Министерство образования Республики Беларусь

Учреждение образования
«Международный государственный экологический институт
им. А.Д. Сахарова» Белорусского государственного университета

Факультет мониторинга окружающей среды

Кафедра экологических информационных систем

Иванюкович В.А.

РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ И ОСНОВЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ SQL

Рекомендовано научно-методическим советом института в качестве учебно-методического пособия для студентов магистратуры и специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)»

Минск 2017

Рекомендовано к изданию НМС МГЭИ им. А.Д. Сахарова БГУ
(протокол №4 от 20 декабря 2016 г.)

Автор:

В.А. Иванюкович, кандидат физико-математических наук, доцент

Рецензенты:

Заведующий кафедрой управления информационными ресурсами
Академии управления при Президенте Республики Беларусь,
кандидат физико-математических наук, доцент Б.В. Новыш;
доцент кафедры физики и высшей математики Международного государственного экологического института
им. А.Д. Сахарова Белорусского государственного университета, кандидат педагогических наук, доцент Т.Е. Кузьменкова

Иванюкович, В.А.

Реляционная модель данных и основы языка программирования SQL: учебное пособие / В.А. Иванюкович. – Минск: МГЭИ им. А.Д. Сахарова БГУ, 2017. – 55 с.

Приведены основы реляционной алгебры. Даны начальные сведения о синтаксисе и операторах SQL. Материал сопровождается многочисленными примерами реляционных выражений и команд SQL. Предлагаются задачи для самостоятельного решения, которые позволят усвоить учебный материал.

Предназначено для магистрантов всех специальностей института и студентов специальности «Информационные системы и технологии (по направлениям)».

ВВЕДЕНИЕ

Современные специалисты широко используют информационные технологии в своей профессиональной деятельности. При работе с большими объемами информации одной из наиболее востребованных технологий являются реляционные базы данных. Широко распространены и программные средства, предназначенные для управления ими – системы управления базами данных (СУБД). Их популярность подтверждается тем, что СУБД MS Access входит в состав интегрированного программного пакета Microsoft Office, а многие известные бренды свои коммерческие продукты сопровождают свободно распространяемыми версиями с сокращенным функционалом, такими как Microsoft SQL Server Express. Возможности таких продуктов достаточны для удовлетворения большинства потребностей специалистов различных направлений.

Эффективность работы с базами данных во многом определяется умением производить отбор и обработку нужных данных. Возможности программных продуктов в этом плане огромны. Язык программирования баз данных SQL (Structured Query Language) позволяет отбирать данные, хранящиеся в различных таблицах, их обрабатывать и представлять в заданном виде.

Математической основой языка программирования SQL является реляционная алгебра. В пособии кратко представлены ее постулаты и операторы. Разобранные примеры помогут сделать первые шаги в понимании идеи реляционной модели данных.

Материал, предназначенный для освоения основ языка SQL, достаточен для того, чтобы научиться составлять команды на выборку данных, хранящихся в различных таблицах, проводить вычисления и представлять результат в нужном виде.

Знакомство с программированием на языке SQL построено на изучении команд, которые реализуют все основные и некоторые дополнительные операторы реляционной алгебры. Изучение построено на использовании версии Jet 4.0 языка SQL, которая используется в СУБД Access. Пособие не содержит сведения об основных понятиях систем баз данных и правилах их

проектирования. С таким материалом можно познакомиться в курсе лекций по дисциплине «Базы данных».

1 РЕЛЯЦИОННАЯ АЛГЕБРА

Реляционная модель данных и построенные на ее основе программные средства обработки массивов данных является одной из самых успешных современных информационных технологий. В основе ее лежит простая идея – все данные упорядочены и хранятся в двумерных таблицах. Хранящийся в таблице набор данных (строк) можно рассматривать как математическое отношение, арность которого равна количеству столбцов таблицы. При этом наборы данных в строках таблицы соответствуют кортежам множества, на котором построено такое отношение.

Действительно, можно считать, что в таблицу, изображенную на рис.1.1, внесены данные, содержащиеся в четырех множествах – П№ = {П1, П2, П3, П4, П5}, Имя_П = {Бык, Волк, Заяц, Лев, Лиса}, Статус = {10, 20, 30} и Город = {Брест, Гродно, Минск}.

Поставщики

П№	Имя_П	Статус	Город
П1	Волк	20	Брест
П2	Заяц	10	Минск
П3	Лев	30	Гродно
П4	Лиса	20	Минск
П5	Бык	30	Брест

Рис. 1.1. Таблица с данными о поставщиках

В математике *декартовым произведением* (или сокращенно *произведением)* N множеств является множество всех таких упорядоченных наборов из N элементов, которые называются N-арными кортежами, что в каждом из них первый элемент берется из первого множества, второй – из второго и т.д. Подмножество такого множества, кортежи которого удовлетворяют заданному условию, называется N-арным отношением. Следовательно, результатом произведения четырех множеств П№×Имя_П×Статус×Город будет множество, состоящее из 4-арных кортежей, представляющих собой всевозможные упорядоченные наборы элементов наших четырех множеств. Выбрав из этого множества кортежи,

которые соответствуют наборам данных в строках таблицы, мы получим подмножество, которое представляет собой математическое отношение. Если по каким-то причинам отсутствуют данные в некоторых ячейках таблицы, то в соответствующие множества (и в пустые ячейки таблицы) можно добавить элемент, который рассматривается как метка, указывающая на отсутствие данных. В теории баз данных его принято называть Null-значением. Таким образом, наборы данных в строках двумерной таблицы могут рассматриваться как математическое отношение, к которому могут быть применены соответствующие правила. Поэтому в учебном пособии для наглядности мы будем часто использовать термин «таблица» вместо более корректного термина «математическое отношение».

Идея реляционной модели была предложена американским математиком Э. Коддом и подробно представлена в статье E.F. Codd. Relational Completeness of Data Base Sublanguages // Randall J. Rustin (ed.). Data Base Systems, Courant Computer Science Symposia Series 6. – Englewood Cliffs, N.J.: Prentice-Hall, 1972.

Часть реляционной модели, которая связана с применением операторов для обработки отношений, основана на реляционной алгебре. Операторы в реляционной алгебре используют отношения в качестве операндов и возвращают отношения в качестве результата. Это реляционное свойство называется свойством *замкнутости*. Благодаря замкнутости, результат одной операции может использоваться в качестве исходных данных для другой. Это, в свою очередь, дает возможность формировать *вложенные выражения*, т.е., выражения, в которых операнды сами представлены выражениями (по аналогии, в арифметике, благодаря свойству замкнутости, результатом арифметических операций с числами получаем также число, к которому могут быть применены арифметические операторы, то есть, могут быть сформированы вложенные выражения).

Это же свойство замкнутости обусловливает необходимость принятия *правил наследования имен атрибутов* для того, чтобы можно было

предсказывать имена атрибутов на выходе произвольной реляционной операции. Задав такие правила для всех операций, можно гарантировать, что для выражения любой сложности будет вычисляться отношение, имеющее вполне определенный набор атрибутов. Более того, поскольку каждое отношение имеет один или множество потенциальных ключей, то возможны ситуации, когда должны быть известны потенциальные ключи для каждого результирующего отношения, т.е., должны быть приняты *правила наследования потенциальных ключей*.

По определению Е. Кодда, основу реляционной алгебры составляют восемь операторов, которые разделены на две группы – *традиционные и специальные реляционные операторы*. Количество операторов могло быть иным. Например, не все предложенные операторы являются примитивными. Оператор соединения можно заменить двумя другими – умножением и выборкой. С другой стороны, можно добавить новые операторы, если они не будут нарушать требования реляционной алгебры.

Некоторые из операторов реляционной алгебры могут выполняться только при определенных условиях, в отличие от подобных операторов, действующих на множествах. Например, в математике объединение двух множеств – это множество всех элементов, принадлежащих или обоим, или одному из исходных множеств. Объединение множества кортежей в приведенном выше отношении *Поставщики* и множества, например, кортежей в отношении, содержащим данные о свойствах деталей, является множеством, но не является отношением – отношения не могут содержать смесь кортежей разных типов. В реляционной алгебре для объединения требуется, чтобы два исходных отношения имели один и тот же тип или, другими словами, были совместимы по типу. *Отношения совместимы по типу*, если каждое из них имеет одно и то же множество имен атрибутов и соответствующие атрибуты определены на одном и том же домене (имели одинаковый смысл).

1.1 Традиционные реляционные операции

К традиционным операциям над множествами относятся операторы объединения, пересечения, вычитания и произведения. Эти операторы подобны соответствующим операторам алгебры множеств, но имеют некоторые ограничения, которые отражены в определениях.

Объединением двух совместимых по типу отношений A и B (A UNION B) называется отношение с тем же заголовком, как и в отношениях A и B, и с телом, состоящим из множества всех кортежей t, принадлежащих A или B или обоим отношениям. При этом совпадающие кортежи записываются один раз. Пример операции объединения показан на рис. 1.2.

П1

П№	Имя_П	Город
П1	Волк	Брест
П2	Заяц	Минск
П3	Лев	Гродно

П2

П№	Имя_П	Город
П2	Заяц	Минск
П4	Лиса	Минск
П5	Бык	Брест

П1 UNION П2

П№	Имя_П	Город
П1	Волк	Брест
П2	Заяц	Минск
П3	Лев	Гродно
П4	Лиса	Минск
П5	Бык	Брест

Рис. 1.2. Пример объединения двух таблиц

Пересечением двух совместимых по типу отношений A и B (A INTERSECT B) называется отношение с тем же заголовком, как и в отношениях A и B, и с телом, состоящим из множества всех кортежей t, которые принадлежат одновременно обоим отношениям A и B. Пример операции пересечения показан на рис. 1.3.

П1

П№	Имя_П	Город
П1	Волк	Брест
П2	Заяц	Минск
П3	Лев	Гродно
П4	Лиса	Минск
П5	Бык	Брест

П2

П№	Имя_П	Город
П1	Волк	Брест
П3	Лев	Гродно
П4	Лиса	Минск
П7	Зубр	Брест

П1 INTERSECT П2

П№	Имя_П	Город
П1	Волк	Брест
П3	Лев	Гродно
П4	Лиса	Минск

Рис. 1.3. Пример пересечения двух таблиц

Вычитанием двух совместимых по типу отношений A и B (A MINUS B) называется отношение с тем же заголовком, как и в отношениях A и B, и с

телом, состоящим из множества всех кортежей t, принадлежащих отношению A и не принадлежащих отношению B. Пример операции вычитания показан на рис. 1.4.

П1

П№	Имя_П	Город
П1	Волк	Брест
П2	Заяц	Минск
П3	Лев	Гродно
П4	Лиса	Минск
П5	Бык	Брест

П2

П№	Имя_П	Город
П1	Волк	Брест
П3	Лев	Гродно
П4	Лиса	Минск
П7	Зубр	Брест

П1 MINUS П2

П№	Имя_П	Город
П2	Заяц	Минск
П5	Бык	Брест

Рис. 1.4. Пример вычитания двух таблиц

Декартово произведение двух отношений A и B (A TIMES B), где A и B не имеют общих имен атрибутов, определяется как отношение с заголовком, который представляет собой сцепление (конкатенацию) двух заголовков исходных отношений A и B, и телом, состоящим из множества всевозможных кортежей t таких, что t представляет собой сцепление кортежа a, принадлежащего отношению A, и кортежа b, принадлежащего отношению B. Кардинальное число нового отношения равняется произведению кардинальных чисел исходных отношений, а степень равняется сумме их степеней (см. рис. 1.5.).

A
a_1
a_2
a_3

TIMES

B
b_1
b_2

=

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_2	b_2
a_3	b_1
a_3	b_2

Рис. 1.5. Пример реляционной операции декартова умножения

1.2 Специальные реляционные операции

К специальным реляционным операциям относятся выборка, проекция, соединение и деление.

Выборка или *сокращение* (WHERE) – это упрощенное название θ-выборки, где θ обозначает любой скалярный оператор сравнения (=, ≠, ≥, > и т.д.). θ-выборкой из отношения A по атрибутам X и Y (A WHERE XθY) (порядок учитывается!) называется отношение, имеющее тот же заголовок,

что и отношение A, и тело, содержащее множество всех кортежей t отношения A, для которых проверка условия «XθY» дает значение истина. Атрибуты X и Y должны быть определены на одном и том же домене, а оператор сравнения θ должен иметь смысл для данного домена. Пример выборки из таблицы *Поставщики* (см. рис. .1.1) по условию *город='Минск' OR город='Гродно'* показан на рис. 1.6:

Поставщики

П№	**Имя_П**	**Статус**	**Город**
П2	Заяц	10	Минск
П3	Лев	30	Гродно
П4	Лиса	20	Минск

Рис. 1.6. Пример действия реляционной операции выборки:
Поставщики WHERE город='Минск' OR город='Гродно'

Проекцией (PROJECT) отношения A по атрибутам X, Y, …, Z, где каждый из атрибутов принадлежит отношению A, называется отношение с заголовком {X,Y,…,Z} и телом, содержащим множество кортежей с атрибутами, совпадающими с соответствующими атрибутами отношения A. Т.е., с помощью операции проекции получается вертикальное подмножество исходного отношения – подмножество, получаемое исключением всех атрибутов, не указанных в списке атрибутов, и последующим исключением дублирующих кортежей из того, что осталось. Пример проекции из таблицы Поставщики по атрибутам Имя_П и Город показан на рис. 1.7 (обозначение проекции: `Поставщики [Имя_П, Город]`).

Поставщики [Имя_П, Город]

Имя_П	**Город**
Волк	Брест
Заяц	Минск
Лев	Гродно
Лиса	Минск
Бык	Брест

Рис. 1.7. Пример проекции из таблицы Поставщики по атрибутам
Имя_П, Город

Соединение (JOIN) – это разновидность операции произведения, в которой сцепление кортежей основывается на задаваемом атрибуте или наборе атрибутов каждого из двух отношений. Значения указанных атрибутов сравниваются с целью наложения определенных ограничений на результат. Отношения можно соединять по атрибутам, имеющим либо общие домены, либо сопоставимые домены, когда значения данных из одного домена можно сравнить со значениями данных из другого домена.

Наиболее часто используется естественное или внутреннее соединение, когда отношения имеют общий атрибут и результат содержит только строки, в которых значения общего атрибута совпадают.

Естественным (внутренним) соединением отношений A и B (A JOIN B) с заголовками X,Y и Y,Z соответственно и с атрибутами Y, определенными на одном и том же домене, называется отношение с заголовком {X,Y,Z} и телом, содержащим множество кортежей с атрибутами, совпадающими с соответствующими атрибутами отношений A и B. Пример естественного соединения отношений Поставки и Поставщики показан на рис. 1.8.

Отношения имеют общий атрибут П№. В результирующем отношении присутствуют соединения кортежей, в которых значения общего атрибута совпадают. Таким образом, благодаря соединению в одном кортеже собраны данные о поставке и поставщике, осуществившим эту поставку. При этом строки, не имеющие одинаковых значений в общем атрибуте, в результирующее отношение не вошли (строки с П№=’П5’ из таблицы Поставки и строка с П№=’П7’ из таблицы Поставщики.

В случае внешнего соединения (Поставки OUTER JOIN Поставщики) кортеж, который невозможно соединить с кортежем соответствующей таблицы из-за отсутствия совпадающих значений, будет помещен в результирующую таблицу, а для присоединенных атрибутов значения определены не будут, т.е., им присвоят Null-значения (рис. 1.9).

Поставки

П№	Д№	Пр№	Кол
П1	Д1	Пр1	200
П1	Д1	Пр4	700
П2	Д3	Пр1	400
П2	Д3	Пр2	200
П2	Д3	ПР7	800
П2	Д5	ПР2	100
П3	Д3	ПР1	200
П3	Д4	ПР2	500
П4	Д6	ПР3	300
П4	Д6	ПР7	300
П5	Д2	ПР2	200
П5	Д6	ПР2	200
П5	Д1	ПР4	100

Поставщики

П№	Имя_П	Статус	Гор
П1	Волк	20	Брест
П2	Заяц	10	Минск
П3	Лев	30	Гродно
П4	Лиса	20	Минск
П7	Бык	30	Брест

Табл1 = Поставки JOIN Поставщики

П№	Д№	Пр№	Кол	Имя_П	Статус	Гор
П1	Д1	Пр1	200	Волк	20	Брест
П1	Д1	Пр4	700	Волк	20	Брест
П2	Д3	Пр1	400	Заяц	10	Минск
П2	Д3	Пр2	200	Заяц	10	Минск
П2	Д3	ПР7	800	Заяц	10	Минск
П2	Д5	ПР2	100	Заяц	10	Минск
П3	Д3	ПР1	200	Лев	30	Гродно
П3	Д4	ПР2	500	Лев	30	Гродно
П4	Д6	ПР3	300	Лиса	20	Минск
П4	Д6	ПР7	300	Лиса	20	Минск

Рис. 1.8. Результат естественного (внутреннего) соединения таблиц

Большинство СУБД используют также разновидности внешнего соединения с операторами SQL – LEFT JOIN и RIGHT JOIN.

Соединение обладает свойствами ассоциативности и коммутативности. Если отношения A и B не имеют общих имен атрибутов, то естественное соединение превращается в декартово произведение.

Поставки

П№	Д№	Пр№	Кол
П1	Д1	Пр1	200
П1	Д1	Пр4	700
П2	Д3	Пр1	400
П2	Д3	Пр2	200
П2	Д3	ПР7	800
П2	Д5	ПР2	100
П3	Д3	ПР1	200
П3	Д4	ПР2	500
П4	Д6	ПР3	300
П4	Д6	ПР7	300
П5	Д2	ПР2	200
П5	Д6	ПР2	200
П5	Д1	ПР4	100

Поставщики

П№	Имя_П	Статус	Гор
П1	Волк	20	Брест
П2	Заяц	10	Минск
П3	Лев	30	Гродно
П4	Лиса	20	Минск
П7	Бык	30	Брест

Тавл1 = Поставки OUTER JOIN Поставщики

П№	Д№	Пр№	Кол	Имя_П	Статус	Гор
П1	Д1	Пр1	200	Волк	20	Брест
П1	Д1	Пр4	700	Волк	20	Брест
П2	Д3	Пр1	400	Заяц	10	Минск
П2	Д3	Пр2	200	Заяц	10	Минск
П2	Д3	ПР7	800	Заяц	10	Минск
П2	Д5	ПР2	100	Заяц	10	Минск
П3	Д3	ПР1	200	Лев	30	Гродно
П3	Д4	ПР2	500	Лев	30	Гродно
П4	Д6	ПР3	300	Лиса	20	Минск
П4	Д6	ПР7	300	Лиса	20	Минск
П5	Д2	ПР2	200			
П5	Д6	ПР2	200			
П5	Д1	ПР4	100			
П7				Бык	30	Брест

Рис. 1.9. Результат внешнего соединения таблиц Поставки и Поставщики

Результатом *Деления* (DIVIDED BY) двух отношений, бинарного и унарного, является отношение, содержащее все значения одного атрибута бинарного отношения, которые сцеплены в другом атрибуте со всеми значениями в унарном отношении.

На рис. 1.10 показаны два примера деления отношений.

П№	Д№
П1	Д1
П1	Д2
П1	Д4
П2	Д1
П2	Д2

DIVIDED BY

Д№
Д2
Д4

=

П№
П1
П4

П№	Д№
П3	Д2
П4	Д1
П4	Д2
П4	Д4

DIVIDED BY

Д№
Д1

=

П№
П1
П2
П4

Рис. 1.10. Примеры деления бинарного отношения на унарное.

Приведенное выше определение *операции деления* можно развить для случая деления n-арного отношения на m-арное отношение (n>m), в результате которого получается (n–m)-арное отношение.

1.3 Дополнительные реляционные операции

Итак, Е.Кодд предложил восемь операторов для построения реляционных выражений. Традиционные операторы соответствуют операциям на множествах (с некоторыми ограничениями, которые учитывают специфику математических отношений), специальные операторы предложены для решения специфических задач, связанных с обработкой табличных данных. Однако могут возникать ситуации, когда этих операторов недостаточно для построения реляционного выражения, описывающего алгоритм обработки данных. В таких случаях предлагается создавать и использовать дополнительные операторы, действия которых не противоречат постулатам реляционной алгебры. Например, необходимо перемножить два отношения, имеющих одинаковые атрибуты, что запрещено правилами декартового произведения. Можно создать и применить оператор переименования атрибутов (Е. Кодд назвал его RENAME) и после этого произведение отношений будет возможно:

```
Поставки TIMES (Поставщики RENAME П№ AS П1№).
```

Описанные выше операторы реляционной алгебры не содержат средств для скалярных вычислений. Например, надо найти стоимость поставок деталей Д1, если одна деталь стоит 235 рублей. Для обеспечения таких возможностей предлагается *операция расширения* EXTEND. С помощью этой операции создается новое отношение, похожее на исходное, но содержащее дополнительный атрибут, значения которого получены посредством некоторых скалярных вычислений:

```
Extend A add expr as z.
```

Результатом этого выражения будет отношение с заголовком, эквивалентным заголовку отношения A, дополненному новым атрибутом z, который рассчитывается скалярным выражением expr для каждого кортежа отношения A. При этом отношение A не должно иметь атрибута z и выражение expr не должно ссылаться на атрибут z. Кардинальное число нового отношения равно кардинальному числу отношения A, степень равна степени отношения A плюс единица.

Сейчас, применив реляционное выражение

```
EXTEND (Поставки WHERE Д№='Д1') ADD Кол*235 AS сумма,
```

получим таблицу поставок деталей Д1, содержащую в поле «сумма» стоимость каждой поставки (рис. 1.11):

П№	**Д№**	**Пр№**	**Кол**	**сумма**
П1	Д1	Пр1	200	47000
П1	Д1	Пр4	700	164500
П5	Д1	ПР4	100	23500

Рис. 1.11. Пример действия оператора EXTEND

Пример более сложного реляционного выражения:

Подсчитать количество поставок, сделанных каждым поставщиком.

```
EXTEND Поставщики ADD COUNT ((Поставки RENAME П№ AS X)
  WHERE X= П№) AS Кол_П;
```

Результат действия этого выражения показан на рис. 1.12. Изменив в таблице Поставки название атрибута П№ на X, мы получили возможность для каждого поставщика из таблицы Поставщики отбирать соответствующие

ему строки в таблице Поставки ((Поставки RENAME П№ AS X) WHERE X=П№) и при помощи итоговой функции COUNT( ) подсчитывать их количество, которое равно числу поставок. Результат записывается в поле Кол_П, которое оператор EXTEND добавляет к таблице Поставщики.

П№	**Имя_П**	**Статус**	**Гор**	**Кол_П**
П1	Волк	20	Брест	6
П2	Заяц	10	Минск	2
П3	Лев	30	Гродно	1
П4	Лиса	20	Минск	3
П5	Бык	30	Брест	0

Рис. 1.12. Пример действия оператора EXTEND

Таким образом, операция расширения обеспечивает возможность горизонтальных или построчных, осуществляемых в пределах кортежа, вычислений.

Итоговые (или статистические) функции – это функции, которые в качестве аргумента используют множество значений и в результате возвращают одно значение. Кроме упомянутой ранее функции COUNT (подсчет строк в таблице), итоговыми функциями являются SUM (сумма), AVG (среднее значение), MAX (максимальное значение), MIN (минимальное значение), StDev (среднеквадратичное отклонение от среднего значения поля), Var (дисперсия значений поля) и некоторые другие. Если аргумент такой функции будет пустым множеством, то функции COUNT и SUM возвращают нуль, функции MAX и MIN возвращают максимальное и минимальное значения соответствующего домена.

Итоговые функции используются, как правило, для обработки группы строк или всех строк таблицы – для так называемых вертикальных вычислений. Для таких случаев предложен оператор агрегирования (или подведения итогов) SUMMARIZE:

```
SUMMARIZE A BY (a1,a2,…,an) ADD expr AS Z,
```

где a1, a2, …, an – отдельные атрибуты отношения A. Результатом этого выражения будет отношение с заголовком {a1, a2, …, an, z} и с телом,

содержащим все такие кортежи t, которые являются кортежами проекции отношения A по атрибутам a1, a2, …, an, расширенного значением для нового атрибута z. Такое новое значение z подсчитывается вычислением итогового значения expr по всем кортежам отношения A, которые имеют те же самые значения для атрибутов a1, a2, …, an, что и кортеж t.

Другими словами, оператор SUMMARIZE собирает в группы кортежи отношения A, имеющие одинаковые значения наборов атрибутов a1, a2, …, an и для них проводит вычисления по формуле expr. В результирующей таблице для каждого набора a1, a2, …, an приводится результат вычислений expr, записанный в поле z.

Кардинальное число результирующего отношения равно кардинальному числу проекции отношения A по атрибутам a1, a2, …, an, а степень равна степени такой проекции плюс единица. Например, рассчитанное в прошлом примере количество поставок, сделанных каждым поставщиком, может быть получено при помощи выражения

```
SUMMARIZE Поставки BY(П№) ADD COUNT AS Кол_П.
```

В отличие от результата действия оператора EXTEND, примененного в предыдущем решении, в этом случае результат вычислений не содержит сведений о поставках поставщика П5(рис. 1.13). Причина состоит в том, что поставщика П5 нет в отношении Поставки.

П№	**Кол_П**
П1	6
П2	2
П3	1
П4	3

Рис. 1.13.

Если группировка кортежей производится по пустому множеству атрибутов, то вычисление выполняется по всем кортежам отношения и итоговая функция выдает единственный результат для всей таблицы. Например, результатом действия реляционного выражения

```
SUMMARIZE Поставки BY( ) ADD COUNT AS Кол_П
```

будет таблица, содержащая одну ячейку с именем атрибута Кол_П.

Популярными дополнительными реляционными операциями являются также:

– операции реляционного присвоения, которые дают возможность «запоминать» результаты действия некоторых алгебраических выражений в базе данных и таким образом изменять состояние базы данных или, иначе говоря, обновлять базу данных, например:

```
Поставки := Поставки MINUS (Поставки WHERE Кол = 0);
```

– операции вставки:

```
INSERT (Поставщики WHERE Гор_П = Минск) INTO Temp;
```

(здесь выбранные данные вставлены в отношение Temp, которое должно быть совместимым по типу с отношением Поставщики);

– операции обновления данных:

```
UPDATE (Поставщики WHERE Гор_П = Брест) СТАТУС := 40;
```

– операции удаления данных:

```
DELETE Поставщики WHERE Статус < 20.
```

Подобные операции действуют на уровне множеств (например, операция DELETE удаляет множество кортежей из целевого отношения) и должны контролироваться с помощью предиката рассматриваемого отношения.

Дополнительные реляционные операции не ограничиваются приведенным здесь списком. Ранее говорилось о том, что при необходимости можно создавать новые реляционные операторы. Как правило, авторами предлагаются составные, построенные на восьми операторах, предложенных Е. Коддом. Например, оператор полусоединения A SEMIJOIN B, который после соединения таблиц A и B выполняет проекцию по атрибутам таблицы A, тем самым такой оператор удаляет из таблицы A кортежи, у которых значение атрибута соединения отсутствует в соответствующем атрибуте таблицы B. Очевидно, такие операторы всего лишь сокращают реляционное выражение.

1.4 Примеры использования реляционной алгебры

Для демонстрации примеров будем использовать базу данных Поставщики–Детали–Проекты, которая была предложена Дж.Дейтом, и широко используется для изучения систем баз данных. Во всех примерах этой главы использовались фрагменты подобной базы данных. Более того, при изучении основ языка программирования SQL, которому посвящена глава 2 настоящего пособия, также будет использоваться эта база данных.

Схема базы данных и таблицы, содержащие учебные данные, приведены на рис. 1.14 и рис. 1.15 соответственно.

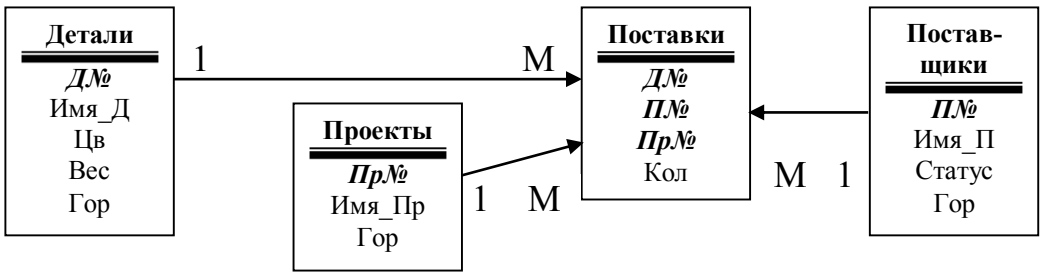


Рис. 1.14. Схема базы данных Поставщики–Детали–Проекты

Процесс проектирования баз данных, включающий анализ предметной области, разработку инфологической модели и проверку отношений на соответствие требованиям нормальных форм, подробно рассмотрен в курсе лекций по дисциплине «Базы данных».

Поставки

П№	Д№	Пр№	Кол
П1	Д1	Пр1	200
П1	Д1	Пр4	700
П2	Д3	Пр1	400
П2	Д3	Пр2	200
П2	Д3	ПР3	200
П2	Д3	ПР4	500
П2	Д3	ПР5	600
П2	Д3	ПР6	400
П2	Д3	ПР7	800
П2	Д5	ПР2	100
П3	Д3	ПР1	200
П3	Д4	ПР2	500
П4	Д6	ПР3	300
П4	Д6	ПР7	300
П5	Д2	ПР2	200
П5	Д2	ПР4	100
П5	Д5	ПР5	500
П5	Д5	ПР7	100
П5	Д6	ПР2	200
П5	Д1	ПР4	100
П5	Д3	ПР4	200
П5	Д4	ПР4	800
П5	Д5	ПР4	400
П5	Д6	ПР4	500

Детали

Д№	Имя_Д	Цв	Вес	Гор
Д1	Тестер	Черный	250	Минск
Д2	Дозиметр	Серый	700	Борисов
Д3	Радиометр	Черный	1400	Гродно
Д4	Часы	Желтый	140	Минск
Д5	Рулетка	Красный	200	Брест
Д6	Лом	Черный	5000	Варшава

Поставщики

П№	Имя_П	Статус	Гор
П1	Волк	20	Брест
П2	Заяц	10	Минск
П3	Лев	30	Гродно
П4	Лиса	20	Минск
П5	Бык	30	Брест

Проекты

Пр№	Имя_П	Гор
Пр1	Спутник	Минск
Пр2	Корунд	Минск
Пр3	Лес	Брест
Пр4	Шина	Бобруйск
Пр5	Азот	Гродно
Пр6	Электро	Молодечно
Пр7	Кристалл	Пинск

Рис. 1.15. База данных Поставщики–Детали–Проекты

Примеры

1. Получить имена поставщиков, которые поставляют деталь Д2.

Требуется построить таблицу с одним полем Имя_П, используя таблицы Поставки и Поставщики:

```
((Поставки JOIN Поставщики) WHERE Д№='Д2') [Имя_П];
```

2. Получить имена поставщиков, которые поставляют по крайней мере одну черную деталь.

Здесь также надо построить унарную таблицу с атрибутом Имя_П, но необходимые нам сведения находятся в трех таблицах – Детали, Поставки и Поставщики:

```
(((Детали WHERE Цв = 'Черный') JOIN Поставки) JOIN
Поставщики) [Имя_П]
```

или

```
(((Детали WHERE Цв = 'Черный') [Д№] JOIN Поставки) [П№]
JOIN Поставщики) [Имя_П].
```

Во втором выражении предлагается делать проекции после операций выборки и соединения, уменьшая при этом степень промежуточных таблиц. Благодаря этому повышается эффективность алгоритма.

3. Получить имена поставщиков, которые поставляют все детали.

Наиболее простое решение этой задачи получаем при использовании реляционной операции деления:

```
((Поставки [П№,Д№] DIVIDED BY Детали [Д№]) JOIN Поставщики)
[Имя_П]
```

Разделив бинарную таблицу Поставки [П№,Д№] на унарную Детали [Д№], получим унарную таблицу с атрибутом П№ и телом, содержащим кортежи с теми номерами поставщиков П№, которые в бинарной таблице сцеплены со всеми значениями атрибута Д№ в таблице Детали [Д№].

4. Получить номера поставщиков, которые поставляют по крайней мере все те детали, которые поставляет поставщик П2.

Алгоритм решения похож на алгоритм прошлой задачи:

```
Поставки [П№,Д№] DIVIDEDBY (Поставки WHERE Имя_П='П2') [Д№]
```

5. Получить имена поставщиков, которые не поставляют деталь Д2.

В этой задаче можно легко найти номера поставщиков, поставляющих деталь д2, и вычесть их из таблицы, содержащей все номера поставщиков:

```
((поставщики [п№] MINUS (поставки WHERE д№='д2') [п№])
JOIN (поставщики) [имя_п]
```

6. Более сложный пример: Получить номера поставщиков, у которых максимальное значение поставок как минимум в 2 раза превышает среднее значение поставок деталей остальными поставщиками.

Итак, требуется построить унарную таблицу с атрибутом П№. Вся необходимая информация находится в таблице Поставки. Выбрать нужных поставщиков мы сможем, если построим таблицу, содержащую для каждого поставщика данные о величине его максимальной поставки и о средних поставках всех остальных. Тогда операцией выборки сможем отобрать поставщиков, у которых первая величина более чем в 2 раза превышает вторую.

На первом шаге строим таблицу с данными о максимальных поставках каждого поставщика:

```
SUMMARIZE поставки BY(П№) ADD MAX(Кол) AS m
```

Получили бинарную таблицу с атрибутами П№ и m. Далее добавим в эту таблицу новый атрибут, в котором для каждого поставщика будут находиться значения средних поставок всех остальных поставщиков. Для этого удобно использовать оператор расширения EXTEND:

```
EXTEND (SUMMARIZE поставки BY(п№) ADD MAX(Кол) AS m) ADD
AVG(((поставки RENAME П№ AS x) WHERE x<>П№) [кол]) AS ср
```

Далее в полученной таблице выбираем строки, в которых m>=2*ср и делаем проекцию по полю п№:

```
(EXTEND (SUMMARIZE поставки BY(п№) ADD MAX(кол) AS m) ADD
AVG(((поставки RENAME п№ AS x) WHERE x<>п№) [кол]) AS ср)
WHERE m>=2*ср)[п№]
```

Полученное реляционное выражение позволяет построить таблицу, удовлетворяющую условию задачи.

Задачу можно усложнить, если потребовать найти поставщиков, у которых максимальное (или минимальное) значение поставок *меньше* среднего. Если использовать предложенный алгоритм, то мы потеряем поставщиков, которые не поставляли детали. Решите такую задачу самостоятельно.

Итак, в главе очень кратко изложены основы реляционной алгебры. Современные системы управления базами данных построены, в основном, на реляционной модели данных и алгебра является фундаментом всех алгоритмов обработки данных. Для работы с данными используется язык программирования баз данных SQL, знакомству с которым посвящена следующая глава.

2 Основы SQL

2.1 Некоторые особенности языка баз данных SQL

Аббревиатура SQL расшифровывается как Structured Query Language – язык структурированных запросов. Он включает операторы, предназначенные для выполнения почти всех необходимых действий с объектами реляционных баз данных – создание структур хранения данных (таблицы, связи и т.п.), организации доступа к данным (права администраторов, пользователей и т.п.), манипулирования данными (поиск, вычисления, изменение, удаление и т.п.). Системы управления базами данных (СУБД) предоставляют среду для использования операторов SQL в интерактивном режиме. Их можно использовать и в программах, написанных на сценарных (VBA, VBScript, PHP) или универсальных (C#, Java) языках.

Язык SQL был разработан фирмой IBM и в 1992 году был принят Американским национальным институтом стандартов (ANSI) в качестве национального стандарта США. Он дает возможность использовать общий набор операторов в любой SQL-совместимой программе управления базами данных. В последующие годы стандарт был существенно дополнен, особенно в связи с бурным развитием web-технологий. Стандарт разнообразен и гибок и позволяет создавать разновидности языка программирования. В пособии приведены учебные материалы для знакомства с версией языка Jet SQL, используемой в СУБД MS Access.

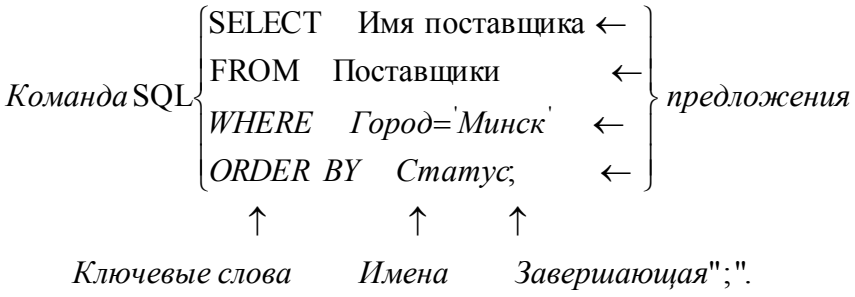
Мы познакомимся с синтаксисом и конструкцией языка SQL и научимся писать запросы для создания таблиц, поиска данных и их обработки.

Все ключевые слова стандарта ANSI SQL, поддерживаемые модификацией Jet SQL, а также зарезервированные уникальные слова, можно найти в справочных пособиях, в том числе и в системе помощи СУБД MS.

Команда SQL – это последовательность заданных операторами действий, выполненных в определенном порядке.

Синтаксис языка включает слова и символы, которые можно применять, а также правила, по которым эти слова и символы можно использовать при создании команд и программ. Структура команды SQL показана на следующем примере (мы продолжаем использовать базу данных Проекты-Поставщики-Детали, приведенную в конце первой главы):

Найти поставщиков из Минска.



Название команды SQL соответствует ее назначению и совпадает с определяющим ее ключевым словом – команда SELECT, команда CREATE TABLE и т.п. Предложение – это фрагмент команды SQL, который начинается с ключевого слова, такого как FROM, WHERE и др. и может содержать имена таблиц, полей, другие ключевые слова, выражения (формулы) или константы. Предложения могут быть обязательными, например, предложение FROM в команде SELECT, или необязательными (предложения WHERE или ORDER BY), без которых команда может быть выполнена.

Ключевые слова – операторы, условия, модификаторы, итоговые функции – зарезервированы в SQL и имеют определенное назначение, которое строго регламентировано.

Имена или идентификаторы – это слова, которые применяют для именования объектов базы данных, в том числе таблиц, столбцов, псевдонимов и представлений. Они не могут совпадать с ключевым словом и их длина ограничена 128 символами.

Синтаксис SQL предусматривает определенное назначение знаков пунктуации:

- запятые используются для разделения компонентов списка параметров (например, при перечислении имен полей в предложении SELECT);
- точки используются для отделения имен таблиц от имен полей при записи полного имени поля (Поставщики.Гор);
- точка с запятой ставится в конце команды SQL;
- квадратные скобки используются для записи имен полей, если в них используются ключевые поля, пробелы или другие знаки пунктуации, не разрешенные в SQL;
- одинарная кавычка применяется для описания строчных переменных;
- при использовании оператора LIKE в SQL Jet для маскирования части слова или одного знака используются символы “*“ и “?“.

Необходимо помнить, что в разных версиях языка SQL имеются различия в использовании знаков пунктуации. Поэтому при использовании новой СУБД приходится часто обращаться к справочникам.

SQL – непроцедурный, или декларативный язык. В нем нужно описывать то, что именно вы хотите сделать, а не то, как вы собираетесь это делать (в отличие от процедурных языков C, Perl, Pascal и др.). Оптимизатор, который входит в состав программного обеспечения СУБД, самостоятельно компилирует команду SQL в процедурный код, предварительно выполнив ее оптимизацию.

Язык SQL может быть встроенным (когда команды SQL как элементы включаются в другие программы, написанные на процедурных языках общего назначения C, Java, Pascal… или языках сценариев VBA, Perl, PHP, Python…) и/или интерактивным.

Итак, SQL – это декларативный язык баз данных, с помощью которого можно создавать базы данных, манипулировать данными и обеспечивать их целостность и безопасность. На основании этого операторы SQL можно классифицировать на три основных типа:

Операторы определения данных – определяют содержимое реляционной базы данных в виде таблиц, представлений, связей;

Операторы манипулирования данных – используются для извлечения, вставки, обновления и удаления данных, содержащихся в таблицах и представлениях, а также выполнения вычислений;

Операторы управления данными – ограничивают доступ к данным для разных пользователей.

2.2 Типы данных

Каждому столбцу таблицы (атрибуту отношения) присваивается определенный тип данных, которые могут в нем храниться. В SQL наиболее часто используются следующие категории типов данных:

- Exact numeric – рациональные (целые и действительные) числа;
- Approximate numeric – вещественные числа (с плавающей точкой);
- Character string – строки символов;
- Bit string – строки битов;
- Date time – значения даты и времени.

Точные числовые типы данных

Точные числовые значения могут быть положительными, отрицательными и нулем, целыми или рациональными (конечная десятичная дробь). Характеризуются фиксированными значениями точности и масштаба. Точность – число значащих цифр в записи числа (т.е. общее количество цифр в десятичной записи без учета десятичной точки). Масштаб – число цифр справа от десятичной точки (масштаб ≤ точности).

Типы:

- tinyint – для хранения целых чисел в интервале от 0 до +255, занимает в памяти 1 байт;
- smallint – для хранения целых чисел в интервале от –32768 до +32768, занимает в памяти 2 байта;
- integer (или int) – для хранения целых чисел в интервале от -2^{31} до $+2^{31}-1$, занимает в памяти 4 байта;
- bigint – для хранения целых чисел в интервале от -2^{63} до $+2^{63}-1$ и занимает в памяти 8 байт;

- bit – допустимые значения 0 и 1;
- numeric (точность [,масштаб]) – представляет произвольное рациональное число, значение точности – до 38 разрядов, т.е., интервал задаваемых чисел от $-10^{38}+1$ до $+10^{38}+1$
- decimal – аналогичен numeric, но только задает нижнюю границу точности, т.е. СУБД может выбрать б*о*льшую точность, чем заказано пользователем.

  В СУБД Access: Decimal, Integer, byte, long integer.

  Пример: Хранение числа 123,55:

Спецификация столбца	Хранится значение
Numeric (5)	124
Numeric (5.0)	124
Numeric (5.1)	123,6
Numeric (5.2)	123,55
Numeric (4.0)	124
Numeric (4.1)	123,6
Numeric (4.2)	Выходит за пределы точности

Вещественные числовые типы данных

Вещественные или числа с плавающей точкой применяются для хранения приближенных числовых значений.

FLOAT (точность) – представляет произвольное приближение вещественного числа с плавающей точкой. Значение точности представляется не в количестве значащих десятичных цифр, а в количестве битов. Точность не должна быть меньше 1. Максимальная точность зависит от СУБД. Для преобразования десятеричной точности в бинарную надо умножить десятеричную точность на 3.32193. Например, 7 знаков точности дают 24 бита.

В СУБД MS SQL Server используется два значения точности – 24 и 53. При задании значения точности от 1 до 24 диапазон значений от $-3{,}4\times10^{38}$ до $-1{,}18\times10^{-38}$; 0; и от $1{,}18\times10^{-38}$ до $3{,}4\times10^{38}$. При точности от 25 до 53 диапазон значений от $-1{,}79\times10^{308}$ до $-2{,}23\times10^{-308}$; 0; и от $2{,}23\times10^{-308}$ до $1{,}79\times10^{308}$.

REAL совпадает с FLOAT(24), но точность вводить не надо, ее автоматически определяет СУБД. Числа типа REAL называют числами одинарной точности с плавающей точкой.

DOUBLE PRECISION – вдвое превышает точность REAL – числа двойной точности с плавающей точкой.

В СУБД Access используют вещественные одинарные и двойные типы данных.

Строковые типы данных

Character (n) – строка фиксированной длины n. Максимальная длина строки определяется конкретной СУБД. Если символов меньше чем n, то добавляются пробелы. Синонимы – char(n). В Access используется тип данных текстовый размером до 256 символов.

Character varying (n) – строка переменной длины, длинной менее n. Максимальное n зависит от СУБД. Синонимы: Char varying, Charvar (в Access – MEMO).

National Character (National Char, NChar) – соответствует типу Char, только хранит лишь стандартизованные двухбайтовые знаки (Unicode). National Character Varying – то же для строк переменной длины.

Unicode – единое множество 16-разрядных чисел, которое представляет знаки почти всех мировых языков. Содержит 65536 = 2^{16} знаков.

Битовые типы данных

В таблицах часто необходимо хранить последовательности битов различного назначения, например, аудио- или видео-данные, файлы pdf, doc и т.п. Для этого используется битовый или бинарный тип данных.

BIT(n) – строка фиксированной длины (фиксированные числа битов). Максимальная длина определяется СУБД. Если длина строки меньше n, то возвращается сообщение об ошибке. В строке BIT перед первой кавычкой должна стоять латинская B, например, B'01001' – это строка типа BIT(5). Bit varying задает бинарный тип данных переменной длины.

В СУБД Access к битовому типу данных относятся: YES, NO, BINARY, OLE OBJECT.

Календарные типы данных

Используются для отображения даты и времени суток.

- date – имеет формат YYYY-MM-DD, точность 1 день.
- time(p) – имеет формат HH:MM:SS.NNN. Параметр p задает точность для долей секунды ($0 \leq p \leq 7$).
- datetime – имеется несколько вариантов полного формата YYYY-MM-DD_ HH:MM:SS.

В СУБД Access формат дата/время имеет несколько модификаций. В полях с такими форматами удобно использовать маски ввода, которые обеспечивают единообразие ввода даты и времени.

Значения NULL

Это метка, которая указывает на то, что данные в ячейке отсутствуют по какой-то причине. Это не 0, не пустая строка (′ ′) и не строка пробелов. Метка NULL не принадлежит ни к какому типу данных и ее можно записать в любой столбец кроме тех, для которых задано ограничение NOT NULL.

Значения NULL можно найти и идентифицировать оператором IS NULL.

Значения NULL не равны друг другу. Поэтому нельзя определить, соответствует ли какое-нибудь значение NULL другому значению в базе данных. Тем не менее, оператор DISTINCT воспринимает все NULL одинаково для удаления повторяющихся строк. Оператор GROUP BY также не различает значения NULL при группировке строк.

При сортировке значения NULL в разных СУБД будут либо больше, либо меньше любых значений.

Значения NULL размножаются в процессе вычислений, но итоговые функции игнорируют NULL в процессе вычислений.

2.3 Создание и обслуживание таблиц

Базовые отношения создаются оператором CREATE TABLE. Следует указать: имя таблицы, имена столбцов, типы данных для столбцов и ограничения на данные, хранящиеся в столбце. Рекомендуется описание каждого столбца начинать с новой строки (не обязательно). Некоторые часто используемые ограничения на элементы столбца:

- NOT NULL – не разрешает присваивать значения NULL;
- DEFAULT – задает значения по умолчанию;
- PRIMARY KEY – задает первичный ключ для таблицы;
- FOREIGN KEY – задает внешний ключ;
- REFERENCES – задает внешний ключ и устанавливает связи между таблицами;
- UNIQUE – не позволяет вводить в столбец повторяющиеся значения;
- CHECK – ограничивает с помощью логических выражений значения, которые могут добавляться в столбец.

Пример:

```
CREATE TABLE Проекты
  (Пр№ CHAR(3)     PRIMARY KEY,
  Имя_Пр CHAR(15)  NOT NULL,
  Гор CHAR(20));
```

Внешний ключ создается ключевыми словами FOREIGN KEY или REFERENCES при описании столбца в команде CREATE TABLE:

```
CREATE TABLE Поставки
  (П№ CHAR(3) REFERENCES Поставщики,
  Пр№ CHAR(5) REFERENCES Проекты,
  Д№ CHAR(3)  REFERENCES Детали,
  Кол INTEGER DEFAULT '???',
  PRIMARY KEY (П№, Пр№, Д№));
```

Для ограничений полезно вводить имена. Это поможет ориентироваться в комментариях, которые генерируются программой, и упростит редактирование команд:

```
[CONSTRAINT имя_ограничения] REFERENCES имя_табл
  [имя_столбца].
```

Сохранение целостности данных по ссылкам организуется при помощи операторов RESTRICT, CASCADE и SET NULL, например:

```
CREATE TABLE Поставки
  (П№ CHAR(3) REFERENCES Поставщики ON UPDATE CASCADE
        ON DELETE SET NULL,
  Пр№ CHAR(5) REFERENCES Проекты RESTRICT,
  Д№ CHAR(3) REFERENCES Детали,
  Кол INTEGER DEFAULT '???',
  CONSTRAINT Ключ PRIMARY KEY (П№, Пр№, Д№));
```

Сейчас при изменении П№ в таблице Поставщики будет изменен и П№ в таблице Поставки, а при его удалении в таблицу Поставки будет внесено значение NULL. Оператор RESTRICT устанавливает, что в таблице Проекты, на которую ссылается внешний ключ, нельзя изменять значение первичного ключа, если в ссылающейся таблице Поставки существует строка с этим значением.

Для обеспечения целостности атрибута может быть использован оператор CHECK (некоторые из приведенных в примерах ниже операторов в версии языка Jet SQL не применяются):

```
CREATE TABLE Детали
  (Д№ CHAR(3) PRIMARY KEY,
  Имя_Д CHAR(15) UNUQUE,
  Цвет CHAR(10) CHECK (Цвет='Черный' OR Цвет ='Красный'
    OR Цвет ='Желтый' OR Цвет ='???'),
  Вес INTEGER,
  Гор CHAR(20));
```

Ограничения на данные можно организовать и созданием доменов:

```
CREATE DOMAIN Color CHAR(10) CHECK (Color='Черный' OR
  Color='Красный' OR Color='Желтый' OR Color='???');
```

Тогда ограничение на поле Цвет можно ввести следующим образом:

```
CREATE TABLE Детали
  (Д№ CHAR(3) PRIMARY KEY,
  Имя_Д CHAR(15) UNUQUE,
  Цвет Color,
  Вес INTEGER,
  Гор CHAR(20));
```

Внести изменение в структуру таблицы можно оператором ALTER с указанием характера изменения – ADD, MODIFY или DELETE:

```
ALTER TABLE Детали ADD [Дата изготовления] DATE;
```

Для удаления объектов базы данных используется оператор DROP:

```
DROP TABLE Детали;
```

2.4 Запросы на выборку

Для создания запроса на выборку используется команда SELECT. Она возвращает таблицу, содержащую поля, выбранные из базовых таблиц или из созданных ранее представлений (представление – это именованная производная таблица, точнее – сохраненная в базе данных команда SELECT, по которой производная таблица построена из базовых таблиц). Запрос на выборку выглядит следующим образом:

```
SELECT [ALL/DISTINCT] [TOP n [PERCENT]] список полей
  FROM имена таблиц
  [WHERE условие отбора]
  [ORDER BY столбцы сортировки [ASC/DESC]];
```

В предложении SELECT указываются поля, которые будут включены в возвращенную таблицу, и их имена в этой новой таблице. Имена полей разделяются запятыми. Предложение SELECT используется для выполнения реляционной операции проекции – в возвращаемую таблицу будут входить только перечисленные в предложении SELECT атрибуты. Ключевые слова ALL и DISTINCT, зарезервированные в ANSI SQL-92, определяют способ отбора строк (в справочных материалах в квадратных скобках размещают необязательные части команды):

- ALL – включает все строки, соответствующие указанным далее условиям отбора;
- DISTINCT – исключает повторяющиеся строки из результирующего набора записей (возвращенной таблицы).

Необязательный параметр TOP n[PERCENT] ограничивает количество выведенных записей в результирующей таблице первыми n или n% из полного набора.

В предложении FROM указываются имена таблиц, из которых будут выбраны данные.

Предложение WHERE определяет условие отбора записей и реализует реляционную операцию выборки. Условие задается оператором LIKE (для текстовых полей) или операторами сравнения: >, <, =, < >, >=, BETWEEN.

Модификатор ORDER BY определяет порядок сортировки записей в созданной таблице. Ключевыми словами ASC и DESC (ascent и descent) можно задать сортировку по возрастанию или убыванию соответственно.

Пример инструкции запроса на выборку:

```
SELECT Имя_Д, Вес
  FROM Детали
  WHERE вес>500
  ORDER BY вес DESC;
```

Результатом выполнения запроса будет новая таблица, содержащая два столбца Имя_Д и Вес и множество строк, удовлетворяющих условию Вес>500, отсортированных по убыванию веса.

2.5 Статистические функции

Статистические функции (применяются также названия агрегатные и итоговые функции) в качестве аргумента используют множество значений, а в результате возвращают одно значение. (сумму, среднее, минимальное, максимальное и т.п.) Они применяются к группам записей с общим значением атрибутов, по которым проводится группировка. Для этого используется инструкция GROUP BY:

```
SELECT статистическая функция (имя поля) AS заголовок поля
  [, список полей]
    FROM имена таблиц
    [WHERE условие отбора]
    [GROUP BY поля, по значениям которых выполняется
  группировка]
    [HAVING условие для результата]
    [ORDER BY столбцы сортировки];
```

Поле, используемое как аргумент статистической функции, должно содержать данные числового типа.

Ключевое слово AS определяет заголовок поля, в котором будет размещен результат вычисления. В предложении GROUP BY указываются поля, по значениям которых записи объединяются в группы, к которым применяется статистическая функция. Оператор HAVING позволяет ввести одно или несколько условий, налагаемых на возвращаемую таблицу. Список

полей в предложении SELECT должен включать все поля, перечисленные в условии группировки.

Если вычисления проводятся по всем строкам таблицы, то оператор GROUP BY не используется.

Примеры команд SQL, применяющих статистические функции

Общее количество деталей можно получить следующим образом:

```
SELECT SUM(Кол) FROM Поставки AS [Общее число];
```

В предложении SELECT можно указывать несколько скалярных выражений:

```
SELECT MIN(Кол), MAX(Кол), SUM(Кол), AVG(Кол)
  FROM Поставки;
```

Такие запросы возвращают в качестве результата таблицу, состоящую из одной строки. Если не указываются названия полей, в которые будут занесены результаты вычислений, то система присвоит им стандартные имена.

Количество строк в таблице можно посчитать следующим образом:

```
SELECT COUNT(*) AS Кол_кортежей FROM Поставки;
```

В результате получим таблицу из одной строки столбца с заголовком “Кол_кортежей”.

Как отмечалось ранее, статистические функции можно применять как ко всем записям таблицы, так и к отдельным группам записей. Например, чтобы получить количество деталей, поставляемых каждым поставщиком, надо провести группировку по атрибуту П№ и в каждой группе выполнить сложение по полю Кол. В возвращенной таблице количество строк будет равно количеству поставщиков и результаты вычисления для каждого П№ будут записаны в новом поле:

```
SELECT Пк.П№, SUM(Пк.Кол)AS КолДет
  FROM Поставки AS Пк
  GROUP BY Пк.П№;
```

(В этом примере используются псевдонимы, которые вводятся в предложении FROM при помощи оператора AS (его можно и опустить).

Использование псевдонимов упрощает запись команд. Они также используются и для организации некоторых запросов.)

В предложении SELECT необходимо указывать атрибуты, по которым производится группировка, и нельзя указывать имена атрибутов, не входящих в предложение GROUP BY (но можно указывать несколько статистических функций).

На полученные результаты можно накладывать ограничения оператором HAVING, например:

```
SELECT П№, SUM(Кол)
  FROM Поставки
  GROUP BY П№
  HAVING COUNT(*)>2;
```

Поскольку используется оператор GROUP BY, то статистические функции будут применяться к группам данных. Предложение HAVING COUNT(*)>2 выбирает из результата только те группы, в которых количество кортежей больше 2 (т.е., поставщики выполнили более 2 поставок).

2.6 Создание соединений

Реляционная операция произведения двух отношений TIMES реализуется в SQL, если указать имена этих отношений в предложении FROM, например:

```
SELECT *
  FROM Проекты, Поставки;
```

Если из полученной таблицы при помощи оператора WHERE выбрать строки, в которых в полях Проекты.Пр№ и Поставки.Пр№ равные значения, то будет реализована реляционная операция соединения JOIN по полю П№. Например, соединение отношений Проекты и Поставки:

```
SELECT *
  FROM Проекты, Поставки
  WHERE Проекты.Пр№=Поставки.Пр№;
```

Подобным образом можно соединить произвольное число отношений:

```
SELECT DISTINCT П.Имя_П, Д.Имя_Д, Пр.Имя_Пр, Пк.Кол
  FROM Поставщики П, Детали Д, Проекты Пр, Поставки Пк
  WHERE Д.Д№=Пк.Д№
  AND П.П№=Пк.П№
  AND Пр.Пр№=Пк.Пр№;
```

Для соединения таблиц может быть использован и специальный оператор JOIN…ON, который записывается в предложении FROM и содержит имена соединяемых таблиц, поле соединения и условие соединения:

```
SELECT список полей
  FROM таблица1 {INNER/LEFT/RIGHT} JOIN таблица2
  ON условие связи
  [WHERE условие отбора]
  [ORDER BY столбцы сортировки];
```

Вместо имени одной из таблиц может использоваться повторно конструкция JOIN … ON, называемая вложенной. В этом случае таблица соединяется с результатом соединения двух других таблиц. Инструкция SQL может содержать набор нескольких вложенных конструкций JOIN … ON. Их число обычно равно общему количеству таблиц, включенных в запрос, минус один. Перед оператором JOIN должен быть указан тип соединения:

- INNER – соединяет записи из двух таблиц, если связующие поля этих таблиц содержат одинаковые значения;
- LEFT (RIGHT) – соединяет записи исходных таблиц, причем левое внешнее соединение включает все записи из первой (левой) таблицы и присоединяет к ним записи из второй таблицы, если связующие поля содержат одинаковые значения. Правое внешнее соединение включает все записи из второй (правой) таблицы и присоединяет к ним записи из первой таблицы, если связующие поля содержат одинаковые значения.

Конструкция «ON условие связи» указывает на поля, по которым производится соединение (их имена могут быть разными, но содержание должно быть одинакового смысла), и условие соединения. Реляционному соединению JOIN соответствует условие равенства значений в полях соединения. Язык SQL позволяет соединять строки и по другим условиям. В выражении «условие связи» присутствует оператор сравнения значений полей, и оно возвращает значения TRUE или FALSE. Если значение

выражения TRUE, то соединенные строки включаются в результирующую таблицу.

Пример инструкции на соединение отношений Проекты и Поставки базы данных Проекты-Поставщики-Детали по значениям полей Пр№:

```
SELECT DISTINCT *
  FROM Проекты AS Пр INNER JOIN Поставки AS Пк
  ON Пр.Пр№=Пк.Пр№;
```

Соединения представляют собой мощный инструмент для организации сложных запросов и используются не только для выборки данных из разных таблиц. Например, соединение таблицы самой с собой позволяет формировать всевозможные комбинации строк, что может быть необходимым для выполнения некоторых поисков. Допустим, надо узнать, какие детали поставляются несколькими поставщиками. Соединение таблицы Поставки с таблицей Поставки по значению поля Д№ позволит построить таблицу, в которой в строках будут присутствовать всевозможные пары поставщиков , поставляющих одну деталь:

```
SELECT F.П№, S.П№, F.Д№
  FROM Поставки AS F, Поставки AS S
  WHERE F.Д№=S.Д№;
```

В результате выполнения этой команды будет создана таблица, фрагмент которого показан на рисунке.

F.ПN	S.ПN	F.ДN
П1	П1	Д1
П1	П1	Д1
П1	П5	Д1
П1	П1	Д1
П1	П1	Д1
П1	П5	Д1
П5	П1	Д1
П5	П1	Д1
П5	П5	Д1
...	...	...

Рис.2.1.

Добавив условие выборки

```
AND F.П№ <> S.П№,
```

убираем строки, содержащие одинаковые номера поставщиков, которые не нужны по условию поиска. Сделав проекцию по полю F.Д№, получим список деталей, поставляемых несколькими поставщиками. Модификатор DISTINCT удалит повторяющиеся кортежи.

Полная команда SQL для решения поставленной задачи выглядит так:

```
SELECT DISTINCT F.ДП№
  FROM Поставки AS F, Поставки AS S
  WHERE F.ДП№=S.ДП№
  AND F.ПП№ <> S.ПП№;
```

2.7 Вложенные запросы

В инструкциях к программным продуктам корпорации Microsoft под вложенным запросом понимают запрос, помещаемый в команду SELECT, INSERT, UPDATE, DELETE или в другой вложенный запрос. Он может быть использован везде, где разрешены выражения.

Это не общепринятое определение. В литературе часто используется другая точка зрения и считается, что вложенный запрос – это применение одного запроса к результирующему набору записей другого. Для этой цели создается команда SELECT, в которой для формирования условия выборки в предложении WHERE используется еще одна команда SELECT.

Такие конструкции могут существенно повысить производительность работы базы данных, но не менее важно и то, что использование вложенных запросов может упростить алгоритм поиска данных.

Вложенные запросы с оператором IN

Синтаксис вложенного запроса, для создания которого используется оператор IN:

```
SELECT список полей
  FROM список таблиц
  WHERE [имя таблицы.] имя поля
    IN (SELECT оператор выборки
          [GROUP BY условие группировки]
          [HAVING условие отбора]
          [ORDER BY столбцы сортировки]);
```

Пример. Найти номера поставщиков, поставляющих хотя бы одну черную деталь.

```
SELECT Пк.ПП№ FROM Поставки Пк
  WHERE Пк.ДП№ IN
          (SELECT Д.ДП№ FROM Детали Д
               WHERE Д.Цвет='Черный');
```

Механизм действия оператора IN такой. Внутренний запрос возвращает множество значений атрибута, удовлетворяющих заданному условию (в нашем примере – номера деталей черного цвета из таблицы Детали). Внешний запрос выбирает те строки из внешней таблицы (Поставки), в которых атрибут сравнения (номера деталей) присутствует в множестве, возвращенном внутренним запросом (номера деталей черного цвета). То есть, из таблицы Поставки выбираются строки с деталями черного цвета. После выбора таких строк делается проекция по полю П№.

Такая конструкция позволяет легко добавлять данные из других таблиц. Например, чтобы в рассмотренном примере найти имена поставщиков, которые хранятся в третьей таблице Поставщики, надо добавить еще один вложенный запрос:

```
SELECT П.Имя_П
  FROM Поставщики П
  WHERE П.П№ IN
    (SELECT Пк.П№
         FROM Поставки Пк
         WHERE Пк.Д№ IN
              (SELECT Д.Д№
                   FROM Детали Д
                   WHERE Д.Цв='Черный'));
```

Такой же результат можно получить при помощи операции соединения трех таблиц:

```
SELECT DISTINCT П.Имя_П
  FROM Поставщики П, Поставки Пк, Детали Д
  WHERE П.П№=Пк.П№
    AND Пк.Д№=Д.Д№
    AND Д.Цв='Черный';
```

Эффективность запросов разного типа можно оценивать только если известны параметры всех объектов базы данных. Однако существуют определенные типы запросов, которые лучше реализовать с помощью вложенных запросов, например, так называемые проверки на существование.

Пример: Найти данные о поставщиках, которые не поставляют детали:

```
SELECT *
  FROM Поставщики П
  WHERE П.П№ NOT IN
    (SELECT Пк.П№ FROM Поставки Пк);
```

Вложенные запросы с оператором EXISTS

Во многих случаях удобно организовывать вложенные запросы при помощи оператора EXISTS.

Оператор EXISTS в предложении WHERE выполняет поиск строк, которые удовлетворяют критериям соответствующего вложенного запроса, и возвращает булево значение «истина», если такая строка обнаружена, или «ложь» в противном случае. При возврате значения «истина» проверяемая строка внешней таблицы отбирается в результат.

Пример. Найти имена поставщиков, которые поставляют деталь Д1:

```
SELECT DISTINCT П.Имя_П FROM Поставщики AS П
  WHERE EXISTS
    (SELECT * FROM Поставки AS Пк
         WHERE Пк.ПN№=П.ПN№
         AND Пк.ДN№='Д1');
```

Для каждой строки таблицы Поставщики, которая обрабатывается во внешнем запросе, из вложенного запроса будет возвращено значение “истина”, если существует хотя бы одна строка в таблице Поставки с тем же значением атрибута П№, которое присутствует в рассматриваемой строке внешнего запроса. Чтобы это проверить, выполняется соединение по условию Пк.П№=П.П№. Во вложенном запросе используется предложение SELECT *, но можно указать имя любого атрибута, т.к. этот вложенный запрос ищет строку и возвращает не данные, а значение истинности. Итак, из внешней таблицы Поставщики будут выбраны те строки, в которых присутствует такой П№, что во внутренней таблице Поставки есть строка с таким же П№ и деталью Д1.

Чтобы получить имена поставщиков, которые не поставляют деталь Д1, можно использовать отрицание предложения EXISTS:

```
SELECT DISTINCT П.Имя_П FROM Поставщики AS П
  WHERE NOT EXISTS
    (SELECT * FROM Поставки AS ПК
         WHERE ПК.ПN№=П.ПN№
         AND ПК.ДN№='Д1');
```

Приведем для сравнения пример решения задачи двумя способами – с операторами EXISTS и IN:

Пример. Найти номера деталей, поставляемых поставщиком из города, название которого начинается на букву М.

```
I. SELECT DISTINCT Пк.Д№ FROM Поставки AS Пк
    WHERE Пк.П№ IN
         (SELECT П.П№ FROM Поставщики AS П
              WHERE П.Гор LIKE 'М*');


II. SELECT DISTINCT Пк.Д№ FROM Поставки Пк
    WHERE EXISTS
         (SELECT * FROM Поставщики П
              WHERE П.П№ = Пк.П№
              AND Гор LIKE 'М*');
```

Как и при использовании оператора IN, оператор EXISTS можно применить для добавления сведений из третьей таблицы. Например, если в прошлой задаче потребовать не номер, а имя детали, которое хранится в таблице Детали, то команда SQL выглядит так:

```
SELECT DISTINCT Д.Имя_Д FROM Детали AS Д
  WHERE EXISTS
    (SELECT DISTINCT Пк.Д№ FROM Поставки AS Пк
         WHERE EXISTS
         (SELECT * FROM Поставщики AS П
              WHERE П.П№=Пк.П№
                   AND Гор LIKE 'М*')
                   AND Д.Д№=Пк.Д№);
```

Обратите внимание на то, что при использовании оператора EXISTS, при добавлении нового внешнего запроса необходимо включить условие соединения его с внутренним запросом. В нашем случае это условие Д.Д№=Пк.Д№, добавленное в последней строке команды.

Вложенные запросы могут содержать статистические функции и другие вычисления, причем, как во внешнем, так и во внутреннем запросах.

Вложенные запросы могут быть организованы и без вспомогательных операторов.

Пример. Сколько деталей Д3 использовалось в проектах, в которых поставщики и детали из одного города?

```
SELECT Sum(Кол)
  FROM Поставки Пк
  WHERE Д№='Д3'
    AND
```

```
    (SELECT Гор FROM Поставщики П
         WHERE П.ПЗ=Пк.ПЗ)
=
    (SELECT Гор FROM Проекты Пр
         WHERE Пр.ПрЗ=Пк.ПрЗ)
```

2.8 Примеры команд для выборки данных

По своей конструкции язык SQL очень гибкий и дает возможность реализовывать различные алгоритмы отбора данных. Рассмотрим на простом примере разнообразие методов создания команды SELECT.

Пример. Получить номера деталей, поставляемых поставщиком из Минска.

По условию задачи команда SELECT должна построить таблицу, состоящую из одного атрибута – Д№. Необходимые данные хранятся в двух таблицах – в таблице Поставки собраны данные о номерах поставщиков и номерах поставляемых ими деталей, а города поставщиков хранятся в таблице Поставщики. Выражение реляционной алгебры для этой задачи имеет вид:

```
((Поставщики JOIN Поставки) WHERE Гор='Минск')[Д№]
```

В командах SQL мы познакомились с двумя способами использования связанных данных, хранящихся в разных таблицах – создание соединений или вложенных запросов. Рассмотрим разные варианты решения.

1. Использование операторов соединения:

```
SELECT DISTINCT Пк.Д№
  FROM Поставки AS Пк INNER JOIN Поставщики AS П
         ON Пк.ПЗ=П.ПЗ
              WHERE Гор='Минск';
```

2. Создание соединения через произведение и выборку:

```
SELECT DISTINCT Пк.Д№
  FROM Поставки AS Пк, Поставщики AS П
    WHERE Пк.ПЗ=П.ПЗ
         AND Гор='Минск';
```

3. Использование вложенного запроса с оператором IN:

```
SELECT DISTINCT Пк.ДN
  FROM Поставки AS Пк
    WHERE Пк.ПN IN
          (SELECT П.ПN
               FROM Поставщики AS П
                    WHERE Гор='Минск');
```

4. Использование вложенного запроса с оператором EXISTS:

```
SELECT DISTINCT Пк.ДN
  FROM Поставки AS Пк
    WHERE EXISTS
          (SELECT *
               FROM Поставщики AS П
                    WHERE Гор='Минск'
                         AND Пк.ПN=П.ПN);
```

5. Использование вложенного запроса без вспомогательных операторов:

```
SELECT DISTINCT Пк.ДN
  FROM Поставки AS Пк
    WHERE
          (SELECT Гор
               FROM Поставщики AS П
                    WHERE Пк.ПN=П.ПN)='Минск';
```

Как правило, выбор способа написания команды зависит от разных факторов – сложности алгоритма, его эффективности, структуры объектов базы данных и даже от предпочтений пользователя.

В следующем примере рассмотрим более сложную задачу, реляционное выражение для которой было подробно рассмотрено в разделе 1.4 (пример 6).

Пример. Получить номера поставщиков, у которых максимальное значение поставок как минимум в 2 раза превышает среднее количество деталей, поставляемых остальными поставщиками.

```
SELECT DISTINCT Пк.ПN
  FROM Поставки Пк
    WHERE
          (SELECT MAX(Кол)
               FROM Поставки Пк1
                    WHERE Пк1.ПN=Пк.ПN)
          >=
          2*(SELECT AVG(Кол)
               FROM Поставки Пк2
                    WHERE Пк2.ПN<>Пк.ПN);
```

2.9 Запрос на объединение

Запросы на объединение таблиц реализуют реляционную операцию UNION и позволяют представить в одной таблице записи, созданные несколькими запросами на выборку, записав их один под другим. Синтаксис запроса на объединение, основой которого является оператор UNION, имеет вид:

```
SELECT оператор выборки
  UNION
  SELECT оператор выборки
         [GROUP BY условие группировки]
         [HAVING итоговое условие]
  [UNION
  SELECT оператор выборки
         [GROUP BY условие группировки]
         [HAVING итоговое условие]]
  [UNION
  …]
  [ORDER BY столбцы сортировки];
```

При использовании операторов UNION необходимо формировать объединяемые таблицы с одинаковым набором имен полей, причем, их последовательность должна быть одинакова в каждой команде SELECT. Модификатор ORDER BY может быть использован только один раз в инструкции за последним оператором UNION SELECT. При необходимости в каждую команду SELECT можно вложить операторы GROUP BY и HAVING. Оператор UNION удаляет из результирующей таблицы все строки-дубликаты. Чтобы это не происходило, используют оператор UNION ALL.

Синтаксис оператора UNION позволяет собирать в одном поле объединенной таблицы значения из различных доменов:

```
SELECT Имя_П AS Наименование
  FROM Поставщики
  WHERE Гор='Минск'
UNION
   SELECT Имя_Пр AS Наименование
    FROM Проекты
    WHERE Гор='Минск';
```

Результатом запроса будет таблица из одного столбца, названного «Наименование», содержащая поставщиков, находящихся в Минске, и проектов, выполняемых в Минске.

2.10 Запросы, выполняющие реляционные операции пересечения, вычитания и деления

В кратком учебном пособии мы ограничимся самыми основными правилами использования языка программирования SQL, к которым относится и программная реализация реляционных операций. Ранее мы уже использовали команды, выполняющие такие операции с таблицами, как произведение, объединение, выборку, соединение, проекцию и некоторые дополнительные операции, позволяющие выполнять вычисления, переименования и т.п. Сейчас научимся выполнять оставшиеся три операции – пересечение, вычитание и деление, которые также могут быть очень полезными при выполнении запросов на выборку данных.

В некоторых модификациях языка SQL, например, используемой в СУБД Oracle, реляционная операция пересечения отношений выполняется оператором INTERSECT, который возвращает только те строки, которые присутствуют в обеих таблицах. Вычитание отношений реализуется оператором EXCEPT, который возвращает строки первой таблицы за исключением тех, которые присутствуют и во второй таблице. Как и в случае оператора UNION, обрабатываемые таблицы должны иметь одинаковый набор и последовательность имен полей.

В таких СУБД, как MS Access, MS SQL Server и некоторых других эти операции удобно реализовывать при помощи вложенных запросов, использующих оператор EXISTS.

Покажем, как при помощи оператора EXISTS можно реализовать реляционные операции пересечения, вычитания и деления.

Пример. Пересечением таблиц Детали и Поставщики по полю Гор является множество городов, в которых есть и детали, и поставщики:

```
SELECT DISTINCT Д.Гор FROM Детали AS Д
  WHERE EXISTS
    (SELECT * FROM Поставщики П
         WHERE Д.Гор=П.Гор);
```

Вычитая из таблицы Детали таблицу Поставщики по полю Гор, получим множество городов, в которых есть детали, но нет поставщиков:

```
SELECT DISTINCT Д.Гор FROM Детали Д
  WHERE NOT EXISTS
    (SELECT * FROM Поставщики П
         WHERE Д.Гор=П.Гор);
```

Операция пересечения может быть реализована и без оператора EXISTS, например так:

```
SELECT DISTINCT Д.Гор FROM Детали Д
  WHERE
    (SELECT COUNT(*) FROM Поставщики П
         WHERE Д.Гор=П.Гор)>0;
```

Если количество найденных строк во вложенном запросе больше нуля, то в таблице Поставщики есть город, равный городу в проверяемой строке внешней таблицы Детали, значит условие сравнения в предложении WHERE даст значение «истина» и проверяемая строка из таблицы Детали будет выбрана. Соответствующий город будет возвращен в итоговой таблице. Операция вычитания будет выполнена, если условие сравнения > заменить на равенство.

При таком запросе производительность ниже, чем при использовании EXISTS, т.к. EXISTS возвращает результат, как только ему встретится хотя бы одна строка, удовлетворяющая условию, а оператор COUNT(*) подсчитывает строки, просматривая таблицу до конца.

Пример реализации реляционной операции деления DIVIDED BY: Получить номера поставщиков, поставляющих все детали.

```
SELECT DISTINCT Пк.П№
  FROM Поставки AS Пк
  WHERE NOT EXISTS
  (SELECT Д.Д№ FROM Детали AS Д
    WHERE NOT EXISTS
         (SELECT Пк1.Д№ FROM Поставки AS Пк1
              WHERE Пк1.П№=Пк.П№
                   AND Пк1.Д№=Д.Д№));
```

Внешний запрос исследует каждый кортеж отношения Поставки и возвращает из него П№, если вложенный запрос возвращает значение «Ложь». Вложенный запрос создает множество всех возможных значений Д№ из отношения Детали. Из этого множества удаляются те значения Д№, которые состоят в паре с П№ в кортеже с тем же значением П№, что и кортеж, рассматриваемый во внешнем запросе. Если рассматриваемый П№ комбинируется со всеми возможными значениями Д№, то результатом будет пустое множество и вложенный запрос возвратит значение «Ложь», т.е., не существует значений Д№, которые не состоят в паре с этим конкретным П№.

Чтобы получить имена таких поставщиков, надо вложить этот запрос в другой:

```
SELECT П.Имя_П
  FROM Поставщики AS П
  WHERE П.П№ IN
    (SELECT DISTINCT Пк.П№
         FROM Поставки AS Пк
         WHERE NOT EXISTS
              (SELECT Д.Д№ FROM Детали AS Д
                   WHERE NOT EXISTS
                        (SELECT Пк1.Д№ FROM Поставки AS Пк1
                             WHERE Пк1.П№=Пк.П№
                             AND Пк1.Д№=Д.Д№)));
```

Используя подобный шаблон, можно выполнять разные выборки, алгоритм которых основан на реляционной операции деления, в том числе при делении n-арных отношений на m-арные.

2.11 Запросы на изменение

Запросы на изменение предназначены для добавления, удаления или обновления записей, а также для создания таблиц.

Синтаксис запроса на добавление записей:

```
INSERT INTO таблица-получатель
  SELECT список полей
  FROM таблица-источник;
```

Если в инструкции на добавление отсутствует предложение WHERE, то в таблицу-получатель будут добавлены все записи таблицы-источника.

Синтаксис запроса на удаление записей:

```
DELETE FROM имя таблицы
  [WHERE условие удаления];
```

В этой инструкции предложение WHERE также не обязательное. При его отсутствии из указанной таблицы будут удалены все записи.

Синтаксис запроса на создание таблицы:

```
SELECT список полей
  INTO новая таблица
  FROM исходная таблица
  [WHERE условие выбора];
```

Полную копию исходной таблицы можно создать, указав вместо списка полей символ * и не используя предложение WHERE.

Запрос на обновление используется для присвоения новых значений отдельным атрибутам при помощи операторов UPDATE и SET:

```
UPDATE имя таблицы
  SET имя_поля_1=значение [,имя_поля_2=значение[,…]]
  [WHERE условие обновления];
```

2.12 Перекрестные запросы

Перекрестные запросы позволяют создавать различные итоговые запросы, использующие статистические функции SQL. Когда данные группируются с помощью перекрестного запроса, можно выбирать значения из заданных полей или выражений как заголовки столбцов и строк. Это позволяет просматривать данные в более компактной форме, чем при работе с запросом на выборку. Для организации перекрестных запросов используются операторы TRANSFORM и PIVOT:

```
TRANSFORM статистическая функция (имя поля) [AS
  наименование]
  SELECT список полей
  FROM имя таблицы
  PIVOT поле [IN (значение_1[, значение_2[, ...]])];
```

Пример перекрестного запроса:

Представить данные о количествах деталей, поставленных каждым поставщиком.

```
TRANSFORM Sum(Пк.Кол)
  SELECT Пк.П№, Sum(Пк.Кол) AS [ВСЕГО:]
  FROM Поставки AS Пк
  GROUP BY Пк.П№
  PIVOT Пк.Д№;
```

Результатом такого запроса будет таблица:

П№	ВСЕГО:	Д1	Д2	Д3	Д4	Д5	Д6
П1	900	900					
П2	3200			3100		100	
П3	700			200	500		
П4	600						600
П5	3110	110	300	200	800	1000	700

Оператор TRANSFORM записывается вначале команды SELECT, указывает на то, что будет создан перекрестный запрос и задает правила вычислений (формулы или статистические функции) для групп данных, которые заданы оператором PIVOT и которые будут использованы как заголовки полей в результирующем наборе запроса. В предложении SELECT задаются атрибуты, значения которых будут использованы как заголовки строк. В этих строках будут размещены результаты вычислений, заданных в предложении TRANSFORM. Здесь же можно заказать новое поле, в котором будут храниться результаты операции над группами данных, записанных в предложении GROUP BY. При необходимости можно включить и другие предложения, например WHERE или ORDER BY для описания дополнительных условий отбора и сортировки. Кроме того, можно использовать вложенные запросы как предикаты в перекрестном запросе.

В предложении PIVOT, которое описывает имена и формат полей результирующего набора записей, содержащих результаты вычислений, заданных в предложении TRANSFORM, аргумент *поле* можно ограничить, чтобы создать заголовки из фиксированных значений (значение_1, значение_2, …), перечисленных в необязательном предложении IN.

2.13 Управление доступом к данным

Это последняя группа команд, которую рассмотрим в данном курсе. Доступ к данным в многопользовательской системе регулируется с помощью операторов GRANT и REVOKE. Указываются вид полномочий (SELECT, UPDATE, ALL), таблица или представление, по отношению к которым создается полномочие, и имя пользователя:

```
GRANT UPDATE ON Поставки TO USER1;
```

Полномочия для всех пользователей устанавливаются через специального пользователя PUBLIC:

```
GRANT SELECT ON Поставки TO PUBLIC;
```

Для повышения безопасности рекомендуется открывать доступ к представлениям, а не базовым отношениям.

Оператор REVOKE аннулирует полномочия:

```
REVOKE UPDATE ON Поставки FROM USER1;
```

Если аннулировать полномочия, используя выражение ...FROM PUBLIC, то полномочий лишаются все пользователи.

3 Задания для практических занятий и самостоятельной работы

I. Построить в СУБД ACCESS базу данных Поставщики–Детали–Проекты, описанную в главе 1 (схема базы данных приведена на рисунке 1.14), используя команды CREATE (ALTER) TABLE. Заполнить ее данными из таблицы, приведенной на рисунке 1.15.

II. Используя операторы реляционной алгебры, записать алгоритмы отбора данных, для приведенных ниже задач. В СУБД Access записать команды SQL и выбрать требуемые данные.

1. Получить полную информацию обо всех проектах.
2. Получить полную информацию обо всех проектах в Минске.
3. Получить номера поставщиков, которые обеспечивают проект IIpl.
4. Получить все отправки, где количество находится в диапазоне от 300 до 750 включительно.
5. Получить все сочетания "цвета деталей - города деталей".

   Замечание. Здесь и в последующих упражнениях термин "все" используется в значении "все, представленные в настоящий момент в базе данных", а не "все возможные".

6. Получить все такие тройки "номера поставщиков» – номера деталей – номера проектов", для которых выводимые поставщик, деталь и проект размещены в одном городе.
7. Получить все такие тройки "номера поставщиков – номера деталей – номера проектов", для которых выводимые поставщик, деталь и проект не размещены в одном городе.
8. Получить все такие тройки "номера поставщиков – номера деталей – номера проектов", для которых никакие из двух выводимых поставщиков, деталей и проектов не размещены в одном городе.
9. Учитывая зафиксированные в базе данных поставки, получить все такие тройки "номера поставщиков – номера деталей – номера

проектов", для которых никакие из двух выводимых поставщиков, деталей и проектов не размещены в одном городе.

10. Получить номера деталей, поставляемых поставщиком в Минске для проекта в Минске.
11. Получить все пары названий городов, для которых поставщик из первого города обеспечивает проект во втором городе.
12. Получить номера деталей, поставляемых для всех проектов, обеспечиваемых поставщиком из того же города, где размещен проект.
13. Получить номера проектов, обеспечиваемых, по крайней мере, одним поставщиком не из того же города.
14. Получить все такие пары номеров деталей, которые обе поставляются одновременно одним поставщиком.
15. Получить общее количество деталей Д2, поставляемых поставщиком П5.
16. Получить общее количество деталей Д2, поставляемых каждым поставщиком.
17. Получить общее число проектов, обеспечиваемых поставщиком П5.
18. Получить общее число проектов, обеспечиваемых каждым поставщиком.
19. Для каждой детали, поставляемой для проекта, получить номер детали, номер проекта и соответствующее общее количество.
20. Получить номера деталей, поставляемых для некоторого проекта со средним количеством больше 320.
21. Получить номера поставщиков, поставивших количество деталей больше среднего количества деталей в одной поставке.
22. Получить имена проектов, обеспечиваемых поставщиком П1.
23. Получить номера деталей, поставляемых для минских проектов.
24. Получить цвета деталей, поставляемых поставщиком П1.

25. Получить номера деталей, поставляемых для какого-либо проекта в Минске.
26. Получить номера проектов, использующих по крайней мере одну деталь, имеющуюся у поставщика П1.
27. Получить номера поставщиков, поставляющих по крайней мере одну деталь, поставляемую по крайней мере одним поставщиком, который поставляет по крайней мере одну красную деталь.
28. Получить номера поставщиков со статусом, меньшим чем у поставщика П1.
29. Получить номера проектов, город которых стоит первым в алфавитном списке городов.
30. Получить номера проектов, для которых среднее количество поставляемых деталей Д1 больше, чем наибольшее количество любых деталей, поставляемых для проекта Пр1.
31. Получить номера поставщиков, поставляющих деталь Д1 для некоторого проекта в количестве, большем среднего количества деталей Д1 в поставках для этого проекта.
32. Получить номера проектов, для которых не поставляются желтые детали поставщиками из Минска.
33. Получить номера проектов, полностью обеспечиваемых поставщиком П1.
34. Получить номера поставщиков, поставляющих одну и ту же деталь для всех проектов.
35. Получить номера проектов, обеспечиваемых, по крайней мере, всеми деталями поставщика П1.
36. Получить имена поставщиков, поставляющих все детали.
37. Получить все города, в которых расположен по крайней мере один поставщик, одна деталь или один проект.

38. .Получить номера деталей, поставляемых либо минским поставщиком, либо для минского проекта. Номера деталей отсортировать по убыванию их веса.
39. Получить пары "номер поставщика – номер детали" такие, что данный поставщик не поставляет данную деталь.
40. Сколько и какие детали поставляет поставщик П5 проектам не в Минске.
41. Найти номера деталей, поставляемых поставщиком из города, который начинается на букву «М».
42. Найти город, где сделано больше всего деталей.
43. В каком проекте используется минимальное количество деталей?
44. Какое количество наименований деталей используется в проектах?
45. Получить номера поставщиков, у которых максимальное значение поставки как минимум в 2 раза меньше среднего значения поставок деталей остальными поставщиками.

Заключение

Освоение материала учебного пособия позволит специалисту, начинающему использовать базы данных в своей профессиональной деятельности, выполнять основные операции с данными – отбирать данные по различным критериям, проводить вычисления с использованием статистических формул и различных математических выражений, формировать результирующие таблицы.

Даже краткое знакомство с технологией реляционных баз данных убедит пользователя в их удобстве и эффективности.

Список литературы

1. К.Дж. Дейт. Введение в системы баз данных. Восьмое издание. – М.: Вильямс, 2005. – 1328 С.)
2. К. Дейт. SQL и реляционная теория. Как грамотно писать код на SQL / Символ, 2014. –480 с.
3. П.Вайнберг, Дж.Гробф, Э.Оппель. SQL. Полное руководство. Третье издание. / Вильямс, 2015. – 960 с.
4. Э. Молинаро. SQL. Сборник рецептов. Символ, 2016. – 672 с.
5. Codd E.F. A Relation Model of Data for Large Share Data Banks //CACM.– 1970.–13,No.6.
6. Джо Селко. Программирование на SQL для профессионалов. – М.: Лори, 2004. – 442 С.
7. Фиайли К. SQL. – СПб.: Питер, 2004. 464 с.
8. Астахова И.Ф., Толстобров А.П., Мельников В.М. SQL в примерах и задачах. Учеб. пособие – Мн.: Новое знание, 2002. – 176 с.
9. Чарльз Е. Браун, Рон Петруша. Access VBA: Программирование в примерах. – М.: КУДИЦ-ОБРАЗ, 2006. – 432 с.
10. Дженнингс Р. Использование Microsoft Access 2000. Специальное издание. –М.: Вильямс, 2000. 1148 с.