ада или APL.
- Навигационная природа ранних систем и доступ к данным на уровне записей заставляли пользователей самих производить всю оптимизацию доступа к БД, без какой-либо поддержки системы.
- После появления реляционных систем большинство ранних систем было оснащено «реляционными» интерфейсами. Однако в большинстве случаев это не сделало их по-настоящему реляционными системами, поскольку оставалась возможность манипулировать данными в естественном для них режиме.

2.3.1. Модель данных инвертированных таблиц

К числу наиболее известных и типичных представителей систем, в основе которых лежит эта модель данных, относятся СУБД Datacom/DB, выведенная на рынок в конце 1960-х гг. компанией Applied Data Research, Inc. (ADR) и принадлежащая в настоящее время компании Computer Associates, и Adabas (ADAptable DAtabase System), которая была разработана компанией Software AG в 1971 г. и до сих пор является ее основным продуктом.

Организация доступа к данным на основе инвертированных таблиц используется практически во всех современных реляционных СУБД, но в этих системах пользователи не имеют непосредственного доступа к инвертированным таблицам (индексам). Кстати, когда мы будем рассматривать внутренние интерфейсы реляционных СУБД, можно будет увидеть, что они очень близки к пользовательским интерфейсам систем, основанных на инвертированных таблицах.

Структуры данных

База данных в модели инвертированных таблиц похожа на БД в модели SQL, но с тем отличием, что пользователям видны и хранимые таблицы, и пути доступа к ним. При этом:

- Строки таблиц упорядочиваются системой в некоторой физической, видимой пользователям последовательности.
- Физическая упорядоченность строк всех таблиц может определяться и для всей БД (так делается, например, в Datacom/DB).
- Для каждой таблицы можно определить произвольное число ключей поиска, для которых строятся индексы. Эти индексы автоматически поддерживаются системой, но явно видны пользователям.

Манипулирование данными

Поддерживаются два класса операций:

1. Операции, устанавливающие адрес записи и разбиваемые на два подкласса:
   - прямые поисковые операторы (например, установить адрес первой записи таблицы по некоторому пути доступа);
   - операторы, устанавливающие адрес записи при указании относительной позиции от предыдущей записи по некоторому пути доступа.
2. Операции над адресуемыми записями.

Вот типичный набор операций:

- `LOCATE FIRST` – найти первую запись таблицы *`T`* в физическом порядке; возвращается адрес записи;
- `LOCATE FIRST WITH SEARCH KEY EQUAL` – найти первую запись таблицы *`T`* с заданным значением ключа поиска *`k`*; возвращается адрес записи;
- `LOCATE NEXT` – найти первую запись, следующую за записью с заданным адресом в заданном пути доступа; возвращается адрес записи;
- `LOCATE NEXT WITH SEARCH KEY EQUAL` – найти следующую запись таблицы *`T`* в порядке пути поиска с заданным значением *`k`*; должно быть соответствие между используемым способом сканирования и ключом *`k`*; возвращается адрес записи;
- `LOCATE FIRST WITH SEARCH KEY GREATER` – найти первую запись таблицы *`T`* в порядке ключа поиска *`k`* со значением ключевого поля, большим заданного значения *`k`*; возвращается адрес записи;
- `RETRIVE` – выбрать запись с указанным адресом;
- `UPDATE` – обновить запись с указанным адресом;
- `DELETE` – удалить запись с указанным адресом;
- `STORE` – включить запись в указанную таблицу; операция генерирует и возвращает адрес записи.

Ограничения целостности

Общие правила определения целостности БД отсутствуют. В некоторых системах поддерживаются ограничения уникальности значений некоторых полей, но в основном вся поддержка целостности данных возлагается на прикладную программу.

2.3.2. Иерархическая модель данных

Типичным представителем (наиболее известным и распространенным) является СУБД IMS (Information Management System) компании IBM. Первая версия системы появилась в 1968 г.

Иерархические структуры данных

Иерархическая БД состоит из упорядоченного набора деревьев; более точно, из упорядоченного набора нескольких экземпляров одного типа дерева. Тип дерева состоит из одного «корневого» типа записи и упорядоченного набора из нуля или более типов поддеревьев (каждое из которых является некоторым типом дерева). Тип дерева в целом представляет собой иерархически организованный набор типов записи.

На рис. 2.1 показан пример типа дерева (схемы иерархической БД). Здесь тип записи `Отдел` является предком для типов записи `Руководитель` и `Служащие`, а `Руководитель` и `Служащие` – потомки типа записи `Отдел`. Смысл полей типов записей в основном должен быть понятен по их именам. Поле `Рук_Отдел` типа записи `Руководитель` содержит номер отдела, в котором работает служащий, являющийся данным руководителем (предполагается, что он работает не обязательно в том же отделе, которым руководит). Между типами записи поддерживаются связи (правильнее сказать, *типы* связей, поскольку реальные связи появляются в экземплярах типа дерева).

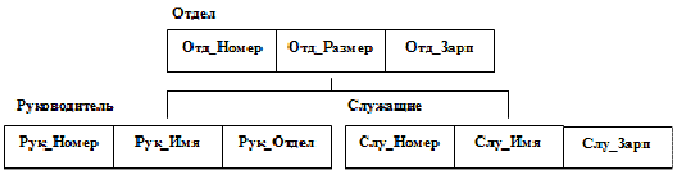

Рис. 2.1. Пример типа дерева

База данных с такой схемой могла бы выглядеть так, как показано на рис. 2.2 (мы показываем один экземпляр дерева).

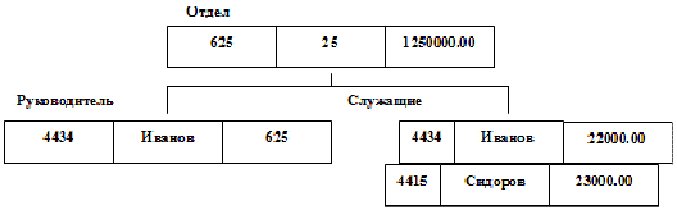

Рис. 2.2. Пример иерархической базы данных

Все экземпляры данного типа потомка с общим экземпляром типа предка называются близнецами. Для иерархической базы данных определяется полный порядок обхода

дерева: сверху-вниз, слева-направо. Заметим, что в терминологии IMS вместо термина *запись* использовался термин *сегмент*, а под *записью базы данных* понималось все дерево сегментов.

Манипулирование данными

Примерами типичных операций манипулирования иерархически организованными данными могут быть следующие:

- найти указанный экземпляр типа дерева БД (например, отдел 310);
- перейти от одного экземпляра типа дерева к другому;
- перейти от экземпляра одного типа записи к экземпляру другого типа записи внутри дерева (например, перейти от отдела к первому сотруднику);
- перейти от одной записи к другой в порядке обхода иерархии;
- вставить новую запись в указанную позицию;
- удалить текущую запись.

Ограничения целостности

В иерархической модели данных автоматически поддерживается целостность ссылок между предками и потомками. Основное правило: *никакой потомок не может существовать без своего родителя*. Заметим, что аналогичная поддержка целостности по ссылкам между записями без связи «предок-потомок», не обеспечивается. Примером такой «внешней» ссылки является содержимое поля `Рук_Отдел` в экземпляре типа записи `Руководитель`.

2.3.3. Сетевая модель данных

Типичным представителем систем, основанных на сетевой модели данных, является СУБД IDMS (Integrated Database Management System), разработанная компанией Cullinet Software, Inc. и изначально ориентированная на использования на мейнфреймах компании IBM. Архитектура системы основана на предложениях Data Base Task Group (DBTG) организации CODASYL (COnference on DAta SYstems Languages), которая отвечала за определение языка программирования COBOL. Отчет DBTG был опубликован в 1971 г., и вскоре после этого появилось несколько систем, поддерживающих архитектуру CODASYL, среди которых присутствовала и СУБД IDMS. В настоящее время IDMS принадлежит компании Computer Associates.

Сетевые структуры данных

Сетевой подход к организации данных является расширением иерархического подхода. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре данных у потомка может иметься любое число предков.

Сетевая БД состоит из набора записей и набора связей между этими записями, а если говорить более точно, из набора экземпляров каждого типа из заданного в схеме БД набора типов записи и набора экземпляров каждого типа из заданного набора типов связи.

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка. Для данного типа связи `L` с типом записи предка `P` и типом записи потомка `C` должны выполняться следующие два условия:

- каждый экземпляр типа записи `P` является предком только в одном экземпляре типа связи `L`;
- каждый экземпляр типа записи `C` является потомком не более чем в одном экземпляре типа связи `L`.

На формирование типов связи не накладываются особые ограничения; возможны, например, следующие ситуации:

- тип записи потомка в одном типе связи `L1` может быть типом записи предка в другом типе связи `L2` (как в иерархии);
- данный тип записи `P` может быть типом записи предка в любом числе типов связи;
- данный тип записи `P` может быть типом записи потомка в любом числе типов связи;
- может существовать любое число типов связи с одним и тем же типом записи предка и одним и тем же типом записи потомка; и если `L1` и `L2` - два типа связи с одним и тем же типом записи предка `P` и одним и тем же типом записи потомка `C`, то правила, по которым образуется родство, в разных связях могут различаться;
- типы записи `X` и `Y` могут быть предком и потомком в одной связи и потомком и предком – в другой;
- предок и потомок могут быть одного типа записи.

На рис. 2.3 показан простой пример схемы сетевой БД. На этом рисунке показаны три типа записи: `Отдел`, `Служащие` и `Руководитель` и три типа связи: `Состоит из служащих`, `Имеет руководителя` и `Является служащим`**.** В типе связи `Состоит из служащих` типом записи-предком является `Отдел`, а типом записи-потомком – `Служащие` (экземпляр этого типа связи связывает экземпляр типа записи `Отдел` со многими экземплярами типа записи `Служащие`, соответствующими всем служащим данного отдела). В типе связи `Имеет руководителя` типом записи-предком является `Отдел`, а типом записи-потомком – `Руководитель` (экземпляр этого типа связи связывает экземпляр типа записи `Отдел` с одним экземпляром типа записи `Руководитель`, соответствующим руководителю данного отдела). Наконец, в типе связи `Является служащим` типом записи-предком является `Руководитель`, а типом записи-потомком – `Служащие` (экземпляр этого типа связи связывает экземпляр типа записи `Руководитель` с одним экземпляром типа записи `Служащие`, соответствующим тому служащему, которым является данный руководитель).

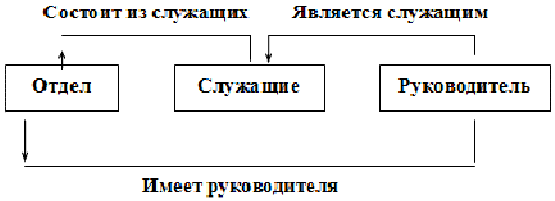

Рис. 2.3. Пример схемы сетевой базы данных

Манипулирование данными

Вот примерный набор операций манипулирования данными:

- найти конкретную запись в наборе однотипных записей (например, служащего с именем Иванов);

- перейти от предка к первому потомку по некоторой связи (например, к первому служащему отдела 625);
- перейти к следующему потомку в некоторой связи (например, от Иванова к Сидорову);
- перейти от потомка к предку по некоторой связи (например, найти отдел, в котором работает Сидоров);
- создать новую запись;
- уничтожить запись;
- модифицировать запись;
- включить в связь;
- исключить из связи;
- переставить в другую связь и т.д.

Ограничения целостности

Имеется (необязательная) возможность потребовать для конкретного типа связи отсутствие потомков, не участвующих ни в одном экземпляре этого типа связи (как в иерархической модели).

---

1 Заметим, что перечисляемые ниже характеристики в полной мере относятся и к другим не реляционным подходам к организации баз данных, которые возникли до появления реляционного подхода или почти одновременно с ним. В частности, подобными свойствами обладают системы, основанные на подходах MUMPS (наиболее известной в России является реализация этого подхода в СУБД Cache компании Intersystems) и Pick (этот подход реализован во многих СУБД, в частности, в СУБД UniVerse и UniData семейства U2 компании IBM).

2.4. Неформальное введение в реляционную модель данных

Как уже говорилось в начале этой лекции, основные идеи реляционной модели данных были предложены Эдгаром Коддом в 1969 г. [2.1]. Следует заметить, что, несмотря на общепризнанную значимость этой и последующих работ Кодда, посвященных реляционной модели данных, эти работы писались на идейном уровне, не были (по теперешним меркам) глубоко технически проработанными, во многих важных местах допускали неоднозначное толкование, и поэтому эти работы невозможно было использовать как непосредственное руководство для реализации СУБД, поддерживающей реляционную модель.

За прошедшие десятилетия реляционная модель развивалась в двух направлениях. Первое направление заложил знаменитый экспериментальный проект компании IBM System R (см. лекцию 12). В этом проекте возник язык SQL, изначально основанный на идеях Кодда (который также работал в IBM), но нарушающий некоторые принципиальные предписания реляционной модели. К настоящему времени в действующем стандарте языка SQL, по сути, специфицирована некоторая собственная, законченная модель данных, обзор которой мы приведем в следующем разделе этой лекции, а более подробно обсудим в лекциях 15-23.

Второе направление, начиная с 1990-х гг., возглавляет Кристофер Дейт, к которому позже примкнул Хью Дарвен. Оба этих ученых также работали в компании IBM и до 1990-х гг. внесли большой вклад в развитие языка SQL. Однако в 1990-е гг. Дейт и Дарвен пришли к выводу, что искажения реляционной модели данных, свойственные языку SQL, достигли настолько высокого уровня, что пришло время предложить альтернативу, опирающуюся на неискаженные идеи Эдгара Кодда и обеспечивающую все возможности как SQL, так и объектно-ориентированного подхода к организации баз данных и СУБД (обзор объектно-ориентированной модели данных приводится в следующем разделе).

Новые идеи Дейта и Дарвена были впервые изложены в их *Третьем манифесте* [2.8], а позже на основе этих идей была специфицирована модель данных [1.5]. Авторы считают, что в [1.5], на самом деле, приводится всего лишь современная и полная интерпретация идей Кодда. С этим можно соглашаться или спорить, но бесспорен один факт – Кодд не участвовал в написании этих материалов и никогда не писал ничего подобного. В следующих лекциях, тем не менее, при обсуждении реляционной модели мы будем использовать, в основном, интерпретацию Дейта и Дарвена.

В этой же лекции мы сначала приведем в данном разделе краткий и неформальный обзор основных идей реляционной модели в том виде, в котором она была предложена Коддом, а в следующем разделе также кратко и неформально обсудим предложения Дейта и Дарвена. Более строгое и формальное описание реляционной модели данных приводится в лекциях 3-6.

2.4.1. Реляционные структуры данных

Основная идея Кодда состояла в том, чтобы выбрать в качестве родовой логической структуры хранения данных структуру, которая, с одной стороны, была бы достаточно удобной для большинства приложений и, с другой стороны, допускала бы возможность выполнения над базой данных ненавигационных операций. Иерархические и, в особенности, сетевые структуры данных являются навигационными по своей природе. Ненавигационному использованию таблиц мешает упорядоченность их столбцов и, в особенности, строк.

По сути, Кодд предложил использовать в качестве родовой структуры БД «таблицы», в которых и столбцы, и строки не являются упорядоченными. Легко видеть, что такая «таблица» со множеством столбцов $\{A_1, A_2, \ldots, A_n\}$, в которой каждый столбец A_i может содержать значения из множества $T_i = \{v_{i1}, v_{i2}, \ldots, v_{im}\}$ (все множества конечны), в математическом смысле представляет собой отношение над множествами $\{T_1, T_2, \ldots, T_n\}$. Напомню, что в математике отношением над множествами $\{T_1, T_2, \ldots, T_n\}$ называется подмножество декартова произведения этих множеств, т.е. некоторое множество кортежей $\{\{v_1, v_2, \ldots, v_n\}\}$, где $v_i \in T_i$. Поэтому для обозначения родовой структуры Кодд стал использовать термин *отношение (relation)*, а для обозначения элементов отношения – термин *кортеж*. Соответственно, модель данных получила название *реляционной модели*.

Схема БД в реляционной модели данных – это набор именованных *заголовков отношений* вида $H_i = \{<A_i^1, T_i^1>, < A_i^2, T_i^2>, \ldots, < A_i^{ni}, T_i^{ni}>\}$. T_i называется *доменом* атрибута A_i. По Кодду, каждый домен T_i является подмножеством значений некоторого базового типа данных T_i^+, а значит, к его элементам применимы все операции этого базового типа (в конце 1960-х гг. базовыми типами данных считались типы данных распространенных тогда языков программирования; в IBM наиболее популярными языками были PL1 и COBOL).

Реляционная база данных в каждый момент времени представляет собой набор именованных отношений, каждое из которых обладает заголовком, таким как он

определен в схеме БД, и телом. Имя отношения R_i совпадает с именем заголовка этого отношения H_{Ri}. Тело отношения B_{Ri} – это множество кортежей вида $\{<A_i^1, T_i^1, v_i^1>, < A_i^2, T_i^2, v_i^2>, \ldots, < A_i^{ni}, T_i^{ni}, v_i^{ni}>\}$, где $t_i^j \in T_i$[2]. Во время жизни БД тела отношений могут изменяться, но все содержащиеся в них кортежи должны соответствовать заголовкам соответствующих отношений.

2.4.2. Манипулирование реляционными данными

Поскольку в реляционной модели данных заголовок и тело любого отношения представляют собой множества, к отношениям, вообще говоря, применимы обычные теоретико-множественные операции: объединение, пересечение, вычитание, взятие декартова произведения. Напомним, что для двух множеств S_1 $\{s_1\}$ и S_2 $\{s_2\}$ результатом операции объединения этих двух множеств S_1 UNION S_2 является множество S $\{s\}$ такое, что $s \in S_1$ *или* $s \in S_2$. Результатом операции пересечения S_1 INTERSECT S_2 является множество S $\{s\}$ такое, что $s \in S_1$ *и* $s \in S_2$. Результатом операции вычитания S_1 MINUS S_2 является множество S $\{s\}$ такое, что $s \in S_1$ *и* $s \notin S_2$[3]. На рис. 2.4 эти операции проиллюстрированы в интуитивной графической форме. Про операцию взятия декартова произведения уже говорилось выше.

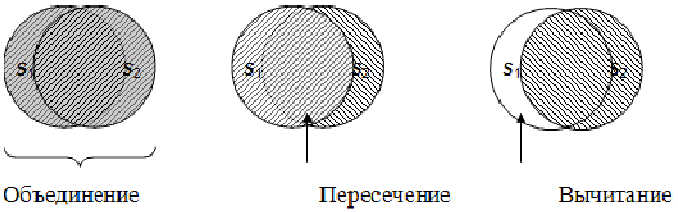

Рис. 2.4. Иллюстрация результатов теоретико-множественных операций

Понятно, что эти операции применимы к любым телам отношений, но результатом не будет являться отношение, если у отношений-операндов не совпадают заголовки. Кодд предложил в качестве средства манипулирования реляционными базами данных специальный набор операций, которые гарантированно производят отношения. Этот набор операций принято называть *реляционной алгеброй Кодда*, хотя он и не является *алгеброй* в математическом смысле этого термина, поскольку некоторые бинарные операции этого набора применимы не к произвольным парам отношений.

В алгебре Кодда имеется десять операций: объединение (UNION), пересечение (INTERSECT), вычитание (MINUS), взятие расширенного декартова произведения (TIMES), переименование атрибутов (RENAME), проекция (PROJECT), ограничение (WHERE), соединение (θ-JOIN), деление (DIVIDE BY) и присваивание. Если не вдаваться в некоторые тонкости, которые мы рассмотрим в лекции 4, то почти все операции предложенного выше набора обладают очевидной и простой интерпретацией.

- При выполнении операции *объединения* (UNION) двух отношений с одинаковыми заголовками производится отношение, включающее все кортежи, входящие хотя бы в одно из отношений-операндов.
- Операция *пересечения* (INTERSECT) двух отношений с одинаковыми заголовками производит отношение, включающее все кортежи, входящие в оба отношения-операнда.
- Отношение, являющееся *разностью* (MINUS) двух отношений с одинаковыми заголовками, включает все кортежи, входящие в отношение-первый операнд, такие, что ни один из них не входит в отношение, являющееся вторым операндом.
- При выполнении *декартова произведения* (TIMES) двух отношений, пересечение заголовков которых пусто, производится отношение, кортежи которого производятся путем объединения кортежей первого и второго операндов.
- Операция *переименования* (RENAME) производит отношение, тело которого совпадает с телом операнда, но имена атрибутов изменены; эта операция позволяет выполнять первые три операции над отношениями с «почти» совпадающими заголовками (совпадающими во всем, кроме имен атрибутов) и выполнять операцию TIMES над отношениями, пересечение заголовков которых не является пустым.
- Результатом *ограничения* (WHERE) отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющее этому условию.
- При выполнении *проекции* (PROJECT) отношения на заданное подмножество множества его атрибутов производится отношение, кортежи которого являются соответствующими подмножествами кортежей отношения-операнда.
- При θ*-соединении* (θ-JOIN) двух отношений по некоторому условию (θ) образуется результирующее отношение, кортежи которого производятся путем объединения кортежей первого и второго отношений и удовлетворяют этому условию.
- У операции *реляционного деления* (DIVIDE BY) два операнда – бинарное и унарное отношения. Результирующее отношение состоит из унарных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) включает множество значений второго операнда.
- Операция *присваивания* (:=) позволяет сохранить результат вычисления реляционного выражения в существующем отношении БД.

2.4.3. Целостность в реляционной модели данных

Кодд предложил два декларативных механизма поддержки целостности реляционных баз данных, которые затверждены в реляционной модели данных и должны поддерживаться в любой реализующей ее СУБД: *ограничение целостности сущности*, или *ограничение первичного ключа* и *ограничение ссылочной целостности*, или *ограничение внешнего ключа*. Мы снова оставим подробности и формализмы на лекцию 3 и приведем здесь только изложение общих идей.

Ограничение целостности сущности звучит следующим образом: для заголовка любого отношения базы данных должен быть явно или неявно определен *первичный ключ*, являющийся таким минимальным подмножеством заголовка отношения, что в любом теле этого отношения, которое может появиться в базе данных, значение первичного ключа в любом кортеже этого тела является уникальным, т.е. отличается от значения первичного ключа в любом другом кортеже. Под минимальностью первичного ключа понимается то, что если из множества атрибутов первичного ключа удалить хотя бы один атрибут, то ограничение целостности изменится, т.е. в БД смогут появляться тела отношений, которые не допускались исходным первичным ключом.

Если первичный ключ не объявляется явно, то в качестве первичного ключа отношения принимается весь его заголовок. Понятно, что поскольку по определению любое тело отношения с заданным заголовком является множеством, следовательно, в нем

отсутствуют дубликаты, и первичный ключ, совпадающий с заголовком отношения, всегда обладает свойством уникальности. Должно быть понятно, что в этом случае определение первичного ключа не задает никакого ограничения целостности.

Чтобы пояснить смысл *ограничения ссылочной целостности*, нужно сначала ввести понятие *внешнего ключа*. В принципе при использовании реляционной модели данных можно хранить все данные, соответствующие предметной области в одной таблице. Пример такой базы данных демонстрировался в лекции 1 на рис. 1.6 , где в одном файле (интуитивном аналоге отношения) хранилась информация и о служащих, и об отделах, в которых они работают. Как было показано в лекции 1, такой подход приводит к избыточности хранения (данные об отделе повторяются в каждой записи о служащем этого отдела) и усложняет выполнение некоторых операций.

На рис. 1.7 для хранения информации о служащих и отделах использовалось два файла, в одном из которых хранились данные, индивидуальные для каждого служащего, а во втором – данные об отделах. Для возможности получения полной информации о служащих и отделах, в которых они работают, в файле СЛУЖАЩИЕ содержалось поле СЛУ_ОТД_НОМЕР, содержащее для каждого служащего его уникальный номер отдела. В то же время, в файле ОТДЕЛЫ имелось поле ОТД_НОМЕР, являющееся уникальным ключом этого файла. На самом деле, введя файлы СЛУЖАЩИЕ и ОТДЕЛЫ, а также обеспечив связь между ними с помощью полей СЛУ_ОТД_НОМЕР и ОТД_НОМЕР, мы смогли обеспечить табличное представление иерархии ОТДЕЛ-СЛУЖАЩИЕ. Если говорить в терминах реляционной модели данных, то в отношении ОТДЕЛЫ поле ОТД_НОМЕР является *первичным ключом*, а в отношении СЛУЖАЩИЕ поле СЛУ_ОТД_НОМЕР является *внешним ключом*, ссылающимся на отношение ОТДЕЛЫ.

Более точно, *внешним ключом отношения* R_1, *ссылающимся на отношение* R_2, называется подмножество заголовка H_{R1}, которое совпадает с первичным ключом отношения R_2 (с точностью до имен атрибутов). Тогда *ограничение ссылочной целостности* реляционной модели данных можно сформулировать следующим образом: в любом теле отношения R_1, которое может появиться в базе данных, для «не пустого»[4] значения внешнего ключа, ссылающегося на отношение R_2, в любом кортеже этого тела должен найтись кортеж в теле отношения R_2, которое содержится в базе данных, с совпадающим значением первичного ключа. Легко заметить, что это почти то же самое ограничение, о котором говорилось в подразделе 2.3.2. Иерархическая модель данных: *никакой потомок не может существовать без своего родителя*, но немного уточненное – *ссылки на родителя должны быть корректными.*

---

2 Обозначение $s \in S$ означает, что элемент s принадлежит множеству S.

3 Обозначение $s \notin S$ означает, что элемент s *не* принадлежит множеству S.

4 Понятие «пустого», или *неопределенного* значения мы уточним в лекции 3.

2.5. Современные модели данных

Я считаю, что история современных моделей данных началась с 1989 г., когда группа известных специалистов в области языков программирования баз данных опубликовала статью под названием «Манифест систем объектно-ориентированных баз данных» [2.6]. К этому времени уже существовало несколько реализаций объектно-ориентированных СУБД (ООСУБД), но каждая из них опиралась на некоторое расширение объектной модели какого-либо объектно-ориентированного языка программирования (Smalltalk, Object Lisp, C++), отсутствовали какие-либо общие подходы.

В [2.6] не предлагалась единая объектно-ориентированная модель данных, но выделялся набор требований к ООСУБД. Базовыми требованиями являлось преодоление несоответствия между типами данных, используемыми в языках программирования, и типами данных, поддерживаемыми в набравших к тому времени силу реляционных (вернее, SQL-ориентированных) СУБД, а также придание СУБД возможностей хранить в БД данные произвольно сложной структуры. Эти требования сопровождались утверждениями об ограниченности реляционной модели данных и языка SQL и потребности использовать более развитые модели данных.

Под влиянием [2.6] в 1991 г. возник консорциум ODMG (Object Database Management Group), задачей которого была разработка стандарта объектно-ориентированной модели данных. В течение более чем десятилетнего существования ODMG опубликовала три базовых версии стандарта, последняя из которых называется ODMG 3.0 [4.4]. На этот документ мы и будем опираться в дальнейшем изложении.

В ответ на публикацию [2.6] группа исследователей, близких к индустрии баз данных, в 1990 г. опубликовала документ «Манифест систем баз данных третьего поколения» [2.7], который во многом направлен на защиту инвестиций крупных компаний-производителей программного обеспечения SQL-ориентированных СУБД. Соглашаясь с авторами [2.6] относительно потребности обеспечения развитой системы типов данных в СУБД, авторы [2.7] утверждали, что можно добиться аналогичных результатов, не производя революцию в области технологии баз данных, а эволюционно развивая технологию SQL-ориентированных СУБД.

За публикацией [2.7] последовало появление *объектно-реляционных* продуктов ведущих компаний-поставщиков SQL-ориентированных СУБД (Informix Universal Server, Oracle8, IBM DB2 Universal Database). В 1999 г. был принят стандарт языка SQL (SQL:1999), в котором был зафиксирован ряд новых черт языка, придающих ему черты полноценной модели данных. В последнем ко времени написания этой книги стандарте SQL:2003 эта модель уточнена и расширена. В Части 4 мы достаточно подробно обсудим стандарт SQL, а в этом разделе остановимся лишь на некоторых особенностях модели данных SQL, отличающих ее от реляционной модели данных.

Итак, в начале 1990-х гг. были провозглашены два манифеста, каждый из которых претендовал на роль программы будущего развития технологии баз данных. В первом манифесте реляционная модель данных отвергалась полностью, а во втором заменялась еще незрелой к тому времени моделью данных SQL, которая уже тогда была далека от реляционной модели. На защиту реляционной модели данных в ее первозданном виде встали Кристофер Дейт и Хью Дарвен, опубликовавшие в 1995 г. статью, под названием «Третий манифест» [2.8].

«Третий манифест» являлся одновременно наиболее консервативным и наиболее радикальным. Консервативность *Третьего манифеста* заключается в том, что его авторы всеми силами утверждают необходимость и достаточность использования в системах базах данных следующего поколения классической реляционной модели данных. Радикальность состоит в том, что (а) авторы полностью отрицают подходы, предлагаемые в первых двух манифестах, расценивая их как необоснованные, плохо проработанные,

избыточные и даже вредные (за исключением одной общей идеи о потребности обеспечения развитой системы типов); (b) фактически, авторы полностью отбрасывают технологию, созданную индустрией баз данных за последние 25 лет, и предлагают вернуться к истокам реляционной модели данных, т.е. начальным статьям Э. Кодда [2.1].

Позже Дейт и Дарвен написали книгу, первое издание которой вышло в 1998 г. под названием «Foundation for Object/Relational Databases: The Third Manifesto» [4.5], второе – в 2000 г. под названием «Foundation for Future Database Systems: The Third Manifesto» [4.6] (издан перевод второго издания на русский язык [1.5]) и третье – под названием «Databases, Types and the Relational Model: The Third Manifesto» в 2006 г. [4.7]. В этих книгах очень подробно излагается подход авторов к построению СУБД на основе, как они утверждают, истинных идей Эдгара Кодда, изложенных им в своих первых статьях про реляционную модель данных. Некоторые более поздние идеи Кодда относительно той же реляционной модели авторами отвергаются. В любом случае, Кодд и Дарвен предлагают некоторый современный вариант реляционной модели данных (далее для определенности мы будем называть ее *истинной* реляционной моделью), который, безусловно, заслуживает внимания и изучения. В данной книге мы ограничимся только кратким очерком основных черт этой модели.

2.5.1. Объектно-ориентированная модель данных

Если не обращать внимания на особенности объектно-ориентированной терминологии (предполагается, что читатели в общих чертах знакомы с ней), то объектно-ориентированная модель данных ODMG, специфицированная в [4.4], отличается от других двух моделей, описываемых в этом разделе, прежде всего, в одном принципиальном аспекте. В модели данных SQL и истинной реляционной модели данных база данных представляет собой набор именованных контейнеров данных одного родового типа: таблиц или отношений соответственно. В объектно-ориентированной модели данных база данных – это набор объектов (контейнеров данных) произвольного типа.

Типы и структуры данных объектной модели

В объектной модели данных вводятся две разновидности типов: *литеральные* и *объектные* типы. Литеральные типы данных – это обычные типы данных, принятые в традиционных языках программирования. Они подразделяются на базовые скалярные числовые типы, символьные и булевские типы (*атомарные литералы*), конструируемые типы записей (*структур*) и *коллекций*.

Литеральный тип записи – это традиционный определяемый пользователем структурный тип, подобный структурному типу языка C или типу записи языка Pascal. Отличие состоит лишь в том, что в объектной модели атрибут типа записи может определяться не только на литеральном, но и на объектном типе, т.е. значение литерального типа записи может в качестве компонентов включать объекты. На первый взгляд это звучит странно и страшновато, но здесь все странности проистекают из особенностей объектно-ориентированной терминологии. У любого существующего объекта имеется одно и только одно местоположение, характеризующееся его идентификатором (OID). Когда в модели говорится, что некоторое структурное значение в качестве компонента имеет некоторый объект, то, конечно, имеется в виду OID этого объекта, являющийся всего лишь аналогом указательного значения в традиционных языках программирования.

Имеются четыре вида типов коллекций: типы множеств, мультимножеств (неупорядоченные наборы элементов, возможно, содержащие дубликаты), списков (упорядоченные наборы элементов, возможно, содержащие дубликаты) и словарей (множества пар `<ключ, значение>`, причем все ключи в этих парах должны быть различными). Типом элемента любой коллекции может являться любой скалярный или объектный тип за исключением того же типа коллекции.

Объектные типы в объектной модели данных по смыслу ближе всего к понятию *класса* в объектно-ориентированных языках программирования. У каждого объектного типа имеется операция создания и инициализации нового объекта этого типа. Эта операция возвращает значение OID нового объекта, который можно хранить в любом месте, где допускается хранение объектов данного типа, и использовать для обращения к *операциям* объекта, определенным в его объектном типе.

Имеются два вида объектных типов. Первый из них называется атомарным объектным типом. Нестрого говоря, при определении атомарного объектного типа указывается его внутренняя структура (набор свойств – атрибутов и связей) и набор операций, которые можно применять к объектам этого типа. Для определения атомарного объектного типа можно использовать механизм наследования, расширяя набор свойств и/или переопределяя существующие и добавляя новые операции.

Атрибутами называются свойства объекта, значение которых можно получить по OID объекта. Значениями атрибутов могут быть и литералы, и объекты (т.е. OID), но только тогда, когда не требуется обратная ссылка. *Связи* – это инверсные свойства. В этом случае значением свойства может быть только объект. Связи определяются между атомарными объектными типами. В объектной модели ODMG поддерживаются только бинарные связи, т.е. связи между двумя типами. Связи могут быть разновидностей «один-к-одному», «один-ко-многим» и «многие-ко-многим» в зависимости от того, сколько экземпляров соответствующего объектного типа может участвовать в связи.

Связи явно определяются путем указания *путей обхода*. Пути обхода указываются парами, по одному пути для каждого направления обхода связи. Например, в базе данных `СЛУЖАЩИЕ-ОТДЕЛЫ` служащий работает (`works`) в одном отделе, а отдел состоит (`consists of`) из множества служащих. Тогда путь обхода `consists_of` должен быть определен в объектном типе `ОТДЕЛ`, а путь обхода `works` – в типе `СЛУЖАЩИЙ`. Тот факт, что пути обхода относятся к одной связи, указывается в разделе `inverse` обоих объявлений пути обхода. Это связь «один-ко-многим». Путь обхода `consists_of` ассоциирует объект типа `ОТДЕЛ` с литеральным множеством объектов типа `СЛУЖАЩИЙ`, а путь обхода `works` ассоциирует объект типа `СЛУЖАЩИЙ` с объектом типа `ОТДЕЛ`. Пути обхода, ведущие к коллекциям объектов, могут быть упорядоченными или неупорядоченными в зависимости от вида коллекции, указанного в объявлении пути обхода.

Заметим, что хотя связь является модельным понятием, другие понятия модели наталкивают на мысль, что единственным способом реализации связей является хранение в объекте OID или коллекции OID связанных объектов в зависимости от вида связи. Это можно сделать и с использованием должным образом типизированных атрибутов. Однако явное определение связи обеспечивает системе дополнительную информацию, которая используется в объектной модели как ограничение целостности (см. ниже).

Второй вид – это объектные типы коллекций. Как и в случае использования литеральных типов коллекций, можно определять объектные типы множеств, мультимножеств, списков и словарей. Типом элемента объектного типа коллекции может быть любой литеральный

или объектный тип за исключением того же типа коллекции. У объектных типов коллекций имеются предопределенные наборы операций. В отличие от литеральных типов коллекций, которые, как и все литеральные типы являются множествами значений, объектные типы коллекций обладают операцией создания объекта, имеющего, как и все объекты, собственный OID.

Интересен и важен один специальный случай неявного использования объектов типа множества. При определении атомарного объектного типа можно в качестве одного из дополнительных свойств этого типа указать, что для него должен быть создан объект типа множества, элементами которого являются объекты данного атомарного типа (*экстент* объектного структурного типа). Поскольку такой объект создается неявно, его OID неизвестен, но зато у него имеется имя, явно задающееся в определении и совпадающее с именем атомарного объектного типа. Наличие этой возможности позволяет создавать объектные базы данных, состоящие из именованных контейнеров однотипных объектов, причем в действительности эти контейнеры содержат OID'ы соответствующих объектов.

Манипулирование данными в объектной модели

В стандарте ODMG в качестве базового средства манипулирования объектными базами данных предлагается язык OQL (Object Query Language). Это небольшой, но достаточно сложный язык запросов. Разработчики в целом характеризуют его следующим образом:

- OQL опирается на объектную модель ODMG (имеется в виду, что в нем поддерживаются средства доступа ко всем возможным структурам данных, допускаемых в структурной части модели).
- OQL очень близок к SQL/92. Расширения относятся к объектно-ориентированным понятиям, таким как сложные объекты, объектные идентификаторы, путевые выражения, полиморфизм, вызов операций и отложенное связывание.
- В OQL обеспечиваются высокоуровневые примитивы для работы с множествами объектов, но, кроме того, имеются настолько же эффективные примитивы для работы со структурами, списками и массивами.
- OQL является функциональным языком, допускающим неограниченную композицию операций, если операнды не выходят за пределы системы типов. Это является следствием того факта, что результат любого запроса обладает типом, принадлежащим к модели типов ODMG, и поэтому к результату запроса может быть применен новый запрос.
- OQL не является вычислительно полным языком. Он представляет собой простой язык запросов.
- Операторы языка OQL могут вызываться из любого языка программирования, для которого в стандарте ODMG определены правила связывания. И, наоборот, в запросах OQL могут присутствовать вызовы операций, запрограммированных на этих языках.
- В OQL не определяются явные операции обновления, а используются вызовы операций, определенных в объектах для целей обновления.
- В OQL обеспечивается декларативный доступ к объектам. По этой причине OQL-запросы могут хорошо оптимизироваться.
- Можно легко определить формальную семантику OQL.

Объем этой лекции не позволяет привести развернутое описание языка OQL. Приведем лишь один характерный пример.

Получить номера руководителей отделов и тех служащих их отделов, зарплата которых превышает 20000 руб.

```
SELECT DISTINCT STRUCT ( ОТД_РУК: D.ОТД_РУК,
СЛУ: ( SELECT E
FROM D.CONSISTS_OF AS E
WHERE E.СЛУ_ЗАРП > 20000.00 ) )
FROM ОТДЕЛЫ D
```

Здесь предполагается, что для атомарного объектного типа `ОТДЕЛ` определен экстент типа множества с именем `ОТДЕЛЫ`. В запросе перебираются все существующие объекты типа `ОТДЕЛ`, и для каждого такого объекта происходит переход по связи к литеральному множеству объектов типа `СЛУЖАЩИЙ`, соответствующих служащим, которые работают в данном отделе. На основе этого множества формируется «усеченное» множество объектов типа `СЛУЖАЩИЙ`, в котором остаются только объекты-служащие с зарплатой, большей 20000.00. Результатом запроса является литеральное значение-множество, элементами которого являются значения-структуры с двумя литеральными значениями, первое из которых есть атомарное литеральное значение типа `INTEGER`, а второе – литеральное значение-множество с элементами-объектами типа `СЛУЖАЩИЙ`. Более точно, результат запроса имеет тип `set < struct { integer ОТД_РУК; bag < СЛУЖАЩИЙ > СЛУ } >`.

В совокупности результатом допустимых в OQL выражений запросов могут являться:

- коллекция объектов;
- индивидуальный объект;
- коллекция литеральных значений;
- индивидуальное литеральное значение.

Ограничения целостности в объектной модели

В соответствии с общей идеологией объектно-ориентированного подхода в модели ODMG два объекта считаются совпадающими в том и только в том случае, когда являются одним и тем же объектом, т.е. имеют один и тот же OID. Объекты одного объектного типа с разными OID считаются разными, даже если обладают полностью совпадающими состояниями. Поэтому в объектной модели отсутствует аналог ограничения целостности сущности реляционной модели данных. Интересно, что при определении атомарного объектного типа можно объявить ключ – набор свойств объектного класса, однозначно идентифицирующий состояние каждого объекта, входящего в экстент этого класса. Для класса может быть объявлено несколько ключей, а может не быть объявлено ни одного ключа даже при наличии определения экстента. Но при этом определение ключа не трактуется в модели как ограничение целостности; утверждается, что объявление ключа способствует повышению эффективности выполнения запросов.

Что же касается ссылочной целостности, то она поддерживается, если между двумя атомарными объектными типами определяется связь вида «один-ко-многим». В этом случае объекты на стороне связи «один» рассматриваются как предки, а объекты на стороне связи «многие» – как потомки, и ООСУБД обязана следить за тем, чтобы не образовывались потомки без предков.

2.5.2. Модель данных SQL

Как отмечалось в начале этого раздела, модель данных SQL в относительно законченном виде сложилась к 1999 г., когда был принят и опубликован стандарт SQL:1999. В приводимом в этом подразделе очерке этой модели данных мы затронем только наиболее важные, с точки зрения автора, ее черты, опуская многие менее существенные моменты.

Типы и структуры данных SQL

SQL-ориентированная база данных представляет собой набор таблиц, каждая из которых в любой момент времени содержит некоторое мультимножество строк, соответствующих заголовку таблицы. В этом состоит первое и наиболее важное отличие модели данных SQL от реляционной модели данных. Вторым существенным отличием является то, что для таблицы поддерживается порядок столбцов, соответствующий порядку их определения. Другими словами, таблица – это вовсе не отношение, хотя во многом они похожи.

Имеется две основных разновидности таблиц, хранимых в базе данных: традиционная таблица и типизированная таблица. Традиционная таблица определяется как множество столбцов с указанными типами данных. В SQL поддерживаются следующие категории типов данных: точные числовые типы; приближенные числовые типы; типы символьных строк; типы битовых строк; типы даты и времени; типы временных интервалов; булевский тип; типы коллекций; анонимные строчные типы; типы, определяемые пользователем; ссылочные типы. Подробно система типов SQL описывается в лекции 15, а здесь мы ограничимся только пояснениями наименее очевидных случаев.

Булевский тип в SQL содержит три значения – *true*, *false* и *uknown*. Это связано с интенсивным использованием в SQL так называемого неопределенного значения (NULL), которое разрешается использовать вместо значения любого типа данных. Как уже говорилось выше, здесь мы не будем более подробно затрагивать запутанную тему неопределенных значений и оставим подробности на следующие лекции.

В модели данных SQL допускается объявление двух видов типов коллекций: типы массива и типы мультимножества. Элементы типа коллекции могут быть любого типа данных, определенного к моменту определения данного типа коллекции. При объявлении типа мультимножества можно явно запретить наличие в его значениях элементов-дубликатов, что фактически приводит к объявлению типа множества.

Анонимный строчный тип – это безымянный структурный тип, значения которого являются строками, состоящими из элементов ранее определенных типов.

Поддерживается два вида типов данных, определяемых пользователями: индивидуальные и структурные типы. Индивидуальный тип – это именованный тип данных, основанный на единственном предопределенном типе. Индивидуальный тип не наследует от своего опорного типа набор операций над значениями. Чтобы выполнить некоторую операцию базового типа над значениями определенного над ним индивидуального типа, требуется явно сообщить системе, что с этими значениями нужно обращаться как со значениями базового типа. Имеется также возможность явного определения методов, функций и процедур, связанных с данным индивидуальным типом.

Структурный тип данных – это именованный тип данных, включающий один или более атрибутов любого из допустимых в SQL типа данных, в том числе другого структурного типа, типа коллекций, анонимного строчного типа и т. д. Дополнительные механизмы определяемых пользователями методов, функций и процедур позволяют определить поведенческие аспекты структурного типа. При определении структурного типа можно использовать механизм наследования от ранее определенного структурного типа.

При определении типизированной таблицы указывается ранее определенный структурный тип, и если в нем содержится *n* атрибутов, то в таблице образуется *n*+1 столбец, из которых последние *n* столбцов с именами и типами данных, совпадающими именам и типам атрибутов структурного типа. Первый же столбец, имя которого явно задается, называется «самоссылающимся» и содержит типизированные уникальные идентификаторы строк, которые могут генерироваться системой при вставке строк в типизированную таблицу, явно указываться пользователями или состоять из комбинации значений других столбцов. Типом «самоссылающегося» столбца является ссылочный тип, ассоциированный со структурным типом типизированной таблицы. Способ генерации значений ссылочного типа указывается при определении соответствующего структурного типа и подтверждается при определении типизированной таблицы.

При определении типизированных таблиц можно использовать механизм наследования. Можно определить подтаблицу типизированной супертаблицы, если структурный тип подтаблицы является непосредственным подтипом структурного типа супертаблицы. Подтаблица наследует у супертаблицы способ генерации значений ссылочного типа и все ограничения целостности, которые были специфицированы в определении супертаблицы. Дополнительно можно определить ограничения, затрагивающие новые столбцы.

С типизированной таблицей можно обращаться, как с традиционной таблицей, считая, что у нее имеются неявно определенные столбцы, а можно относиться к строкам типизированной таблицы, как к объектам структурного типа, OID которых содержатся в «самоссылающемся» столбце. Ссылочный тип можно использовать для типизации столбцов традиционных таблиц и атрибутов структурных типов, на которых потом определяются типизированные таблицы. В последнем случае можно считать, что значениями атрибутов соответствующих объектов являются объекты структурного типа, с которыми ассоциирован данный ссылочный тип.

Манипулирование данными в SQL

Средства манипулирования данными составляют значительную часть языка SQL и сравнительно подробно обсуждаются в лекциях 17-21. Здесь же мы ограничимся общей характеристикой оператора SQL SELECT, предназначенного для выборки данных и имеющего следующий синтаксис:

```
SELECT [ ALL | DISTINCT ] select_item_commalist
FROM table_reference_commalist
[ WHERE conditional_expression ]
[ GROUP BY column_name_commalist ]
[ HAVING conditional_expression ]
[ ORDER BY order_item_commalist ]
```

Выборка данных производится из одной или нескольких таблиц, указываемых в разделе FROM запроса. В последнем случае на первом этапе выполнения оператора SELECT образуется одна общая таблица, получаемая из исходных таблиц путем применения операции расширенного декартова умножения. Таблицы могут быть как базовыми, реально хранимыми в базе данных (традиционными или типизированными), так и порожденными, т.е. задаваемыми в виде некоторого оператора SELECT. Это допускается, поскольку результатом выполнения оператора SELECT в его базовой форме является

традиционная таблица. Кроме того, в разделе FROM можно указывать выражения соединения базовых и/или порожденных таблиц, результатами которых опять же являются традиционные таблицы.

На следующем шаге общая таблица, полученная после выполнения раздела, подвергается фильтрации путем вычисления для каждой ее строки логического выражения, заданного в разделе WHERE запроса. В отфильтрованной таблице остаются только те строки общей таблицы, для которых значением логического выражения является *true*.

Если в операторе отсутствует раздел GROUP BY, то после этого происходит формирование результирующей таблицы запроса путем вычисления выражений, заданных в списке выборки оператора SELECT. В этом случае список выборки вычисляется для каждой строки отфильтрованной таблицы, и в результирующей таблице появится ровно столько же строк.

При наличии раздела GROUP BY из отфильтрованной таблицы получается сгруппированная таблица, в которой каждая группа состоит из кортежей отфильтрованной таблицы с одинаковыми значениями столбцов группировки, задаваемых в разделе GROUP BY. Если в запросе отсутствует раздел HAVING, то результирующая таблица строится прямо на основе сгруппированной таблицы. Иначе образуется отфильтрованная сгруппированная таблица, содержащая только те группы, для которых значением логического выражения, заданного в разделе HAVING, является *true*.

Результирующая таблица на основе сгруппированной или отфильтрованной сгруппированной таблицы строится путем вычисления списка выборки для каждой группы. Тем самым, в результирующей таблице появится ровно столько строк, сколько групп содержалось в сгруппированной или отфильтрованной сгруппированной таблице.

Если в запросе присутствует ключевое слово DISTINCT, то из результирующей таблицы устраняются строки-дубликаты, т.е. запрос вырабатывает не мультимножество, а множество строк.

Наконец, в запросе может присутствовать еще и раздел ORDER BY. В этом случае результирующая таблица сортируется в порядке возрастания или убывания в соответствии со значениями ее столбцов, указанных в разделе ORDER BY. Результатом такого запроса является не таблица, а отсортированный список, который нельзя сохранить в базе данных. Сам же запрос, содержащий раздел ORDER BY, нельзя использовать в разделе FROM других запросов.

Приведенная характеристика средств манипулирования данными языка SQL является не вполне точной и полной. Кроме того, она отражает *семантику* оператора SQL, а не то, как он обычно исполняется в SQL-ориентированных СУБД.

Ограничения целостности в модели SQL

Как отмечалось в начале этого раздела, наиболее важным отличием модели данных SQL от реляционной модели данных является то, что таблицы SQL могут содержать мультимножества строк. Из этого, в частности, следует, что в модели SQL отсутствует обязательное предписание об ограничении целостности сущности. В базе данных могут существовать таблицы, для которых не определен первичный ключ. С другой стороны, если для таблицы определен первичный ключ, то для нее ограничение целостности сущности поддерживается точно так же, как это требуется в реляционной модели данных.

Ссылочная целостность в модели данных SQL поддерживается в обязательном порядке, но в трех разных вариантах, лишь один из которых полностью соответствует реляционной модели. Это связано с уже упоминавшимся в этом разделе интенсивным использованием в SQL неопределенных значений. Подробнее особенности ограничений ссылочной целостности в SQL рассматриваются в лекции 16.

Кроме того, в SQL имеются развитые возможности явного определения ограничений целостности на уровне столбцов таблиц, на уровне таблиц целиком и на уровне базы данных.

2.5.3. Истинная реляционная модель

Дейт и Дарвен очень подробно и тщательно разработали предлагаемый ими вариант реляционной модели данных. В последнем издании книги [4.7], изданном в крупном формате, около 600 страниц, причем это очень насыщенный текст. Поэтому в кратком очерке истинной реляционной модели, предлагаемом в этом разделе, мы можем описать только ее самые общие и внешние черты. За подробностями отсылаю читателей к [1.5].

Типы и структуры данных истинной реляционной модели

Кристофер Дейт и Хью Дарвен поставили перед собой трудную задачу: показать, что на основе идей Эдгара Кодда можно реализовать СУБД, обеспечивающие возможности по части представления и хранения данных сложной структуры, не меньшие тех, которые обеспечивают объектные и SQL-ориентированные СУБД. Этому мешал, прежде всего, тезис Кодда о нормализации отношений: в реляционной базе данных должны содержаться только отношения, атрибуты которых определены на «доменах, элементы которых являются атомарными (не составными) значениями» [2.1]. В [3.12] Дейт пишет: «Я согласен с Коддом, что желательно оставаться в рамках логики первого порядка, если это возможно. В то же время я отвергаю идею "атомарных значений", по крайней мере, в смысле абсолютной атомарности. В Третьем манифесте мы допускаем наличие доменов, содержащих значения произвольной сложности. (Они могут быть даже отношениями.) Тем не менее, мы остаемся в рамках логики первого порядка.» Если учесть, что [2.1] является первой официальной публикацией Кодда по поводу реляционной модели данных, то трудно сказать, что Дейт очень уж строго следует всем его заветам. Те постулаты Кодда, которые вредят достижению цели Третьего манифеста, просто отвергаются.

В истинно реляционной модели очень большое внимание уделяется типам данных. Предлагаются три категории типов данных: скалярные типы, кортежные типы и типы отношений. Скалярный тип данных – это привычный инкапсулированный тип, реальная внутренняя структура которого скрыта от пользователей. Предлагаются механизмы определения новых скалярных типов и операций над ними. Типом атрибута определяемого скалярного типа может являться любой определенный к этому моменту скалярный тип, любой кортежный тип и тип отношения. Некоторые базовые скалярные типы данных должны быть предопределены в системе. В число этих типов должен входить тип truth value (так Дейт и Дарвен называют булевский тип) ровно с двумя значениями *true* и *false*.

Кортежный тип – это безымянный тип данных, определяемый с помощью генератора типа TUPLE с указанием множества пар <имя_атрибута, тип_атрибута> (заголовка кортежа). Типом атрибута кортежного типа может являться любой определенный к этому моменту скалярный тип, любой кортежный тип и тип отношения. Значением кортежного типа

является кортеж, представляющий собой множество триплетов `<имя_атрибута, тип_атрибута, значение_атрибута>`, которое соответствует заголовку кортежа этого кортежного типа.

Тип отношения – это безымянный тип данных, определяемый с помощью генератора типа `RELATION` с указанием некоторого заголовка кортежа. Значением типа отношения является заголовок отношения, совпадающий с заголовком кортежа этого типа отношения, и тело отношения, представляющее собой множество кортежей, соответствующих этому заголовку. Кортежные типы и типы отношений не являются инкапсулированными: имеется возможность прямого доступа к атрибутам.

Для всех разновидностей типов данных разработана модель множественного наследования, позволяющая определять новые типы данных на основе уже определенных типов. Модель наследования по Дейту и Дарвену не является частью истинной реляционной модели данных.

Понятно, что при таких определениях значениями атрибутов отношения могут быть не только значения произвольно сложных скалярных типов, типами атрибутов которых могут быть, в частности, отношения, но и просто отношения. Тем не менее, в [2.8] Дейт и Дарвен говорят: «Каждый кортеж в [отношении] R содержит в точности одно значение v для каждого атрибута A в [заголовке отношения] H. Иными словами, R находится в *первой нормальной форме*, 1NF.» Это хорошее и понятное определение первой нормальной формы, но трудно сказать, согласился ли бы с ним Кодд.

База данных в истинной реляционной модели – это набор долговременно хранимых именованных *переменных отношений*, каждая из которых определена на некотором типе отношений. В каждый момент времени каждая переменная отношения базы данных содержит некоторое значение отношения соответствующего типа.

Манипулирование данными в истинной реляционной модели

Вообще говоря, в качестве эталонного средства манипулирования данными в истинной реляционной модели можно использовать упоминавшуюся в подразделе 2.4.2. Манипулирование реляционными данными реляционную алгебру Кодда. Однако в [4.5–4.7] Дейт и Дарвен предложили новую реляционную алгебру, названную ими Алгеброй A, которая основывается на реляционных аналогах булевских операций *конъюнкции*, *дизъюнкции* и *отрицания*. В лекции 5 мы опишем эту алгебру и покажем, что через ее операции выражаются все операции алгебры Кодда.

Ограничения целостности в истинной реляционной модели

В число обязательных требований истинной реляционной модели входит требование определения хотя бы одного возможного ключа для каждой переменной отношения (возможный ключ – это одно из подмножеств заголовка переменной отношения, обладающее упоминавшимися в подразделе 2.4.3 Целостность в реляционной модели данных свойствами первичного ключа). Кроме того, говорится, что «любое условное выражение, которое является (или логически эквивалентно) замкнутой правильно построенной формулой (WFF) реляционного исчисления, должно быть допустимо в качестве спецификации ограничения целостности» [1.5].

Средства поддержки декларативной ссылочной целостности фигурируют только в разделе рекомендуемых возможностей: «В **D** [конкретную реализацию истинной реляционной модели] следует включить некоторую декларативную сокращенную форму для выражения ссылочных ограничений (называемых также ограничениями внешнего ключа)».

2.6. Заключение

В этой лекции было введено важнейшее в технологии баз данных понятие модели данных. Кратко рассмотрены особенности трех ранних моделей данных: модели инвертированных таблиц, иерархической модели и сетевой модели данных. В отдельном разделе представлена исходная реляционная модель данных, определенная Эдгаром Коддом. Описаны основные черты трех современных моделей данных, системы типов данных которых позволяют сохранять в базе данных и обрабатывать данные произвольно сложной структуры: объектно-ориентированная модель данных, модель данных SQL и истинно реляционная модель данных.