М.В. Копейкин, В.В. Спиридонов, Е.О. Шумова

БАЗЫ ДАННЫХ

ОСНОВЫ SQL РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Учебное пособие

Санкт-Петербург
2005

ПРЕДИСЛОВИЕ

Развитие автоматизированных систем обработки данных в настоящее время характеризуется переносом акцента с процедурной обработки на структуру и хранение данных, что подразумевает использование в их контуре банков данных. Банки данных стали важнейшей частью информационных систем. Их основное назначение - обеспечение хранения и поддержания в системе интегрированной базы данных, которая является динамической информационной моделью предметной области (реального мира), в рамках которой функционирует система управления базами данных (СУБД).

Необходимость эффективного доступа к базам данных разнообразных пользователей, требование мобильности, безопасности, целостности информационных систем, учет организации взаимодействия баз данных и многие другие факторы привели к постановке проблемы обеспечения независимости данных от структур хранения и специфики используемого программного обеспечения. Исследование этой проблемы в таких СУБД, как Oracle, Informix, Sybase, ORD, Db2 и многих других, показывает, что для ее практического решения наиболее перспективна теория реляционных баз данных [13], которая в настоящее время интенсивно развивается в системах объектно-реляционного и объектно-ориентированного типа.

Курс «Базы данных» относится к циклу системных дисциплин. Его освоение базируется на знаниях, полученных студентами при изучении целого ряда других дисциплин: «Информатика», «Математическая логика и теория алгоритмов», «Программирование» и т.д., так как практически все знания и навыки дисциплин специальности 220100 - «Вычислительные машины, комплексы, системы и сети», необходимы для качественного изучения «Баз данных».

Предмет и содержание курса «Базы данных» предопределяется профилем подготавливаемых специалистов, учебным планом и содержанием изучаемых по нему дисциплин. Данное пособие адресуется, прежде всего, студентам специальности 220100 - «Вычислительные машины, комплексы, системы и сети», но может оказаться полезным студентам и аспирантам других специальностей.

Целью написания данной работы стало создание учебного пособия, в котором ясно и точно были бы изложены теоретические и практические основы теории баз данных,

интересные как для профессионального разработчика, так и для студента, впервые изучающего базы данных.

При проектировании баз данных обычно используются четыре уровня восприятия и отображения информации предметной области в банке данных: инфологический, концептуальный, внешний и внутренний. Инфологический уровень предусматривает создание инфологической модели данных предметной области, независящей от каких-либо характеристик СУБД, в которой будет реализован проект. Концептуальный уровень предполагает создание концептуальной модели, в которой из инфологической модели удаляются (или преобразуются) элементы, которые не могут быть реализованы в СУБД, выбранной в качестве целевой. Внешний уровень формирует индивидуальные представления о хранимой информации для пользователей системы с помощью специального языка (обычно это SQL - Structured Query Language или QBE – Query By Example). Концептуальная модель преобразуется в физическую модель, предназначенную для реализации (хранения в виде баз данных и таблиц) в среде конкретной целевой СУБД.

Каждый из указанных уровней (и соответствующих им моделей) – это отдельный этап в проектировании информационной системы, использующий свои методы и средства.

Материала по рассматриваемой теме так много, что просто невозможно поместить его в одно учебное пособие. Это учебное пособие писалось одновременно с книгами [30, 31], поэтому и читать их целесообразно вместе, так как знакомство с одной из них облегчит понимание другой.

Порядок изучения этапов проектирования и эксплуатации баз данных не является строго обязательным и зависит от уровня подготовки читающего по таким дисциплинам, как системное программирование, математическая логика и теория алгоритмов, организация ЭВМ и систем, сети ЭВМ и средства телекоммуникаций и т.д.

Многолетняя практика чтения курса лекций по дисциплине «Базы данных» и практическое использование баз данных в различных системах управления показывает, что наиболее рациональным является следующий вариант успешного усвоения материала.

Изучение теоретических основ реляционной алгебры как базы языка SQL и ознакомление с основными конструкциями языка SQL (данное учебное пособие), воспринимая при этом базу данных как алгебраическую систему (набор взаимосвязанных плоских таблиц и определенных на них алгебраических операций). Изучение концепций баз данных [30] и методов инфологического этапа проектирования баз данных [31], учитывая особенности манипулирования данными с помощью SQL.

В заключение авторы выражают искреннюю благодарность сотрудникам кафедры компьютерных технологий и программного обеспечения за ценные замечания, способствовавшим улучшению пособия.

Введение

В начале 70-х годов появились работы, в которых обсуждались возможности применения различных моделей данных. Наиболее перспективной из них была статья сотрудника фирмы IBM Э. Кодда (Codd E.F., A Relational Model of Data for Large Shared Data Banks. Comm. of ACM vol. 13, N 6, 1970), где, вероятно, впервые был применен термин "реляционная модель данных". Э. Кодд предложил использовать для обработки баз данных аппарат теории множеств и теории отношений. Он использовал для описания данных предметной области особый вид двумерных таблиц, известных в математике как *отношение*, и совокупность правил (алгебру) для работы с этими таблицами. Предложенная им теория нормализации явилась толчком для создания языков манипулирования данными реляционного типа. Среди них наиболее распространены SQL (*структурированный язык запросов*) и QBE (Query By Example – *запросы по образцу*), с помощью которых пользователь указывает, какие данные необходимо получить, не уточняя процедуру их поиска в базе данных.

Глава 1. Основы реляционной модели данных

1.1. Отношения

Пусть имеется **n** множеств $\{\mathbf{D_1, D_2,...,D_n}\}$.

R есть отношение на этих множествах, если оно представляет собой множество элементов вида $< d_1, d_2, ...,d_n >$, где $d_i \in \mathbf{D_i}$ (i=1,..n).

Более строго, **R** - это подмножество декартова произведения указанных множеств и формально записывается:

$$\mathbf{R \subseteq D_1 \times D_2 \times ... \times D_n},$$

где $\subseteq$ - математический символ нестрогого включения;
$\times$ - математический символ операции декартова произведения.

Примечание. Отношение R допускает нестрогое включение и строгое включение.

В дальнейшем будем называть исходные множества $\mathbf{D_i}$ доменами (областями определения) отношения **R**, а элементы отношения $< d_1, d_2, ..., d_n >$ - кортежами или выборками.

Будем считать, что элементы доменов элементарны, т.е. сами по себе они не есть множества. Элементарность подразумевает, что это некоторое неразделимое понятие предметной области, например идентификатор информационных объектов или число. Так как отношение есть множество, а все элементы множества должны быть попарно различны, то в отношении не может быть двух идентичных кортежей.

Количество элементов в кортеже (в данном случае **n**) называется степенью отношения. Отношения могут быть унарные (домены, в частности) бинарные и **n**-арные. Количество кортежей в отношении называется мощностью отношения.

Приведем пример отношения. Пусть имеется два множества:

D1: {д1, д2, д3} - множество деталей;

D2: {м1, м2, м3, м4} - множество материалов.

На этих множествах можно задать отношение **R**, интерпретируемое как "возможность изготовления детали из материала", которое формально есть множество, содержащее все возможные пары из доменов **D1** и **D2**:

R = D1 × D2

< д1 , м1 >
< д1 , м2 >
< д1 , м3 >
< д1 , м4 >
< д2 , м1 >
< д2 , м2 >
< д2 , м3 >
< д2 , м4 >
< д3 , м1 >
< д3 , м2 >
< д3 , м3 >
< д3 , м4 >

Отношение **R**, которое существует в реальности, чаще всего является подмножеством декартова произведения и для некоторой предметной области,

например, оно может иметь вид

R ⊆ D1 × D2

< д1 , м3 >
< д2 , м2 >
< д2 , м4 >
< д3 , м1 >

Данное отношение, содержит только те выборки (кортежи), из декартова произведения, которые соответствуют реальности.

Так как элементы отношения представляют собой кортежи, идентичные по своей структуре, то все отношение может быть представлено в форме таблицы **R** (A1, A2), где A1 и A2 - так называемые имена атрибутов, которые можно рассматривать как имена подмножеств, определенных на множествах. A1 на **D1** , A2 на **D2** соответственно.

Атрибут есть подмножество соответствующего домена, на котором определено отношение. При этом, так как один и тот же домен может быть использован более одного раза при образовании отношения, то, следовательно, на нем может быть определено более одного атрибута. Им необходимо дать различные имена, так как соответствующие подмножества, в общем случае, выполняют различные роли и могут быть различимы. Таким образом, атрибут мы будем рассматривать как подмножество соответствующего домена, на котором определено отношение. В дальнейшем смысловое имя атрибута в тексте будем выделять курсивом, а имя отношения записывать прописными буквами или выделять жирным шрифтом. Иногда, для лучшей читаемости текста, отношение, заданное на множестве атрибутов, будем обозначать **R** (A_1, A_2) или **R** (A1, A2).

В общем случае отношение **R** может иметь n атрибутов, что определяет степень отношения. В рассмотренном примере n = 2. Количество выборок (кортежей) определяет мощность отношения M (в примере M=4).

Отношение обладает следующими свойствами:

- Отношение имеет имя, отличающее его от других отношений.
- Каждый атрибут имеет уникальное имя.
- Значения атрибута берутся из одного и того же домена.
- Каждый кортеж является уникальным, т.е. в отношении не может быть дублированных строк.
- Атрибут (или набор атрибутов), уникально идентифицирующий каждый кортеж, является первичным (primary) ключом отношения.
- Атрибут (или множество атрибутов) называется внешним (foreign) ключом, если в другом отношении он является первичным.

Идентификатор отношения и множество имен атрибутов составляют схему отношения. Введенное понятие схемы будет формально определено при рассмотрении вопросов, касающихся теории нормализации [30].

Множество кортежей может изменяться со временем. Например, в приведенном примере кортеж <д1, м3> может быть заменен в некоторый момент на <д1, м4>, что будет соответствовать определенному событию в отражаемой предметной среде. Этим

понятие отношения, используемое в реляционной модели баз данных, отличается от понятия отношения, используемого в математике. В отличие от самого отношения (в случае баз данных) его схема со временем не изменяется, поэтому естественно говорить о состоянии схемы отношения, имея в виду конкретное множество кортежей, которое имеется в отношении в данный момент. На рис. 1.1 приведена схема отношения и ее состояние на определенный момент времени, где:

Схема отношения:
Ведомость_ оплаты (Ид_Сотр, Ид_Отд, Период, Сумма), а

имена атрибутов: Ид_Сотр - идентификатор сотрудника,
Ид_Отд - идентификатор отдела,
Период - выплата за период,
Сумма - количество по ведомости.

Таблица Ведомость_ оплаты

Ид_Сотр	Ид_Отд	Период	Сумма
1	1	Март	1200
2	1	Март	1200
3	1	Март	1000
1	1	Апрель	1200
…	…	…	…

Рис. 1.1. Схема отношения и ее состояние

*Примечание***.** Некоторые имена атрибутов (в связи с особой их ролью в схеме отношения) получают приставку Ид_ и связаны с понятием первичного реляционного ключа. Реляционная база данных представляется пользователю как совокупность таблиц и ничего кроме таблиц.

Содержимое таблицы Ведомость_оплаты принято называть **состоянием схемы.** Совокупность схем отношений составляют **схему базы данных**. Соответственно состояние схемы базы данных (собственно сама база) - есть совокупность состояний схем отношений.

Следует заметить, что имена доменов явно не входят ни в схему отношения, ни в схему базы данных.

Ниже, на рис. 1.2, представлен пример описания оператора создания отношения (таблицы) базы данных CREATE TABLE языка SQL, синтаксис которого в большинстве СУБД совпадает со следующей формой:

CREATE TABLE [имя_базовой таблицы] (Имя_столбца Тип_данных [`NOT NULL`], [Имя_столбца Тип_данных [`NOT NULL`],…).

```
CREATE    TABLE      [dbo].[Ведомость_оплаты ]
(Ид_Сотр  CHAR       (5)    NOT NULL,
Ид_Отд    CHAR       (4)    NOT NULL,
Период    VARCHAR    (8)    NOT NULL,
Сумма     DECIMAL    (8,2)  NOT NULL,
PRIMARY KEY   (Ид_Сотр, Ид_Отд, Период));
```

Рис. 1.2. CREATE TABLE для таблицы Ведомость_оплаты

На рис. 1.2 указано имя создаваемой таблицы (`dbo.Ведомость_оплаты`) и список ее столбцов с указанием типов данных (пункт 2.2), используемых в столбцах, при этом для каждого столбца указано правило (`NOT NULL` – пункт 3.1.2), требующее обязательное означивание элементов каждого столбца. Для таблицы определен первичный ключ (`PRIMARY KEY`) на атрибутах (`Ид_Сотр, Ид_Отд, Период`), назначение которого на данном этапе следует понимать как средство, позволяющее уникально отличать одну строку таблицы от другой. На систему (СУБД) возложено автоматическое поддержание целостности данных в базе, которое в данном примере выражается в том, что каждый элемент таблицы должен быть означен и в ней не может быть двух строк, относящихся к одному и тому же сотруднику за определенный период.

Правила описания схемы базы данных и элементов, ее составляющих, указываются в документации, сопутствующей каждой СУБД, которые регламентируются стандартом SQL.

*Примечание***. Стандарт SQL** определяется **ANSI (Американским Национальным Институтом Стандартов)** и в данное время также принимается **ISO (Международной организацией по стандартизации).** Однако большинство коммерческих СУБД расширяют **SQL** без уведомления **ANSI,** добавляя разные другие особенности в этот язык, которые, как они считают, будут продуктивны. Иногда они несколько нарушают стандарт языка, хотя хорошие идеи имеют тенденцию развиваться и вскоре становиться стандартами "рынка" сами по себе в силу полезности своих качеств.

Например, грамматические и синтаксические правила описания схемы базы данных на языке определения данных (DDL языка Transact-SQL), в СУБД SQL SERVER 7.0 [25] приведены в приложении.

1.2. Алгебра отношений

В реляционном подходе ответ на конкретный запрос к базе данных также представляется в форме отношения. Поэтому в основе средств, используемых для формулировки запроса, может лежать алгебра отношений. Алгебра отношений как самостоятельная математическая дисциплина была развита достаточно давно, но,

видимо, Е. Кодд был первым, кто предложил использовать алгебру отношений для моделирования поисковых процессов в базах данных.

Существует много вариантов операций для реляционной алгебры, но мы рассмотрим только операции, использованные Коддом (основоположником реляционной модели).

Как известно, алгебра есть множество вида

A: < H, S > , где

H - носитель (в данном случае - множество отношений);
S - сигнатура (в данном случае - множество операций над отношениями).

Все множество **S** операций реляционной алгебры разбиты на два подмножества:

Стандартные теоретико-множественные операции:

- ***объединение*** ∪ **Rrez=R1 ∪ R2,**
- ***пересечение*** ∩ **Rrez=R1 ∩ R2,**
- ***разность*** \ **Rrez=R1 \ R2,**
- ***декартово произведение*** **Rrez= R1 × R2.**

Специальные операции:

- ***проекция*** **Rrez** (A) = **R**[A],
- ***ограничение*** **Rrez** = **R1**[булевское выражение],
- ***соединение*** **Rrez** = **R1**[булевское выражение]**R**2,
- ***деление*** **Rrez** = **R1 ÷ R2**.

Операции реляционной алгебры либо унарные (т.е. используют в качестве операнда только одно отношение), либо бинарные, когда имеются два операнда.

1.2.1. Теоретико-множественные операции

Рассмотрим более подробно каждую из операций.

Операции объединения, пересечения и разности двух отношений, представленных схемами отношений:

R1 (A1, A2, ..., An) и **R2** (B1, B2, ..., Bn),

возможны только в том случае, когда отношения **R1** и **R2** являются объединимыми. Это означает, что степени объединяемых отношений идентичны, а соответственные атрибуты определены на одних и тех же доменах. Например, атрибут A1 из **R1** и B1 из **R2** определены на домене **D1**.

В результате операции объединения получающееся отношение имеет ту же степень, что и исходные отношения.

Rrez (C1, ..., Cn) = **R1** (A1, ..., An) ∪ **R2** (B1, ..., Bn),

а мощность Mrez лежит в пределах: Max (M1, M2) <= Mrez <= M1+M2.

Имена атрибутов результирующего отношения могут быть выбраны произвольно, но атрибуты определяются на тех же доменах, что и в исходных отношениях. В частности, во всех трех отношениях могут быть использованы идентичные имена атрибутов, если это не ведет к путанице.

В результате операции пересечения:

Rrez (C1, …, Cn) = **R1** (A1, ..., An) ∩ **R2** (B1, ..., Bn),

степень отношения также сохраняется, а мощность лежит в пределах:

0 <= Mrez <= Min (M1, M2).

При операции взятия разности двух отношений:

Rrez (C1, …, Cn) = **R1** (A1, ..., An) \ **R2** (B1, ..., Bn),

степень сохраняется, а мощность лежит в пределах: 0 <= Mrez <= M1.

Пример с указанными операциями представлен на рис. 1.3. Для имен атрибутов приняты следующие сокращения: ШД – шифр детали, ШМ – шифр материала, ЕИ – единица измерения, НР – норма расхода, НМ – наименование материала, ШМ" – шифр материала на складе (рис. 1.4).

Исходные отношения:

R1 (ШД, ШМ, ЕИ, НР)			
д1	**м2**	**1**	**15**
д2	**м5**	**3**	**3**
д2	**м9**	**3**	**5**
д3	**м2**	**1**	**10**

R2 (ШД, ШМ, ЕИ, НР)			
д2	**м5**	**3**	**3**
д2	**м3**	**2**	**15**

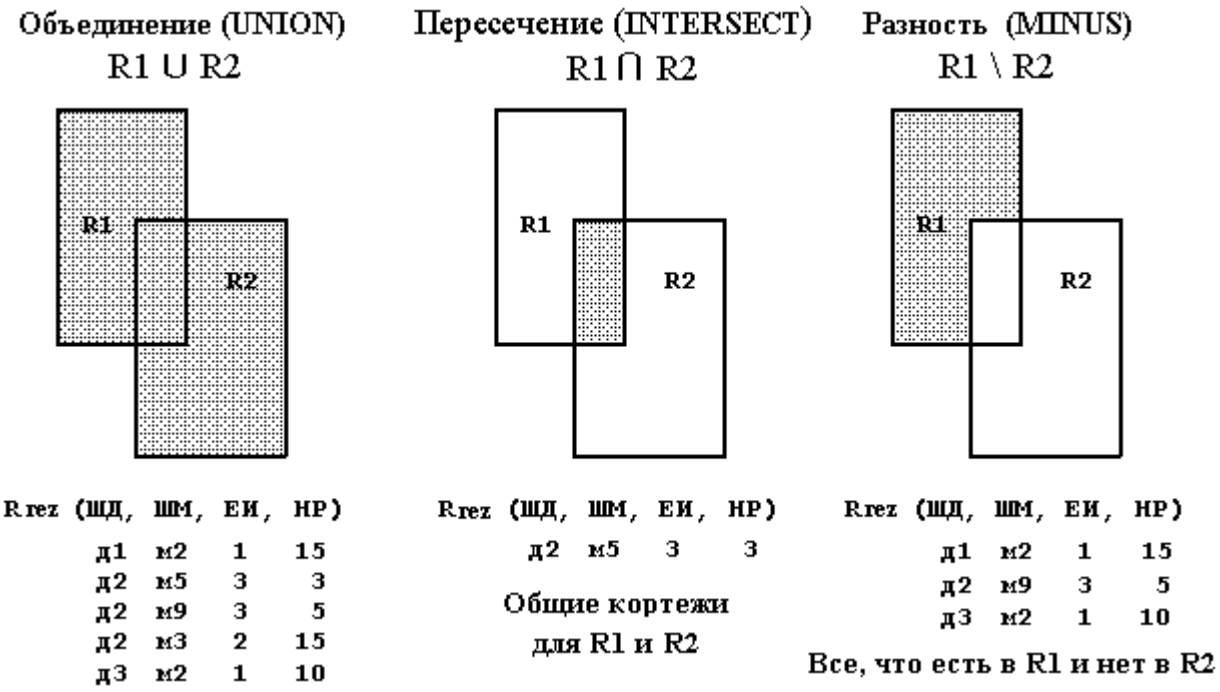

Рис. 1.3.Теоретико-множественные операции над отношениями

*Примечание***.** В SQL (см. пункт 5.2) **UNION** обозначает операцию объединения, **INTERSECT** - операцию пересечения, а **MINUS** - операцию взятия разности.

Последней из теоретико-множественных операций является операция декартового произведения отношений:

Rrez (A1, ..., An, B1, ..., Bm) = **R1** (A1, ..., An) × **R2** (B1, ..., Bm).

При этой операции степень результирующего отношения есть сумма степеней исходных, а мощность

Mrez=M1 * M2, где * - символ арифметического умножения.

В результате этой операции осуществляется конкатенация (приписывание одного кортежа к другому) строк (кортежей) отношений, причем каждая строка конкатенирует с каждой. Никаких ограничений на степени исходных отношений и природу атрибутов не накладывается. На рис. 1.4 представлен пример декартова произведения отношений.

Исходные отношения:

R1(ШД, ШМ, ЕИ, НР)

д1 м2 1 15
д2 м5 3 3
д2 м9 3 5
д3 м2 1 10

R3(ШМ", НМ)

м2 ст-5
м9 ст-7

Результат декартова произведения отношений R1 и R2:

Rrez (ШД, ШМ, ЕИ, НР, ШМ", НМ)

д1	**м2**	**1**	**15**	**м2**	**ст-5**
д2	**м5**	**3**	**3**	**м2**	**ст-5**
д2	**м9**	**3**	**5**	**м2**	**ст-5**
д3	**м2**	**1**	**10**	**м2**	**ст-5**
д1	**м2**	**1**	**15**	**м9**	**ст-7**
д2	**м5**	**3**	**3**	**м9**	**ст-7**
д2	**м9**	**3**	**5**	**м9**	**ст-7**
д3	**м2**	**1**	**10**	**м9**	**ст-7**

Рис. 1.4. Декартово произведение двух отношений

1.2.2. Специальные операции

Перейдем к рассмотрению специальных операций.

ПРОЕКЦИЯ

Операция является унарной и предназначена для уменьшения степени отношения.

Пусть имеется исходное отношение **R1**(A1,..., An) и список атрибутов A. Проекция отношения **R1** на список A обозначается следующим образом:

Rrez (A)=**R1** [A],

где - [ ] (квадратные скобки) являются операционными, т.е. используются для обозначения операции ПРОЕКЦИЯ.

При выполнении операции **проекции** на первом этапе из таблицы, представляющей исходное отношение, вычеркиваются все столбцы, соответствующие атрибутам, не вошедшим в список A. В результате получается **срез** отношения. Срез отношения не является отношением в полном смысле, так как там могут быть идентичные строки. Чтобы получить результирующее отношение, необходимо выбросить дублированные строки среза.

Пример: **R1**(ШД, ШМ, ЕИ, НР)

д1	м2	1	15
д2	м5	3	3
д2	м9	3	5
д3	м2	1	10

Проекция на атрибут ШД (*Шифр детали*):

Rrez (ШД) = **R1** [ШД].

В результате вычеркивания столбцов, соответствующих атрибутам, не вошедших в список A = (ШД), получаем срез:

д1
д2
д2
д3

В полученном срезе необходимо удалить продублированный элемент д2, в результате чего получим **Rrez** (ШД) = **R1** [ШД]

д1
д2

Если спроектировать это же отношение на список A = {ШМ, ЕИ}, то срез будет (ШМ, ЕИ), а проекция [ШМ, ЕИ]

м2	1	м2	1
м5	3	м5	3
м9	3	м9	3
м2	1		

Таким образом, проектируя некоторое исходное отношение, мы, прежде всего, уменьшаем степень отношения, но при этом может уменьшиться и мощность отношения:

1<= Mrez <= M1.

ОГРАНИЧЕНИЕ

Операция ОГРАНИЧЕНИЕ (селекция, выборка) также, как и проекция, является унарной операцией, но в отличие от проекции ориентирована на выделение нужных строк отношения.

Пусть исходное отношение задано схемой: **R1** (A1, ..., An).

При ограничении производится либо сужение области определения некоторых атрибутов, что символически записывается в виде: (A1<"Константа"), причем константа выбирается из домена, на котором определен атрибут A1, либо происходит сопоставление атрибутов отношения, например (A2 = A4), при этом A2 и A4 должны быть определены на одном и том же домене.

При проверке очередной выборки отношения на введенные условия при подстановке вместо имен атрибутов их конкретных значений получающиеся элементарные высказывания могут быть либо истинными, либо ложными. Если булевское выражение, составленное из подобных элементарных высказываний (их иногда называют термами сравнения) с помощью связок & (И - AND), (ИЛИ - OR), будет истинным, то кортеж попадает в результирующее отношение, если нет - не попадает.

В общем случае термы сравнения могут иметь следующий вид:

(A_i # "Константа") или (A_i # A_k),

где – символ # - одна из операций, являющаяся элементом из множества: { = , <> , > , < , >= , <= }.

Операция ограничения записывается следующим образом:

Rrez (A1, ..., An) = **R1**[булевское выражение],

при этом "булевское выражение" имеет вид:

AND ... AND (терм или дизъюнкция термов).

Как видно, степень результата остается прежней, а мощность лежит в пределах: 0 <= Mrez <= M1.

Пример для отношения **R1**:

R1 (ШД, ШМ, ЕИ, НР)

д1	м2	1	15
д2	м5	3	3
д2	м9	3	5
д3	м2	1	10

Найти все кортежи, в которых *единица измерения* (ЕИ) имеет код "1", а *норма расхода* (НР) больше или равна 10:

Rrez (ШД, ШМ, ЕИ, НР) = **R1**[(ЕИ = "1") AND (НР >= 10)].

В результате имеем:

Rrez (ШД, ШМ, ЕИ, НР)

д1	м2	1	15
д3	м2	1	10

Примечание. В языке SQL для обозначения операции ограничения используется конструкция FROM **R** WHERE <условие>, где **R** - ограничиваемое отношение, а <условие> - простое условие сравнения. Пусть comp1 и comp2 - два простых условия ограничения. Тогда по определению:

- **R** WHERE comp1 AND comp2 обозначает то же самое, что и (**R** WHERE comp1) INTERSECT (**R** WHERE comp2)
- **R** WHERE comp1 OR comp2 обозначает то же самое, что и (**R** WHERE comp1) UNION (**R** WHERE comp2)
- **R** WHERE NOT comp1 обозначает то же самое, что и **R** MINUS (**R** WHERE comp1)

С использованием этих определений можно использовать операции ограничения, в которых условием ограничения является произвольное булевское выражение, составленное из простых условий с использованием логических связок AND, OR, NOT и скобок.

На интуитивном уровне операцию ограничения лучше всего представлять как взятие некоторой "горизонтальной" вырезки из отношения-операнда**.**

СОЕДИНЕНИЕ

Операция СОЕДИНЕНИЕ двух отношений является важнейшей и часто используемой операцией. При этой операции кортежи одного отношения конкатенируют (соединяются) с выборками другого, точно также, как и при декартовом произведении двух отношений, но при условии, что булевское выражение, в каждый терм которого входят атрибуты обоих отношений, принимает значение "истина".

Итак, для двух исходных отношений, заданных схемами

R1 (A_1, ..., A_n) и **R2** (B_1, ..., B_m),

в результате операции соединения получается отношение со схемой

Rrez (A_1, ..., A_n,B_1, ...,B_m) = **R1** [булевское выражение] **R2**.

В булевском выражении термы имеют вид:

(A_i # B_j),

где - атрибуты A_i и B_j определены на одном домене, а термы сравнения # выбираются из множества: { = , < > , > , < , >= , <= }.

Очень часто булевское выражение состоит из единственного терма. Если таким термом является "=" (знак равенства), то этот частный случай называется операцией **эквисоединения** двух отношений по атрибутам A_i и B_j.

В дальнейшем, в связи с особой важностью этой операции, будем ее обозначать "**x°**", если использование терма "=" (знак равенства) приводит к неоднозначности прочтения формулы.

В этом случае булевское выражение есть конъюнкция термов, выражающих равенство соответствующих атрибутов, входящих в список и принадлежащих соединяемым отношениям:

(A_1=B_1) AND (A_2=B_2) AND ... AND (A_k=B_k).

Смысл операции соединения двух отношений более четко можно выразить сведением операции к ранее введенным:

Rrez(A_1, ..., A_n, B_1, ..., B_m) = **R1** [(булевское выражение)] **R2** =

= (**R1** × **R2**) [(булевское выражение)],

т.е. операция соединения сводится к декартовому произведению отношений с последующим ограничением получаемого промежуточного отношения.

Это означает, что мощность результата

0 <= Mrez <= M1 * M2.

Пример: Пусть заданы отношения **R1** и **R2**:

R1(ШД,	ШМ,	ЕИ,	НР)	и	**R2**	(ШМ",	НМ)
д1	м2	1	15			м2	ст-5
д2	м5	3	3			м9	ст-7
д2	м9	3	5				
д3	м2	1	10				

и требуется приписать к *шифру материала* (ШМ) его *наименование* (НМ).

Это можно получить с помощью операции эквисоединения:

Rrez (ШД, ШМ, ЕИ, НР, ШМ", НМ) = **R1** [(ШМ=ШМ")] **R2**.

Следующая запись эквивалентна предыдущей записи.

Rrez (ШД, ШМ, ЕИ, НР, ШМ", НМ) = **R1** [(ШМ **x°** ШМ")] **R2**.

В результате получим:

Rrez (ШД, ШМ, ЕИ, НР, ШМ", НМ)

д1	м2	1	15	м2	ст-5
д2	м9	3	5	м9	ст-7
д3	м2	1	10	м2	ст-5

при этом избавиться от лишних столбцов и поменять столбцы местами (если нужно) можно с помощью последующей операции проекции.

Рассмотрим более подробно, как получается результат, сведя операцию соединения к последовательности операций декартова произведения и ограничения:

R1 [(ШМ=ШМ")] **R2** = (**R1** × **R2**) [(ШМ=ШМ")].

Промежуточное отношение (**R1** × **R2**) со схемой

Rпр (ШД, ШМ, ЕИ, НР, ШМ", НМ) - декартово произведение, представлено на рис. 1.4.

Чтобы получить результат, необходимо ограничить промежуточное отношение **Rпр**:

Rrez (ШД, ШМ, ЕИ, НР, ШМ", НМ) = **Rпр** [(ШМ = ШМ")]

д1	м2	1	15	м2	ст-5
д3	м2	1	10	м2	ст-5
д2	м9	3	5	м9	ст-7

Отметим, что в результирующем отношении нет сведений о нормах расхода материала м5. Это произошло потому, что в справочнике наименований материалов (отношение **R2**) нет сведений об м5 и формальное эквисоединение двух отношений выбросило строки с м5 из результата.

При выполнении операции эквисоединения может возникнуть **ловушка соединения**, т.е. при эквисоединении семантически верных отношений может получиться семантически ложное отношение.

Пример:

Пусть **R1**(Изготовитель, Цех, Адрес, Деталь)

S2	Ц1	Т2	Д1
S3	Ц1	Т2	Д2
S1	Ц1	Т1	Д1
S4	Ц1	Т2	Д3
S1	Ц2	Т1	Д1

Пусть **R2**(Потребитель, Адрес, Деталь)

Р1	Т2	Д1
Р2	Т2	Д1
Р3	Т1	Д2
Р1	Т2	Д3
Р2	Т2	Д1

Попытка произвести эквисоединение **R1** и **R2** по атрибуту *Деталь* может привести к ложному выводу, так как не обязательно, чтобы *Потребитель* получал все детали от всех *Изготовителей*. Для устранения указанного недостатка была предложена **т е о р и я НОРМАЛИЗАЦИИ** [1, 2, 4, 6, 13, 27, 30].

ДЕЛЕНИЕ

Чтобы понять сущность операции деления отношений, целесообразно рассмотреть упрощенный пример.

Пусть задано отношение **R1**(ШД, ШМ) (рис. 1.5), которое задает возможные варианты изготовления деталей из разных материалов. И задано отношение **R2** (рис. 1.6), указывающее, какие материалы хранятся на складе.

R1 (ШД, ШМ)		**R2** (ШМ")
д1	м2	м5
д2	м5	м9
д2	м9	

д3 м2
д3 м5
д2 м3

Рис. 1.5. Варианты изготовления

Рис. 1.6. Материалы на складе

Если необходимо определить детали, которые могут быть изготовлены из всех материалов, хранимых на складе, то неформально мы можем определить, что такой деталью является только деталь д2, так как она может быть изготовлена как из материала м5, так и из м9.

Фактически **R2** есть подмножество материалов, из которых может быть сделана деталь д2: {м5, м9} $\subseteq$ {м3, м5, м9}.

Формально этот запрос соответствует операции деления реляционной алгебры, которая обозначается:

Rrez (ШД) = **R1**[ШМ ÷ ШМ"]**R2** , где ÷ - символ операции деления.

В общем случае имеется два отношения:

R1 (A1, ..., An) и **R2** (O1, ..., Om),

и задан список атрибутов A так, что, не теряя общности, отношения **R1** и **R2** можно представить в виде:

R1 (**Ã**, A) и R2 (A, **Õ**),

где **Ã** и **Õ** - дополнение списка A до полного списка атрибутов отношения **R1** и **R2** соответственно.

A ∪ **Ã** = Ar (полный список атрибутов отношения r). **Ã** -это подмножество Ar, являющееся результатом операции деления.

Для нашего примера:

A = ШД; **Ã** = ШМ; **Õ** - пустое множество.

Тогда отношения **R1** и **R2** становятся объединяемыми и результат можно представить:

Rrez (A) = **R1** [**Ã** ÷ A] **R2**.

Степень результирующего отношения определяется количеством атрибутов в списке **Ã**, а мощность Mrez << M1. Чтобы более точно объяснить смысл операции деления, запишем ее через ранее введенные операции. Для этого необходимо дать

перечень кортежей из **R1** [**Ã**], которые идентифицируют некоторые объекты предметной области, при этом каждый из объектов обладал бы совокупностью свойств **R2** [A]. Для того чтобы это сделать, наделим все объекты, задаваемые совокупностью атрибутов **Ã** и принадлежащие **R1**, совокупностью свойств **R2**[A]. Это можно сделать с помощью декартова произведения:

R1 [**Ã**] × **R2** [A].

В конкретном примере с *деталями* и *материалами* получим

R1 [ШД] × **R2** (ШМ")	
д1	м5
д1	м9
д2	м5
д2	м9
д3	м5
д3	м9

Получаемое в результате произведения промежуточное отношение объединимо с отношением **R1**. Поэтому можно взять разность:

R1 [**Ã**] × **R2** [A] \ **R1** (**Ã**, A),

которая, будучи спроецированной на список **Ã**:

(**R1** [**Ã**] × **R2** [A] \ **R1** (**Ã**, A)) [**Ã**], -

дает перечень объектов, не обладающих списком свойств **R2** [A].

На примере:

R1 [ШД] × R2 (ШМ")		R1 (ШД, ШМ)			Результат
д1	м5	д1	м2		д1 м5
д1	м9 \	д2	м5	=	д1 м9
д2	м5 \	д2	м9		д3 м9
д2	м9	д2	м3		
д3	м5	д3	м2		
д3	м9	д3	м5		

и после проектирования получаем список деталей {д1, д3}, которые не могут быть сделаны как из материала м5, так и из м9.

Таким образом, список объектов, обладающих свойствами **R2** [A], т.е. операция деления, может быть записан следующей формулой:

Rrez (**Ã**) = **R1** [**Ã**] \ (**R1** [**Ã**] × **R2** [A] \ **R1** (**Ã**, A)) [**Ã**].

Полученное выражение точно описывает смысл операции деления и включает в себя ранее введенные операции реляционной алгебры.

Продолжая конкретный пример, получим

Rrez (ШД) = R1[ШД] \ (R1[ШД] × R2[ШМ] \ R1(ШД, ШМ)) [ШД]

{д1, д2, д3} \ {д1, д3} = {д2}

1.2.3. Алгоритм операции деления

Алгоритм операции деления фактически определен ее формулой:

R1 [A ÷ O] **R2** = **R1** [**Ã**] \ ((**R1** [**Ã**] × **R2** [O]) \ **R1**)[**Ã**].

Необходимо только соблюдать старшинство операций:

1) Операции в скобках,

2) ×, [], \

Рассмотрим алгоритм на примере отношений **R1**(ЩД, ШМ) (рис. 1.5), **R2**(ШМ") (рис. 1.6):

1) R1 [**Ã**] = R1 [ШД]

d1
d2
d3

2) R2 [**Ã**] × R2 [O] = [ШД, ШМ] = R3 Из R3 вычитаем R1

д1	м5		R1(ШД, ШМ)
д1	м9		д1 м2
д2	м5		д2 м5
д2	м9		д2 м9
д3	м5		д3 м2
д3	м9		д3 м5
			д2 м3

3) R2[**Ã**] × R2[O] \ R1 = R3 \ R1 = R4(ШД, ШМ)

д1	м5
д1	м9
д3	м9

4) В полученном отношении R4 берется проекция по **Ã**=ШД, т.е. получаем отношение R5:

R5(ШД)

d1 т.е. список деталей, которые не могут быть сделаны
d3 как из материала м5, так и м9.

5) Полученное отношение R5 должно быть вычтено из R1[**Ã**],
т.е. {d1,d2,d3} \ {d1,d3} = d2.

Заканчивая рассмотрение операции деления, укажем на следующее свойство данной операции:

(**R1** (A) × **R2** (O)) [O ÷ O] **R2** (O) = **R1** (A),

т.е. операция деления обратна операции умножения, что проливает свет на название операции.

И еще один семантический пример операции деления. Пусть, в базе данных поддерживаются два отношения: СОТРУДНИКИ (Имя, Номер отдела) и ИМЕНА (Имя), причем унарное отношение ИМЕНА содержит все *фамилии*, которыми обладают сотрудники организации.

Тогда после выполнения операции реляционного деления отношения СОТРУДНИКИ на отношение ИМЕНА будет получено унарное отношение, содержащее *номера отделов*, сотрудники которых обладают всеми возможными в этой организации именами (фамилиями).

Рассмотрим использование алгебры отношений при составлении запросов к базе данных. Пусть БД содержит отношения **R1**(ШД, ШМ, НР) и **R2**(ШМ", НМ).

Запрос. Перечислить *шифры материалов* и их *наименования* (НМ), которые идут на изготовление одной *детали* в количестве большем чем 25 кг.

На языке реляционной алгебры запрос будет реализован на основе следующих операций:

Rrez (ШМ, НМ) = ((R1[НР > 25]) [ШМ]) [ШМ=ШМ"] R2[ШМ, НМ]

Круглые скобки определяют последовательность действий:

1) Ограничение отношения R1,
2) Проектирование промежуточного отношения,
3) Соединение с R2,
4) Проектирование полученного результата.

Нетрудно видеть, что при таком подходе закладывается "процедурность" запроса, поэтому для языков манипулирования данными (ЯМД), построенных на алгебре отношений принято говорить как о процедурных языках.

Рассмотренная алгебра, безусловно, имеет преимущества, но требует от пользователя четкого представления (алгоритма) выполнения операций алгебры, возлагая тем самым на пользователя вопросы эффективности реализации запроса. Указанный недостаток в некоторой мере устранен в реляционном исчислении.

Четкость в последовательности действий при обработке запроса характеризует степень " процедурности " средств, используемых для его формулировки. Говорят, что алгебра отношений лежит в основе процедурных языков манипулирования данными в реляционной модели БД.

1.3. Предпосылки введения исчисления отношений

Реляционная алгебра определяет набор операций (алгебраических), которые должны быть реализованы системой для получения ответа на запрос.

Взаимодействуя с системой на языке алгебры отношений, пользователь должен уметь манипулировать соответствующими операциями реляционной алгебры при конструировании запросов. При этом от пользователя-непрофессионала требуются определенные знания в области математики, и на него же возлагаются вопросы, связанные с построением таких запросов на языке алгебры, которые для своей реализации требовали бы минимальное время.

Пусть, например, есть база данных (БД), состоящая из следующих отношений (рис. 1.7).

Табл. 1.Изделие - R1

Шифр изделия	Характеристики изделия
Ш1	X1
Ш2	X2
Ш3	X3

Табл. 2. Цех - R2

N цеха	Имя цеха
N1	И1
N2	И2

Табл. 3. ЦЕХ_ИЗДЕЛИЕ - R3

N цеха	Шифр изделия	Количество по плану
N1	Ш1	10
N1	Ш2	10
N2	Ш1	10
N2	Ш2	10
N2	Ш3	20

Рис. 1.7. Отношения ИЗДЕЛИЕ, ЦЕХ, ЦЕХ_ИЗДЕЛИЕ

Запрос к базе:

Найти шифры цехов (*N цеха*), которые изготавливают все детали (*Шифр изделия*), выпускаемые на данном предприятии.

Для реализации данного запроса пользователь может построить несколько вариантов предложений на языке алгебры отношений.

Например, запрос типа:

1) Rrez(N цеха) = (R3[N цеха, Шифр изд.] **x°** R2[N цеха]) ÷ R1[Шифр изд.]

или

2) Rrez(N цеха)=(R3[N цеха, Шифр изд.] ÷ R1[Шифр изд.]) **x°** R2[N цеха]

Здесь **x°** - символ операции эквисоединения

÷ - символ операции деления

При реализации запроса типа 1 или 2 СУБД будет строго придерживаться порядку операций, предписанных в предложении. При этом существенна в данном случае операция ÷ (деление):

R [A ÷ O] **S**=**R** [**Ã**] \ ((**R** [**Ã**] × **S** [B]) \ **R**)[**Ã**],

так как она и будет определять время выполнения данного запроса.

Рассмотрим процесс выполнения запроса для предложения типа 1 и типа 2 для нашего примера. При этом атрибут *N цеха* обозначим "Nц", атрибут *Шифр изделия* обозначим как "Ш".

1.3.1. Пример исполнения запросов

ВЫПОЛНЕНИЕ ЗАПРОСА ТИПА 1.

Rrez(Nц)=(R3 [Nц, Ш] **x°** R2 [Nц]) ÷ R1 [Ш]

Первый шаг.

R4=R3 **x°** R2 R4 (Nц, Ш)

N1 Ш1
N1 Ш2
N2 Ш1
N2 Ш2
N2 Ш3

Второй шаг.

R4|R1 = R4 [Nц] \ ((R4 [Nц] × R1 [Ш]) \ R4)[Nц]

а) R4 [Nц] = R5 [Nц]

N1
N2

б) R4 [Nц] × R1 [Ш] = R5 × R1 [Ш]=R6

R6(Nц, Ш)

N1 Ш1
N1 Ш2
N1 Ш3
N2 Ш1
N2 Ш2
N2 Ш3

в) R6 \ R4 = R7

R7(Nц, Ш)

N1 Ш3

г) R7 [Nц] = {N1} = R8(Nц)

д) R4 [Nц] \ R8 = Rrez = R5 \ R8

Rrez= Nц R5(Nц) \ R8(Nц)

N2 N1 N1

N2

ВЫПОЛНЕНИЕ ЗАПРОСА ТИПА 2

Rrez (Nц) = (R3 [Nц, Ш)] ÷ R1 [Ш]) **x°** R2 [Nц]

R3 [Nц, Ш] ÷ R1 [Ш] = R3 [Nц] \ ((R3 [Nц] × R1 [Ш]) \ R3)[Nц]

а) R3 [Nц] = R4 [Nц] (промежуточное отношение)

N1

N2

б) R3 [Nц] × R1 [Ш] = R4 × R3 [Ш] = R5(Nц, Ш)

N1 Ш1

N1 Ш2

N1 Ш3

N2 Ш1

N2 Ш2

N2 Ш3

в) R5 \ R3 = R6(Nц, Ш)

N1 Ш3

г) R6 [Nц] = {N1}

д) R3 [Nц]\R6(Nц) = R7 = {N2}

е) Rrez = R7(Nц) **x°** R2 [Nц] = R7(Nц) **x°** R2 [Nц]

N2 N1

N2

Rrez=N2

Анализ выполнения запросов типа 1 и 2 показывает, что существенное влияние на скорость ответа будет оказывать число операций эквисоединения, что в свою очередь определяется уже особенностями физической организации хранимой информации.

Цель же реляционного подхода - сделать язык манипулирования данными независимым от способов физической организации реляционной базы и упростить "навигационные" процессы пользователя с физической моделью данных.

В основе средств, которые в некоторой степени устраняют этот недостаток, лежит реляционное исчисление. Оно только описывает (декларирует) результат, предоставляя системе решить, какие операции и в какой последовательности должны быть в действительности выполнены. Однако следует понимать, что алгебра и исчисление эквивалентны. Каждому выражению в алгебре соответствует эквивалентное выражение в исчислении, и точно также каждому выражению в исчислении соответствует эквивалентное выражение в алгебре, т.е. между ними существует взаимно однозначное соответствие [3, 4, 27]. Подробное изложение исчисления отношений не входит в задачи данного учебного пособия, но важно уяснить, как используется исчисление в языке SQL.

1.4. Исчисление отношений и SQL

Реляционное исчисление или исчисление отношений основано на разделе математической логики, которое называется исчислением предикатов.

Исчисление отношений лежит в основе декларативного подхода к формулировке запроса к базе данных. При декларативном подходе запросу к базе данных (БД) соответствует формула реляционного исчисления. Ответом на запрос служит множество объектов из области интерпретации (в нашем случае этой областью является БД), на которой истинна формула, соответствующая запросу.

Базисными понятиями исчисления являются понятие **переменной** с определенной для нее областью допустимых значений и понятие **правильно построенной формулы** (**WFF** - Well-Formed Formula), опирающейся на переменные, предикаты и кванторы.

В зависимости от того, что является областью определения переменной, различаются исчисление кортежей и исчисление доменов. В исчислении кортежей областями определения переменных являются отношения базы данных, т.е. допустимым значением каждой переменной является кортеж некоторого отношения. В исчислении доменов областями определения переменных являются домены, на которых определены атрибуты отношений базы данных, т.е. допустимым значением каждой переменной

является значение некоторого домена. Мы рассмотрим более подробно исчисление кортежей, так как именно это исчисление лежит в основе таких языков, как QUEL, SQURE, SQL [3, 13, 11, 15, 19, 25, 26, 29]. Языки, подобные QBE, используют исчисление доменов, и некоторые вопросы его реализации рассмотрены в [13, 17].

В исчислении кортежей все переменные связаны квантором существования и область определения каждой из них ограничена одним из отношений (таблиц). Объявление переменной и ее привязка к определенному отношению производится в указанных языках различными операторами. Так, в языках, подобных QUEL (QUEry Language – язык системы Ingres [13]) такой оператор имеет вид

Range of <переменная> is <имя отношения>

В языках, подобных SQL [25, 26, 29], объявление переменных производится следующей группой операторов:

SELECT <список атрибутов>

FROM <имя отношения>

WHERE<условие>

В последнем случае <список атрибутов> и есть не что иное, как список переменных, участвующих в запросе. Синтаксически SQL близок к исчислению кортежей, но не более, так как он также использует и операции реляционной алгебры.

Предположим, что мы работаем с базой данных, обладающей схемой

ИЗДЕЛИЕ (Шифр, Имя изделия, Материал, Вес, Заказчик),
ЦЕХ(N цеха, Имя цеха, ФИО начальника),
ЦЕХ_ИЗДЕЛИЕ (N цеха, Шифр изделия, Количество по плану).

И хотим узнать имена *Заказчиков*, для которых планируется выпуск изделий больше 10.

Если бы для формулировки такого запроса использовалась реляционная алгебра, то мы получили бы алгебраическое выражение, которое читалось бы, например, следующим образом:

- выполнить соединение отношений ИЗДЕЛИЕ и ЦЕХ_ИЗДЕЛИЕ по условию Шифр = Шифр изделия;
- ограничить полученное отношение по условию *Количество по плану > 10;*

- спроецировать результат предыдущей операции на атрибут *Заказчик.*

Мы четко сформулировали последовательность шагов выполнения запроса, каждый из которых соответствует одной реляционной операции. Если же сформулировать тот же запрос с использованием реляционного исчисления, которому посвящается этот раздел, то мы получили бы формулу, которую можно было бы прочитать, например, следующим образом: Выдать *Заказчиков*, для которых планируется выпуск изделий со значением *Количество по плану > 10.*

Во второй формулировке мы указали лишь характеристики результирующего отношения, но ничего не сказали о способе его формирования. В этом случае система должна сама решить, какие операции, и в каком порядке нужно выполнить над отношениями ИЗДЕЛИЕ и ЦЕХ_ИЗДЕЛИЕ.

Обычно говорят, что алгебраическая формулировка является процедурной, т.е. задающей правила выполнения запроса, а формулировка на языке исчисления носит описательный (или декларативный) стиль, поскольку она всего лишь описывает свойства желаемого результата. Как мы указывали в начале параграфа, на самом деле эти два механизма эквивалентны и существуют не очень сложные правила преобразования одного формализма в другой.

Так на языке SQL запрос будет оформлен следующим образом:

SELECT ИЗДЕЛИЕ.Заказчик

FROM ИЗДЕЛИЕ, Цех_Изделие

WHERE ИЗДЕЛИЕ.Шифр=Цех_Изделие.Шифр_изделия AND Цех_Изделие.Количество_по_плану > 10;

Как видно из запроса, в любой момент времени переменная - ИЗДЕЛИЕ.Заказчик представляет некоторый кортеж отношения ИЗДЕЛИЕ. При использовании кортежных переменных в формулах можно ссылаться на значение атрибута переменной (это аналогично тому, как, например, при программировании на языке Паскаль и Си можно сослаться на значение поля структурной переменной типа Record и Struct). Точка между именем отношения и атрибутом этого отношения и указывает на этот факт, и является обязательной в конструкциях языка.

Отметим, что результатом запроса всегда будет таблица. Если проанализировать инструкцию FROM, то ее результатом будет декартово произведение отношений ИЗДЕЛИЕ и ЦЕХ_ИЗДЕЛИЕ. После выполнения инструкции WHERE получается

выборка из полученного произведения с использованием операции эквисоединения. Наконец, после исполнения оператора SELECT получается проекция выборки по столбцам, указанным в инструкции SELECT. Нестрого говоря, инструкция FROM в SQL соответствует декартову произведению, инструкция WHERE – выборке, а конструкция SELECT-FROM-WHERE представляет проекцию выборки произведения. Если использовать выражение **SELECT * From R** (где **R**-имя таблицы), то результатом запроса будет копия всей таблицы; **звездочка** - это указание на вывод всех имен столбцов в таблице (или таблиц), на которую указывает ссылка в инструкции FROM. При этом имена столбцов в результирующем отношении будут представлены в том же порядке, что и в таблице.

Правильно построенные формулы служат для выражения условий, накладываемых на кортежные переменные. Основой WFF являются простые сравнения, представляющие собой операции сравнения скалярных значений (значений атрибутов переменных или литерально заданных констант). Например, конструкция " ЦЕХ_ИЗДЕЛИЕ.Количество_по_плану > 10 " является простым сравнением. По определению, простое сравнение является WFF, а WFF, заключенная в круглые скобки, также является простым сравнением.

Более сложные варианты WFF строятся с помощью логических связок NOT (отрицание), AND (конъюнкция), OR (дизъюнкция) и IF … THEN (условие) и кванторов. Для обозначения и реализации кванторов существования и всеобщности используются функции EXISTS и FORALL соответственно.

Впервые язык манипулирования данными с кортежными переменными был реализован в системе Ingres [13].

Язык SQL является фактически гибридом алгебры и исчисления [3]. В нем присутствуют и квантор существования **EXISTS**, и квантор **FORALL**, получаемый отрицанием **NOT EXISTS**, что относится к исчислению. Также в нем используется, например, и оператор UNION (объединение), что относится к алгебре.

Самый общий вид запроса на языке SQL представляет теоретико-множественное алгебраическое выражение, составленное из элементарных запросов. В SQL развитых систем допускаются все базовые теоретико-множественные операции (UNION, INTERSECT, MINUS,…).

Если в алгебре Кодда в ее первоначальном виде были только восемь операторов: выборка, проекция, произведение, объединение, пересечение, вычитание, соединение и

деление, то в настоящее время добавлен еще один оператор переименования атрибутов – RENAME. Назначение данного оператора - обеспечение уникальности имен атрибутов в результирующем отношении в случае необходимости создания виртуальных (вычисляемых) столбцов отношения. Однако данный оператор трудно отнести к алгебраическим, потому что он просто снимает проблему нежелательности иметь в таблице два или более одинаковых имен атрибутов. Например, использование данного оператора позволяет изменить в отношении

ИЗДЕЛИЕ (Шифр, Имя_ изделия, Материал, Вес, Заказчик)

имя атрибута *Вес* на имя *Вес_ в_ Кг*, следующим оператором:

ИЗДЕЛИЕ RENAME ВЕС AS Вес_ в_ Кг

В SQL подобный механизм иногда используется для замены латинских имен атрибутов (в большинстве СУБД схемы должны прописываться только латинскими буквами и имена не должны содержать символов пробела и точки) на более привычное изображение (русское) при выводе информации на экран дисплея. Например, пусть необходимо вывести содержимое отношения ИЗДЕЛИЕ на экран, заменив в нем имя атрибута, *Вес* на имя *Вес_ в_ Кг*, и при этом разделить содержимое столбца *Вес* на 1000.

SELECT ИЗДЕЛИЕ.ВЕС / 1000 AS Вес_ в_ Кг FROM ИЗДЕЛИЕ

При рассмотрении операторов SQL следует учитывать, что синтаксические правила описания языка в различных СУБД различны. Существует стандарт SQL ANSI, но существует и множество его диалектов, на которых работают конкретные системы. Например, Foxpro, Sybase, SQL Server и Microsoft SQL используют синтаксис, существенно отличающийся от стандарта ANSI. InterBase, Oracle (и многие другие серверы) во многом поддерживают указанный стандарт, но каждый разработчик вносит в своих системах усовершенствования, что естественно приводит к изменению синтаксиса языка в конкретной СУБД [9, 17, 20, 22, 23, 25, 26, 28, 29].

Поскольку будут излагаться только основные операторы языка, расхождения в их синтаксисе между различными диалектами невелико. Чтобы действительно изучить SQL, надо обратиться к технической документации системы, которую вы собираетесь осваивать.

Глава 2. Диалекты SQL

2.1. Способы реализации языка SQL

SQL - это язык реляционных баз данных, а не язык системного программирования. SQL ориентирован на работу с множествами и стандарт ANSI SQL не включает ни средств управления выполнением программы (ветвления и циклов), ни средств для создания форм и отчетов (каждая СУБД поставляет свои собственные утилиты для реализации указанных функций). Функции управления реализуются с помощью языков программирования, например xBase, C, C++ , Паскаль [8, 9, 12, 24]. Каждый диалект SQL включает (добавляет) в базовый SQL некоторые дополнительные ключевые слова, которые на практике могут существенно отличаться от стандарта. Каждая фирма, разработчик SQL, утверждает, что ее SQL наиболее близок к стандарту [14, 20], и достаточно трудно определить, какой из диалектов SQL, например SQL-PLUS фирмы Oracle [11, 28] или TRANSACT-SQL [25] фирмы Microsoft более ему соответствует. Большинство развитых СУБД предлагают два варианта реализации SQL: интерактивный и вложенный, о которых более подробно говорится в третьей главе.

2.2. Типы данных и язык определения схем DDL

Прежде чем перейти к обсуждению основных операций SQL, отметим, что любая СУБД, естественно, имеет свой язык описания схем (DDL) и в ней определен допустимый набор типов данных (как для полей столбцов так и для включающего языка), как и в любой среде программирования. Чаще всего набор этих типов данных у большинства систем совпадает и в этот набор входят следующие типы:

INTEGER- целое число (обычно до 10 значащих цифр и знак);
SMALLINT- "короткое целое" (обычно до 5 значащих цифр и знак);
DECIMAL (p, q)- десятичное число, имеющее p цифр ($0 < p < 16$) и знак; с помощью q задается число цифр, справа от десятичной точки ($q < p$, если $q = 0$, оно может быть опущено);

FLOAT- вещественное число с 15 значащими цифрами и целочисленным порядком, определяемым типом СУБД;
CHAR(n)- символьная строка фиксированной длины из n символов (0 < n < 256);
VARCHAR(n)- символьная строка переменной длины, не превышающая n символов (n > 0 и разное в разных СУБД, но не меньше 4096);
DATE- дата в формате, определяемом специальной командой (по умолчанию mm/dd/yy); поля даты могут содержать только реальные даты, начинающиеся за несколько тысячелетий до н.э. и ограниченные пятым-десятым тысячелетиями н.э.;
TIME- время в формате, определяемом специальной командой (по умолчанию hh.mm.ss);
DATETIME - комбинация даты и времени;
MONEY- деньги в формате, определяющем символ денежной единицы ($, руб., ...) и его расположение (суффикс или префикс), точность дробной части и условие для показа денежного значения.

В некоторых СУБД еще существует тип данных LOGICAL, DOUBLE и ряд других. Кроме того, некоторые СУБД предоставляют пользователю возможность самостоятельного определения новых типов данных, например плоскостные или пространственные координаты, дроби, графика и т.п.

SQL ориентирован на работу с таблицами и не имеет достаточных средств для создания сложных прикладных программ. Поэтому в разных СУБД он либо используется вместе с языками программирования высокого уровня (например, такими, как Си или Паскаль), либо включен в состав команд специально разработанного языка СУБД (язык систем dBASE, RBASE и т.п.).

Унификация полных языков современных профессиональных СУБД достигается за счет внедрения объектно-ориентированного языка четвертого поколения 4GL [29]. Последний позволяет организовывать циклы, условные предложения, меню, экранные формы, сложные запросы к базам данных с интерфейсом, ориентированным как на алфавитно-цифровые терминалы, так и на оконный графический интерфейс типа X-Windows.

Заметим, что, хотя с использованием языка SQL можно определить схему БД, содержащую данные любого из перечисленных типов, возможность использования этих данных в прикладных системах зависит от применяемого языка программирования. В большинстве реализаций правила встраивания SQL в программы на языке Си имеют следующее соответствие между типами данных SQL и типами данных Си:

CHARACTER соответствует строкам Си;

INTEGER соответствует long;

SMALLINT соответствует short;

REAL соответствует float;

DOUBLE PRECISION соответствует double .

Так, например, в СУБД Oracle набор этих типов следующий:

CHAR - Длина столбца данного типа не более 255,определяется при описании таблицы.
NUMBER -Тип данных NUMBER используется для хранения чисел (с фиксированной и плавающей точкой, с точностью до 38 цифр). Возможно хранение чисел до 9.99 * 10 в 124 степени, то есть - единица и за ней 125 нулей.
Например, при создании таблицы ЦЕХ_ИЗДЕЛИЕ с именем столбца *Количество_ по _плану* можно указать:
CREATE TABLE ЦЕХ_ИЗДЕЛИЕ (Количество _по_ плану number (5));
DATE - Данные даты хранятся в фиксированных полях длиной семь байтов, в которых кодируется следующая информация:

Год Месяц День Час Минута Секунда

LONG - Столбец, описанный как LONG, может содержать строку переменной длины до 65536 символов. Данные типа LONG представляют собою просто неструктурированный набор байтов. Данные типа LONG могут содержать массивы двоичных данных, произвольные символы или даже короткие документы.
RAW и **LONG RAW -** Данные типа RAW и LONG RAW используются для байториентируемых данных, которые не интерпретируются системой ORACLE. RAW аналогичен данным типа CHAR (соответственно LONG RAW - LONG), затем лишь исключением, что не делается никаких предположений относительно значения байтов.
Эти типы данных предназначаются для двоичных данных или байтовых строк, например - для хранения последовательностей графических символов. Данные типа RAW эквивалентны CHAR (соответственно LONG RAW - LONG), кроме того, что при передаче с помощью SQL*Net данные типа CHAR преобразовываются при необходимости в другой набор символов, а RAW - нет.

Теперь остановимся коротко на некоторых физических особенностях хранения пользовательских таблиц.

База данных - это упорядоченный набор данных, с которым обращаются как с единым целым. База данных состоит из файлов операционной системы. Физически это собственно файл базы данных и файлы журнала повторного выполнения (redo log files).

Логически файлы базы данных содержат набор словарей и пользовательских таблиц, а файлы журнала повторного выполнения содержат данные для восстановления. Дополнительно базе данных требуется одна или несколько копий управляющего файла. Практически каждая СУБД предлагает свой способ организации и хранения баз данных.

Например, DBVISTA, FOXPRO, ORD, ALASKA, RBASE используют для каждой таблицы отдельный файл. В этом случае база данных – это каталог, в котором хранятся файлы таблиц. В Microsoft Access, InterBase несколько таблиц хранится как один файл. В этом случае база данных – это имя файла с путем доступа к нему. Системы типа Sybase или Microsoft SQL хранят все данные на отдельном компьютере и обращаются с клиентом на основе специального языка – SQL_NET.

2.3. Создание базы данных

Перед тем, как использовать базу данных для хранения информации, она должна пройти процесс, называемый созданием базы данных. Для создания базы данных используется оператор **CREATE DATABASE.**

В каждой СУБД естественно есть утилита для создания базы данных и фиксации в ней таблиц пользователя. Основой этой утилиты является команда CREATE DATABASE, которая (например, в Microsoft SQL SERVER 7.0 [25]) имеет следующий синтаксис:

```
CREATED DATABASE имя_базы_данных
[ON [PRIMARY]
[ <спецификация_файла> [,…n]
[, <группа> [,…n] ]
]
[LOAD ON {<спецификация файла>}]
[FOR LOAD │ FOR ATTACH]
```

Спецификации отдельных файлов, составляющих базу данных, определяются так:

```
<спецификация_файла> -
( [ NAME = логическое_имя_файла,]
FILENAME = "имя_файла_операционной_системы"
[, SIZE = размер]
[, MAXSIZE = { максимальный_размер │ UNLIMITED }]
[, FILEGROWTH = приращение]  ) [,…n]
```

Спецификации групп файлов определяются так:

<группа> _
FILEGROUP имя_группы<спецификация_файла> [,…n]
Рассмотрим параметры этой команды:
 имя_базы данных – имя создаваемой базы данных. Оно должно быть уникальным в рамках сервера и соответствовать требованиям, предъявляемым к идентификаторам. SOL Server 7.0 не рекомендует использовать для имени символы кириллицы.
 ON – ключевое слово, обозначающее начало описания файлов, где будет размещаться база данных.
 PRIMARY – ключевое слово для определения основного (первичного) файла в группе. Если оно опущено, то выбирается первый файл из списка.
 ,…*n* – перечень дополнительных файлов, в которых будет размещаться база данных.
 LOAD ON - открывает начало списка файлов для размещения журнала транзакций. Если этот параметр опущен, файл создается автоматически с размером равным 25% общего размера файлов данных.
 FOR LOAD – оставлен для совместимости с предыдущими версиями.
 FOR ATTACH – определяет, что база данных создается путем присоединения существующих файлов. Если число файлов не превышает 16, то вместо использования этого параметра при создании базы данных лучше воспользоваться системной хранимой процедурой *SP_ATTACH_DB*.
 NAME = логическое_имя_файла – определяет логическое имя, которое SQL Server использует для ссылки на файл. Оно должно быть уникальным в базе данных.
 FILENAME = "имя_файла_операционной системы" – определяет полный путь и имя физического файла операционной системы
 SIZE = размер – задает начальный размер файла.
 MAXSIZE = максимальный_размер – определяет максимальное значение, до которого может увеличиться файл. Если это значение не указано, то рост файла может продолжаться пока будет свободное место на диске.
 UNLIMITED – явным образом указывает на возможность неограниченного увеличения размера файла, естественно, пока есть свободное место на диске.
 FILEGROWTH = приращение – задает приращение, на которое будет изменяться размер файла при его увеличении или уменьшении. Этот параметр можно указывать и в процентах. Если значение равно 0, то указанный начальный размер меняться не будет.

В простейшем случае оператор создания базы данных с именем PROBA (для хранения таблиц и некоторой другой информации) выглядит следующим образом:

```
CREATED DATABASE PROBA
ON PRIMARY
(NAME=PROBA_Data, FILENAME="D:\mssql7\data\Proba_Data.MDF",
      SIZE=1MB,MAXSIZE=UNLIMITED, FILEGROWTH=10%)
LOAD ON (NAME=PROBA_Data,
            FILENAME="D:\mssql7\data\Proba_Data.MDF",
      SIZE=1MB,MAXSIZE=UNLIMITED, FILEGROWTH=10%)
```

Например, в системе ORACLE создание базы данных осуществляется с помощью утилиты SQL*DBA. Описания полного синтаксиса оператора занимает значительный объем и поэтому не приводится.

CREATE DATABASE имя_базы
[CONTROLFILE REUSE]
[LOGFILE файл [, файл] ...]
[MAXLOGFILES целое]
[DATAFILE файл [, файл] ...]
[MAXDATAFILES целое]
[MAXINTANCES целое]
[ARCHIVELOG | NOARCHIVELOG]
[SHARED | EXCLUSIVE]

База данных идентифицируется именем базы данных, которое дается в момент ее создания. Имя указывается в операторе CREATE DATABASE и должно быть уникальным в системе. Некоторые операторы SQL и SQL*DBA при использовании администратором (DBA) базы данных требуют указания имени базы данных, большинство же пользователей это имя не интересует.

Как видно из приведенных примеров, синтаксис команд в различных СУБД существенно отличен, но назначение команд совпадает.

После создания базы необходимо определить пользовательскую таблицу с помощью оператора Create Table (синтаксис команды приведен в приложении 1), который создает реальную (базовую) таблицу, именно эта таблица и хранится в физической памяти машины в разделе, определяемом командой CREATE DATABASE.

Базовые таблицы словаря данных должны быть первыми объектами, которые должны быть созданы в любой базе, так как они должны присутствовать при создании любых других объектов. Таблицы словаря данных создаются автоматически при исполнении оператора CREATE DATABASE.

Словарь данных является одной из основных частей любой СУБД. Словарь данных - это набор таблиц, используемых для справочного руководства по базе данных. Например, он может содержать следующую информацию:

- имена пользователей системы,

- права и привилегии, им предоставленные,

- имена объектов базы (таблицы, обзоры, индексы, алиасы, синонимы),

- информацию об основных (Primary) и внешних (Foreign) ключах,

- умалчиваемые значения для столбцов,

- ограничения, поддерживающие непротиворечивость таблиц,

- контрольную информацию, например кто запрашивал или изменял различные объекты базы,

- и другую важную для базы информацию.

Словарь данных, как и другая информация базы, структурирован по таблицам и обзорам, и доступ к ней может быть ограничен. Полный доступ к такой информации имеет только администратор базы данных.

Словарь данных создается в процессе создания базы. Затем, уже во время работы базы, словарь данных модифицируется системой в ответ на каждый оператор DDL [11, 15, 25, 29].

Примечание. Некоторые термины, например: обзоры, индексы остались не определенными. Однако каждый из них будет раскрыт по мере необходимости.

ANSI SQL включает набор стандартных команд, сгруппированный по шести категориям: описание данных (здесь же и инструкции для обеспечения целостности данных), выполнение запросов, манипулирование данными, управление курсором, управление транзакциями, административное управление.

Полное описание синтаксиса конструкций SQL занимает существенный объем, поэтому будут приводиться только необходимые инструкции языка для используемых примеров.

Здесь же отметим, что при обращении к базовой таблице посредством оператора SELECT (запрос пользователя с терминала или прикладной программы) СУБД создает и использует в своей работе ряд вспомогательных (виртуальных) таблиц, с помощью которых пользователь и получает доступ к базовым таблицам. К таким виртуальным (будто не существующим) таблицам относятся: представления, курсоры, неименованные рабочие таблицы. Такие таблицы являются "окнами" к базовым таблицам, и их иногда называют буферами или областями доступа. В этих таблицах формируются результаты поиска (запросов) данных из базовых таблиц. Еще раз подчеркнем, что рабочие таблицы, представления и курсоры не существуют в базе данных - это лишь окно доступа к реально хранящимся данным.

2.4. Учебный фрагмент схемы базы

В операторе определения **схемы** содержится идентификатор полномочий и список элементов схемы, каждый из которых может быть определением таблицы, определением представления (**view**) или определением привилегий. Каждое из этих определений представляется отдельным оператором SQL, и все они должны быть указаны в описании схемы. Для этих операторов мы приведем синтаксис, поскольку это позволит более четко описать их особенности.

Рекомендация. На данном этапе прочтения основ языка SQL понятие СХЕМЫ пока не определено формально. Поэтому пока и неизвестны также такие формальные понятия реляционной модели, как **реляционный ключ** (**Primary key (первичный), Foreign key (внешний)**), **функциональная зависимость**, и многое другое [3, 5, 27, 30]. Ознакомившись с данным разделом, необходимо изучить материал следующей главы и вновь вернуться к прочтению данного раздела.

Для дальнейшего изложения материала воспользуемся небольшим фрагментом схемы модели базы данных некоторого учебного заведения, которое состоит из следующих взаимосвязанных таблиц (рис. 2.1 – 2.9):

СОТРУДНИК (Ид_Сотрудника, Фамилия, Имя, Отчество, ИНН, Год_рождения, Пол, Город, Район, Индекс, Ид_Совместителя) – рис. 2.1.

ОТДЕЛ (Ид_Отдела, Название_отдела, Ид_Начальника, Вид_отдела, Год_основания) – рис. 2.2.

ОТДЕЛ_СОТРУДНИК (Ид_Отдела, Ид_Сотрудника, Должность, Оклад, Дата_приема, Дата_увольнения) – рис. 2.3.

ВЕДОМОСТЬ_ОПЛАТЫ (Ид_Сотрудника, Ид_Отдела, Период, Сумма, Вид_оплаты) – рис. 2.4.

КОМАНДИРОВКИ (Ид_Сотрудника, Дата_отбытия, Точка, Дата_убытия) - рис. 2.5.

СОВМЕСТИТЕЛИ (Ид_Совместителя, Фамилия, Имя, Отчество, ИНН, Город, Район, Индекс) – рис. 2.6.

ЗАМЕЩЕНИЕ (Порядковый номер, Дата_замещения, Ид_Сотрудника, Ид_Совместителя) – рис. 2.7.

РАБОТЫ (Ид_ Сотрудника, Ид_ Вида, Количество, Цена, Период_с, Период_по) – рис. 2.8.

ВИД_РАБОТЫ (Ид_Вида, Лекции, Практика, Лаб_занятия, Консультации, Рецензии, Зачеты, Экзамены, Дипломное_проектирование, Курсовое_проектирование, Итого) – рис. 2.9.

В таблицах, для удобства восприятия, некоторым именам атрибутов присвоены сокращения:

Ид_Сотр Ид_Сотрудника (идентификатор сотрудника),
Ид_Совм Ид_Совместителя (идентификатор совместителя),
№ п/п Порядковый номер,
и т.д.

Кроме того, в имени атрибута (столбца) в данном учебном пособии иногда допускаются пробелы, что на практике будет являться ошибкой.

Обратим Ваше внимание на смысловую нагрузку имен отношений и используемых имен атрибутов. Например, рассматривая отношение, представленное на рис. 2.2, имя которого определено как ОТДЕЛ у читающего может вызвать раздражение. Почему рассматриваются **объекты** "Кафедра", а называют их ОТДЕЛАМИ. Если это объект КАФЕДРА, то почему ее атрибут - руководитель (заведующий) получил имя *Ид_Начальника.*

Ответить полно и правильно на этот вопрос можно только после изучения так называемого ИНФОЛОГИЧЕСКОГО этапа проектирования модели базы данных [4, 8, 16, 17, 18, 31]. Более того, мы здесь использовали понятие "объект", который также определяется в инфологической модели предметной области. Инфологическое моделирование здесь не рассматривается (наиболее интересные работы в этом направлении, на наш взгляд, это работы [4, 16, 18, 26]), и поэтому такие понятия как "объект", "свойство объекта" остаются не поясненными и их трактовка остается за Вами.

Предположим, что база данных будет "расти" и придется еще хранить информацию об объектах БУХГАЛТЕРИЯ и ОТДЕЛ_КАДРОВ. Тогда, наверное, это пояснит введение подобных имен. Кроме того, на практике число таблиц может достигать сотен и более, но подобранный (реальный) пример фрагмента предметной области позволяет наиболее полно осветить особенности манипулирования данными на основе SQL.

Сотрудник

Ид_ Сотр	Фамилия	Имя	Отчество	ИНН	Год рожд.	Пол	Город	Район	Индекс	Ид_ Совм
1	Иванов	Иван	Петрович	101	1949	М	СПБ	Центр	97000	1
2	Петров	Иван	Иванович	102	1949	М	Колпино	Колп	92210	3
3	Сидоров	Петр	Петрович	103	1947	М	СПБ	Фрун	94555	2
4	Панов	Антон	Михайлович	104	1975	М	СПБ	Фрун	94556	2
5	Петухов	Виктор	Борисович	105	1940	М	Пушкин	Пушк	99481	1
6	Иванова	Вера	Васильевна	116	1970	Ж	Гатчина	Гатч	92123	
7	Петрова	Нина	Николаевна	217	1970	Ж	СПБ	Калин	95257	4
8	Сидрова	Ада	Ивановна	308	1970	Ж	СПБ	Фрун	94556	7
9	Никитин	Виктор	Сергеевич	489	1952	М	СПБ	Центр	97007	4
10	Мухин	Степан	Михайлович	510	1964	М	СПБ	Моск	94001	
11	Попов	Михаил	Михайлович	611	1947	М	СПБ	Центр	97000	
12	Иванов	Иван	Иванович	712	1980	М	СПБ	Фрун	94555	
13	Хохлов	Иван	Васильевич	713	1960	М	СПБ	Фрун	94555	
14	Яковлев	Иван	Васильевич	714	1980	М	СПБ	Фрун	94550	

Рис. 2.1. Сотрудники подразделений

Отдел

Ид_ Отдела	Название_отдела	Ид_ Начальника	Вид_ отдела	Год_ основания
1	Кафедра КТиПО	8	Выпускающая	1960
2	Кафедра механики	9	Нет	1930
3	Кафедра физики	10	Нет	1930
4	Кафедра математики	13	Нет	1945
5	Кафедра автоматики	14	Выпускающая	1945

Рис. 2.2. Отделы организации

Отдел_Сотрудник

Ид_ Отдела	Ид_ Сотр.	Должность	Оклад	Дата_ приема	Дата_ увольнения
1	1	Доцент	2500	1977	
1	2	Доцент	2500	1979	
1	3	Ст. препод.	2000	1987	
1	5	Профессор	3700	1970	
1	6	Инженер	1500	1999	
1	8	Зав. кафедрой	5000	1975	2003
2	7	Ассистент	1700	2003	
2	4	Доцент	3100	2000	
2	10	Зав. кафедрой	5000	1977	
3	11	Зав. кафедрой	5000	1980	
2	9	Профессор	4800	1980	
4	13	Зав. кафедрой	5000	1980	
5	14	Зав. кафедрой	5000	1980	

Рис. 2.3. Закрепление сотрудников по отделам

Ведомость_оплаты

Ид_Сотр	Ид_Отдела	Период	Сумма	Вид_оплаты
1	1	Март	1200	
2	1	Март	1200	
3	1	Март	1000	
1	1	Апрель	1200	
1	1	Март	800	
……				

Рис. 2.4. Ведомость оплаты за основной вид работы

Командировки

Ид_ Сотр	Дата_ отбытия	Точка	Дата_ убытия
1	01.02.03.	Мурманск	02.02.03
2	01.02.03	Мурманск	
3	01.03.03.	Петрозаводск	11.03.03
1	03.03.03.	Петрозаводск	
5	01.04.03	Мурманск	12.04.03

Рис. 2.5. Командировки сотрудников

Совместители

Ид_ Совм	Фамилия	Имя	Отчество	ИНН	Город	Район	Индекс
1	Петров	Петр	Петрович	836	СПБ	Центр	97000
2	Смирнов	Федор	Николаевич	976	Мурманск	Центр	36232
4	Михайлов	Иван	Михайлович	915	СПБ	Моск.	94091
7	Русов	Игорь	Васильевич	918	Мирный	Север	31000
3	Алехин	Армен	Сергеевич	900	Колпино	Колп.	92210

Рис. 2.6. Совместители, подменяющие преподавателей

Замещение

№ п/п	Дата_замены	Ид_Сотр	Ид_Совм
1	10.02.03	8	7
2	10.02.03	1	1
3	10.02.03	7	4
4	10.02.03	3	2
5	10.03.03	8	7
6	10.03.03	2	3
7	10.04.03	4	2
8	10.04.03	5	1
9	20.05.03	4	2
10	20.05.03	5	1

Рис. 2.7. Реестр замещений основных преподавателей совместителями

Работы

Ид_Сотр	Ид_Вида	Количество	Цена	Период_С	Период_По
1	10	88	1.7	01.04.03	30.04.03
1	11	80	1.7	01.05.03	31.05.03
2	13	72	1.7	01.04.03	30.04.03
3	10	88	1.5	01.04.03	30.04.03
1	11	80	1.75	01.05.03	31.05.03
2	14	56	1.75	01.05.03.	31.05.03
3	14	56	1.6	01.05.03	31.05.03

Рис. 2.8. Реестр выполненных работ (нагрузка) основными преподавателями

Вид_работы

Ид_Вида	Лекции	Практика	Лаб._занятия	Консультации	Рецензии	Зачеты	Экзамены	Дипломное_проектирование	Курсовые проекты	Итого
1	4					2				6
2		8								8
3				2						2
4			8							8
5					8					8
6				2			6			8
7						8				8
8									12	12
9				2				32		34
10	16	8	8	8	8	2	6	20	12	88
11	16		8	8	8	2	6	32		80
12		4								4
13		8	8	8	8	2	6	32		72
14	16	8	8	8	8	2	6			56
15										

Примечание. Пустые клетки содержат ноль.

Рис. 2.9. Вид работы (учебная работа - часов в день)

Приведем некоторые пояснения к описанию таблиц и их содержимому. Если для таблиц на рис. 2.1 – 2.7 практически все ясно (некоторые моменты будут раскрыты позже), то таблицы на рис. 2.8 и 2.9 требуют пояснения.

Каждый преподаватель в начале месяца представляет отчет о проделанной работе в виде документа (так называемая форма 2), представленного на рис. 2.10.

Каждая строка этой формы относится к определенному дню месяца, в которой указывается какие виды учебной работы исполнялись преподавателем в указанный день. Последний столбец этой таблицы является итоговой суммой каждой строки. Просуммировав значения элементов указанного столбца, получается "нагрузка" преподавателя за месяц, фамилия которого указывается в шапке формы 2.

Учитывая, что в данном учебном фрагменте информационной подсистемы форма 2 физически не отображается в таблицах базы данных, подсчитанная нагрузка (с расшифровкой на ее составляющие) и является одной из строк базовой таблицы ВИД_ РАБОТЫ (рис. 2.9). Так, например, учебная нагрузка из формы 2 рис. 2.10 составляет 88 часов. Эта величина и проставлена в первой строке в столбце *Количество* таблицы РАБОТЫ (рис. 2.8). Если оказывается, что в таблице ВИД_ РАБОТЫ (рис. 2.9) итоговая строка из формы 2 с подобной расшифровкой отсутствует, то она добавляется в базовую таблицу ВИД_ РАБОТЫ (рис. 2.9.). Такая новая строка получает новый уникальный идентификатор *Ид_Вида* в таблице ВИД_ РАБОТЫ.

Для нашего примера нагрузка в объеме 88 часов с расшифровкой, представленной в форме 2 (рис. 2.10), получила уникальный идентификатор *Ид_Вида*=10. В базовой таблице РАБОТЫ (рис. 2.8) *Ид_Вида,* равный 10, зафиксирован в первой строке таблицы РАБОТЫ для сотрудника Иванова Ивана Петровича с уникальным идентификатором *Ид_Comp,* равным единице за апрель месяц, и для преподавателя с *Ид_Comp*=3 за тот же месяц.

Форма 2

СЕВЕРО-ЗАПАДНЫЙ ГОСУДАРСТВЕННЫЙ ЗАОЧНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Сведения об учебной работе штатного преподавателя

за апрель месяц 2003 г.

Фамилия, имя, отчество Иванов Иван Петрович_____

Должность, ученая степень и звание: Доцент_____

Кафедра ___КТиПО_____

Дисциплина _Схемотехника_____

Число, месяц	Место занятий	Лекции	Практические занятия	Лабораторные занятия	Консультации	Рецензии	Зачеты	Экзамены	Курсовое проектирование	Преддипломная практика	Дипломное проектирование	Заседание ГАК	Контроль занятий	Всего
1	418	4					2							6
2	325		8											8
4	325				2									2
7	320			8										8
8	325					8								8
9	418	4												4
10	418	4												4
11	325				2			6						8
14	325				2									2
15	418	4												4
17	325								12					12
24	325										20			20
27	325				2									2
		16	8	8	8	8	2	6	12		20			88

Всего:

Сведения сдавать на кафедру не позднее 3-го числа каждого месяца
Преподаватель Зав. кафедрой

Рис. 2.10. Пример отчета об учебной работе преподавателя

В реальной ситуации все это происходит автоматически. При вводе новой формы 2 (которая в свою очередь должна быть представлена совокупностью взаимосвязанных

таблиц) автоматически определяется уникальный идентификатор *Вида_работы* (Ид_Вида) и автоматически формируется новая строка (за прошедший месяц) в базовой таблице РАБОТЫ (рис. 2.8) для каждого преподавателя, представившего форму 2 за прошедший месяц.

Остановимся теперь более подробно на базовых таблицах:

СОТРУДНИК, ОТДЕЛ, и ОТДЕЛ_СОТРУДНИК.

Таблица ОТДЕЛ_СОТРУДНИК (рис. 2.3) связывает таблицы СОТРУДНИК (рис. 2.1) и ОТДЕЛ (рис. 2.2) посредством того, что значения атрибута *Ид_Comp* (**первичный ключ** в отношении СОТРУДНИК**)** и сам атрибут из таблицы СОТРУДНИК заносится в таблицу ОТДЕЛ_ СОТРУДНИК, где он получает статус **внешнего ключа.** Аналогичным образом первичный ключ *Ид_Отд* таблицы ОТДЕЛ, попадая в таблицу ОТДЕЛ_ СОТРУДНИК получает статус внешнего ключа (это надо указать явно при описании таблицы ОТДЕЛ_СОТРУДНИК средствами языка SQL). При этом устанавливается ограничение, существующее в предметной области "Каждый служащий может работать только в одном отделе", посредством первичного ключа таблицы ОТДЕЛ_СОТРУДНИК задаваемого на атрибутах *Ид_Отд* и *Ид_Comp.*

Воспользуемся этими тремя таблицами для пояснения, как описываются Primary key и Foreign key.

Для того чтобы работать с таблицами, их необходимо обязательно описать с помощью специальных средств СУБД, с которой Вы собираетесь работать. Например, чтобы описать таблицу ОТДЕЛ (рис. 2.2), необходимо использовать следующую конструкцию языка описания данных (синтаксис этой команды представлен в параграфе 3.1):

```
CREATE TABLE Отдел
(Ид_Отд   CHAR (3)       NOT NULL,
Название_ отдела CHAR (27) NOT NULL,
Ид_Начальника     SMALLINT          UNIQUE,
Вид_ отдела          CHAR (1),
Год_ основания    INT (4),
PRIMARY KEY (Ид_Отд));
```

Если при описании таблицы не указать PRIMARY KEY, то на нее нельзя ссылаться при описании таблицы ОТДЕЛ_СОТРУДНИКИ.

После описания таким же образом таблицы СОТРУДНИК (рис. 2.1) можно описывать таблицу ОТДЕЛ_СОТРУДНИК (рис. 2.3). Примеры описания и создания указанных таблиц рассмотрены в разделе 3.1.2. Обратите внимание, что в таблице СОТРУДНИК (рис. 2.1) присутствует внешний ключ *Ид_Совм* – ключевой атрибут из таблицы СОВМЕСТИТЕЛИ (рис. 2.6).

Ключи - это строго логические понятия [1]. Основной ключ (Primary key) - это атрибут (или комбинация атрибутов), который может использоваться для уникальной идентификации строки в указанной таблице. Внешний ключ (Foreign key) - это атрибут (или группа атрибутов) в одной таблице, которые соответствуют основному ключу в другой таблице. Внешние ключи полезны при поиске данных из нескольких таблиц, как это делается при объединении. Так, для таблицы ЗАМЕЩЕНИЕ (рис. 2.7) основным реляционным ключом может быть атрибут ***№ п/п*** (номер по порядку). Для таблицы ВЕДОМОСТЬ_ ОПЛАТЫ (рис. 2.4) основным ключом будут два атрибута ***Ид_ Сотр*** и ***Ид_Отд***. Для таблицы КОМАНДИРОКИ (рис. 2.5) основной ключ также состоит из двух атрибутов ***Ид_Сотр*** и ***Дата_Отбытия***. В таблице РАБОТЫ (рис. 2.8) первичным ключом будут атрибуты ***Ид_Сотр*** и ***Ид_Вида***. Для таблицы ОТДЕЛ (рис. 2.2) есть два претендента на реляционный ключ - это ***Ид_Отд*** или ***Ид_Начальника***. Также два претендента на реляционный ключ в таблице СОТРУДНИК – это атрибут ***ИНН*** и атрибут ***Ид_Сотр***.

Внешний ключ всегда указывает на первичный ключ в другой таблице.
Введенные понятия тесно связаны с понятием физического индекса. Практически всегда основные и внешние ключи имеют индекс.

Индекс – это специальным образом организованный файл, с помощью которого осуществляется доступ к интересующей информации [3, 5]. Индексы - это реально записанные в базе структуры, которые пользователь может создать с помощью операторов SQL или используя встроенные средства оболочки СУБД [17]. Индексы могут быть непосредственно встроены в таблицу или быть отдельным файлом. Именно использование всевозможных типов индексов позволяет практически мгновенно получить из базы данных интересующую Вас информацию. Физическая организация индексов и математические методы, используемые для обработки индексов, во многом определяют производительность СУБД. Индексы, как впрочем, и упоминавшиеся файлы

относятся к так называемой **внутренней (физической)** модели данных [5, 27], и здесь не рассматриваются.

В заключение рассмотрения данного фрагмента предметной области отметим, что между таблицами СОТРУДНИК (рис. 2.1) и СОВМЕСТИТЕЛЬ (рис. 2.6) установлена следующая смысловая связь. Внешние совместители – это преподаватели, которые могут подменять штатных преподавателей на местах или на базовой точке (в университете). Наличие атрибута *Ид_совм* из таблицы СОВМЕСТИТЕЛЬ (рис. 2.6) в таблице СОТРУДНИК (рис. 2.1) и фиксирует этот факт. Т.е. строки таблицы СОТРУДНИК, содержащие значения *Ид_Совм* – это список преподавателей кому в приказном порядке назначены совместители. При этом один совместитель может подменять нескольких основных преподавателей. После приведенного объяснения становится ясно назначение таблицы ЗАМЕЩЕНИЕ (рис. 2.7.), которая и фиксирует факт, и дату такого замещения (подмены), и кого из основных преподавателей (*Ид_Сотр*) подменял совместитель (*Ид_Совм*).

Глава 3. Основные операторы языка SQL

Существуют два вида (формы) SQL: ИНТЕРАКТИВНЫЙ и ВЛОЖЕННЫЙ. Конструкции этих форм работают одинаково, но используются различно. Интерактивный SQL используется для функционирования непосредственно в базе данных, чтобы производить вывод на экран дисплея или принтер информацию из базы данных для использования ее заказчиком. В этой форме SQL, когда Вы введете команду, она сейчас же выполнится, и можно увидеть вывод - немедленно.

Вложенный SQL состоит из команд SQL, помещенных внутри программ, которые обычно написаны на некотором другом языке (типа Си или Паскаля).

Это делает эти программы более мощными и эффективными. Однако, допуская эти языки, приходится иметь дело со структурой SQL и стилем управления данных, который требует некоторых расширений к интерактивному SQL. Передача команд SQL во вложенный SQL является выдаваемой ("passed off") для переменных или параметров, используемых программой, в которую они были вложены.

Примечание. В этом учебном пособии мы будем представлять SQL в интерактивной форме. Это даст нам возможность обсуждать команды и их эффекты, не заботясь о том, как они связаны с помощью интерфейса с другими языками. Интерактивный SQL - это форма наиболее удобная для непрограммистов. Все, что Вы узнаете относительно интерактивного SQL, в основном применимо и к вложенной форме.

3.1. Определение таблицы CREATE TABLE

Оператор определения таблицы имеет следующий синтаксис:

<table definition> ::=

<table name> (<table element>

[{,<table element>}...])

<table element> ::=

<column definition>

| <table constraint definition>

Кроме имени таблицы (**table name**) , в операторе специфицируется список элементов таблицы (**table element**), каждый из которых служит либо для определения столбца, либо для определения ограничения целостности определяемой таблицы (**table constraint definition**). Требуется наличие хотя бы одного определения столбца.

Оператор **CREATE TABLE** определяет так называемую **базовую** таблицу, т.е. реальное хранилище данных.

В дальнейшем термин таблица – используется для обобщения таких видов таблиц, как базовая таблица, представление или псевдоним (псевдоним см. параграф 5.1.1), и служит для временного (на момент выполнения запроса) переименования и (или) создания рабочей копии базовой таблицы (представления).

3.1.1. Обозначения в синтаксических конструкциях

Здесь (так же, как и в других разделах) в синтаксических конструкциях используются следующие обозначения:

- **двойное двоеточие и равняется (:: =) означают - то, что следует за ними, является определением того, что им предшествует;**
- **слова, обозначенные в угловых скобках (< >) - специальные термины, которые объясняют, что они собой представляют.**
- **квадратные скобки ([ ]) – означают, что конструкции, заключенные в эти скобки, могут быть опущены (т.е. могут не использоваться);**
- **фигурные скобки ({ }) – означают, что конструкции, заключенные в эти скобки, должны рассматриваться как целые синтаксические единицы, т.е. они позволяют уточнить порядок разбора синтаксических конструкций, заменяя обычные скобки, используемые в синтаксисе SQL;**
- **многоточие (...) – указывает на то, что все предшествующее им может, повторяться любое количество раз;**
- **прямая черта (|) – означает наличие выбора из двух или более возможностей. Например, обозначение ASC | DESC указывает, что можно выбрать один из терминов ASC или DESC. Когда же один из элементов выбора заключен в квадратные скобки, то это означает, что он выбирается по умолчанию (так, [ASC] | DESC означает, что отсутствие всей этой конструкции будет восприниматься как выбор ASC) .Термины ASC и DESC обозначают возрастание и убывание соответственно;**
- **точка с запятой (;) – ставится в конце предложений SQL и завершает конструкцию языка;**

- **запятая (,) – используется для разделения элементов списков;**
- **точка (.) – используется для уточнения (связывания) элементов списков. Например, Сотрудник.Имя;**
- **пробелы ( ) – могут использоваться между любыми синтаксическими конструкциями предложений SQL;**
- **прописные латинские буквы и символы – используются для написания конструкций языка SQL и должны (если это специально не оговорено) записываться в точности так, как показано;**
- **строчные буквы – используются для написания конструкций, которые должны заменяться конкретными значениями, выбранными пользователем, причем для определенности отдельные слова этих конструкций связываются между собой символом подчеркивания (_).**

Для определения столбцов таблицы и ограничений целостности используются специальные операторы, которые должны быть вложены в операторы определения таблицы.

3.1.2. Определение столбца

Оператор определения столбца описывается следующими синтаксическими правилами:

<column definition> ::=

<column name> <data type>

[<default clause>] [<column constraint>...]

<default clause> ::=

DEFAULT { <literal> | USER | NULL }

<column constraint> ::=

NOT NULL [<unique specification>]

| <references specification>

| CHECK (<search condition>)

Как видно, кроме обязательной части, в которой определяется имя столбца (column name) и его тип данных (data type), определение столбца может содержать два необязательных раздела: раздел значения столбца по умолчанию и раздел ограничений

целостности столбца (CHECK). Имя столбца не должно содержать пробелов. Например, *Дата_приема* содержит символ ‘ _ ‘, чтобы избежать пробелов в наименовании. Однако в данном тексте (для читаемости) мы иногда отходим от этого правила.

В разделе значения по умолчанию (DEFAULT) указывается значение, которое должно быть помещено в строку, заносимую в данную таблицу, если значение данного столбца явно не указано. Значение по умолчанию может быть указано в виде литеральной (literal) константы (с типом, соответствующим типу столбца) или путем задания ключевого слова USER. В последнем случае, при выполнении оператора занесения строки, ему соответствует символьная строка, содержащая имя текущего пользователя (в этом случае столбец должен иметь тип CHAR). Можно использовать ключевое слово NULL, означающее, что значением по умолчанию является неопределенное значение. Если значение столбца по умолчанию не специфицировано и в разделе ограничений целостности столбца указано NOT NULL, то попытка занести в таблицу строку с неспецифицированным значением данного столбца приведет к ошибке.

Указание в разделе ограничений целостности NOT NULL приводит к неявному порождению проверочного ограничения целостности для всей таблицы (см. подраздел 3.1.3, оператор NULL)

"CHECK (<имя столбца> IS NOT NULL)"

(где <имя столбца>, на значение полей которого накладывается ограничение).

Если ограничение NOT NULL не указано и раздел умолчаний отсутствует, то неявно порождается раздел умолчаний DEFAULT NULL.

Если указана спецификация уникальности (unique specification), то порождается соответствующая спецификация уникальности для таблицы по указанному столбцу.

Если в разделе ограничений целостности указано ограничение по ссылкам данного столбца (<reference specification>), то порождается соответствующее определение ограничения по ссылкам для таблицы:

FOREIGN KEY((<имя столбца>) <reference specification>.

Если указано проверочное ограничение столбца, то условие поиска этого ограничения должно ссылаться только на данный столбец и неявно порождается соответствующее проверочное ограничение для всей таблицы.

Примеры, связанные с ограничениями целостности, приводятся ниже, здесь же приведен пример создания базовой таблицы СОТРУДНИК (рис. 2.1) с помощью команды **CREATE TABLE**:

```
CREATE   TABLE    Сотрудник
(Ид_Сотр  SMALLINT,
Фамилия   CHAR     (17)  NOT NULL,
Имя       CHAR     (15),
Отчество  CHAR     (17),
ИНН       CHAR     (12)  NOT NULL,
Год_рожд  INT  (4)  NOT NULL,
Пол  CHAR      (1),
Город     CHAR     (15),
Район     CHAR     (15),
Индекс    CHAR     (8),
Ид_Совм   SMALLINT);
```

Предложение CREATE TABLE (создать таблицу) должно содержать имя базовой таблицы, которая должна быть создана, имена столбцов и типы данных полей (столбцов). При вводе данного предложения с терминала или из пользовательской программы СУБД построит пустую таблицу, которая будет содержать только заголовочную часть, но строк с данными в ней не будет. Однако если появится желание сразу ее заполнить, то с помощью оператора **INSERT** (операция присваивания, глава 6) это делается просто, и Вы можете создать таблицу, аналогичную таблице СОТРУДНИК. В приведенном примере Вы также увидели простое ограничение целостности для столбца *Фамилия*, для которого введено требование обязательного присутствия значения фамилии во вводимых строках создаваемой базовой таблицы.

Теперь если потребуется увидеть список сотрудников 1940 года рождения, среди всех введенных сотрудников, то для этого на терминале следует набрать следующий текст (синтаксис конструкции SELECT (выбрать) FROM (из) WHERE (где) <условие> - приведен в четвертой главе):

```
SELECT    ИД_Сотр, Фамилия

FROM      Сотрудник

WHERE     Год_рожд=1940;
```

И сразу на экране Вы получите следующий результат запроса:

ИД_Сотр	Фамилия
5	Петухов

СУБД, исполняя команду SELECT (выбрать), проверит синтаксис команды и создаст пустую рабочую таблицу с именами столбцов, которые указаны во фразе SELECT. Заметим, что типы полей столбцов будут соответствовать типам соответствующих столбцов базовой таблицы. Далее система выберет из базовой таблицы строки (записи), у которых в столбце *Год_рожд* есть значение 1940, и выполнит процедуру вывода рабочей таблицы на экран. Естественно, после исполнения запроса система должна уничтожить рабочую таблицу, если Вы не укажите ей что-то другое.

Так же просто встраиваются и любые другие выражения от значений хранимых полей (разрешены все арифметические операции: { +, - , * , / }).

3.1.3. Переопределение имени столбца AS

После выражения может записываться его псевдоним в форме **AS<выражение>**. В качестве псевдонима может фигурировать любой идентификатор, на который потом можно будет, при необходимости сослаться. Указанный псевдоним будет при отображении результатов фигурировать в заголовке таблицы.

Приведем пример использования выражения:

SELECT Фамилия, Имя, 2003-Год_рожд AS Возраст

FROM Сотрудник;

Примечание. В предложении указана арифметическая операция минус (-).

В этом случае результат запроса даст следующую рабочую таблицу (рис. 3.1)

Фамилия	Имя	Возраст
Иванов	Иван	54
Петров	Иван	54
Сидоров	Петр	56
…	…	…

Рис. 3.1. Таблица возрастов сотрудников

Надо иметь в виду, что с русскими буквами в операторах SQL могут возникать всевозможные неприятности. Некоторые символы могут восприниматься системой неверно и приводить якобы к синтаксическим ошибкам. Почему-то чаще всего вызывают недоразумения символы “ч” и “я” русского алфавита.

Полученные данные вычислены по хранимым значениям столбца (атрибута) *Год_ рожд* и будут существовать на экране до следующей смены изображения на экране. Если возникнет необходимость сохранить эти данные в какой-нибудь базовой таблице, то это обеспечит команда INSERT (вставить, присвоить, - рассмотренная в 6-й главе).

Если пользователь хочет уточнить выводимые данные информацией из других таблиц, то следует написать следующее предложение:

SELECT Фамилия, Имя, 2003-Год_рожд AS Возраст , Должность, Оклад

FROM Сотрудник, Отдел_Сотрудник

WHERE Сотрудник.ИД_Сотр=Отдел_Сотрудник.ИД_Сотр;

Результатом запроса будет рабочая таблица (рис. 3.2).

Фамилия	Имя	Возраст	Должность	Оклад
Иванов	Иван	54	Доцент	2500
Петров	Иван	54	Доцент	2500
Сидоров	Петр	56	Ст.препод	2000
…	…	…		

Рис. 3.2. Связь таблиц

Для получения рабочей таблицы была выполнена операция эквисоединения таблиц СОТРУДНИК (рис. 2.1) и ОТДЕЛ_ СОТРУДНИК (рис. 2.3) по условию соединения равенства значений уникальных идентификаторов *ИД_Сотр* в таблице СОТРУДНИК и таблице, закрепляющей сотрудников за отделами ОТДЕЛ_ СОТРУДНИК.

Если подобный запрос требуется достаточно часто, то его целесообразно оформить в виде представления(**VIEW -** параграф 3.2).

Рассмотренный выше пример показывает, что во фразе SELECT может содержаться не только перечень столбцов, но и выражение, с помощью которого вычисляются значения столбца.

Как же работает СУБД, если элемент выражения к данному моменту не определен или, как принято говорить, имеет NULL значение ?. Посмотрим внимательно на таблицу ОТДЕЛ_ СОТРУДНИК, точнее, на столбец *Дата_ увольнения*. В рассматриваемой таблице только один сотрудник помечен как уволенный. Все остальные элементы этого столбца имеют неопределенные значения. В СУБД, чтобы единым образом представлять "неизвестные значения" и введено понятие **NULL значения**, которое просто указывает,

что значение данного поля в данный момент может быть не обозначено. При выводе на экран NULL значения каждая СУБД использует свои механизмы показа таких значений. Например, может быть выведено “NULL” в соответствующей строке столбца, для которого значение неизвестно [25, 28, 29], или какая-либо другая информация, не противоречащая здравому смыслу.

Работа с NULL значениями

Часто бывает, что записи в таблице имеют какие-то незаполненные поля, например, потому что ввод информации не завершен или потому что это поле просто не заполнялось. SQL учитывает такой вариант, позволяя Вам вводить значение NULL (ПУСТОЙ) в поле, вместо значения. Когда значение поля равно NULL, это означает, что программа базы данных специально промаркировала это поле как не имеющее никакого значения для этой строки (или записи). Это отличается от просто назначения полю значения нуля или пробела, которые база данных будет обрабатывать так же, как и любое другое значение. Точно так же, как NULL не является техническим значением, оно не имеет и типа данных. Оно может помещаться в любой тип поля. Тем не менее NULL в SQL часто упоминается как нуль.

Предположим, что вы приняли нового сотрудника в отдел. Естественно, что поле с *Годом_увольнения* у такой записи в данный момент будет не заполнено. Вы можете заполнить это поле только в том случае, если сотрудник уволится. В данный момент времени данное поле будет иметь NULL значение.

NULL ОПЕРАТОР

Так как NULL указывает на отсутствие значения, вы не можете знать, каков будет результат любого сравнения с использованием NULL. Когда NULL сравнивается с любым значением, даже с другим таким же NULL, результат будет ни верным, ни неверным, он неизвестен.

Заметим, что поскольку SQL допускает наличие в базе данных неопределенных значений, то вычисление условия поиска производится не в булевой, а в трехзначной логике со значениями true, false и unknown (неизвестно). Для любого предиката известно, в каких ситуациях он может порождать значение unknown. Булевские операции AND, OR и NOT работают в трехзначной логике следующим образом:

true AND unknown = unknown,

unknown AND true = unknown,

unknown AND unknown = unknown,

true OR unknown = true,

unknown OR true = true

unknown OR unknown = unknown,

NOT unknown = unknown.

Следовательно, выражение типа "Фамилия = NULL" будет неизвестно, независимо от значения атрибута *Фамилия*.

Вы должны уметь различать "неверно" и "неизвестно" для строк, содержащих значения столбцов, которые не соответствуют условию предиката и которые содержат NULL в столбцах. В SQL присутствует специальный оператор **IS,** который используется с ключевым словом NULL для размещения значения NULL.

Найдем все записи в нашей таблице СОТРУДНИК (рис. 2.1) с NULL значениями в столбце Фамилия:

SELECT *

FROM Сотрудник

WHERE Фамилия IS NULL;

Здесь не будет никакого вывода, потому что нет никаких значений NULL в столбце *фамилия* нашей таблицы.

Примечание. В некоторых диалектах SQL, указав *****, Вы просто приказываете показать **ВСЕ** атрибуты отношения.

NOT с NULL

Например, если какой-то столбец имеет NULL-значение и возникает необходимость устранить NULL из вывода, то используется **NOT**, чтобы изменить на противоположное значение предиката:

SELECT *

FROM Сотрудник

```
WHERE Город NOT NULL;
```

В таблице СОТРУДНИК (рис. 2.1) отсутствуют значения NULL в столбце *Город*, поэтому будет выведена вся таблица СОТРУДНИКОВ. Следующие два запроса эквивалентны предыдущему.

```
SELECT *
FROM Сотрудник
WHERE NOT Город IS NULL;
```

и

```
SELECT *
FROM Сотрудник
WHERE Город      NOT  IN ("Париж");
```

В заключение рассмотрения работы с NULL значениями рассмотрим следующий запрос:

```
SELECT      ИД_Сотр, Должность, Дата_приема, Дата_увольнения,
        ”Проработал=”, Дата_увольнения - Дата_приема
FROM        Сотрудник;
```

Смысл этого запроса вроде бы ясен – сколько лет проработали уволенные сотрудники в своих отделах. Результат запроса можно увидеть на рис. 3.3.

Отметим, что каждая СУБД с помощью специальных команд может установить свой режим представления NULL-значений при выполнении числовых расчетов, запрещая или разрешая замены NULL-значений нулем. Если будет разрешено заместить NULL-значения, то результат вычислений будет иметь численное значение. В противном случае результат будет иметь неопределенное значение. На практике чаще всего стараются избежать NULL значений и заменяют, например, значение символьного поля на известное слово для пользователя ("Не известен").

В заключение рассмотрения использования ключевого слова AS покажем, как используется данное слово в команде **CREATE VIEW (создать представление).** Подробно данное предложение языка SQL рассмотрено в параграфе 3.2. Здесь же только укажем, что представление (обзор) - это именованная таблица (**пустая, виртуальная**), в которой приведен перечень имен столбцов таблиц и имен базовых таблиц, связанных каким-то предикатным условием между собой. Образно говоря представление есть "окно" в одну или несколько базовых таблиц.

Ид_ Сотр.	Должность	Дата приема	Дата увольнения	Проработал
1	Доцент	1977		-0-
2	Доцент	1979		-0-
3	Ст.препод.	1987		-0-
5	Профессор	1970		-0-
6	Инженер	1999		-0-
8	Зав. кафедрой	1975	2003	28
7	Ассистент	2003		-0-
4	Доцент	2000		-0-
10	Зав. кафедрой	1977		-0-
11	Зав. кафедрой	1980		-0-
9	Профессор	1980		-0-

Рис. 3.3. Вывод NULL значений

Приведем пример предложения **CREATE VIEW** для предыдущего примера:

```
CREATE VIEW    Все_ о_ Сотруднике
SELECT     Фамилия, Имя, 2003-Год_Рожд    AS  Возраст, Долж, Оклад
FROM       Сотрудник, Отдел_ Сотрудник
WHERE      Сотрудник.ИД_Сотр=Отдел_ Сотрудник.ИД_Сотр;
```

Приведенная конструкция описывает пустую таблицу с именем Все_о_Сотруднике, в которую при исполнении запроса будут закачиваться данные из базовых таблиц с выборкой из тех столбцов, которые указаны после команды SELECT. Причем выборка из каждой строки базовых таблиц может находиться под управлением функции Dbsetrelation() (установить реляционную связь с использованием физических индексов) той СУБД, на которой Вы работаете. Различные СУБД на физическом уровне используют различные механизмы реализации запросов и здесь не рассматриваются.

Как Вы уже догадываетесь, полученную таблицу можно использовать как основу для другого запроса:

```
SELECT     Фамилия, Имя, Долж, Оклад
FROM       Все_о_сотруднике;
```

Одна из целей создания представлений это также и возможность скрыть от некоторых пользователей реальный состав базовых отношений (таблиц), упростить их восприятие и в конечном итоге повысить уровень безопасности хранимых данных.

Другим представителем виртуальных таблиц является понятие курсора (**CURSOR**). По определению **курсор –** это пустая поименованная таблица, в которой определен список необходимых полей для запроса и признак тех строк этой таблицы, которые в ней будут использованы.

Если для обычных пользователей представления воспринимаются как базовые таблицы (с некоторыми ограничениями) и доступны только в интерактивном режиме, то для пользователей программистов представления и курсоры доступны также и в их прикладных программах. Но курсор, прежде всего, используется для процедурной работы с таблицами в прикладных программах.

Например, после объявления курсора:

```
DECLARE  O_сотруднике_все        CURSOR FOR
SELECT      Фамилия, Имя, Долж, Оклад
FROM        Все_ о_ сотруднике;
```

*Примечание***.** DECLARE - оператор объявления курсора.

И при его активизации (производится оператором **OPEN <имя таблицы>** во вложенном SQL) будет инициализирована временная рабочая таблица с соответствующим содержанием данных из таблиц СОТРУДНИК (рис. 2.1) и ОТДЕЛ_СОТРУДНИК (рис. 2.3), для которой будет создан специальный указатель, установленный в момент открытия рабочей таблицы на ее первую строку. Далее с помощью предложения **FETCH** (выбрать) можно указанным столбам (полям таблицы) присваивать необходимые значения. Обычно **FETCH** используется в цикле, и после обработки текущей строки автоматически указатель устанавливается на следующую строку курсора и т.д., до последней строки виртуальной таблицы включительно.

3.1.4. Ограничения целостности таблицы

Ограничения целостности таблицы имеют следующий синтаксис:

<table constraint definition> ::=

<unique constraint definition>

| <referential constraint definition>

| <check constraint definition>

<unique constraint definition> ::=

<unique specification> (<unique column list>)

<unique specification> ::= UNIQUE | PRIMARY KEY

<unique column list> ::= <column name> [{,<column name>}...]

<referential constraint definition> ::=

FOREIGN KEY (<referencing columns>) <references specification>

<references specification> ::=

REFERENCES <referenced table and columns>

<referencing columns> ::= <reference column list>

<referenced table and columns> ::=

<table name> [(<reference column list>)]

<reference column list> ::= <column name> [{,<column name>}...]

<check constraint definition> ::= CHECK (<search condition>)

Для одной таблицы может быть задано несколько ограничений целостности, в том числе те, которые неявно порождаются ограничениями целостности столбцов. Стандарт

SQL устанавливает, что ограничения таблицы фактически проверяются при выполнении каждого оператора SQL.

Замечание. Наличие правильно подобранного набора ограничений БД очень важно для надежного функционирования прикладной информационной системы. Вместе с тем в некоторых СУБД ограничения целостности практически не поддерживаются. Поэтому при проектировании прикладной системы необходимо принять решение о том, что более существенно: рассчитывать на поддержку ограничений целостности, но ограничить набор возможных СУБД, или отказаться от их использования на уровне SQL. Кроме того, как показано в [1, 2, 3, 6, 13, 16] набор таблиц и их состав не может определяться только на интуитивном уровне, иначе прикладная информационная система будет работать крайне неустойчиво или придется постоянно переделывать схему модели и прикладные программы пользователей. SQL не несет никакой ответственности за вторую составляющую алгебры (носитель алгебры).

SQL – это средство для манипулирования набором отношений (таблица не является полным аналогом отношения) реляционной модели с указанием некоторых ограничений для каждого отношения модели и всей схемы в целом.

Пример создания таблицы ЦЕХ_ИЗДЕЛИЕ (рис. 1.8) с помощью оператора CREATE TABLE:

```
CREATE TABLE Цех_Изделие
(Nцеха  CHAR (2) Nцеха NOT NULL,
Шифр_изделия CHAR (7) Шифр_изделия NOT NULL,
Кол-во_ по_ плану INT  (4) Кол-во_ по_ плану NOT NULL,
PRIMARY KEY (Nцеха, Шифр_изделия),
FOREIGN KEY  (Nцеха) REFERENCES    ЦЕХ
                    ON DELETE CASCADE
                    ON UPDATE CASCADE,
FOREIGN KEY  (Шифр_ изделия) REFERENCES       ИЗДЕЛИЕ
                    ON DELETE CASCADE
                    ON UPDATE CASCADE,
CHECK (Кол-во_ по_ плану>0 AND Кол-во_ по_ плану < 3000));
```

Здесь подразумевается, что таблицы ЦЕХ и ИЗДЕЛИЕ уже определены и в них явно проставлены определения PRIMARY KEY.

Далее представленная базовая таблица может быть изменена с помощью оператора **ALTER TABLE**.

Поддерживаются следующие ограничения:

- добавление новых столбцов;
- определение для существующего столбца нового значения по умолчанию (замещается старое значение):
- удаление для столбца существующего значения по умолчанию;
- удаление существующего столбца;
- указание нового ограничения целостности для базовой таблицы;
- удаление существующего ограничения целостности для базовой таблицы.

Приведем пример лишь для первого случая:

ALTER TABLE Цех_изделие ADD COLUMN Факт_Кол-во INTEGER (5) DEFAULT 0;

С помощью этого оператора добавляется столбец *Факт_ Кол-во* типа INTEGER (5) в базовой таблице ЦЕХ_ИЗДЕЛИЕ. Все существующие строки в этой таблице расширяются с трех столбцов до четырех; во всех случаях значение полей добавленного столбца будет 0.

Существующая таблица может быть удалена с помощью оператора DROP TABLE, который имеет следующий синтаксис:

DROP TABLE Цех_Изделие < option >;

Здесь опция (option) может быть **RESTRICT** или **CASCADE**. Если будет указана опция **RESTRICT** и на базовую таблицу есть ссылки в каком-нибудь определении представления (**VIEW**) или ограничения целостности, то операция **DROP** будет завершена сообщением о невозможности ее исполнения. Если указана опция **CASCADE**, то операция **DROP** удалит таблицу со всем ее содержимым.

Любые определения представлений, которые имеют ссылку на удаляемую таблицу, будут уничтожены.

Пример создания таблицы ОТДЕЛ_ СОТРУДНИК (рис. 2.3) с помощью оператора CREATE TABLE:

```
CREATE TABLE Отдел_ Сотрудники
(Ид_Сотр SMALLINT     Ид_Сотр NOT NULL,
Ид_Отдела CHAR (3)    Ид_Отдела NOT NULL,
Должность CHAR (15) Должность NOT NULL,
Оклад         INT    (4)     Оклад         NOT NULL,
Дата_ приема INT  (4) Дата_ приема NOT NULL,
Дата_ увольнения INT  (4) NULL,
PRIMARY KEY (Ид_Отдела, Ид_Сотр),
FOREIGN KEY  (Ид_Отдела) REFERENCES      Отдел
                         ON DELETE CASCADE
                         ON UPDATE CASCADE,
FOREIGN KEY  (Ид_Сотр) REFERENCES         Сотрудник
                         ON DELETE CASCADE
                         ON UPDATE CASCADE,
    CHECK (Оклад >0 AND Оклад < 30000));
```

Попытка добавить в таблицу ОТДЕЛ_СОТРУДНИК информацию о *Сотруднике* или *Отделе*, которые отсутствуют в базовых таблицах ОТДЕЛ (рис. 2.2) или СОТРУДНИК (рис. 2.1) приведет к сообщению системы о некорректности операции (сработает ссылочная целостность REFERENCES либо по таблице ОТДЕЛ, либо СОТРУДНИК). Удаление же какого-то сотрудника или отдела приведет к автоматическому удалению информации в таблице ОТДЕЛ_СОТРУДНИК (сработает ссылочная целостность по операции ON DELETE). Подробно этот вопрос рассмотрен в главе 6. Заметьте, что оператор CREATE TABLE для таблицы ОТДЕЛ_ СОТРУДНИК (рис. 2.3) запрещает (или, как говорят, накладывает явное ограничение целостности)

присутствие строк, для которых выполняется условие: CHECK (Оклад > 0 AND Оклад < 30000).

Для дальнейшего изложения обозначим абстрактную таблицу символом R, а множество атрибутов, ее составляющих, символом A, для которой определяются ограничения целостности.

Ограничение уникальности (PRIMARY)

Каждое имя столбца в списке уникальности (unique column list) должно именовать столбец R и не должно входить в этот список более одного раза. При определении столбца, входящего в список уникальности, должно быть указано ограничение столбца NOT NULL. Среди ограничений уникальности R не должно быть более одного определения первичного ключа (ограничения уникальности с ключевым словом PRIMARY KEY).

Действие ограничения уникальности состоит в том, что в таблице R не допускается появление двух или более строк, в столбце для которого определено ограничение уникальности.

Ограничение по ссылкам (REFERENCES)

Ограничение по ссылкам от заданного набора столбцов A таблицы R на заданный набор столбцов A1 некоторой определенной к этому моменту таблицы R1 определяет условие на содержимое обеих этих таблиц, при котором ссылки можно считать корректными.

Если список столбцов A1 явно специфицирован в определении ограничения по ссылкам, то требуется, чтобы этот список явно входил в какое-либо определение уникальности таблицы R1. Если же список A1 не специфицирован явно в определении ограничения по ссылкам таблицы R, то требуется, чтобы в определении таблицы R1 присутствовало определение первичного ключа, и список A1 неявно полагается совпадающим со списком имен столбцов из определения первичного ключа таблицы R1. Имена столбцов списков A и A1 должны именовать столбцы таблиц R и R1 соответственно, и не должны появляться в списках более одного раза. Списки столбцов A и A1 должны содержать одинаковое число элементов, и столбец таблицы R, идентифицируемый i-м элементом списка A, должен иметь тот же тип, что столбец таблицы R1, идентифицируемый i-м элементом списка A1.

По определению, таблицы R и R1 удовлетворяют заданному ограничению по ссылкам, если для каждой строки s таблицы R такой, что все значения столбцов, идентифицируемых списком A, не являются неопределенными, существует строка s1 таблицы R1 такая, что значения столбцов s1, идентифицируемых списком A1, позиционно равны значениям столбцов s, идентифицируемых списком A. Другими словами, это можно сформулировать так: ограничение по ссылкам удовлетворяется, если для каждой корректной ссылки существует объект, на который она ссылается. В понятной (привычной) программистам терминологии ограничение по ссылкам не позволяет производить "висячие" ссылки, не ведущие ни к какому объекту.

Проверочное ограничение (CHECK)

Проверочное ограничение специфицирует условие, которому должна удовлетворять в отдельности каждая строка таблицы R. Это условие не должно содержать подзапросов (см. пункт 5.3), спецификаций агрегатных функций (см. пункт 4.4), а также ссылок на внешние переменные или параметры. В него могут входить только имена столбцов данной таблицы и литеральные константы.

Таблица удовлетворяет проверочному ограничению целостности в том и только в том случае, когда вычисление условия для каждой строки таблицы дает true.

Замечание. В некоторых реализациях допускаются расширенные механизмы ограничений по ссылкам и проверочных ограничений. Однако такое предоставляют только немногие СУБД.

3.2. Определение представлений (VIEW обзоров)

Механизм представлений (**view**) является средством языка SQL, позволяющим скрыть реальную структуру БД от пользователей за счет определения представления БД, которое реально является некоторым хранимым в БД запросом с именованными столбцами, а для пользователя ничем не отличается от базовой таблицы БД (с учетом технических ограничений). Любая реализация должна гарантировать, что состояние представляемой таблицы точно соответствует состоянию базовых таблиц, на которых определено представление. Обычно вычисление представляемой таблицы (актуализация соответствующего запроса) производится каждый раз при использовании представления.

Реально обзоры не являются таблицами, но упоминаются здесь, так как во многих случаях они трактуются как таблицы. Обзоры - это "запомненные запросы", которые представляются в формате таблиц. Таким образом, обзор может содержать (как и таблица) необходимое число столбцов (обычно до 254 столбцов). Ссылки на обзоры и таблицы во многих SQL -операторах идентичны (особенно в операторах языка манипулирования данными (**DML)** и запросах), в других же случаях использование обзоров запрещено. Обзоры, скрывая физическое представление данных, обеспечивают их независимость от приложений, которые заботит только логическая структура данных.

Обзоры полезны во многих случаях, включая удобство и безопасность. Например - обзоры, используемые для изменения и созданные с опцией WITH CHECK, могут дополнять проверки данных в столбцах, подразумеваемые при создании таблицы с помощью оператора CREATE TABLE. Использование обзоров для безопасности здесь не рассматриваются.

Обзоры не требуют физической памяти на диске, кроме строк в словаре данных, используемых для запоминания определения обзора (сохраненный запрос). Напомним, что словарь данных - это базовая системная таблица, создаваемая СУБД в момент исполнения оператора CREATE DATABASE, в которой фиксируются и созданные, и сохраненные обзоры.

Если удаляются таблицы, на которых определены обзоры, то некоторые системы не удаляют обзоры, исходя из предположения, что таблица может быть создана заново или импортирована. Однако попытка использовать обзор при отсутствии таблицы приведет к ошибке. Если структура вновь созданной таблицы отличается от структуры предыдущей, результат использования обзора непредсказуем.

В стандарте SQL оператор определения представления имеет следующий синтаксис:

<view definition> ::=

CREATE VIEW <table name> [(<view column list>)]

AS <query specification> [WITH CHECK OPTION]

<view column list> ::= <column name> [{,<column name>}...]

Определяемая представляемая таблица V является изменяемой (т.е. по отношению к V можно использовать операторы DELETE и UPDATE - глава 6) в том и только в том случае, если выполняются следующие условия для спецификации запроса:

- в списке выборки не указано ключевое слово DISTINCT;
- каждое арифметическое выражение в списке выборки представляет собой одну спецификацию столбца, и спецификация одного столбца не появляется более одного раза;
- в разделе FROM указана только одна таблица, являющаяся либо базовой таблицей, либо изменяемой представляемой таблицей;
- в условии выборки раздела WHERE не используются подзапросы;
- в табличном выражении отсутствуют разделы GROUP BY и HAVING.

Замечание. Эти ограничения являются очень сильными. В реализациях они могут быть ослаблены. Но, если стремиться к мобильности, не следует пользоваться расширенными возможностями.

Если в списке выборки спецификации запроса имеется хотя бы одно арифметическое выражение, состоящее не из одной спецификации столбца, или если одно имя столбца участвует в списке выборки более одного раза, определение представления должно содержать список имен столбцов представляемой таблицы. Более просто, нужно явно именовать столбцы представляемой таблицы, если эти имена не наследуются от столбцов таблиц раздела FROM спецификации запроса.

Требование WITH CHECK OPTION в определении представления имеет смысл только в случае определения изменяемой представляемой таблицы, которая определяется спецификацией запроса, содержащей раздел WHERE. При наличии этого требования не допускаются изменения представляемой таблицы, которые приводят к появлению в базовых таблиц строк, не видимых в представляемой таблице (т.е. таких строк, которые не удовлетворяют условию поиска раздела WHERE спецификации запроса). Если WITH CHECK OPTION в определении представления отсутствует, такой контроль не производится.

3.3. Определение прав доступа (привилегий)

В соответствии с идеологией языка SQL контроль прав доступа данного пользователя к таблицам БД производится на основе механизма привилегий. Фактически, этот механизм состоит в том, что для выполнения любого действия над таблицей пользователь должен обладать соответствующей привилегией (реально все возможные действия описываются фиксированным стандартным набором привилегий). Пользователь, создавший таблицу, автоматически становится владельцем всех

возможных привилегий на выполнение операций над этой таблицей. В число этих привилегий входит привилегия на передачу всех или некоторых привилегий по отношению к данной таблице другому пользователю, включая привилегию на передачу привилегий. Иногда поддерживается и обратная операция изъятия привилегий от пользователя, ранее их получившего.

Практически каждая СУБД решает проблемы, связанные с определением прав доступа своими уникальными средствами [17, 20, 22, 25], и по понятным причинам не афиширует эти средства. Однако мы должны понимать, что эти средства должны позволять обеспечить необходимую нам информационную безопасность.

Глава 4. Запросы

4.1. Структура запросов

Запрос – это команда, которая указывается в программе или вводится с командной строки для поиска информации в базе данных и которая сообщает ей, чтобы она вывела определенную информацию из таблиц в память. Эта информация обычно посылается непосредственно на экран компьютера или терминала, которым вы пользуетесь, хотя в большинстве случаев ее можно также послать принтеру, сохранить в файле (как объект в памяти компьютера) или представить как входную информацию для другой команды или процесса.

Запросы обычно рассматриваются как часть языка DML. Однако, так как запрос не меняет информацию в таблицах, а просто показывает ее пользователю, мы будем рассматривать запросы как самостоятельную категорию среди команд DML**,** которые производят действие, а не просто показывают содержание базы данных.

Все запросы в SQL состоят из одиночной команды. Структура этой команды достаточно проста, но возможность ее расширения иногда затрудняет ее понимание. Эта команда называется – **SELECT** (ВЫБОР).

4.1.1. Команда SELECT

В самой простой форме команда SELECT просто инструктирует базу данных, чтобы извлечь информацию из таблицы. Например, вы могли бы вывести таблицу ВЕДОМОСТЬ_ОПЛАТЫ (рис. 4.1) напечатав следующее:

```
SELECT      ИД_Отдела, ИД_Сотр, Период, Сумма, Вид_оплаты
FROM  Ведомость_ оплаты;
```

Заметьте, что содержимое выводимой таблицы будет отражать то состояние базовой таблицы (рис. 2.4), которое ей присуще в данный момент времени.

При написании имен атрибутов и имен отношений, используемых в запросе, регистр, в принципе, не важен, так как СУБД обычно использует стандартную функцию Upper() для обеспечения однозначности. Однако в написании указанных имен должны отсутствовать пробелы. В данном тексте мы иногда отходим от данного правила, для обеспечения его читаемости.

Ведомость оплаты

Ид_ Отд	Ид_ Сотр.	Период	Сумма	Вид_ оплаты
1	1	Март	1200	
2	1	Март	1200	
3	1	Март	1000	
1	1	Апрель	1200	
1	1	Март	800	
……				

Рис. 4.1. Простейший запрос к таблице

Другими словами, эта команда просто выводит все данные из таблицы. Большинство программ будут также давать заголовки столбца, как показано выше, а некоторые позволяют детальное форматирование вывода, но это уже вне стандартной спецификации языка SQL.

Что же обозначает каждый элемент этого предложения.

SELECT — Ключевое слово, которое сообщает базе данных, что эта команда - запрос. Все запросы начинаются этим словом, сопровождаемым пробелом.

Ид_Отд, ИД_Сотр, Период, Сумма, Вид_оплаты — Это - список столбцов из таблицы, которые выбираются запросом. Любые столбцы, не перечисленные здесь, не будут включены в вывод команды. Это, конечно, не значит, что они будут удалены или их информация будет стерта из таблиц, потому что запрос не воздействует на информацию в таблицах; он только показывает данные.

FROM — Ключевое слово, подобно SELECT, которое должно быть представлено в каждом запросе. Оно сопровождается пробелом и затем именем таблицы используемой в качестве источника информации. В данном случае - это таблица ВЕДОМОСТЬ_ОПЛАТЫ.

; Точка с запятой используется во всех интерактивных командах SQL, чтобы сообщать базе данных, что команда заполнена и готова выполниться. В некоторых системах наклонная черта влево (\) в строке является индикатором конца команды.

Естественно, запрос такого характера не будет упорядочивать вывод строк в таблице. Та же самая команда, выполненная с теми же самыми данными, но в разное время не сможет вывести тот же самый порядок. Обычно строки обнаруживаются в том порядке, в котором они найдены в таблице. И обычно этот порядок произволен. Это не обязательно будет тот порядок, в котором данные вводились или сохранялись. Вы можете упорядочивать вывод командами SQL непосредственно: с помощью специального предложения. Позже, мы покажем, как это делается. А сейчас просто усвойте, что в отсутствии явного упорядочения, нет никакого определенного порядка в выводе.

Использование возврата (Клавиша ENTER) при наборе предложения является произвольным. Пользователь должен только установить, как удобнее составить запрос, в несколько строк или в одну строку.

В то же время, если бы набрали следующее предложение:

```
SELECT *
FROM    Ведомость_ оплаты;
```

то сразу же на экране дисплея увидели бы таблицу ВЕДОМОСТЬ_ ОПЛАТЫ, т.е. предыдущий и приведенный запросы эквивалентны по результату.

Звездочка (*) - это обозначение "все", т.е. "все случаи, удовлетворяющие определению". Звездочка (*) применяется для вывода полного списка столбцов.

4.1.2. Описание SELECT

В общем случае команда SELECT начинается с ключевого слова SELECT, сопровождаемого пробелом. После этого может следовать список имен столбцов,

которые вы хотите видеть, отделяемые запятыми. Результатом реализации предложения SELECT будет другая таблица, к этой новой таблице снова можно применить команду SELECT и т.д. Таким образом, получается **вложенный запрос**.

Ключевое слово FROM, следующее далее, сопровождается пробелом и именем таблицы, запрос к которой делается. В заключение точка с запятой (;) должна использоваться, чтобы закончить запрос и указать, что команда готова к выполнению.

Оператор SQL - это строка на языке SQL, передаваемая на обработку СУБД. Содержание строки состоит из зарезервированных слов SQL (слов, имеющих в SQL специальное значение и обычно не используемых в других целях). Например - слова SELECT и UPDATE - зарезервированные слова и они не могут использоваться в качестве названий таблиц. Оператор должен быть эквивалентен "предложению" SQL как, например:

```
SELECT    Название_ отдела

FROM      Отдел;
```

Только такой оператор может быть выполнен (в отличие от фрагмента) например, задание строки:

```
SELECT    Название_ отдела;
```

Это приведет к сообщению, запрашивающему дополнительную информацию (для интерактивного режима), или в случае использования вложенного SQL к ошибке исполнения программного кода.

Просмотр только определенного столбца

Команда SELECT способна извлечь строго определенную информацию из таблицы. Предоставляется возможность увидеть только определенные столбцы таблицы. Это выполняется легко, простым перечислением имен столбцов, которые Вы хотите видеть.

Например, запрос:

```
SELECT Фамилия, Имя          FROM Сотрудник;
```

будет производить вывод, показанный на рис. 4.2.

Фамилия	Имя
Иванов	Иван
Петров	Иван
Сидоров	Петр
Панов	Антон
Петухов	Виктор
Иванова	Вера
Петрова	Нина
Сидорова	Екатерина
Никитин	Валентин
Мухин	Александр
Попов	Анатолий
Иванов	Иван
Хохлов	Иван
Яковлев	Иван

Рис. 4.2. Выбор определенных столбцов

Могут иметься таблицы, включающие большое количество столбцов, содержащих данные, не все из которых являются относящимися к поставленной задаче. Следовательно, можно найти способ подбора и выбора только полезных для Вас столбцов.

4.1.3. Сортировка результирующей таблицы

Даже если столбцы таблицы, по определению, упорядочены, это не означает, что вы будете восстанавливать их в том же порядке. Конечно, звездочка (*) покажет все столбцы в их естественном порядке, но, если вы укажете столбцы отдельно, Вы можете получить их в том порядке, который Вам необходим.

Необходимый порядок отображения строк задается с помощью фразы **ORDER BY** (Порядок) команды SELECT.

Например, если к таблице СОТРУДНИК (рис. 2.1), осуществить запрос, в котором первым указать атрибут *Год_ рождения* сотрудника и перечислить его атрибуты *Фамилия, Имя, Отчество*, указав при этом после фразы WHERE (где)**,** что нас

интересуют только люди с годом рождения меньше 1970, то получим следующий результат (рис. 4.3).

```
SELECT      Год_ рожд, Фамилия, Имя, Отчество
FROM    Сотрудник
WHERE       Год_рожд <1970
ORDER BY Год_рожд   ASC;
```

Год_рожд	Фамилия	Имя	Отчество
1940	Петухов	Виктор	Борисович
1947	Сидоров	Петр	Петрович
1947	Попов	Анатолий	Михайлович
1949	Иванов	Иван	Петрович
1949	Петров	Иван	Иванович
1952	Никитин	Валентин	Сергеевич
1964	Мухин	Александр	Михайлович

Рис. 4.3. Сотрудники определенного диапазона возрастов

Фраза ORDER BY с параметром ASC заставит систему отсортировать результирующую таблицу. Если бы потребовалось вывести таблицу СОТРУДНИКОВ в обратном порядке столбца *Год_рождения*, то следует написать следующее предложение:

```
SELECT      Год_ рожд, Фамилия, Имя, Отчество
FROM        Сотрудник
ORDER BY    Год_ рожд    DESC;
```

Обратите внимание, результирующая таблица будет содержать столько же строк, сколько их находится в базовой таблице. Ключевое слово DESC заставит систему отсортировать показываемую виртуальную таблицу по убыванию. Аналогичный запрос может быть подготовлен и для показа результирующей таблицы, отсортированной по *фамилии* сотрудника. Например:

```
SELECT Фамилия, Имя, Отчество, Год_ рожд
FROM Сотрудник
```

ORDER BY Фамилия DESC;

Сортировку можно задавать по нескольким столбцам. Если указать после имени столбца еще имя другого столбца, то по значениям второго столбца будут упорядочены строки, содержащие одинаковые значения в первом столбце. Порядок сортировки задается отдельно для каждого столбца, входящего в список фразы ORDER BY. Например, если необходим список фамилий сотрудников, сгруппированный по годам рождения, то в окне командной строки системы следует набрать следующую конструкцию:

SELECT Фамилия, Имя, Отчество, Год_ рожд

FROM Сотрудник

ORDER BY Фамилия ASC, Год_ Рожд DESC;

Пусть базовая таблица до сортировки имела вид (рис. 4.4):

Ид_ Сотр	Фамилия	Имя	Отчество	ИНН	Год рожд.	Пол
1	Иванов	Иван	Петрович	101	1949	М
2	Петров	Иван	Иванович	102	1949	М
3	Сидоров	Петр	Петрович	103	1947	М
4	Панов	Антон	Михайлович	104	1975	М
5	Петухов	Виктор	Борисович	105	1940	М
6	Иванова	Вера	Васильевна	116	1970	Ж
7	Петрова	Нина	Николаевна	217	1970	Ж
8	Сидрова	Ада	Ивановна	308	1970	Ж
9	Никитин	Виктор	Сергеевич	489	1952	М
10	Мухин	Степан	Михайлович	510	1964	М
11	Попов	Михаил	Михайлович	611	1947	М
12	Иванов	Иван	Иванович	712	1980	М
13	Хохлов	Иван	Васильевич	713	1960	М
14	Яковлев	Иван	Васильевич	714	1980	М

Рис. 4.4. Состояние таблицы до сортировки

После сортировки по приведенному выше предложению SELECT результат будет (рис. 4.5):

Ид_ Сотр	Фамилия	Имя	Отчество	ИНН	Год рожд.	Пол
5	Петухов	Виктор	Борисович	105	1940	М
3	Сидоров	Петр	Петрович	103	1947	М
11	Попов	Михаил	Михайлович	611	1947	М
1	Иванов	Иван	Петрович	101	1949	М
2	Петров	Иван	Иванович	102	1949	М
9	Никитин	Виктор	Сергеевич	489	1952	М
13	Хохлов	Иван	Васильевич	713	1960	М
10	Мухин	Степан	Михайлович	510	1964	М
6	Иванова	Вера	Васильевна	116	1970	Ж
7	Петрова	Нина	Николаевна	217	1970	Ж
8	Сидрова	Ада	Ивановна	308	1970	Ж
4	Панов	Антон	Михайлович	104	1975	М
12	Иванов	Иван	Иванович	712	1980	М
14	Яковлев	Иван	Васильевич	714	1980	М

Рис. 4.5. Сортировка по группам и в группе

Замечание. SQL в предложении ORDER BY не использует существующий порядок следования столбцов хранимой таблицы, например таблицы СОТРУДНИК (рис. 4.4), поскольку в реляционной базе данных синтаксис запросов к БД не зависит от способа хранения данных [25,28,29]. Так, например, при обращении к столбцам в том порядке, в котором они хранятся в таблице, возникла бы проблема. Поскольку в команде SELECT допускается задание столбцов из различных таблиц, при необходимости сослаться, например, на третий столбец двух разных столбцов, во фразе ORDER BY пришлось бы указывать один и тот же порядковый номер, что невозможно.

Некоторые другие способы упорядочивания рассмотрены в параграфе 4.6.

4.1.4. Удаление повторяющихся данных

DISTINCT (ОТЛИЧИЕ) - аргумент, который позволяет устранять двойные значения из результирующей таблицы предложения SELECT. Предположим, что вы хотите узнать список неповторяющихся фамилий сотрудников из таблицы рис. 2.1.

SELECT DISTINCT Фамилия

FROM Сотрудник;

Вывод для этого запроса показан в рис. 4.6 SQL всегда выдает ответ в виде таблицы, которую мы представили как список:

Фамилия
Иванов
Петров
…..

Фамилия {Иванов, Петров, Сидоров, Панов, Петухов, Иванова, Петрова, Сидорова, Никитин, Мухин, Попов, Хохлов, Яковлев}.

Рис. 4.6. Список уникальных значений

DISTINCT следит за тем, чтобы значения не были продублированы в списке. Фактически при добавлении новой строки в результирующую таблицу проверяется, не присутствует ли подобное значение уже в рабочей таблице. Это способ избежать избыточности данных, но важно, что при этом можно потерять нужную информацию. Так как если бы Вы хотели увидеть список фамилий сотрудников, а задали запрос с DISTINCT, то потеряли бы информацию об одном из Ивановых.

DISTINCT может указываться только один раз в данном предложении SELECT. Если вместо DISTINCT указать - ALL, то дублирование строк вывода сохранится. Параметр ALL работает в SELECT по умолчанию.

SELECT с DISTINCT – эквивалентен операции **проекции** реляционной

алгебры.

4.2. Использование фразы WHERE

Если нас интересуют из таблицы только определенные строки в данное время, то SQL дает возможность устанавливать критерии, определяющие, какие строки будут выбраны для вывода. Это делается с помощью фразы WHERE, точнее, с помощью предикатов которые указываются в этой фразе.

Фраза **WHERE** (где) - предложения команды SELECT позволяет Вам устанавливать предикаты, условие которых может быть или верным или неверным для любой строки таблицы. Команда извлекает только те строки из таблицы, для которой такое утверждение верно. Например, предположим, необходимо увидеть *Оклады* сотрудников, находящихся на *должностях* Доцент. Вы можете ввести такую команду:

```
SELECT Должность, Оклад

FROM   Отдел_ Сотрудники

WHERE Должность= "Доцент";
```

Когда предложение WHERE представлено, программа базы данных просматривает всю таблицу по одной строке и исследует каждую строку, чтобы определить истинность утверждения. Следовательно, для каждой записи таблицы ОТДЕЛ_СОТРУДНИК (рис. 2.3), программа рассмотрит текущее значение столбца *Должность*, определит, что оно равно "Доцент", и включит эту строку в рабочую область. Записи, не удовлетворяющие данному требованию, не будут включаться в результирующую таблицу. Вывод для вышеупомянутого запроса показан на рис. 4.7.

Должность	Оклад
Доцент	2500
Доцент	2500
Доцент	3100

Рис. 4.7. SELECT с предложением WHERE

Пример с числовым полем в предложении WHERE. Для таблицы ОТДЕЛ_СОТРУДНИК выбрать всех Доцентов с окладом 2500:

```
SELECT *        FROM Отдел_ сотрудник WHERE     Оклад= 2500;
```

Кавычки не используются здесь потому, что Оклад - это числовое поле. Заметим, что некоторые СУБД требуют, чтобы кавычка была одинарной. Результаты запроса показаны в рис. 4.8.

Ид_Отд	Ид_Сотр.	Должность		Оклад	Дата_ приема	Дата_ увольнения
1	1	Доцент		2500	1977	
1	2	Доцент		2500	1979	

Рис. 4.8. SELECT с числовым полем в предикате

Фраза WHERE совместима с предыдущим материалом в этой главе. Другими словами, Вы можете использовать номера столбцов, устранять дубликаты или переупорядочивать столбцы в команде SELECT, которая использует WHERE. Однако Вы можете изменять порядок столбцов для имен только в предложении SELECT, но не в предложении WHERE.

Естественно, критерий отбора, указываемый во фразе WHERE, может содержать несколько условий, соединенных логическими операторами:

AND и NOT AND - когда должны удовлетворяться оба разделяемых с помощью AND условия;

OR - когда должно удовлетворяться одно из разделяемых с помощью OR условий;

AND NOT - когда должно удовлетворяться первое условие и не должно второе;

OR NOT - когда или должно удовлетворяться первое условие или не должно удовлетворяться второе.

Причем существует приоритет AND над OR (сначала выполняются все операции AND и только после этого операции OR). Для получения желаемого результата WHERE-условия должны быть введены в правильном порядке, который можно организовать введением скобок.

Пусть базовая таблица СОТРУДНИК1 (рис 4.9) изначально имеет вид:

Ид_Отд	Ид_Сотр.	Должность	Оклад	Фамилия
1	1	Доцент	2500	Иванов
1	2	Доцент	2500	Петров
1	3	Ст. препод.	2000	Сидоров
1	5	Профессор	3700	Петухов
1	6	Инженер	1500	Сидорова
1	8	Зав. кафедрой	5000	Иванова
2	7	Ассистент	1700	Петрова
2	4	Доцент	3100	Панов
2	10	Зав. кафедрой	5000	Мухин
3	11	Зав. кафедрой	5000	Попов
2	9	Профессор	4800	Иванов

Рис. 4.9. Исходное состояние отношения СОТРУДНИК1

```
SELECT      Фамилия, Ид_Отд
        FROM        Сотрудник1
        WHERE Должность = "Доцент"
        AND Ид_Отд  = 1
        OR    Ид_Отд  = 2;
```

Здесь надо понимать, что мы хотим получить от базы:

1. "Выдать фамилии и отделы всех сотрудников с должностью Доцент в отделе с номером 1 и всех сотрудников в отделе с номером 2".

или

2. "Выдать фамилии и отделы всех сотрудников с должностью Доцент, но только если они работают в отделах с номерами 1 или 2".

Если цель вопроса - запрос номер 1: фамилии и номера отделов всех сотрудников с должностью Доцент, которые работают в отделах 1 или 2, то мы получим результат, в котором будет список всех сотрудников с должностью Доцент, работающих в отделе 1 и всех без исключения сотрудников отдела 2:

Фамилия	Ид_Отд
Иванов	1
Петров	1
Панов	2
Мухин	2
Иванов	2

Если же мы хотим получить только сотрудников с должностью Доцент работающих только в отделах 1 и 2 , то запрос должен быть оформлен иначе:

```
SELECT    Фамилия, Ид_Отд
FROM      Сотрудник1
WHERE     Должность = "Доцент"
AND       (Ид_Отд  = 1
OR        Ид_Отд  = 2 );
```

Результат:

Фамилия	Ид_Отд
Иванов	1
Петров	1
Панов	2

Если мы добавим скобки в другом месте:

```
SELECT    Фамилия, Ид_Отд
FROM      Сотрудник1
WHERE     (Должность = "Доцент"
```

```
AND        Ид_Отд     = 1 )
OR    Ид_Отд     = 2;
```

то получим результат первого запроса: все сотрудники с должностью Доцент отдела 1 и все сотрудники отдела 2.

Рассмотрим следующую конструкцию предложения SELECT:

```
SELECT *
FROM       Сотрудник1
WHERE      NOT Должность = "Доцент"
OR         Оклад > 4500;
```

NOT применяется здесь только к выражению Должность = "Доцент", но не к выражению Оклад > 4500 . SQL может применять NOT только к выражению непосредственно за ним следующим. Вы получите другой результат при команде:

```
SELECT *
FROM       Сотрудник1
WHERE      NOT (Должность = "Доцент"
OR         Оклад > 4500);
```

Здесь SQL понимает круглые скобки как означающие, что все внутри них будет оцениваться первым и обрабатываться как единое выражение с помощью всего, что снаружи них (это является стандартной интерпретацией в математике). Другими словами, SQL берет каждую строку и определяет, соответствует ли истине равенство Должность="Доцент" или неравенство Оклад > 4500. Если любое условие верно, то выражение внутри круглых скобок верно. Однако, если выражение внутри круглых скобок верно, предикат как единое целое неверен, потому что NOT преобразует верно в неверно и наоборот.

Вывод для этого запроса показывается в рис. 4.10.

Ид_Отд	Ид_Сотр.	Должность	Оклад	Фамилия
1	3	Ст.препод.	2000	Сидоров
1	5	Профессор	3700	Петухов
1	6	Инженер	1500	Сидорова
2	7	Ассистент	1700	Петрова

Рис. 4.10. Учет логических связок

Ниже приведен достаточно сложный пример для базовой таблицы РАБОТЫ (рис. 2.8). Попробуйте проследить его логику:

```
SELECT *
    FROM Работы
WHERE    NOT ((Период_с = “01.04.03” AND ИД_Вида >11)
OR           ИД_Сотр  =2);
```

4.3. Операторы IN, BETWEEN, LIKE в фразе WHERE

SQL использует специальные операторы IN, BETWEEN, LIKE, и IS NULL в дополнение к булевым операциям.

ОПЕРАТОР IN

Оператор IN определяет набор значений, в которое данное значение может или не может быть включено.

В нашей учебной базе, если Вы хотите найти всех сотрудников 1940 или 1980 года рождения необходимо использовать следующий запрос (вывод показывается на рис. 4.11):

```
SELECT *
FROM      Сотрудник
WHERE     Год_ рожд = 1940
OR        Год_ рожд = 1980;
```

Однако SQL предлагает более простой способ получить ту же информацию:

```
SELECT *
FROM    Сотрудник
WHERE   Год_ рожд  IN    (1940, 1980);
```

Ид_ Сотр	Фамилия	Имя	Отчество	ИНН	Год_ рожд.	Пол	Город	Район	Ин-декс	Ид_ Совм
5	Петухов	Виктор	Борисович	105	1940	М	Пушкин	Пушк	99481	1
12	Иванов	Иван	Иванович	712	1980	М	СПБ	Фрун	94555	
14	Яковлев	Иван	Васильевич	714	1980	М	СПБ	Фрун	94550	

Рис. 4.11. Поиск сотрудников определенного года рождения

Как Вы можете видеть, IN определяет набор значений с помощью имен членов набора, заключенных в круглые скобки и отделенных запятыми. Он затем проверяет различные значения указанного поля, пытаясь найти совпадение со значениями из набора. Если это случается, то предикат верен. Когда набор содержит значения номеров, а не символов, одиночные кавычки опускаются.

Поэтому следующий запрос Вам будет понятен без вывода результирующей таблицы (в ответе будет три строки):

```
SELECT *
FROM    Сотрудник
WHERE   Фамилия   IN    ("Иванов", "Петров");
```

Рассмотренная форма IN является просто краткой записью последовательности отдельных сравнений, соединенных операторами OR.

Еще один пример с использованием IN. Выдать ведомость оплаты за март месяц отдела с номером 1.

```
SELECT *
FROM    Ведомость_ оплаты
WHERE   Ид_Отд    IN ( 1 ) AND Период="Март";
```

В результате запроса будет получена таблица (рис. 4.12):

Ид_сотр	Ид_Отд	Период	Сумма	Вид оплаты
1	1	Март	1200	
2	1	Март	1200	
3	1	Март	1000	
1	1	Март	800	

Рис. 4.12. Ведомость за март

ОПЕРАТОР **BETWEEN**

Оператор BETWEEN похож на оператор IN. В отличие от определения по номерам из таблицы, как это делает IN, BETWEEN определяет диапазон, значения которого делают предикат верным. Для этого необходимо ввести ключевое слово BETWEEN с начальным значением, ключевое AND и конечное значение. В отличие от IN BETWEEN чувствителен к порядку, и первое значение в предложении должно быть первым по алфавитному или числовому порядку. Следующий пример будет извлекать из таблицы Сотрудник (рис. 2.1) всех сотрудников с годами рождения между 1964 и 1970 (вывод показывается в рис. 4.13):

```
SELECT    Ид_Сотр, Фамилия, Имя, Отчество, Год_ рожд
FROM      Сотрудник
WHERE     Год_ рожд  BETWEEN 1964 AND 1970;
```

Для включенного оператора BETWEEN значение, совпадающее с любым из двух значений границы (в этом случае, 1964 и .1970) заставляет предикат быть верным.

Ид_Сотр.	Фамилия	Имя	Отчество	Год рожд.
6	Сидорова	Екатерина	Ивановна	1970
7	Петрова	Нина	Николаевна	1970
8	Иванова	Вера	Васильевна	1970
10	Мухин	Александр	Михайлович	1964

Рис. 4.13. Диапазон на основе BETWEEN

Чтобы исключить граничные элементы выбираемого списка, можно определить граничные значения так, чтобы включающая интерпретация была приемлема, или сделать, например, так:

```
SELECT *
```

```
FROM    Сотрудник
WHERE   (Год_рожд BETWEEN 1964 AND 1975)
AND NOT Год_рожд IN (1964, 1975);
```

Вывод для этого запроса показывается в рис. 4.14.

Ид_Сотр.	Фамилия	Имя	Отчество	Год рожд.
6	Сидорова	Екатерина	Ивановна	1970
7	Петрова	Нина	Николаевна	1970
8	Иванова	Вера	Васильевна	1970

Рис. 4.14. BETWEEN – с IN

С помощью BETWEEN ... AND ... (находится в интервале от ... до ...) можно отобрать строки, в которых значение какого-либо столбца находится в заданном диапазоне.

Следующий пример иллюстрирует запрос по выбору сотрудников, находившихся в командировках (рис. 2.5) в диапазоне дат между 02.02.2003 и 10.03.2003.

```
SELECT *
FROM   Командировки
WHERE  Дата_отбытия BETWEEN '02.02.03' AND '10.03.03';
```

Ид _ Сотр	Дата_отбытия	Точка	Дата_прибытия
1	01.02.03.	Мурманск	02.02.03
2	01.02.03	Мурманск	02.02.03
3	01.03.03.	Петрозаводск	02.03.03
1	03.03.03.	Петрозаводск	04.03.03

Рис. 4.15. BETWEEN – по дате

Заметим, что вводимые даты, в зависимости от используемой СУБД, либо должны быть заключены в апострофы, либо нет.

ОПЕРАТОР **LIKE**

Оператор LIKE необходим, если Вам потребуется поиск в столбцах таблицы, которые имеют тип CHAR или VARCHAR, для нахождения какой-либо подстроки. Т.е.

он ищет в символьном поле, совпадает ли заданное условие в LIKE с частью символов, хранящихся в ячейке столбца.

Давайте найдем всех сотрудников, чьи имена начинаются с символа “А” (вывод показывается в рис. 4.16):

```
SELECT  *
FROM        Сотрудник
WHERE      Имя LIKE "А%";
```

Ид_Сотр.	Фамилия	Имя	Отчество	Год рожд.	Пол
4	Панов	Антон	Михайлович	1975	М
10	Мухин	Александр	Михайлович	1964	М
11	Попов	Анатолий	Михайлович	1947	М

Рис. 4.16. Использование LIKE "А%"

В качестве условия задаются символы поискового образа и добавляются специальные символы, которые могут соответствовать чему-нибудь (такая конструкция обычно называется маской поиска). Оператор LIKE принуждает систему осуществлять поиск любого совпадения с заданной Вами маской, просматривая и анализируя каждую строку указанного столбца.

В качестве таких специальных символов используются символы:

% - символ процента

_ - символ подчеркивания

[ ] - квадратные скобки

^ - каретка

Символ % указывает SQL ,что на место (позицию), в котором находится, % подходят любые символы из ячейки поискового столбца.

Следующий запрос к таблице ОТДЕЛ_СОТРУДНИК (рис. 2.3) с оператором LIKE создаст таблицу, содержащую информацию только о заведующих кафедрах:

```
SELECT  *
```

FROM Отдел_ Сотрудники

WHERE Должность LIKE "Зав%";

Если предположить, что в таблицу ОТДЕЛ_СОТРУДНИК (рис. 2.3) добавлена строка с информацией о заведующем лабораторией отдела:

Ид_Отд	Ид_Сотр.	Должность	Оклад	Дата_ приема	Дата_ увольнения
2	17	Лаб. Зав	1900	2002	

Рис. 4.17. Добавление новой информации

То следующий запрос:

SELECT *

FROM Отдел_ Сотрудники

WHERE Должность LIKE "%Зав%";

найдет все строки в таблице, которые в столбце Должность имеют сочетание символов "Зав"

Если запрос сформировать иначе:

SELECT *

FROM Отдел_ Сотрудники

WHERE Должность LIKE "Зав%";

то информация о заведующем лабораторией будет не найдена.

Использование _ **символа подчеркивания**

Каждое подчеркивание означает любой символ. Например, маска _a_ _ _ позволяет найти совпадение с любым значением столбца, которое состоит из пяти символов, второй из которых символ "a" (рис. 4.18).

SELECT *

FROM Сотрудник

WHERE Фамилия LIKE "_а_ _ _";

Ид_Сотр.	Фамилия	Имя	Отчество	Год_рожд.	Пол
4.	Панов	Антон	Михайлович	1975	М

Рис. 4.18. Использование LIKE "_а_ _ _"

Использование **[ ] - квадратных скобок** задает диапазон значений. Например, маска [Д-М]% позволяет получить строки, в которых фамилия сотрудника начинается с букв, расположенных в алфавите между Д и М (рис 4.19):

SELECT *

FROM Сотрудник

WHERE Фамилия LIKE "[Д-М]% ";

Ид_Сотр.	Фамилия	Имя	Отчество	Год_рожд.	Пол
1	Иванов	Иван	Петрович	1949	М
6	Иванова	Вера	Васильевна	1970	Ж
12	Иванов	Иван	Иванович	1980	М

Рис. 4.19. Использование LIKE "[Д-М]% "

Использование символа ^ - каретка

Задает диапазон значений, которые не должны содержаться в считываемых строках. Например, маска "^[Д-М]% " позволяет получить только те строки, в которых фамилия сотрудника начинается с любой буквы, кроме букв, находящихся между Д и М:

SELECT *

FROM Сотрудник

WHERE Фамилия LIKE "^[Д-М]% ";

Ид_Сотр.	Фамилия	Имя	Отчество	Год_рожд.	Пол
2	Петров	Иван	Иванович	1949	М
3	Сидоров	Петр	Петрович	1947	М
4	Панов	Антон	Михайлович	1975	М
5	Петухов	Виктор	Борисович	1940	М
7	Петрова	Нина	Николаевна	1970	Ж
8	Сидорова	Екатерина	Ивановна	1970	Ж
9	Никитин	Валентин	Сергеевич	1952	М
11	Попов	Анатолий	Михайлович	1947	М

Рис. 4.20. Использование LIKE "^[Д-М]% "

РЕЗЮМЕ

Теперь Вы можете создавать предикаты в терминах связей специально определенных SQL. Вы можете искать значения в определенном диапазоне (BETWEEN) или в числовом наборе (IN), или Вы можете искать символьные значения, которые соответствуют тексту внутри параметров (LIKE).

4.4. GROUP BY и агрегатные функции SQL

Результатом запроса может быть обобщенное групповое значение полей, точно так же, как и значение одного поля. Это делается с помощью стандартных агрегатных функций SQL, список которых приведен ниже:

COUNT - число значений в столбце,

SUM - арифметическая сумма значений в столбце,

AVG - арифметическое среднее значение в столбце,

MAX - самое большое значение в столбце,

MIN - самое маленькое значение в столбце.

Кроме специального случая COUNT(*), каждая из этих функций оперирует совокупностью значений столбца некоторой таблицы и выдает только одно значение.

Аргументу всех функций, кроме COUNT(*), может предшествовать ключевое слово DISTINCT (различный), указывающее, что дублирующие значения столбца

должны быть исключены перед тем, как будет применяться функция. Специальная же функция COUNT(*) служит для подсчета всех без исключения строк в таблице (включая дубликаты).

Агрегатные функции используются подобно именам полей в предложении запроса SELECT, но с одним исключением: они берут имена поля как аргументы. Только числовые поля могут использоваться с SUM и AVG.

С COUNT, MAX, и MIN могут использоваться и числовые или символьные поля. Когда они используются с символьными полями, MAX и MIN будут транслировать их в эквивалент ASCII кода, который должен сообщать, что MIN будет означать первое, а MAX - последнее значение в алфавитном порядке.

Чтобы найти SUM всех окладов в таблице ОТДЕЛ_СОТРУДНИК (рис. 2.3) надо ввести следующий запрос:

SELECT SUM ((Оклад)) AS СУММА

FROM Отдел_ Сотрудники;

И на экране увидим результат: 46800 (в таблице будет один столбец с именем СУММА).

Подсчет среднего значения по окладам также прост:

SELECT AVG ((Оклад))

FROM Отдел_ Сотрудники;

И на экране увидим результат: 3342.85

Функция COUNT несколько отличается от всех. Она считает число значений в данном столбце или число строк в таблице. Когда она считает значения столбца, она используется с DISTINCT (различных) чтобы производить счет чисел уникальных значений в данном поле.

SELECT COUNT (DISTINCT ОКЛАД)
FROM Отдел_ Сотрудники;

Результат: 8.

В таблице восемь строк, в которых находятся различные значения окладов.

Отметим, что в последних трех примерах учитывается информация и об уволенных сотрудниках.

Следующее предложение позволяет определить число подразделений на предприятии:

SELECT COUNT (DISTINCT Ид_Отд)

FROM Отдел_Сотрудники;

Результат: 3.

DISTINCT, сопровождаемый именем поля, с которым он применяется, помещенный в круглые скобки, с COUNT применяется к индивидуальным столбцам.

Чтобы подсчитать общее число строк в таблице, используется COUNT со звездочкой вместо имени поля:

SELECT COUNT (*)
FROM Отдел_ Сотрудники;
Ответ будет: 11.

COUNT (*) подсчитывает все без исключения строки таблицы.

DISTINCT **не применим** с COUNT (*).

Предположим, что таблица ВЕДОМОСТЬ_ОПЛАТЫ (рис. 2.4) имеет еще один столбец, который хранит сумму произведенных вычетов (поле Вычет) для каждой строки ведомости. Тогда если Вас интересует вся сумма, то содержимое столбца *Сумма* и *Вычет* надо сложить.

Если же Вас интересует максимальная сумма с учетом вычетов, содержащаяся в ведомости, то это можно сделать с помощью следующего предложения:

SELECT MAX (Сумма + Вычет)

FROM Ведомость_ оплаты;

Для каждой строки таблицы этот запрос будет складывать значение поля *Сумма* со значением поля *Вычет* и выбирать самое большое значение, которое он найдет.

ПРЕДЛОЖЕНИЕ **GROUP BY** (перекомпоновка, порядок)

Предложение GROUP BY позволяет определять подмножество значений в особом поле в терминах другого поля и применять функцию агрегата к подмножеству. Это дает возможность объединять поля и агрегатные функции в едином предложении SELECT.

Например, предположим, что Вы хотите определить, сколько сотрудников находятся в каждом отделе (ответ приведен на рис. 4.21):

```
SELECT      Ид_Отд, COUNT (DISTINCT Ид_Отд) AS Кол-во_сотрудников
FROM        Отдел_ Сотрудники
WHERE       Дата_ увольнения        NOT NULL
GROUP BY        Ид_Отд;
```

Ид_Отд	Кол-во_сотрудников
1	5
2	4
3	1

Рис. 4.21. Запрос с группировкой

В результате выполнения предложения GROUP BY остаются только уникальные значения столбцов, *по умолчанию* отсортированные *по возрастанию*. В этом аспекте предложение GROUP BY отличается от предложения ORDER BY тем, что последнее хотя и сортирует записи по возрастанию, но не удаляет повторяющиеся значения. В приведенном примере запрос группирует строки таблицы по значениям столбца *Ид_Отд* (по номерам отделов). Строки с одинаковыми номерами отделов сначала объединяются в группы, но при этом для каждой группы отображается только одна строка. Во втором столбце выводится количество строк в каждой группе, т.е. число сотрудников в каждом отделе.

Значение поля, к которому применяется GROUP BY, имеет, по определению, только одно значение на группу вывода так же, как это делает агрегатная функция. Результатом является совместимость, которая позволяет агрегатам и полям объединяться таким образом.

Пусть, например, таблица ВЕДОМОСТЬ_ОПЛАТЫ имеет вид рис. 4.22 и мы хотим определить максимальную сумму, выплаченную по ведомости каждому сотруднику.

Ид_сотр	Ид_Отд	Дата	Сумма	Вид_оплаты
1	1	02.03.03	1200	
2	1	01.03.03	1200	
3	1	01.03.03	1000	
1	1	01.04.03	1200	
1	1	01.03.03	800	
2	1	02.04.03	2000	

Рис. 4.22. Состояние таблицы Ведомость1

```
SELECT     ИД_Сотр, MAX (Сумма)  AS  "Сумма максимальная"
FROM       Ведомость1
GROUP      BY     ИД_Сотр;
```

В результате получим.

Ид_сотр	Сумма_максимальная
3	1000
1	1200
2	2000

Рис. 4.23. Агрегатная функция с AS

Группировка может быть осуществлена и по нескольким атрибутам:

```
SELECT Ид_сотр, Дата, MAX ((Сумма))
FROM Ведомость1
GROUP BY Ид_сотр, Дата;
```

Результат:

Ид_сотр	Дата	
1	01.03.03	800
1	02.03.03	1200
1	01.04.03	1200
2	01.03.03	1200
2	02.04.03	2000
2	01.03.03	1000

Рис. 4.24. Группировка по нескольким атрибутам

Если же возникает необходимость ограничить число групп, полученных после GROUP BY, то, используя предложение HAVING, можно это реализовать.

4.5. Использование фразы HAVING

Фраза HAVING играет такую же роль для групп, что и фраза WHERE для строк: она используется для исключения групп, точно так же, как WHERE используется для исключения строк. Эта фраза включается в предложение

лишь при наличии фразы GROUP BY, а выражение в HAVING должно принимать единственное значение для группы.

Например, пусть надо выдать количественный состав всех отделов (рис. 2.3), исключая отдел с номером 3.

```
SELECT    Ид_Отд, COUNT (DISTINCT Ид_Отд)    AS    Кол-во _сотрудников
    FROM      Отдел_ Сотрудники
    WHERE     Дата_ увольнения        NOT NULL
    GROUP BY      Ид_Отд     HAVING    Ид_Отд < > 3;
```

Ид_Отд	Кол_во_сотрудников
1	5
2	4

Рис. 4.25. Запрос с группировкой и ограничением

Последним элементом при вычислении табличного выражения используется раздел **HAVING** (если он присутствует). Синтаксис этого раздела следующий:

<having clause> ::=

HAVING <search condition>

Условие поиска этого раздела задает условие на группу строк сгруппированной таблицы. Формально раздел HAVING может присутствовать и в табличном выражении, не содержащем GROUP BY. В этом случае полагается, что результат вычисления предыдущих разделов представляет собой сгруппированную таблицу, состоящую из одной группы без выделенных столбцов группирования.

Условие поиска раздела HAVING строится по тем же синтаксическим правилам, что и условие поиска раздела WHERE, и может включать те же самые предикаты.

Однако имеются специальные синтаксические ограничения по части использования в условии поиска спецификаций столбцов таблиц из раздела FROM данного табличного выражения. Эти ограничения следуют из того, что условие поиска раздела HAVING задает условие на целую группу, а не на индивидуальные строки.

Поэтому в арифметических выражениях предикатов, входящих в условие выборки раздела HAVING, прямо можно использовать только спецификации столбцов, указанных в качестве столбцов группирования в разделе GROUP BY. Остальные столбцы можно специфицировать только внутри спецификаций агрегатных функций COUNT, SUM, AVG, MIN и MAX, вычисляющих в данном случае некоторое агрегатное значение для всей группы строк. Аналогично обстоит дело с подзапросами, входящими в предикаты условия выборки раздела HAVING: если в подзапросе используется характеристика текущей группы, то она может задаваться только путем ссылки на столбцы группирования.

Пусть запрос вида (в качестве базовой таблицы см. рис. 4.22):

SELECT Ид_сотр, Дата, MAX ((Сумма))

FROM Ведомость1

GROUP BY Ид_сотр, Дата;

необходимо уточнить тем, чтобы были показаны только выплаты, превышающие 1000.

Однако по стандарту агрегатную функцию в предложении WHERE использовать запрещено (если вы не используете подзапрос, описанный позже), потому что предикаты оцениваются в терминах одиночной строки, а агрегатные функции оцениваются в терминах группы строк.

Следующее предложение будет неправильным:

SELECT Ид_сотр, Дата, MAX (Сумма)

FROM Ведомость1

WHERE MAX ((Сумма)) > 1000 GROUP BY Ид_сотр, Дата;

Правильным предложением будет:

SELECT Ид_сотр, Дата, MAX ((Сумма))

FROM Ведомость1

GROUP BY Ид_сотр, Дата

HAVING MAX ((Сумма)) > 1000;

Вывод для этого запроса представлен на рис. 4.26.

Ид_сотр	Дата	
1	02.03.03	1200
1	01.04.03	1200
2	01.03.03	1200
2	02.04.03	2000

Рис. 4.26. Удаление групп агрегатных значений

4.6. Упорядочение вывода по номеру столбца

Ранее уже были приведены некоторые способы упорядоченного вывода содержимого таблицы. Сейчас мы рассмотрим способ сортировки по номеру столбца.

Если вместо имен столбцов использовать их порядковые номера для указания полей вывода, то мы получаем еще один способ сортировки. Эти номера могут ссылаться не на порядок столбцов в таблице, а на их порядок в выводе. Другими словами, поле, упомянутое в предложении SELECT первым, для ORDER BY - это поле 1, независимо от того каким по порядку оно стоит в таблице. Например, Вы можете использовать следующую команду, чтобы увидеть определенные поля таблицы ВЕДОМОСТЬ1 (рис. 4.22), упорядоченными в порядке убывания к наименьшему значению суммы (вывод показан на рис. 4.27):

SELECT Дата, Сумма

FROM Ведомость1

GROUP BY 2 DESC;

Дата	Сумма
02.04.03	2000
02.03.03	1200
01.03.03	1200
01.04.03	1200
01.03.03	1000
01.03.03	800

Рис. 4.27. Сортировка, использующая номера столбцов

Одна из целей этой возможности ORDER BY - дать возможность использовать GROUP BY со столбцами вывода так же, как и со столбцами таблицы. Столбцы, производимые агрегатной функцией, константы, или выражения в предложении SELECT запроса, пригодны и для использования с GROUP BY, если они ссылаются к ним с помощью номера. Например, давайте подсчитаем, сколько раз совместители замещали основного преподавателя (рис. 2.7), и выведем результаты в убывающем порядке совмещений, как показано в рис. 4.28.

```
SELECT Ид_Совм, COUNT ( DISTINCT № п/п )

FROM Замещение

GROUP BY Ид_Совм

ORDER BY 2 DESC;
```

Ид_Совм	
1	3
2	3
7	2
4	1
3	1

Рис. 4.28. Упорядочение с помощью столбца вывода

В этом случае необходимо использовать номер столбца, так как столбец вывода не имеет имени; и Вы не должны использовать саму агрегатную функцию.

РЕЗЮМЕ

Теперь Вы знаете несколько способов заставить систему давать ту информацию, которая необходима, а не только все ее содержание. Можно переупорядочивать столбцы таблицы или устранять любой из них. Вы можете решать, хотите Вы видеть дублированные значения или нет.

Наиболее важно то, что можно ставить условие, называемое предикатом, которое определяет или не определяет указанную строку таблицы для вывода.

Предикаты могут быть очень сложными, что делает запросы SQL мощными. Следующие разделы будут посвящены, в большей мере, особенностям, которые

расширяют мощность предикатов. Также будут представлены операторы иные, чем те, которые используются в условиях предиката, а также способы объединения многочисленных условий в единый предикат.

О предложении **SELECT** нельзя все рассказать сразу. Для того чтобы можно было более или менее точно рассказать про структуру запросов в SQL, необходимо привести сводку синтаксических правил предложения SELECT.

Предложение SELECT (выбрать) имеет следующий формат [23]:

подзапрос [UNION [ALL] подзапрос]

[ORDER BY {[таблица.]столбец | номер_ элемента_ SELECT} [[ASC] | DESC]

[,{[таблица.]столбец | номер_ элемента_ SELECT} [[ASC] | DESC]] ...;

и позволяет объединить (UNION), а затем упорядочить (ORDER BY) результаты выбора данных, полученных с помощью нескольких "подзапросов". При этом упорядочение можно производить в порядке возрастания - ASC (ASCending) или убывания DESC (DESCending), по умолчанию принимается ASC.

В этом предложении подзапрос позволяет указать условия для выбора нужных данных и (если требуется) их обработки.

SELECT (выбрать) данные из указанных столбцов и (если необходимо) выполнить перед выводом их преобразование в соответствии с указанными выражениями и (или) функциями,

FROM (из) перечисленных таблиц, в которых расположены эти столбцы,

WHERE (где) строки из указанных таблиц должны удовлетворять указанному перечню условий отбора строк,

GROUP BY (группируя по) указанному перечню столбцов с тем, чтобы получить для каждой группы единственное агрегированное значение, используя во фразе SELECT SQL-функции SUM (сумма), COUNT (количество),

MIN (минимальное значение), MAX (максимальное значение) или AVG (среднее значение),

HAVING (имея) в результате лишь те группы, которые удовлетворяют указанному перечню условий отбора групп.

Предложение имеет формат:

SELECT [[ALL] | DISTINCT]{ * | элемент_SELECT [,элемент_SELECT] ...}]

FROM {базовая_ таблица | представление} [псевдоним]

[,{базовая_ таблица | представление} [псевдоним]] ...

[WHERE фраза]

[GROUP BY фраза [HAVING фраза]];

Элемент_ SELECT - это одна из следующих конструкций:

[таблица.]* | значение | SQL_ функция | системная_ переменная,

где значение – это:

[таблица.]столбец | (выражение) | константа | переменная

Синтаксис выражений имеет вид:

({[[+] | -] {значение | функция_ СУБД} [+ | - | * | **]}...)

Синтаксис SQL_ функций – одна из следующих конструкций:

{SUM|AVG|MIN|MAX|COUNT} ([[ALL]|DISTINCT][таблица.]столбец

{SUM|AVG|MIN|MAX|COUNT} ([ALL] выражение)

COUNT(*)

Фраза WHERE включает набор условий для отбора строк:

WHERE [NOT] WHERE_условие [[AND|OR][NOT] WHERE_условие]...

где WHERE_ условие – одна из следующих конструкций:

значение { = | <> | < | <= | > | >= } { значение | (подзапрос) };

значение_1 [NOT] BETWEEN значение_2 AND значение_3;

значение [NOT] IN { (константа [,константа]...) | (подзапрос) };

значение IS [NOT] NULL.

[таблица.]столбец [NOT] LIKE "строка_ символов" [ESCAPE "символ"]

EXISTS (подзапрос)

Кроме операторов сравнения (= | <> | < | <= | > | >=) в WHERE используются условия BETWEEN (между), LIKE (похоже на), IN (принадлежит), IS NULL (не определено) и EXISTS (существует), которым могут предшествовать оператор NOT (не). Критерий отбора строк формируется из одного или нескольких условий, соединенных логическими операторами.

При обработке условия числа сравниваются алгебраически - отрицательные числа считаются меньшими, чем положительные, независимо от их абсолютной величины. Строки символов сравниваются в соответствии с их представлением в коде, используемом в конкретной СУБД, например, в коде ASCII. Если сравниваются две строки символов, имеющих разные длины, более короткая строка дополняется справа пробелами для того, чтобы они имели одинаковую длину перед осуществлением сравнения.

Синтаксис фразы GROUP BY имеет вид

GROUP BY [таблица.]столбец [,[таблица.]столбец] ... [HAVING фраза]

GROUP BY производит перекомпоновку формируемой таблицы по группам, каждая из которых имеет одинаковое значение в столбцах, включенных в перечень GROUP BY. Далее к этим группам применяются агрегирующие функции, указанные во фразе SELECT, что приводит к замене всех значений группы на единственное значение (сумма, количество и т.п.).

Синтаксис фразы HAVING:

HAVING [NOT] HAVING_условие [[AND|OR][NOT] HAVING_условие]...

С помощью ее можно исключить из результата группы, не удовлетворяющие заданным условиям:

значение { = | <> | < | <= | > | >= } { значение | (подзапрос)
| SQL_ функция }

{значение_1 | SQL_функция_1} [NOT] BETWEEN
{значение_2 | SQL_функция_2} AND {значение_3 | SQL_функция_3}
{значение | SQL_ функция} [NOT] IN { (константа [,константа]...)
| (подзапрос) }
{значение | SQL_функция} IS [NOT] NULL
[таблица.]столбец [NOT] LIKE "строка_ символов" [ESCAPE "символ"]
EXISTS (подзапрос)

В дальнейшем будем предполагать, что все операторы вводятся интерактивно, а не как часть прикладной программы (хотя отличий не так уж много).

В общем случае шаги обработки SQL – операторов следующие:

- создание курсора,
- разбор оператора,
- связывание переменных,
- описание результатов,
- определение выхода,
- выполнение.

Глава 5. Объединение таблиц

5.1. Выполнение реляционных объединений

В данном учебном пособии не рассматриваются вопросы, связанные с проектированием моделей баз данных, а эти вопросы по существу являются центральными в проектировании баз данных [13, 27]. Именно правильно построенная модель, а точнее, методология, используемая при построении концептуальной модели, обеспечивает жизнеспособность проекта [4, 17, 18, 30].

На данном этапе можно просто принять к сведению, что базы данных - **это множество взаимосвязанных таблиц** в терминологии традиционных реляционных СУБД. При проектировании стремятся создавать таблицы, в каждой из которых содержалась бы информация об одном и только об одном типе сущностей. Это облегчает модификацию базы данных и поддержание ее целостности.

В предыдущих примерах Вы уже использовали SQL для работы с одной таблицей. В SQL также реализованы возможности "соединять" или "объединять" несколько таблиц и так называемые "вложенные подзапросы".

5.1.1. Естественное соединение таблиц (natural join)

Например, чтобы получить перечень фамилий преподавателей кафедры КТиПО, которые были в командировке в Мурманске (базовые таблицы рис. 2.3, рис. 2.1, рис. 2.2, рис. 2.5), возможен запрос:

```
SELECT Фамилия, Название_ отдела, Точка
FROM   Отдел_ Сотрудник, Сотрудник, Отдел, Командировки
WHERE  Сотрудник.Ид_Сотр=Отдел_ Сотрудник.Ид_Сотр     AND
       Отдел_ Сотрудник.Ид_Отд=Отдел.Ид_Отд           AND
       Отдел_ Сотрудник.Ид_Сотр=Командировки.Ид_Сотр        AND
       Отдел.Название_ отдела='Кафедра КТиПО'         AND
       Командировки.Точка="Мурманск"                  AND
       Командировки.Дата_ убытия  IS NOT NULL;
```

Результатом запроса будет отношение (см. рис. 5.1).

Фамилия	Название_ отдела	Точка
Иванов	КТиПО	Мурманск
Петухов	КТиПО	Мурманск

Рис. 5.1. Сотрудники, бывшие в командировке в Мурманске

Ответ получен следующим образом: СУБД последовательно формирует строки декартова произведения таблиц, перечисленных во фразе FROM, проверяет, удовлетворяют ли данные сформированной строки условиям фразы WHERE, и если удовлетворяют, то включает в ответ на запрос те ее поля, которые перечислены во фразе SELECT.

Следует подчеркнуть, что в SELECT и WHERE ссылки на все атрибуты (отдельные столбцы таблицы) рекомендуется уточнять именем соответствующей таблицы. Например, Командировки.Точка, Сотрудник.Город, Сотрудник.* и т.п.

Такой вид соединения называют обычно **естественным** соединением, так как таблицы объединяются только на основе первичных и внешних ключей, с указанием во фразе SELECT определенного количества атрибутов, подлежащих выводу в результирующей таблице.

Отметим одну важную особенность описания запросов, которую используют программисты. Вместо громоздких имен таблиц используются псевдонимы (alias). Так, для предыдущего запроса запись в алиасной форме будет следующей:

```
SELECT P.Фамилия, B.Название_отдела, K.Точка
FROM   Отдел_Сотрудник  S, Сотрудник P,Отдел B, Командировки  K
WHERE          P.Ид_Сотр=S.Ид_Сотр
AND    S.Ид_Отд=B.Ид_Отд   AND    S.Ид_Сотр=K.Ид_Сотр
AND    B.Название_ отдела="Кафедра КТиПО"
AND    K.Точка="Мурманск"
AND    K.Убытия        IS NOT NULL;
```

Некоторые системы требуют указывать ключевое слово AS перед определением псевдонимов. Псевдоним как бы заменяет имя таблицы во фразе FROM, а затем этим новым именем пользуются для обозначения имен таблиц. Общее правило создания псевдонимов для всех СУБД очень простое. После набора имени таблицы поставьте пробел и затем набирайте псевдоним для таблицы. Отметим, что "время жизни" псевдонима определено только внутри запроса.

Очевидно, что с помощью соединения несложно сформировать запрос на обработку данных из нескольких таблиц. Кроме того, в такой запрос можно включить любые части предложения SELECT, рассмотренные в главе 4 (выражения с использованием функций, группирование с отбором указанных групп и упорядочением полученного результата). Следовательно, соединения позволяют обрабатывать множество взаимосвязанных таблиц как единую таблицу. Фактически если схема модели верна, то весь фрагмент предметной области, описанный в базе данных, должен восприниматься как единое отношение. Это постулат реляционного подхода, и какие проблемы теоретического и практического плана [1, 2, 6, 13, 14, 16, 18, 26, 29, 30, 31] при этом возникают выходят за рамки данного пособия.

Кроме механизма соединений в SQL есть механизм вложенных подзапросов, позволяющий объединить несколько простых запросов в едином предложении SELECT. Иными словами, вложенный подзапрос - это уже знакомый нам подзапрос (с небольшими ограничениями), который вложен в WHERE фразу другого вложенного подзапроса или WHERE фразу основного запроса.

Для иллюстрации вложенного подзапроса, вернемся к предыдущему примеру и попробуем получить перечень сотрудников, которые были в Мурманске, и какова длительность этих командировок.

Результат запроса имеет вид

```
SELECT      Фамилия, Название_ отдела, Точка
FROM  Отдел_ Сотрудник, Сотрудник, Отдел, Командировки
WHERE       Сотрудник.Ид_Сотр=Отдел_Сотрудник.Ид_Сотр      AND
            Отдел_ Сотрудник.Ид_Отд=Отдел.Ид_Отд           AND
            Отдел_ Сотрудник.Ид_Сотр=Командировки.Ид_Сотр          AND
            Отдел.Название_ отдела="Кафедра КТиПО"         AND
            Командировки.Точка="Мурманск"                  AND
            Командировки.Дата_убытия  IS NOT NULL          AND
            (SELECT     Командировки.Дата_прибытия -
            Командировки.Дата_ убытия) AS Кол-во_ дней
            FROM        Командировки        X
            WHERE       X.Ид_Сотр=Отдел_Сотрудник.Ид_Сотр);
```

Результатом запроса будет отношение (рис. 5.2):

Фамилия	Название_ отдела	Точка	Кол-во_ дней
Иванов	КТиПО	Мурманск	8
Петухов	КТиПО	Мурманск	10

Рис. 5.2. Длительность командировки в Мурманске

Здесь с помощью подзапроса, размещенного в четырех последних строках запроса, описывается процесс определения длительности командировки для каждого, приезжавшего в Мурманск и закончившего там работу. Механизм реализации подзапросов будет подробно описан в параграфе 5.3. Там же будет рассмотрено, как и для чего вводится псевдоним **X** для имени таблицы КОМАНДИРОВКИ (рис. 2.5).

Для закрепления материала по данному разделу сформулируем следующий запрос:

```
SELECT      P.Фамилия, B.Название_ отдела, K.Точка
FROM Отдел_Сотрудник S , Сотрудник P , Отдел B , Командировки K
    WHERE P.Ид_Сотр=S.Ид_Сотр
          AND   S.Ид_Отд=B.Ид_Отд  AND   S.Ид_Сотр=K.Ид_Сотр
          AND   B.Название_ отдела="Кафедра КТиПО"
          AND   K.Точка="Мурманск"  AND K.Дата_убытия IS NOT NULL          AN
```

На обычном языке этот запрос уточняет: "Кто из сотрудников указанной кафедры был в Мурманске после февраля" (рис. 5.3).

Фамилия	Название_отдела	Точка
Петухов	КТиПО	Мурманск

Рис. 5.3. Уточняющий запрос по Мурманску

5.1.2. Эквисоединение таблиц

Объединения, которые используют предикаты, основанные на равенствах, называются объединениями по равенству (эквисоединениями). Практически все наши примеры, использующиеся ранее, относились именно к этой категории, потому что все условия в предложениях WHERE базировались на математических выражениях, использующих знак равно (=). Строки вида

```
Отдел_ Сотрудник.Ид_Отд=Отдел.Ид_Отд
Отдел_ Сотрудник.Ид_Сотр=Командировки.Ид_Сотр
Отдел.Название_ отдела="Кафедра КТиПО"
Командировки.Точка="Мурманск"
```

примеры таких типов равенств. Объединения по равенству - это вероятно наиболее общий вид объединения, но имеются и другие (рассмотрены далее). Вы можете, фактически, использовать любой из реляционных операторов в объединении.

В качестве простейшего примера операции эквисоединения таблиц рассмотрим следующий запрос.

Выдать все сведения о начальниках отделов (результат см. рис. 5.4), используя информацию, указанную на рис. 2.2 и 2.1

```
SELECT   Отдел.*, Сотрудник.*
FROM     Отдел, Сотрудник
WHERE    Отдел.Ид_начальника=Сотрудник.Ид_Сотрудника
```

Ид_ Сотр	Фамилия	Имя	Отчество	ИНН	Год_ рожд.	Пол	Город	Район	Ин-декс	Ид_ Совм
8	Сидрова	Ада	Ивановна	308	1970	Ж	СПБ	Фрун	94018	7
9	Никитин	Виктор	Сергеевич	489	1952	М	СПБ	Центр	94540	4
10	Мухин	Степан	Михайлович	510	1964	М	СПБ	Моск	95421	
13	Хохлов	Иван	Васильевич	713	1960	М	СПБ	Фрун	90852	
14	Яковлев	Иван	Васильевич	714	1980	М	СПБ	Фрун	90853	

Ид_ Отд	Название_ отдела	Ид_ Начальника	Вид_ отдела	Год_ основания
1	Кафедра КТиПО	8	Выпускающая	1960
2	Кафедра Механики	9	Нет	1930
3	Кафедра Физики	10	Нет	1930
4	Кафедра Математики	13	Выпускающая	1945
5	Кафедра Автоматики	14	Нет	1945

Рис. 5.4. Эквисоединение

Отметим, что таблицы, представленные на рис. 5.4, на самом деле представляют одну таблицу (в ней 16 столбцов и пять строк). Естественно, что в зависимости от разрешающей возможности СУБД не все эти столбцы будут представлены на экране. Что-то может оказаться за границей окна, куда выводятся данные, но, используя соответствующие кнопки скроллинга, постепенно можно добраться до правой последней колонки результирующей таблицы. В большинстве СУБД число столбцов в результирующей таблице не может превышать числа 254.

5.1.3. Декартово произведение таблиц

В самом начале пособия показано (см. пункт 1.2.2), что соединения - это подмножества декартова произведения. Так как декартово произведение **n** таблиц - это таблица, содержащая все возможные строки **r**, такие, что **r** является сцеплением какой-

либо строки из первой таблицы со строками из второй таблицы, и т.д. - со строками из **n**-й таблицы, то можно показать, как с помощью SELECT формируется декартово произведение. Для получения декартова произведения нескольких таблиц, надо указать во фразе FROM перечень перемножаемых таблиц, а во фразе SELECT – все их столбцы.

Так, чтобы получить декартово произведение Отдел (рис. 2.2) и Командировки (рис. 2.5) надо выдать запрос:

```
SELECT Отдел.*, Командировки.*
FROM   Отдел, Командировки;
```

В результате получим таблицу, содержащую 5 x 5 = 25 строк (рис. 5.5):

Ид_ Отд	Название_ Отдела	Ид_ Нач	Вид_ отдела	Год_ осн.	Ид_ Сотр	Дата_ отбытия	Точка	Дата_ убытия
1	Каф. КТиПО	8	Выпуск	1930	1	01.02.03.	Мурманск	02.02.03
1	Каф. КТиПО	8	Выпуск	1930	2	01.02.03	Мурманск	
1	Каф. КТиПО	8	Выпуск	1930	3	01.03.03.	Петрозаводск	11.03.03
1	Каф. КТиПО	8	Выпуск	1930	1	03.03.03.	Петрозаводск	
1	Каф. КТиПО	8	Выпуск	1930	5	01.04.03	Мурманск	12.04.03
2	Каф. механики	9	Нет	1930	1	01.02.03.	Мурманск	02.02.03
2	Каф. механики	9	Нет	1930	2	01.02.03	Мурманск	
2	Каф. механики	9	Нет	1930	3	01.03.03.	Петрозаводск	11.03.03
2	Каф. механики	9	Нет	1930	1	03.03.03.	Петрозаводск	
2	Каф. механики	9	Нет	1930	5	01.04.03	Мурманск	12.04.03
3	Каф. физики	10	Нет	1930	1	01.02.03.	Мурманск	02.02.03
3	Каф. физики	10	Нет	1930	2	01.02.03	Мурманск	
3	Каф. физики	10	Нет	1930	3	01.03.03.	Петрозаводск	11.03.03
3	Каф. физики	10	Нет	1930	1	03.03.03.	Петрозаводск	
3	Каф. физики	10	Нет	1930	5	01.04.03	Мурманск	12.04.03
4	Каф. математики	13	Выпуск	1945	1	01.02.03.	Мурманск	02.02.03
4	Каф. математики	13	Выпуск	1945	2	01.02.03	Мурманск	
4	Каф. математики	13	Выпуск	1945	3	01.03.03.	Петрозаводск	11.03.03
4	Каф. математики	13	Выпуск	1945	1	03.03.03.	Петрозаводск	
4	Каф. математики	13	Выпуск	1945	5	01.04.03	Мурманск	12.04.03
5	Каф. автоматики	14	Нет	1945	1	01.02.03.	Мурманск	02.02.03
5	Каф. автоматики	14	Нет	1945	2	01.02.03	Мурманск	
5	Каф. автоматики	14	Нет	1945	3	01.03.03.	Петрозаводск	11.03.03
5	Каф. автоматики	14	Нет	1945	1	03.03.03.	Петрозаводск	
5	Каф. автоматики	14	Нет	1945	5	01.04.03	Мурманск	12.04.03

Рис. 5.5. Результат декартова произведения двух таблиц

Важно отметить, что каждая строка одной таблицы соединилась с каждой строкой другой таблицы. Семантический смысл этой таблицы, да в принципе и любой строки полученного результата, достаточно трудно объяснить.

В другом примере, где перемножаются таблицы ОТДЕЛ_СОТРУДНИК и СОТРУДНИК (рис. 2.2 и 2.1 соответственно):

SELECT Отдел_Сотрудники.*, Сотрудники.*

FROM Отдел_Сотрудники, Сотрудники;

получается таблица, содержащая 13 * 14 = 182 строки. Из них только 14 строк соответствуют действительному положению дел, остальные – "информационный шум" операции декартова произведения.

5.1.4. Соединение с дополнительным условием

Стандарт SQL позволяет использовать любой из реляционных операторов в объединении. Здесь показан пример объединения с использованием во фразе WHERE предиката "< >".

Пусть, например, необходимо найти фамилии штатных преподавателей, которые живут в других городах, чем совместители, и указать даты, когда совместители их замещали при проведении занятий (вывод показывается на рис. 5.6).

Дата_ замены	Фамилия	Ид_ Сотр	Ид_ Совм
10.02.03	Иванова	8	7
10.02.03	Петрова	7	4
10.02.03	Сидоров	3	2
10.03.03	Иванова	8	7
10.04.03	Панов	4	2
20.05.03	Панов	4	2

Рис. 5.6. Select с дополнительным условием

Сам запрос выглядит следующим образом:

SELECT Дата_замены, Сотудник.Фамилия, Замещение.Ид_Сотр, Замещение.Ид_Совм

FROM Совместители, Сотрудник, Замещение

WHERE Сотрудник.Город < > Совместитель.Город

AND Замещение.Ид_Сотр = Сотрудник.Ид_Совм;

Каким же образом реализуется данный запрос ?

Во-первых, после выполнения инструкции FROM мы получаем **декартово произведение** по трем отношениям (таблицам Совместители, Сотрудник и Замещение).

Далее, после выполнения инструкции WHERE, мы получаем **выборку** из полученного произведения. Для этого в каждой строке декартова произведения (это наша виртуальная таблица) проверяется в соответствующих столбцах условие, чтобы значение из ячейки (поля) рабочей таблицы Замещения.Ид_Сотр равнялось значению ячейки Сотрудник.Ид_Совм. При этом же в этой строке проверяется, чтобы значение ячейки Сотрудник.Город не равнялось значению ячейки Совместитель.Город. Строка, удовлетворяющая указанному условию, заносится в рабочую таблицу, которую мы и увидим на экране.

Замечание. Ячейка - это элемент двухмерного массива (если таблицу представить **в** виде массива).

Наконец, после выполнения оператора SELECT мы получаем проекцию выборки по столбцам, указанным в инструкции SELECT.

Следовательно, не строго говоря, инструкция FROM в SQL соответствует декартову произведению, инструкция WHERE соответствует выборке, а совместная инструкция SELECT – FROM - WHERE выражает собой проекцию выборки произведения.

Теперь, когда алгоритм работы любой инструкции типа **SELECT – FROM WHERE** Вам ясен, не трудно будет расширить предыдущий запрос так, чтобы на экран выдавалась и фамилия *Совместителя*. Это необходимо для читаемости результирующей таблицы.

Измененная конструкция запроса будет:

SELECT Дата_замены, Сотрудник.Фамилия, Замещение.Ид_Сотр,

Совместитель.Фамилия, Замещение.Ид_Совм

FROM Совместители, Сотрудник, Замещение

WHERE Сотрудник.Город < > Совместитель.Город

AND Замещение.Ид_Сотр = Сотрудник.Ид_Совм;

А ответ нахлдится на рисунке 5.7.

Дата_замены	Фамилия	Ид_Сотр	Фамилия	Ид_Совм
10.02.03	Иванова	8	Русов	7
10.02.03	Петрова	7	Михайлов	4
10.02.03	Сидоров	3	Смирнов	2
10.03.03	Иванова	8	Русов	7
10.04.03	Панов	4	Смирнов	2
20.05.03	Панов	4	Смирнов	2

Рис. 5.7 Расширенное соединение

Для однозначного прочтения таблицы можно воспользоваться следующим приемом:

```
SELECT    Дата_замены, Сотудник.Фамилия, Замещение.Ид_Сотр,
          "Заменял" Совместитель.Фамилия, Замещение.Ид_Совм
FROM      Совместители, Сотрудник, Замещение
WHERE     Сотрудник.Город < > Совместитель.Город
AND       Замещение.Ид_Сотр = Сотрудник.Ид_Совм;
```

С помощью этого приема удается вывести в столбце таблицы значение, которое фактически не хранится в базе данных. Слово "**Заменял**" появится в каждой строке таблицы, представленной на рис. 5.8. Отметим, что подобная хитрость выходит за рамки стандарта SQL. Если же потребуется ввести имя столбца, то можно воспользоваться ключевым словом AS (см. пункт 5.2).

Дата_ замены	Фамилия	Ид_ Сотр		Фамилия	Ид_ Совм
10.02.03	Иванова	8	Заменял	Русов	7
10.02.03	Петрова	7	Заменял	Михайлов	4
10.02.03	Сидоров	3	Заменял	Смирнов	2
10.03.03	Иванова	8	Заменял	Русов	7
10.04.03	Панов	4	Заменял	Смирнов	2
20.05.03	Панов	4	Заменял	Смирнов	2

Рис. 5.8. Виртуальный столбец

5.1.5. Самообъединение таблиц

В ряде задач возникает необходимость одновременной обработки данных какой-либо таблицы и одной или нескольких ее копий, создаваемых на время выполнения запроса. Самообъединение – объединение таблицы с этой же таблицей – является другим вариантом объединения таблиц на основе операции эквисоединения. В этом случае сравниваются значения внутри столбца одной таблицы.

Например, требуется узнать, какие сотрудники (фамилии и их имена) проживают во Фрунзенском районе и имеют одинаковые почтовые индексы. Поскольку в этом запросе таблица СОТРУДНИК (рис. 2.1) выступает в двух ролях, она будет объединяться сама с собой (в таблице СОТРУДНИК значению атрибута *Район* "Фрунзенский" соответствует сокращенное "Фрун").

Для различения этих ролей в списке таблиц ей необходимо назначить два разных псевдонима. Пусть имена этих псевдонимов будут С1 и Копия соответственно

```
SELECT С1.Фамилия, С1.Имя, С1.Индекс
FROM        Сотрудник  С1, Сотрудник  Копия
WHERE  С1.Район = "Фрун"
AND         С1.Индекс =  Копия.Индекс;
```

В результате выполнения указанных операторов получим таблицу (рис.5.9), содержимое которой хотя бы по объему (в реальной ситуации) может быть значительным.

Поэтому уберем повторяющиеся строки (рис. 5.10), используя ключевое слово DISTINCT (вспомните алгебраическую операцию **проекции,** которая выполняет аналогичную работу**)** в исходном описании запроса:

```
SELECT      DISTINCT    С1.Фамилия, С1.Имя, С1.Индекс
FROM            Сотрудник     С1, Сотрудник  Копия
WHERE           С1.Район = "Фрун"
AND             С1.Индекс = Копия.Индекс;
```

Фамилия	Имя	Индекс
Сидоров	Петр	94555
Сидоров	Петр	94555
Сидоров	Петр	94555
Панов	Антон	94556
Панов	Антон	94556
Сидрова	Ада	94556
Сидрова	Ада	94556
Иванов	Иван	94555
Иванов	Иван	94555
Иванов	Иван	94555
Хохлов	Иван	94555
Хохлов	Иван	94555
Хохлов	Иван	94555
Яковлев	Иван	94550

Рис. 5.9. Произведение самим с собой

Фамилия	Имя	Индекс
Сидоров	Петр	94555
Панов	Антон	94556
Сидрова	Ада	94556
Иванов	Иван	94555
Хохлов	Иван	94555
Яковлев	Иван	94550

Рис. 5.10. Проекция по таблице из рис. 5.9

Полученный результат вновь не соответствует тому, чего мы хотим, так как он содержит всех сотрудников, проживающих во Фрунзенском районе. Например, сотрудник Яковлев Иван попал в этот список, но он не имеет сослуживцев, которые бы пользовались таким же почтовым индексом, как и он сам.

Проанализируем запрос еще раз: при самообъединении таблицы все значения почтовых индексов сравниваются между собой, и это заставляет систему автоматически добавлять в результирующую таблицу всех проживающих во Фрунзенском районе. Чтобы отсечь таких людей, необходимо во фразе WHERE добавить дополнительное условие:

```
SELECT  DISTINCT  C1.Фамилия, C1.Имя, C1.Индекс
FROM              Сотрудник  C1, Сотрудник  Копия
WHERE             C1.Район = ‘Фрун’
```

AND С1.Индекс = Копия.Индекс

AND С1.Ид_Сотр <> Копия.Ид_Сотр;

Именно приведенное предложение и позволяет получить то, что мы хотели (рис. 5.11).

Фамилия	Имя	Индекс
Сидоров	Петр	94555
Панов	Антон	94556
Сидрова	Ада	94556
Иванов	Иван	94555
Хохлов	Иван	94555

Рис. 5.11. Истинный результат запроса

Следующий запрос (с самообъединением таблицы) позволяет найти, например, для совместителя Михайлова всех его коллег проживающих с ним в одном городе, т.е в Санкт-Петербурге (СПб). Ответ приведен на рис. 5.12.

SELECT Копия2.Фамилия, Копия2.Имя

FROM Сотрудник Копия1, Сотрудник Копия2

WHERE Копия1.Ид_Совм = 4 AND Копия2.Город = Копия1.Город;

Фамилия	Имя
Петрова	Нина
Никитин	Валентин

Рис. 5.12. Необязательное использование Alias

Пример показывает, что не обязательно использовать во фразе SELECT каждый псевдоним (алиас) или таблицу, которые указаны во фразе FROM запроса.

Псевдоним Копия1 отсекает всех совместителей, кроме совместителя с номером 4. Псевдоним Копия2 будет истинен для всех строк с тем же самым значением города, что и текущее значение города для Копия1. Нахождение этих строк псевдонима Копия2 - единственная цель псевдонима Копия1.

В SQL используются и другие виды объединения, они носят названия **внешних**, но здесь они не рассматриваются.

5.2. Оператор UNION

Оператор UNION объединяет вывод двух или более SQL запросов в единый набор строк и столбцов. UNION - это аналог операции объединения реляционной алгебры. Оператор полезен, если требуется просмотреть аналогичные данные из разных таблиц. Его синтаксис следующий:

SELECT
UNION [ALL]
Оператор SELECT
UNION [ALL]
Оператор SELECT
[, … n];

Например, чтобы получить всех сотрудников и совместителей, хранящихся в базе и живущих в СПБ или в Колпино, необходимо написать следующее предложение (результат запроса представлен на рис. 5.13):

```
SELECT    Фамилия, Имя, "Штатный"    AS    "Кто"
FROM      Сотрудник
WHERE     Город      IN     ("СПБ", "Колпино")
UNION
SELECT Фамилия, Имя, "Совместитель"  AS    "Кто"
FROM      Совместители
WHERE Город    IN     ("СПБ", "Колпино");
```

Рис. 5.13.

Фамилия	Имя	Город	Кто
Иванов	Иван	СПБ	Штатный
Петров	Иван	Колпино	Штатный
Сидоров	Петр	СПБ	Штатный
Панов	Антон	СПБ	Штатный
Петрова	Нина	СПБ	Штатный
Сидрова	Ада	СПБ	Штатный
Никитин	Виктор	СПБ	Штатный
Мухин	Степан	СПБ	Штатный
Попов	Михаил	СПБ	Штатный
Иванов	Иван	СПБ	Штатный
Хохлов	Иван	СПБ	Штатный
Яковлев	Иван	СПБ	Штатный
Петров	Петр	СПБ	Совместитель
Алехин	Игорь	Колпино	Совместитель

Фамилия	Имя	Город	Кто
Алехин	Игорь	Колпино	Совместитель
Иванов	Иван	СПБ	Штатный
Иванов	Иван	СПБ	Штатный
Мухин	Степан	СПБ	Штатный
Никитин	Виктор	СПБ	Штатный
Панов	Антон	СПБ	Штатный
Петров	Иван	Колпино	Штатный
Петров	Петр	СПБ	Совместитель
Петрова	Нина	СПБ	Штатный
Попов	Михаил	СПБ	Штатный
Сидоров	Петр	СПБ	Штатный
Сидрова	Ада	СПБ	Штатный
Хохлов	Иван	СПБ	Штатный
Яковлев	Иван	СПБ	Штатный

UNION в запросе Рис. 5.14. Сортировка по первому столбцу

Обратите внимание, что в обоих запросах используется одинаковое число столбцов с совместимыми типами данных (иначе будет противоречие по определению операции UNION). В качестве заголовков столбцов используются имена столбцов (или псевдонимы) первого запроса. В результат включен еще один результирующий столбец, которого фактически в базе нет. Каждая из инструкций SELECT может иметь свое собственное предложение WHERE, но весь запрос может использовать только одно предложение ORDER BY. Применяется оно ко всему результату и должно находиться в последней инструкции SELECT.

Если в конец предыдущего запроса (перед точкой с запятой) добавить фразу ORDER BY 1, то получим следующую таблицу (рис. 5.14).

По умолчанию оператор UNION автоматически устраняет из результата повторяющиеся строки. Чтобы избежать этого, необходимо после оператора UNION добавить ключевое слово ALL

Кроме того, оператор UNION можно использовать для отображения различных значений одного поля в зависимости от значений в других полях.

Пусть, например, в таблице РАБОТЫ (рис. 2.8) значения столбца *Цена* необходимо изменить по следующему правилу: если значение меньше 1.6, то добавить 0.1, иначе вычесть 0.1. Запрос приведен ниже, а результат представлен на рисунке 5.15.

```
SELECT  “на+0.1”, Цена AS Старая , Цена+0.1 AS Новая
FROM Работа
WHERE Цена < 1.6
UNION       ALL
SELECT  “на-0.1”, Цена AS Старая , Цена-0.1 AS Новая
FROM Работа
WHERE Цена <> 1.6
SELECT
ORDER BY          1;
```

	Старая цена	Новая цена
На-0.1	1.7	1.6
На-0.1	1.7	1.6
На–0.1	1.7	1.6
На+0.1	1.5	1.6
На-0.1	1.7	1.6
На-0.1	1.7	1.6
На+0.1	1.5	1.6

Рис. 5.15**.** Сложный UNION

Если в столбцах, по которым производится объединение, имеются значения NULL, то они пропускаются, поскольку нет основания считать, что одно неизвестное значение совпадает с другим неизвестным значением.

5.3.Структурированные запросы

5.3.1. Виды вложенных подзапросов

Вложенный подзапрос - это предложение SELECT, вложенное:

- во фразу WHERE, HAVING или другой инструкции SELECT,
- в инструкцию INSET, UPDATE, DELETE,
- в другой подзапрос.

Название “вложенный” говорит, что данная конструкция может быть вложена в другую (ей подобную) конструкцию. Поэтому термин подзапрос используется для обозначения как всей конструкции, включающей один или несколько подзапросов, так и для обозначения отдельного вложения в конструкции запроса. Проще говоря, подзапрос – это выражение SELECT-FROM-WHERE-GROUP BY-HAWING, которое вложено в другое такое же выражение.

Упрощенно синтаксис подзапроса:

SELECT [DISTINCT]

```
FROM        список_таблиц
WHERE
{выражение {[NOT]     IN     оператор_ сравнения}  [ANY  |  ALL]}
        [NOT]        EXIST}
(SELECT    [DISTINCT]        список_ выбора_ подзапроса
        FROM        список_ таблиц
        WHERE      условия)
[GROUP BY        список        группировки
[HAVING   условия]
[ORDER BY        порядок_ сортировки];
```

В зависимости от того, каким образом вложенный подзапрос взаимодействует с внешним подзапросом, их подразделяют на два вида: **простые** и **коррелированные**.

Простые вложенные подзапросы обрабатываются системой "снизу вверх". Первым обрабатывается вложенный подзапрос самого нижнего уровня. Множество значений, полученное в результате его выполнения, используется при реализации подзапроса более высокого уровня и т.д..

Запросы с коррелированными вложенными подзапросами обрабатываются системой в обратном порядке. Сначала выбирается первая строка рабочей таблицы, сформированной основным запросом, и из нее выбираются значения тех столбцов, которые используются во вложенном подзапросе (вложенных подзапросах). Если эти значения удовлетворяют условиям вложенного подзапроса, то выбранная строка включается в результат. Затем выбирается вторая строка и т.д., пока в результат не будут включены все строки, удовлетворяющие вложенному подзапросу (последовательности вложенных подзапросов). Как простые, так и коррелированные подзапросы начинаются с ключевого слова IN или оператора сравнения и содержат ключевые слова ANY или ALL.

Подзапросы, которые начинаются с ключевого слова EXISTS (существует), представляют собой проверку на существование.

Обычно внутренний запрос генерирует значение, которое проверяется в предикате WHERE внешнего запроса, определяющего верно оно или нет. Например, предположим,

что мы знаем фамилию совместителя: Петров, но не знаем значение его поля *Ид_Совм*, и хотим извлечь все записи из таблицы ЗАМЕЩЕНИЕ (рис. 2.7), тогда запрос к базе будет:

```
SELECT *
FROM Замещение
WHERE Ид_Совм  =
(SELECT Ид_Совм
    FROM Совместители
    WHERE Фамилия = "Петров");
```

Чтобы исполнить внешний (основной) запрос, SQL должен сначала выполнить внутренний подзапрос внутри предложения WHERE. Он "пробежит" все строки таблицы СОВМЕСТИТЕЛЕЙ и выберет те строки, где поле *Фамилия* равно значению Петров, и извлечет значения поля *Ид_Совм* этих строк.

Единственной найденной строкой, естественно, будет Ид_Совм = 1. Однако SQL не просто выдает это значение, а помещает его в предикат основного запроса вместо самого подзапроса так. чтобы предикат прочитал что:

```
WHERE Ид_Совм = 1
SELECT *
FROM Замещение
WHERE Ид_Совм =
(SELECT Ид_Совм
FROM Совместители
WHERE Фамилия = "Петров");
```

Ид_Совм	Фамилия	Имя	Отчество	ИНН	Город	Район	Индекс
1	Петров	Петр	Петрович	836	СПБ	Центр	97000

Рис. 5.16. Использование подзапроса

Основной запрос затем выполняется как обычно, но уже с известным значением Ид_Совм. Конечно же, подзапрос должен выбрать один и только один столбец, а тип данных этого столбца должен совпадать с тем значением, с которым он будет сравниваться в предикате.

Подчеркнем, что представленный запрос будет работать правильно, пока среди совместителей не появится сотрудник с такой же фамилией.

При использовании подзапросов в предикатах, основанных на реляционных операторах, Вы должны быть уверены, что использовали подзапрос, который будет выдавать одну и только одну строку вывода. Если подзапрос не выводит никаких значений вообще, команда не будет ошибочной, но основной запрос не выведет никаких значений

Можно использовать DISTINCT в подзапросе или подзапросы с IN (см. п. 5.3.2).

5.3.2. Простые вложенные подзапросы

Простые вложенные подзапросы используются для представления множества значений, обработка которых должна осуществляться в каком-либо предикате IN.

Проиллюстрируем простой подзапрос на следующем примере. Найти фамилии сотрудников, для которых планировались командировки в Мурманск:

```
SELECT DISTINCT     Фамилия
FROM    Сотрудник
WHERE  Ид_Сотр      IN
  (       SELECT      Ид_Сотр
          FROM  Командировки
             WHERE    Точка = "Мурманск");
```

Внутренний подзапрос выполняется первым и готовит для более старшего подзапроса результат, затем исполняется вышестоящий запрос.

Этот вложенный подзапрос выдает множество идентификаторов сотрудников, для которых планировались командировки в Мурманск, а именно множество {1, 2, 5}. Поэтому первоначальный запрос эквивалентен такому простому запросу с IN:

```
SELECT DISTINCT     Фамилия
FROM                Сотрудник
WHERE               Ид_Сотр  IN (1,2,5);
```

Следующий тип запроса относится к коррелированному, который также позволяет получить тот же результат, что и рассмотренные нами выше:

```
SELECT              Фамилия
FROM                Сотрудник  P
WHERE  "Мурманск"   IN
  (          SELECT          Точка
             FROM            Командировки K
                WHERE        K.Ид_Сотр = P.Ид_Сотр );
```

В данном случае вложенный подзапрос (внутренний) для своего выполнения должен получить данные из внешнего запроса. Но результат для первого и второго запросов будет одинаков (рис. 5.17).

Фамилия
Иванов
Петров
Петухов

Рис. 5.17. Результат запроса

Полученный результат можно сформировать и с помощью операции соединения, например, так:

```
SELECT DISTINCT     Фамилия
FROM                Сотрудник, Командировки
WHERE  Сотрудник.Ид_Сотр = Командировки.Ид_Сотр
AND    Командировки.Точка = "Мурманск";
```

Или

```
SELECT DISTINCT     Фамилия
FROM                Сотрудник P, Командировки K
```

```
WHERE  Р.Ид_Сотр = К.Ид_Сотр
AND    К.Точка = "Мурманск";
```

При выполнении этого запроса, использующего объединение, система должна одновременно обрабатывать данные из двух таблиц, тогда как в предыдущем примере эти таблицы обрабатываются поочередно. Естественно, что для их реализации требуются различные ресурсы памяти и времени.

5.3.3. Коррелированные вложенные подзапросы

В коррелированном запросе внутренний подзапрос не может быть реализован отдельно: он ссылается на внешний запрос и выполняется последовательно для каждой строки внешнего запроса. Так, например, для запроса:

```
SELECT Фамилия
FROM   Сотрудник Р
WHERE  “Мурманск”   IN
  (    SELECT       Точка
       FROM         Командировки К
          WHERE     К.Ид_Сотр = Р.Ид_Сотр );
```

внутренний запрос будет выполняться столько раз, сколько строк находится во внешней таблице СОТРУДНИК (рис. 2.1), а строк в ней 14.

Рассмотрим более детально, каким образом осуществляется обработка запроса. Внешний запрос отыскивает первую фамилию в таблице СОТРУДНИК, например, как у нас в таблице Иванов. Далее система передает

управление вложенному подзапросу, для которого в качестве параметров передается значение переменной - это Р.Ид_Сотр. Теперь внутренний запрос на основании этих данных ищет в таблице КОМАНДИРОКИ записи, соответствующие указанному условию. Таких записей две.

После этого внутренний запрос возвращает результат в приложение IN внешнего запроса, где значения столбца *Точка* из промежуточной таблицы результата (рис. 5.18) вложенного запроса сравниваются со строкой "Мурманск".

N строки базовой таблицы	Ид_Сотр	Точка
1	1	Мурманск
4	1	Петрозаводск

Рис. 5.18. Промежуточный результат

И только те строки промежуточной таблицы (рис. 5.18) попадают в результирующую таблицу запроса, которые в столбце Точка имеют значение равное "Мурманск". Затем снова начинает работать внешний запрос, переходя на строку 2 базовой таблицы СОТРУДНИК (рис. 2.1), тем самым определяя новый Ид_Сотр, который будет передан во вложенный подзапрос.

Результатом работы подзапроса будет (рис. 5.19) одна строка базовой таблицы КОМАНДИРОВКИ (рис. 2.5).

N строки базовой таблицы	Ид_Сотр	Точка
2	2	Мурманск

Рис. 5.19. 1-я строка ответа

Эта строка попадет, подвергаясь обработке внешним запросом, в ответ. Переход в базовой таблице СОТРУДНИК на третью строку передаст во внутренний подзапрос новый идентификатор сотрудника Ид_Сотр=3. Внутренний подзапрос выдаст ответ {Петрозаводск} (рис. 5.20).

N строки базовой таблицы	Ид_Сотр	Точка
3	3	Петрозаводск

Рис. 5.20. Строка не попадает в ответ

Полученный результат после обработки внешним запросом будет отклонен в качестве ответа в результирующую таблицу.

Таким образом, внешний запрос пробегает все строки базовой таблицы СОТРУДНИК, передавая идентификаторы сотрудников во внутренний запрос, на основании которых создается промежуточная таблица, которая в свою очередь подвергается обработке со стороны внешнего запроса.

Такой подзапрос отличается от простого вложенного подзапроса тем, что вложенный подзапрос не может быть обработан прежде, чем будет обрабатываться внешний подзапрос. Это связано с тем, что вложенный подзапрос зависит от значения Сотрудник.Ид_Сотр. Это значение изменяется по мере того, как система проверяет различные строки таблицы СОТРУДНИК (рис. 2.1).

Одна таблица во внешнем и внутреннем подзапросе

Рассмотрим пример использования одной и той же таблицы во внешнем подзапросе и коррелированном вложенном подзапросе: "Показать номера работ для всех видов работ, выполняемых только одним сотрудником".

Заметим, что для реализации подобных запросов необходимо использовать алиасы (псевдонимы).

```
SELECT DISTINCT      Раб1.Ид_Вида
FROM                 Работы Раб1
WHERE  Раб1.Ид_Вида  NOT  IN
  (          SELECT          Раб2.Ид_Вида
             FROM            Работы Раб2
                WHERE        Раб2.Ид_Сотр <> Раб1.Ид_Сотр);
```

Алгоритм работы запроса можно пояснить следующим образом:
"В цикле для каждой **i**-й строки таблицы РАБОТЫ (рис. 2.8), выделить значение Ид_Сотр, если и только если это значение не входит в некоторую строку, скажем, **j** той же таблицы, а значение столбца Ид_Вида в строке **j** не равно его значению в строке **i**".

5.3.4. Запросы, использующие EXISTS

Квантор EXISTS (существует) - понятие, заимствованное из формальной логики. В языке SQL предикат с квантором существования представляется выражением EXISTS (SELECT * FROM ...).

Такое выражение считается истинным только тогда, когда результат вычисления "SELECT * FROM ..." является непустым множеством, т.е. когда существует какая-либо запись в таблице, указанной во фразе FROM подзапроса, которая удовлетворяет условию WHERE подзапроса.

Слово EXISTS неформально обозначает отбор только тех записей, для которых вложенный запрос возвращает только одно или более значений. Например, оператор:

```
SELECT     Ид_Сотр, Фамилия, Имя, Год_ Рожд, Пол
FROM       Сотрудник          p1
WHERE EXISTS  (SELECT Ид_Сотр, Год_ Рожд FROM Сотрудник  P2
WHERE (p1.Год_Рожд = p2.Год_Рожд) AND (p1.Ид_Сотр  != p2.Ид_Сотр));
```

Вернет список сотрудников, которые имеют хотя бы одного сверстника.

Ид_Сотр	Фамилия	Имя	Год_рожд.	Пол
1	Иванов	Иван	1949	М
2	Петров	Иван	1949	М
3	Сидоров	Петр	1947	М
6	Иванова	Вера	1970	Ж
7	Петрова	Нина	1970	Ж
8	Сидрова	Ада	1970	Ж
11	Попов	Михаил	1947	М
13	Иванов	Иван	1980	М
14	Яковлев	Иван	1980	М

Рис. 5.21. Сотрудники, имеющие сверстников

Имеется также ключевое слово **SINGULAR**, которое означает отбор только тех записей, для которых вложенный запрос возвращает только одно значение.

Например, оператор

```
SELECT     Ид_Сотр, Фамилия, Год_Рожд
FROM       Сотрудник          p1
```

```
        WHERE SUNGULAR
(SELECT    Ид_Сотр, Год_ рожд     From  Сотрудник  p2
        WHERE p1. Год_ рожд = p2. Год_ рожд);
```

вернет список сотрудников, которые не имеют ни одного сверстника (год рождения которых совпадает только с собственным годом рождения и больше не повторяется в таблице).

Рассмотрим пример. Выдать состав сотрудников (*Фамилии, Имена*), входящих в отдел с номером 1. (Ответ представлен на рис. 5.22).

```
SELECT Фамилия, Имя
FROM   Сотрудник
WHERE          EXISTS
  (        SELECT        *
           FROM  Отдел_ Сотрудники
           WHERE         Ид_Сотр = Сотрудник.Ид_Сотр
              AND  Ид_Отд = 1 AND Дата_ увольнения        NOT NULL);
```

Система последовательно выбирает строки таблицы СОТРУДНИК (рис. 2.1), выделяет из них значения столбцов *Фамилия* и *Ид_Comp*, а затем проверяет, является ли истинным условие существования, т.е. существует ли в таблице ОТДЕЛ_СОТРУДНИК (рис. 2.3) хотя бы одна строка со значением *Ид_Отд*=1 и значением *Ид_Comp*, равным значению *Ид_Comp*, выбранному из таблицы СОТРУДНИК. Если условие выполняется, то полученное значение столбца *Фамилия* включается в результат.

Первая фамилия значения поля *Фамилия* таблицы СОТРУДНИК равна "Иванов", и *Ид_Comp* равен 1. Так как в таблице ОТДЕЛ_СОТРУДНИК есть строка с *Ид_Отд*=1 и *Ид_Отд*=1, то значение "Иванов" должно быть включено в результат (рис. 5.22).

Фамилия	Имя
Иванов	Иван
Петров	Иван
Сидоров	Петр
Панов	Антон
Петухов	Виктор
Иванова	Вера
Петрова	Нина

Рис. 5.22.Сотрудники отдела номер 1

Хотя этот пример только показывает иной способ формулировки запроса для задачи, решаемой и другими путями (с помощью оператора IN или соединения), EXISTS представляет собой одну из наиболее важных возможностей SQL. Фактически любой запрос, который выражается через IN, может быть альтернативным образом сформулирован также с помощью EXISTS. Однако обратное высказывание несправедливо.

Если бы нам потребовалось показать всех сотрудников организации, исключая сотрудников первого отдела, то следует написать следующий запрос:

```
SELECT Ид_Отд, Фамилия, Имя, Отчество, Год_ рожд
FROM   Сотрудник
WHERE  NOT EXISTS
  (        SELECT        *              FROM  Отдел_ Сотрудники
WHERE  Ид_Сотр = Сотрудник.Ид_Сотр              AND    Ид_Отд = 1);
```

Ид_Сотр	Фамилия	Имя	Отчество	Год рожд.
9	Никитин	Виктор	Сергеевич	1952
10	Мухин	Степан	Михайлович	1964
11	Попов	Михаил	Михайлович	1947
12	Иванов	Иван	Иванович	1980
13	Хохлов	Иван	Васильевич	1960
14	Яковлев	Иван	Васильевич	1980

Рис. 5.23. Сотрудники, не принадлежащие отделу с номером 1

5.3.5. Использование функций в подзапросе

Любая агрегатная функция (см. п. 4.4) по определению выдает одиночное значение на любом количестве строк, для которых она использована.

Запрос, использующий одиночную функцию агрегата без предложения GROUP BY, будет выбирать одиночное значение для использования в основном предикате.

Пояснением к сказанному будет следующий пример. Найти год рождения самого молодого сотрудника в базовой таблице СОТРУДНИК (рис. 2.1)

```
SELECT     MIN (Год_рожд)
FROM       Сотрудник;
```

Ответ: 1980.

Заметим, что в таблице три таких строки с данным значением, но запрос возвращает только одно значение.

Имейте в виду, что сгруппированные агрегатные функции, которые являются агрегатными функциями, определенными в терминах предложения GROUP BY, могут производить многочисленные значения. Они, следовательно, не позволительны в подзапросах такого характера. Даже если GROUP BY и HAVING используются таким способом, что только одна группа выводится с помощью подзапроса, команда будет отклонена в принципе. Вы должны использовать одиночную агрегатную функцию с предложением WHERE, что устранит нежелательные группы. Например, следующий запрос, который должен найти среднее значение окладов сотрудников в отделе 2:

```
SELECT AVG (Оклад)
FROM Отдел_Сотрудники
GROUP BY Ид_Отд
HAVlNG     Ид_Отд = 2;
```

не может использоваться в подзапросе! Правильный запрос: -

```
SELECT AVG (Оклад)
FROM Отдел_сотрудники
WHERE Ид_Отд = 2;
```

Вернемся к первому запросу данного параграфа, который указал нам, что самый молодой сотрудник в таблице сотрудников имеет год рождения 1980.Теперь на основании данной информации можно найти всех самых молодых сотрудников в таблице СОТРУДНИК (рис. 2.1), с помощью следующего запроса:

```
SELECT *
FROM Сотрудник
WHERE Год_рожд=1980
```

И получим следующий ответ (рис. 5.24.).

Ид_ Сотр	Фамилия	Имя	Отчество	ИНН	Год_ рожд.	Пол	Город	Район	Ин- декс	Ид_ Совм
12	Иванов	Иван	Иванович	712	1980	М	СПБ	Фрун	94555	
14	Яковлев	Иван	Васильевич	714	1980	М	СПБ	Фрун	94550	

Рис. 5.24. Сотрудники 1980 года рождения

Подзапрос позволяет соединить рассмотренные два запроса в один.

Найти всех самых молодых сотрудников в организации:

```
SELECT *
FROM    Сотрудник
WHERE   Год_рожд=
(SELECT  MIN (Год_рожд)
FROM    Сотрудник;
```

Теперь, после знакомства с различными формулировками вложенных подзапросов и псевдонимами, легче понять текст и алгоритм реализации запроса:

Найти список сотрудников, сгруппировав их по отделам, средний возраст которых больше среднего.

```
SELECT

FROM    Сотрудник a, Отдел_Сотрудник b
```

```
ORDER BY        Ид_Отд
WHERE           Ид_Сотр.a =Ид_Сотр.b
AND
    WHERE   Год_Рожд >
    (SELECT   (AVG (Год_рожд)
    FROM Сотрудник C ,Отдел_ Сотрудник  d
    WHERE( Ид_Сотр.a =Ид_Сотр.b     AND
    Ид_Отд.b=Ид_Отд.d);
```

Естественно, что это коррелированный подзапрос: здесь сначала определяется средний возраст сотрудников и только затем выясняется, кто же попадает в этот диапазон.

На этом примере мы закончим знакомство с вложенными подзапросами и с конструкцией SELECT понимая, что многие тонкости и более существенные моменты остались нерассмотренными. Только практика использования языка позволяет снять эти пробелы. Здесь отметим, что в развитых СУБД механизмы, используемые в SQL, реализованы во всевозможных генераторах запросов (Query), которые значительно облегчают поиск информации в базах данных.

Глава 6. Модификация данных

6.1. Синтаксис инструкций модификации данных

Операции добавления, изменения и удаления данных выполняются с помощью инструкций (команд) **INSERT** (вставить), **DELETE** (удалить) и **UPDATE** (обновить). Подобно команде **SELECT** они могут оперировать как базовыми таблицами, так и логическими представлениями (view), объединяющими информацию из нескольких таблиц. Однако не все представления являются обновляемыми. Кроме изменения значений в существующих столбцах, может потребоваться изменить существующие или добавить новые столбцы в существующие таблицы. Подобная модификация осуществляется с помощью инструкции **ALTER TABLE**, упоминавшейся ранее. Создание новой таблицы (**CREATE TABLE**) - это тоже модификация, цель же данной главы - ознакомление с командами манипулирования данными: DELETE, INSERT, UPDATE (конечно, при условии, что у нас имеются соответствующие полномочия на модификацию хранимой информации). Вопросы безопасности и связанные с ними различные режимы работы (совместное или монопольное использование данных) в этом учебном пособии не рассматриваются.

Инструкция **INSERT** добавляет новые строки в базу данных и имеет один из следующих форматов:

INSERT

INTO {имя базовой таблицы | представление} [(список столбцов)]

VALUES ({константа | переменная} [,{константа | переменная}] ...);

Или

INSERT

INTO {имя базовой таблицы | представление} [(список столбцов)]

VALUES(подзапрос);

В первом варианте в таблицу вставляется строка со значениями полей, указанными в перечне фразы VALUES (значения), причем i-е значение соответствует i-му столбцу в списке столбцов (столбцы, не указанные в списке, заполняются NULL-значениями). Если

в списке фразы VALUES указаны все столбцы модифицируемой таблицы и порядок их перечисления соответствует порядку столбцов в описании таблицы, то список столбцов во фразе INTO не присутствует.

Во втором варианте сначала выполняется подзапрос, формирующий в памяти рабочую таблицу, а потом строки рабочей таблицы загружаются в модифицируемую таблицу. При этом i-й столбец рабочей таблицы (i-й элемент списка SELECT) соответствует i-му столбцу в списке столбцов модифицируемой таблицы. Здесь также при выполнении указанных выше условий может быть опущен список столбцов фразы INTO. Фраза INTO не обязательна, но позволяет улучшить читаемость инструкции.

Инструкция **DELETE** удаляет существующие строки и имеет следующий синтаксис:

DELETE

FROM {имя базовой таблицы | представление}

[**WHERE** <условия поиска>];

и позволяет удалить содержимое всех строк указанной таблицы, при отсутствии фразы WHERE или тех ее строк, которые выделяются во фразе WHERE. Инструкцию DELETE следует использовать крайне осторожно. Особенно если она используется без фразы WHERE. После исполнения данной конструкции таблица (файл) физически удаляется с диска и восстановить ее можно только из резервной копии. Ключевое слово FROM в инструкции не обязательно, но обычно используется для читаемости предложения.

Предложение **UPDATE** изменяет существующие в базе данных строки и имеет два формата описания.

Формат 1:

UPDATE {имя базовой таблицы | представление}

SET {имя_ столбца = значение [,имя_ столбца = значение] ...}

[**WHERE** <условия поиска>]

где значение: <имя_ столбца | выражение | константа | переменная>

может включать имена столбцов только из обновляемой таблицы, т.е. значение одного из столбцов модифицируемой таблицы может заменяться на значение ее другого столбца

или выражения (содержащего значения нескольких ее столбцов, включая изменяемый столбец).

Так же, как и в инструкции SELECT, для задания модифицируемых строк используется фраза WHERE. При отсутствии фразы WHERE обновляются значения указанных столбцов во всех строках модифицируемой таблицы. WHERE <условия поиска> позволяет сократить число обновляемых строк, указывая условия их отбора.

Второй формат описывает предложение, которое позволяет производить обновление значений модифицируемой таблицы по значениям столбцов из других таблиц.

Формат 2:

UPDATE {имя базовой таблицы | представление}

SET {имя столбца = значение [, имя столбца = значение] ...}

FROM {имя базовой таблицы | представление} [псевдоним],

{имя базовой таблицы | представление} [псевдоним]

[,{имя базовой таблицы | представление} [псевдоним]] ...

[**WHERE** <условия поиска>]

Здесь перечень таблиц фразы FROM содержит имя модифицируемой таблицы и тех таблиц, значения столбцов которых используются для обновления. При этом, естественно, таблицы должны быть связаны между собой во фразе WHERE, которая, кроме того, служит для указания условий отбора обновляемых строк модифицируемой таблицы.

6.2. Инструкция INSERT

6.2.1. Добавление одной строки в таблицу

Все новые строки в таблицу в SQL вводятся с использованием команды модификации данных INSERT.

Так, например, чтобы ввести строку в таблицу ОТДЕЛ (рис. 2.2), можно использовать следующее предложение:

```
INSERT INTO Отдел
VALUES (7, "Бухгалтерия", 15, "Управление", 1930);
```

Команды типа INSERT не производят вывода на экран, но СУБД должна дать Вам подтверждение того, что данные были добавлены в таблицу.

Имя таблицы (в нашем случае - ОТДЕЛ) должно быть предварительно определено в команде CREATE TABLE (см. главу 3), а каждое значение, указанное в списке значений VALUES, должно совпадать с типом данных столбца, в который оно вставляется.

Значения вводятся в таблицу в том порядке, в котором они присутствуют в списке значений фразы VALUES, поэтому первое значение с идентификатором отдела автоматически попадает в столбец 1, второе - в столбец 2, и так далее. Заметим, что сотрудник с идентификатором *Ид_Начальника*=15 уже должен быть зафиксирован в таблице СОТРУДНИК (рис. 2.1).

Если нужно ввести пустое значение (NULL), то его вводят точно так же, как и обычное значение. Предположим, что еще не определилось значение поля "Название отдела" для создаваемого отдела. Тогда оператор вставки будет выглядеть следующим образом:

```
INSERT INTO Отдел
VALUES (7, NULL, 15, "Управление", 1930);
```

Так как значение NULL - это специальный маркер, а не просто символьное значение, он не включается в одиночные кавычки.

Некоторые СУБД допускают и такое использование команды:

```
INSERT INTO Отдел
VALUES (7,        , 15, "Управление", 1930 );
```

Что эквивалентно предыдущей инструкции.

Порядок полей в INSERT не обязательно должен совпадать с порядком полей, в котором они определялись при создании таблицы. Вполне допустима и такая версия предыдущего предложения:

INSERT

INTO

Отдел (Название_отдела, Ид_Отдела, Ид_Начальника, Вид_отдела,

Год_основания)

VALUES (NULL, 7, 15, "Управление", 1930);

Следующее предложение тоже допустимо:

INSERT INTO Отдел (Название_ отдела, Ид_Отдела, Вид_ отдела)

VALUES ("АСУ", 8, "Управление");

Обратите внимание, что столбцы *Ид_Начальника* и *Год_основания* - отсутствуют. Это значит, что эти элементы строки будут автоматически установлены в значение - по умолчанию. По умолчанию может быть введено или значение NULL, или другое значение, определяемое как - по умолчанию. Если ограничение, наложенное на значения столбца, запрещает использование значения NULL в данном столбце и этот столбец не имеет установки по умолчанию, то система в принципе должна оповестить пользователя о недопустимости подобной операции.

В предыдущих примерах проводилась модификация таблицы ОТДЕЛ (рис. 2.2), т.е. таблицы с первичным ключом *Ид_Отдела*. Практически все СУБД имеют механизмы для предотвращения ввода неуникального первичного ключа. Т.е. попытка добавления следующей строки в таблицу ОТДЕЛ (рис. 2.2) вида

INSERT INTO Отдел

VALUES (1, "Бухгалтерия", 15, "Управление", 1930);

должна быть отклонена системой, иначе произойдет нарушение целостности системы в целом.

А как поступать со значениями внешних ключей ?

Пусть в момент исполнения предложения

INSERT INTO Отдел

VALUES (7, "Бухгалтерия", 17, "Управление", 1930);

в таблице СОТРУДНИК (рис. 2.1) еще не добавлена строка для сотрудника с идентификационным номером *Ид_Comp*=17. Заметим, что в таблице СОТРУДНИК (рис. 2.1) столбец *Ид_Comp* определен как первичный (PRIMARY) ключ. В таблице ОТДЕЛ (рис. 2.2) первичным ключом должен быть определен атрибут (столбец) *Ид_Отдела*. Атрибут *Ид_Начальника* не может быть определен первичным ключом по определению. **SQL запрещает иметь более одного (реляционного первичного) ключа в таблице.** Поэтому атрибут *Ид_Начальника* получает статус внешнего (FOREIGN) ключа для указания системе о необходимости поддержания ссылочной целостности между таблицами СОТРУДНИК (рис. 2.1) и ОТДЕЛ (рис. 2.2), который по определению не может содержать NULL значения. Поэтому для схемы базы данных, с указанными ограничениями ссылочной целостности выполнить команду без ее нарушения нельзя. Другими словами, до тех пор, пока не появится начальник для нового создаваемого отдела, информацию об отделе нельзя занести в базу данных.

Проблемы, возникающие при использовании внешних ключей, подробно рассмотрены в литературе [1, 6, 23, 30]. Здесь только подчеркнем, что операция INSERT может нарушить непротиворечивость базы данных. Если не принять специальных мер, то СУБД, основываясь только на знаниях первичных и внешних ключей отношений, может допустить ссылку на несуществующую запись. Отметим, что не все СУБД имеют механизмы для предотвращения ввода записей со значениями внешних ключей, отсутствующих среди значений соответствующих первичных ключей.

6.2.2. Добавление нескольких строк

Команду INSERT также можно использовать, чтобы получать или выбирать значения из одной таблицы и помещать их в другую. Для этого ее надо использовать вместе с запросом. Формат команды в этом случае:

INSERT

INTO {имя базовой таблицы | представление} [(список столбцов)]

VALUES (**предложение_SELECT)**;

Т.е. по сравнению с вариантом добавления одной строки в таблицу, фраза VALUES дополняется соответствующим подзапросом. При этом базовая таблица или представление к моменту исполнения команды должна существовать или должна быть создана командой CREATE TABLE. Число и тип столбцов базовой таблицы должны совпадать с такими же параметрами создаваемой рабочей таблицы предложения SELECT.

Параметр **предложение_SELECT** позволяет заполнить строку таблицы данными, которые уже хранятся в одной или нескольких таблицах базы данных.

Использование подобной конструкции оказывается очень удобным, когда приходится резервировать данные или когда администратор базы запрещает Вам работать с базовыми таблицами и требует, чтобы Вы работали с их копиями.

Пусть пустая базовая таблица с именем КОПИЯ_СОВМ ранее была создана с помощью команды CREATE TABLE и содержит те же атрибуты (столбцы) что и наша базовая таблица СОВМЕСТИТЕЛИ (рис. 2.6).

```
INSERT INTO Копия_Совм

VALUSE (   SELECT      *

FROM Совместители

WHERE Город  = "СПБ");
```

Выбираются все значения, произведенные запросом, то есть все строки из таблицы СОВМЕСТИТЕЛИ со значениями *Город* = "СПБ", и помещаются в таблицу с именем КОПИЯ_СОВМ.

КОПИЯ_СОВМ - это теперь независимая таблица, которая получила некоторые значения из таблицы СОВМЕСТИТЕЛИ. Если значения в таблице СОВМЕСТИТЕЛИ будут изменены, это никак не отразится на таблице КОПИЯ_СОВМ.

В рассмотренном примере таблица СОВМЕСТИТЕЛИ содержит входные данные, которые будут скопированы в таблицу КОПИЯ_СОВМ. Естественно, если нас интересуют не все столбцы таблицы СОВМЕСТИТЕЛИ, то команду INSERT следует написать, например, так:

```
INSERT INTO Копия_Совм

(Ид_Совм, Фамилия, ИНН)
```

```
VALUSE    (SELECT    (Ид_Совм, Фамилия, ИНН)
FROM Совместители
WHERE Город  = "СПБ");
```

Обратим Ваше внимание, что при явном указании имен столбцов фактический физический порядок их расположения не важен. Важно другое, их расположение в базовой и клонированной таблице должны совпадать. Если данное требование не соблюдается, СУБД не в состоянии отследить возникающие противоречия.

Предположим, что мы сформировали таблицу с именем КОПИЯ_СОВМ1, столбцы которой совпадают со столбцами нашей таблицы СОТРУДНИК (рис. 2.1). Таблица КОПИЯ_СОВМ1 заполнена на основании следующего предложения:

```
INSERT INTO Копия_Совм1
SELECT *
FROM Совместители
WHERE Город = "Мурманск"  or       Город = "Мирный";
```

Теперь мы можем использовать подзапрос, чтобы добавить к таблице КОПИЯ_СОВМ1 всех СОТРУДНИКОВ, которых они замещают:

```
INSERT INTO Копия_Совм1
SELECT *
FROM Совместители
WHERE Ид_Совм = ANY
(SELECT      Ид_Совм
FROM Сотрудник
WHERE Сотрудник.Ид_Совм = Совместители.Ид_Совм;
```

Оба запроса в этом предложении функционируют так же, как если бы они не являлись частью выражения INSERT.

6.3. Инструкция DELETE

6.3.1. Удаление записи

Пользоваться командой DELETE следует с ***величайшей осторожностью***.

Пусть, например, оператор случайно или преднамеренно решил удалить запись о сотруднике с идентификационным номером *Ид_Comp*=3 в таблице СОТРУДНИК (рис. 2.1).

```
DELETE
FROM    Сотрудник
WHERE  Ид_Сотр = 3;
```

Если таблица РАБОТА (рис. 2.8) содержит в момент выполнения этой команды какие-то виды работ для сотрудника с *Ид_Comp* = 3, то такое удаление нарушит синтаксическую и семантическую непротиворечивость базы данных. Подобное удаление может задеть многие базовые таблицы и представления и в некоторых случаях может потребовать пересмотреть макет информационной схемы Вашей предметной области.

В стандарте SQL отсутствуют операции удаления, воздействующие одновременно на несколько таблиц. Однако в некоторых СУБД предложены и реализованы механизмы поддержания целостности [17, 22, 25, 28, 29], позволяющие отменить некорректное удаление или каскадировать удаление на несколько таблиц с одновременным созданием всевозможных архивных и резервных копий. Приведенный пример или пример типа:

```
DELETE
FROM    Отдел
```

WHERE Ид_Отдела = 1;

показывает, что, используя только средства SQL, невозможно добиться адекватного отображения предметной области в традиционной реляционной модели данных.

6.3.2. Удаление нескольких строк

С помощью инструкции DELETE FROM можно удалять и множество строк. Если в команде не указать фразу WHERE , то будут удалены все записи из таблицы.

Например, чтобы сделать таблицу СОВМЕСТИТЕЛИ (рис. 2.6) пустой, необходимо выдать следующую инструкцию:

DELETE

FROM Совместители;

Для удаления пустой таблицы надо выполнить операцию:

DROP TABLE Совместители (см. пункт 3.1.4).

Исполняется команда DELETE FROM всегда построчно, с соответствующей пометкой в журнале транзакций, поэтому существует возможность отката от проделанной операции.

Если возникает необходимость в удалении всех строк таблицы без возможности отката, то обычно для этого используют инструкцию **TRANCATE**, которая имеет следующий синтаксис:

TRANCATE Имя_ таблицы;

Данная команда выполняется значительно быстрее [25], чем команда DELETE, но восстановить информацию после ее исполнения невозможно.

Во фразе WHERE команды DELETE допустимы все рассмотренные ранее предикаты.

Например, удалить из ВЕДОМОСТИ_ОПЛАТЫ (рис. 2.4) все строки, связанные с месяцем апрель:

DELETE FROM Ведомость_оплаты

```
WHERE  Период = "Апрель";
```

Удалить все работы (рис. 2.8) для сотрудника, проживающего в Гатчине.

```
DELETE
FROM     Работы
WHERE  Ид_Сотр IN
  (SELECT         Ид_Сотр
  FROM  Сотрудник
  WHERE           Город = "Гатчина");
```

Например, если мы закрыли нашу точку в Мурманске, следующий запрос удалит всех совместителей из Мурманска:

```
DELETE
FROM Совместители
WHERE Город = "Мурманск"   and   Ид_Совм = ANY
(SELECT     Ид_Совм
FROM Сотрудник
WHERE Сотрудник.Ид_Совм= Совместители.Ид_Сотр );
```

Как и в случае удаления одной записи, при удалении множества записей в базе данных возможно рассогласование таблиц по ссылкам.

6.4. Инструкция UPDATE

6.4.1. Модификация одной записи

Для модификации значений в одной записи необходимо в инструкции UPDATE указать имя используемой таблицы и в опции предложения SET указать на изменение, которое нужно сделать для определенного столбца. Фраза WHERE должна содержать квалифицирующее выражение, позволяющее выделить только одну строку таблицы.

Например, чтобы изменить у сотрудника район и индекс почты в таблице СОТРУДНИК (рис. 2.1), Вы можете ввести:

```
UPDATE Сотрудник
SET        Район = "Фрун", Индекс = "94555"
```

WHERE Ид_Сотр = 1;

Фактически мы зарегистрировали в базе данных новый факт смены адреса жительства сотрудника. Даже столь безобидная модификация полей одной строки может вызвать много неприятностей, а именно:

- как определить, что новый адрес соответствует действительности;

- что делать, если по каким-то причинам потребуется вспомнить старый адрес сотрудника.

Подобные задачи должны решаться уже встроенными средствами СУБД, а точнее, наличием возможностей в концептуальной модели системы отслеживать подобные ситуации.

6.4.2. Модификация нескольких строк

Если фраза WHERE в инструкции UPDATE содержит квалифицирующее выражение, в результате исполнения которого будет получено несколько строк, то действие опции SET распространяется на все эти строки.

Увеличить оклад сотрудников (рис. 2.3) на 1000 рублей, но только для тех, кто был принят на работу до 2003 года.

UPDATE Отдел_ Сотрудники

SET Оклад = Оклад +1000

WHERE Дата_ приема < 2003 and Дата_ увольнения=NULL;

Установить расценки на 0.3 больше предыдущей, только для штатных преподавателей начиная с мая месяца.

```
UPDATE Работы
SET Цена = Цена+0.3
WHERE Работы.Период_с > 30.04.03 AND Ид_Сотр IN
  (SELECT Ид_Сотр
        FROM Сотрудник, Отдел_ Сотрудники
        WHERE Сотрудник. Ид_Сотр = Отдел_ Сотрудник.
Ид_Сотр AND Отдел_ Сотрудники. Ид_Сотр <> NULL);
```

По определению инструкция **UPDATE** может производить обновление данных только в одной таблице, что на практике приводит к нежелательным эффектам, если возникает необходимость модифицировать информацию в нескольких связанных таблицах.

Пусть необходимо изменить идентификатор (номер) *Ид_Вида* в таблице ВИД_РАБОТЫ (рис. 2.9) например 10 на 16. Атрибут с именем *Ид_Вида* встречается также в таблице РАБОТЫ (рис. 2.8). Поэтому изменение, производимое в таблице ВИД_РАБОТЫ, должно быть отражено в таблице РАБОТЫ.

```
UPDATE Вид_ работы
SET Ид_Вида = 16
WHERE Ид_Вида = 10;
UPDATE Работы
SET Ид_Вида = 16
WHERE Ид_Вида = 10;
```

В интервале времени между первым и вторым оператором UPDATE база данных будет находиться в противоречивом состоянии (нарушается целостность по ссылкам), поскольку после выполнения первой транзакции таблица РАБОТЫ (рис. 2.8) ссылается

на уже несуществующий ВИД_РАБОТЫ. Для решения подобных проблем используются всевозможные блокировки таблиц или их записей [3, 4, 9, 25, 28, 29].

Предложение SET - это не предикат. Он может вводить пустые NULL значения так же, как он вводит другие значения, не используя какого-то специального синтаксиса (такого, например, как IS NULL).

Если Вы хотите запретить использовать Совместителей для штатных преподавателей, живущих в Санкт - Петербурге, то предложение модификации должно быть:

```
UPDATE Сотрудник
SET Ид_Совм = NULL
WHERE Город = "СПБ";
```

Заключение

Подведем некоторые итоги по рассмотренным конструкциям языка SQL.

Оператор **SELECT** позволяет возвращать не только множество значений полей, но и некоторые совокупные (агрегированные) характеристики, подсчитанные по всем или по указанным записям таблицы. Одна из функций, возвращающих также совокупные характеристики, **Count (<условие>)** - количество записей в таблице, удовлетворяющих заданным условиям.

Например, оператор:

```
SELECT count (*)
```

FROM Сотрудник;

подсчитает полное количество записей в таблице Сотрудник. А оператор:

```
SELECT    count (*)
FROM      Отдел_ Сотрудник
WHERE     Ид_Отд = 1;
```

выдаст число записей сотрудников отдела 1.

Оператор, использующий ключевое слово **DISTINCT** (уникальный) выдаст число неповторяющихся значений в указанном поле. Например, оператор:

```
SELECT    count (DISTINCT Ид_Отд)
FROM      Отдел_ Сотрудник;
```

вернет число различных подразделений, указанных в поле Ид_Отд таблицы Отдел_ Сотрудник (рис. 2.3).

Функции **min (<поле>), max (<поле>), avg (<поле>), sum (<поле>)** возвращают соответственно минимальное, максимальное, среднее и суммарное значения указанного поля. Например, оператор:

```
SELECT min (Год_ рожд), max (Год_ рожд), avg (Год_ рожд)
FROM      Сотрудник;
```

вернет минимальное, максимальное и среднее значения года рождения.

Оператор:

```
SELECT (2003 - Год_ рожд), max (2003 - Год_ рожд), avg (2003- Год_ рожд)
FROM Отдел_ Сотрудник    a,    Сотрудник  b
WHERE     Ид_Сотр.a = Ид_Сотр.b AND  Название.b="Бухгалтерия";
```

выдаст Вам аналогичные данные, но относящиеся к возрасту сотрудников бухгалтерии. Вы понимаете, что база данных со временем будет расширяться.

В операторе SELECT вы можете указывать не только суммарные характеристики, но и любые выражения от них.

Например, оператор:

```
SELECT    2003 - (min (Год_ рожд) + Max (Год_ рожд)) / 2
FROM      Отдел_ Сотрудник a, Сотрудник    b
WHERE     Отдел_ Сотрудник a, Сотрудник    b
AND       Название.b="Бухгалтерия";
```

выдаст интервал (среднее между максимальным и минимальным значениями) возраста сотрудников бухгалтерии. Здесь надо обратить внимание на то, что при работе с базой данных функция вернет округленное до целого значение интервала, поскольку в выражении использованы только целые числа, и поэтому осуществляется целочисленное деление. Если же Вы в том же операторе замените делитель «2» на «2.», т.е. укажите его как действительное значение, то и результат будет представлен действительным числом. Отметим, что различные **драйверы** (ODBC, BDE) ведут себя иначе и могут не оправдать наши надежды.

При использовании суммарных характеристик надо учитывать, что в списке возвращаемых значений после ключевого слова **SELECT** могут фигурировать или поля (в том числе вычисляемые), или совокупные характеристики, но не могут фигурировать и те и другие (без указания на группирование данных, о чем будет сказано ниже). Это очевидно, так как оператор может возвращать или множество значений полей записей, или суммарные характеристики по таблице, но не может возвращать эти несовместимые друг с другом данные.

Поэтому нельзя, например, записать оператор:

```
SELECT Фамилия, max (Год_ рожд)
FROM Сотрудник;
```

В этом операторе мы пытаемся определить фамилию самого молодого сотрудника. Впрочем, эту задачу можно решить с помощью вложенных запросов, которые мы уже рассматривали.

Смешение в одном операторе полей совокупных характеристик возможно, если использовать группировку записей, задаваемую ключевыми словами **GROUP BY**. После этих ключевых слов перечисляются все поля, входящие в список **SELECT.** В этом случае смысл совокупных характеристик изменяется: они проводят вычисления не по

всем записям таблицы, а по тем, которые соответствуют одинаковым значениям указанных полей. Например, оператор:

```
SELECT    Ид_Отд    Отдел, count (*) Всего сотрудников
FROM      Отдел_ Сотрудники
GROUP BY      Ид_Отд;
```

вернет таблицу, в которой будет два столбца: столбец *'Отдел'* с идентификаторами отделов и столбец *'Всего_сотрудников'*, в котором будет отображено число сотрудников в каждом отделе (рис. 6.1).

Отдел	Всего_ сотрудников
1	6
2	4
3	1
4	1
5	1

Рис. 6.1. Число сотрудников в отделах

Заметьте, в число сотрудников попадают и уволенные сотрудники.

При группировании записей с помощью **GROUP BY** можно вводить условия отбора записей с помощь ключевого слова **HAVING.**

Например, если переписать приведенный выше оператор следующим образом:

```
SELECT    Ид_Отд Отдел, count (*) Всего сотрудников
FROM      Отдел_ Сотрудники
GROUP BY      Ид_Отд
HAVING        Ид_Отд <> 1;
```

то первая строка в приведенной выше таблице должна исчезнуть.

Вложенные запросы

Результаты, возвращаемые оператором SELECT**,** можно использовать в другом операторе SELECT**.** Причем это относится и к операторам, возвращающим совокупные характеристики, и к операторам, возвращающим множество значений.

Для того чтобы узнать фамилию самого старого сотрудника, можно воспользоваться вложенным запросом:

```
SELECT    Фамилия, Год_ рожд
FROM      Сотрудник
WHERE     Год_ рожд =
    (SELECT    max (Год_ рожд)
    FROM       Сотрудник);
```

В этом операторе второй вложенный оператор:

```
SELECT    max (Год_рожд)
FROM      Сотрудник
```

возвращает максимальный год рождения, который используется в элементе WHERE основного оператора SELECT для поиска сотрудника (или сотрудников), чей год рождения совпадает с максимальным значением.

Вложенные запросы могут обращаться к разным таблицам. Пусть, например, мы имеем две аналогичных по структуре таблицы СОТРУДНИК и СОВМЕСТИТЕЛЬ, относящиеся к разным организациям, и хотим в таблице СОТРУДНИК найти всех однофамильцев сотрудников из таблицы СОВМЕСТИТЕЛЕЙ. Это можно сделать оператором:

```
SELECT *
FROM Сотрудник
WHERE Фамилия IN
(SELECT    Фамилия
    FROM       Совместители)
```

Вложенный оператор

```
SELECT    Фамилия
FROM      Совместители
```

возвращает множество фамилий из таблицы СОВМЕСТИТЕЛИ, а конструкция WHERE основного оператора SELECT отбирает из фамилий в таблице СОТРУДНИК те, которые имеются во множестве фамилий из таблицы СОВМЕСТИТЕЛИ.

С помощью предложения SELECT можно реализовать любую операцию реляционной алгебры, которые были подробно рассмотрены в первой главе данного учебного пособия.

Селекция (горизонтальное подмножество) таблицы создается из тех ее строк, которые удовлетворяют заданным условиям:

```
SELECT*
FROM    Вид_работ
WHERE  Лекции = 16  AND  Лаб_ занятия  > 8;
```

Проекция (вертикальное подмножество) таблицы создается из указанных ее столбцов (в заданном порядке) с последующим исключением избыточных дубликатов строк:

```
SELECT          DISTINCT Лекции, Практика, Экзамены
FROM    Вид_ работ;
```

Объединение двух таблиц содержит те строки, которые есть либо в первой, либо во второй, либо в обеих таблицах:

```
SELECT Фамилия, Город
FROM    Сотрудник
UNION
SELECT
FROM    Совместитель;
```

Пересечение двух таблиц содержит только те строки, которые есть и в первой, и во второй:

```
SELECT Фамилия, Город
FROM    Сотрудник
WHERE  Фамилия IN
```

(SELECT Фамилия, Город FROM Совместитель);

Разность двух таблиц содержит только те строки, которые есть в первой, но отсутствуют во второй. Пример:

```
SELECT   Фамилия, Город
FROM   Сотрудник
WHERE  Фамилия   NOT IN
       (SELECT   Фамилия, Город
       FROM      Совместитель);
```

Декартово произведение таблиц и различные виды соединений были подробно рассмотрены ранее.

Здесь не приводится описание операции деления [3], которая также может быть реализована предложением SELECT с коррелированными вложенными подзапросами.

Краткое знакомство с возможностями предложения SELECT показало, что с его помощью можно реализовать все реляционные операции. Кроме того, в предложении SELECT выполняются разнообразные вычисления, агрегирование данных, их упорядочение и ряд других операций, позволяющих описать в одном предложении ту работу, для выполнения которой потребовалось бы написать несколько страниц программы на алгоритмических языках Си, Паскаль или на внутренних языках ряда распространенных СУБД.

Значения могут быть помещены и удалены из полей тремя командами языка SQL:

INSERT (ВСТАВИТЬ),

UPDATE (МОДИФИЦИРОВАТЬ),

DELETE (УДАЛИТЬ).

Последние три команды переводят базу данных из одного состояния в другое и потенциально могут привести базу данных в рассогласованное состояние. К сожалению, в ряде СУБД форматы этих команд отличаются друг от друга и от стандарта.

Приложение

Синтаксис оператора *CREATE TABLE* языка Transact-SQL [25].

CREATE TABLE ИМЯ ТАБЛИЦЫ

({<определение столбца>
| имя столбца AS рассчитываемое _ выражение
| <условия _ на _ значения _ таблицы>
} [, …n]
)
[ON { группа | DEFAULT}]
[TEXTIMAGE_ON { группа | DEFAULT}],

где столбцы определяются следующим образом:

<определение столбца> ::= {имя_ столбца тип_ данных }

[NULL | NOT NULL]
[IDENTITY [(начало, приращение)
[NOT FOR REPLICATION]]]
[ROWGUИДСОL]

Условия на значения таблицы и столбца задаются следующим образом:

[<условия _ на _ значения _ столбца:: =
[CONSTRAINT имя _ условия _ на _ значения]

{ { PRIMARY KEY | UNIQUE}}
[CLUSTERED | NONCLUSTERED]
[WITH FILLFACTOR= фактор _ заполнения]
]
[ON { группа | DEFAULT}]
| FOREIGN KEY]
REFERENCES ссылка _ таблица
[(ссылка столбец)]
[NOT FOR REPLICATION]
| DEFAULT константное _ выражение
| CHECK [NOT FOR REPLICATION]
(логическое выражение)
{ […n]

<условия _ на _ значения _ таблицы>:: = [CONSTRAINT имя _ условия _на _значения]
{ [{ PRIMARY KEY | UNIQUE}
[CLUSTERED | NONCLUSTERED]

```
{       (столбец [,…n]) }
[ WITH      FILLFACTOR = фактор_заполнения]
]
[ ON { группа | DEFAULT} ]
]
| FOREIGN KEY
[ (столбец [,…n ] ) ]
REFERENCES ссылка _ таблица [ (ссылка _ столбец) ] [,…n ] ) ]
[ NOT FOR REPLICATION ]
| CHECK [ NOT FOR REPLICATION ]
(условие _ поиска)
}
```

Параметры команды:

 имя таблицы - имя создаваемой таблицы, которое должно быть уникальным и подчиняться правилам для идентификаторов. Длина имени не должна превышать 128 символов, а для временных таблиц (начинающихся с символа #) –116 символов

 ON *группа* – определяет группу файлов, в которых будет храниться таблица. Эта группа должна существовать в базе данных. При указании ключевого слова DEFAULT таблица помещается в группу, определенную по умолчанию

 TEXTIMAGE _ ON *группа* – позволяет определить отдельную группу для столбцов с типами данных text, ntext, или image. Если этот параметр не указан, то такие столбцы размещаются в той же группе, что и вся таблица.

Определение столбцов таблицы:

 имя столбца – уникальное имя столбца таблицы. Этот параметр можно пропустить для столбцов с типом данных timestamp, название которого будет использовано в качестве имени.

 тип данных – один из предопределенных или созданных пользователем типов данных.

 NULL | NOT NULL - ключевые слова, определяющие, разрешено или нет использование в столбце значения NULL.

 IDENTITY - ключевое слово, показывающее, что за формирование значений в этом столбце будет отвечать SQL Server. В этом случае при добавлении в таблицу новой строки SQL Server предоставляет уникальное (в пределах таблицы) значение, которое получается суммированием значения последней загруженной строки с заданным приращением. Начальное значение задается параметром SEED, а приращенное – параметром INCREMENT. Это свойство может быть определено только для одного столбца таблицы, для которого к тому же определен тип данных tinyint, smallint, int, decimal (p,0) или numeric(p,0). Обычно используется совместно с условием на значение PRIMARY REY для автоматического получения уникального идентификатора строки. Однако если для столбца указать ключевое слово NOT FOR REPLICATION, то это свойство не будет действовать, когда строка в таблицу вставляется посредством репликации.

 ROWGUИДCOL – ключевое слово, показывающее, что новый столбец определяет глобальный уникальный идентификатор. Тип данных в этом случае должен быть uniqueIdentifier. И его можно указывать только для одного столбца таблицы.

 имя_ столбца AS - рассчитываемое выражение, которое позволяет определить виртуальный, физически не хранящийся в таблице столбец. В качестве выражения может

выступать имя столбца, константа, функция, переменная или любая их комбинация. Такой столбец имеет некоторые ограничения.

Он не может использоваться в качестве ключевого при построении индекса, и к нему не могут применять условия на значения PRIMARY KEY, UNIQUE, FOREIGN KEY и DEFAULT.

Его нельзя использовать в качестве целевого в инструкциях INSERT и UPDATE.

Остальные параметры позволяют задать для таблицы и отдельно для столбца условия на значения.

Библиографический список

1. Дрибас В.П. Реляционные модели баз данных. – БГУ. Минск. БССР, 1982. – 297 с.

2. Неклюдова Е.А., Цаленко М.Ш. Синтез логической схемы реляционных баз данных. Программирование N 6, 1979. – С. 58 – 68.

3. Дейт. К. Введение в системы баз данных. – Киев-Москва: Диалектика, 1998. – 781 с.

4. Цикритзис Д., Лоховски. Ф. Модели данных. – М.: Финансы и статистика, 1985. – 334 с.

5. Мартин Д. Организация баз данных в вычислительных системах. – М.: Мир, 1989. – 662 с.

6. Воронин Г.П., Копейкин М.В., Осмоловский Л.Г., Петухов О.А. Проектирование объектно-реляционных баз данных. / Под ред. О. А. Петухова. – Л.: Судостроение, 1986. – 180 с.

7. Хаббард Дж. Автоматизированное проектирование баз данных. – М.: Мир, 1984. – 294 с.

8. Копейкин М.В., Спиридонов В.В., Шумова Е.О. Общие принципы построения объектно-реляционной модели данных. М.: – ВИНИТИ N 2929-В96, 1996. – 28 с.

9. Зубов В.С.. CLIPPER & FOXPRO. Практикум пользователя. Изд.2-е, перераб. и дополн. – М.: Информационно-издательский дом "Филинъ", 1996. –. 496 с.

10. Компьютерные сети: Учебный курс. / Пер. с англ. – М.: ТОО Channel Trading, Ltd, 1997. – 696 с.

11. Архипенков С. Я. Oracle Express OLAP. – М.: Диалог МИФИ, 1999. – 465 с.

12. Ален И., Голуб. С и С++. Правила программирования. – М.: Бином, 1996. – 272 с.

13. Мейер М. Теория реляционных баз данных. – М.: Мир, 1987. – 608 с.

14. Ковалевский С.С., Малярский А.Н. Критический анализ организации СУБД на IBM PC. – М.: Мир, 1991. – 242 с.

15. Мамаев Е. MS SQL SERVER 2000. – СПб.: БХВ-Петербург, 2001. – 1280 с.

16. Stonebraker M., Moore D. Object Relational DBMSs: The Next Great. –San Francisco, Wave, Morgan Kufmann Publishers, 1996. – 367 p.

17. Копейкин М.В., Спиридонов В.В., Шумова Е.О. Базы данных. Объектно-реляционный подход. – СПб.: СЗПИ, 1998. – 96 с.

18. Цаленко М.Ш. Моделирование семантики в базах данных. – М.: Наука, 1989. – 287 с.

19. Баженова И. Ю. SQL Windows. SAL – язык приложений баз данных с архитектурой клиент / сервер. – М.: Диалог МИФИ, 1999. – 456 с.

20. СУБД 1/97 (Серверы Баз Данных). – М. 1997 – 1999. Журнал СУБД.

21. Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ. – М.: Мир, 1991. – 252 с.

22. Архангельский А.Я. Программирование в Delpfi 5. – М.: Бином, 2000. – 1070 с.

23. Кириллов В.В. Структурированный язык запросов (SQL): Учеб. пособие. – СПб.: ИТМО, 1994. – 80 с.

24. Ульман Дж. Базы данных на Паскале. – М.: Машиностроение, 1990. – 386 с.

25. Тихомиров Ю. SQL Server 7.0. – СПб.: БХВ-Петербург, 2001. – 720 с.

26. СУБД Cache. Объектно-ориентированная разработка приложений: Учебный курс / В. Кирстен, М. Иринге р, Б. Рериг, П. Шульте. – СПб.: Изд-во Питер, 2001. – 384 с.

27. Мальцев М.Г., Хомоненко А.Д., Цыганков В.М. Базы данных: Учеб. пособие. – СПб.: Изд-во Корона принт, 2002. – 672 с.

28. Кайт Т. Oracle для профессионалов. Книга 1. Архитектура и основные особенности. – СПб.: Изд-во ООО ДиаСофтЮП, 2003. – 672 с.

29. Грачев А.Ю. Введение в СУБД Informix. – М.: Диалог МИФИ, 2000. – 272 с.

30. Копейкин М.В., Спиридонов В.В., Шумова Е.О. Базы данных. Концепция баз данных: Учеб. пособие. – СПб.: СЗТУ, 2004. – 117 с.

31. Копейкин М.В., Спиридонов В.В., Шумова Е.О. Базы данных. Инфологические модели баз данных: Учеб. пособие. – СПб.: СЗТУ, 2004. – 187 с.

32. Дженнингс Р. Access 97 в подлиннике. – СПб.: BHV – Санкт-Петербург, 1999. – 1268 с.

33. Вейскас Д. Эффективная работа с Microsoft Access 97. – СПб.: Питер Ком, 1999. – 976 с.

Предметный указатель

Оглавление

Копейкин Михаил Васильевич
Спиридонов Виктор Валентинович
Шумова Елена Олеговна

БАЗЫ ДАННЫХ

ОСНОВЫ SQL РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Учебное пособие

Редактор И.Н. Садчикова

Сводный темплан 2004 г.
Лицензия ЛР N 020308 от 14.02.97

Санитарно-эпидемиологическое заключение № 78.01.07.953.П.005641.11.03 от 21.11.2003 г.

Подписано в печать Формат 60x84 1/16

Б.кн.-журн. П.л. . Б.л. . РТП РИО СЗТУ

Тираж Заказ

Северо-Западный государственный заочный технический университет
РИО СЗТУ, член Издательско-полиграфической ассоциации
вузов Санкт-Петербурга
191186, Санкт-Петербург, ул. Миллионная, 5