Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра экономической информатики

ОСНОВЫ И ЛИНГВИСТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ БАЗ ДАННЫХ

Учебно-методическое пособие
для студентов специальности 40 01 02-02
«Информационные системы и технологии в экономике»
всех форм обучения

Минск 2007

УДК 004.6 (075.8)
ББК 32.973 – 018.2 я 73
О-75

Рецензент :
зав. кафедрой интеллектуальных информационных технологий БГУИР,
д-р техн. наук, проф. В. В. Голенков

Авторы:
И. Г. Орешко, В. Н. Комличенко, А. А. Бутов,
Е. Н. Унучек, О. П. Едемская, Н. А. Кириенко

О-75 **Основы** и лингвистическое обеспечение баз данных : учебно-метод. пособие для студ. спец. 40 01 02-02 «Информационные системы и технологии в экономике» всех форм обуч. / И. Г. Орешко [и др.]. – Минск : БГУИР, 2007. – 64 с. : ил.
ISBN 978-985-488-131-7

В пособии рассмотрены основы теории реляционных баз данных и реляционной алгебры. Рассматриваются основные приемы проектирования баз данных на различных этапах. Более подробно рассмотрены основные элементы реляционного языка SQL. В состав пособия включено описание трех лабораторных работ.

УДК 004.6 (075.8)
ББК 32.973 – 018.2 я 73

ISBN 978-985-488-131-7

СОДЕРЖАНИЕ

1. Базы данных. Основные понятия и определения

1.1. Понятие базы данных

1.1.1. Основные понятия и термины

База данных (БД) – *совокупность сведений, логически связанных таким образом, чтобы составлять единую совокупность данных, хранимых в запоминающих устройствах вычислительной машины. Эта совокупность выступает в качестве исходных данных задач, решаемых в процессе функционирования автоматизированных систем управления, систем обработки данных, а также информационных и вычислительных систем.*

Предметная область *– часть реального мира, которая описывается или моделируется с помощью БД и разрабатываемого программного обеспечения [2].*

Предметной областью может быть: предприятие, его некоторая функциональная часть, процесс, система и т.д. Модель предметной области создается с использованием понятий информационных объектов и функций, выполняемых этими объектами или для них.

Информационный объект *– идентифицируемый объект реального мира, некоторое понятие или процесс, относящиеся к предметной области, о которых хранятся описательные данные (люди, счета, изделия, события и т.д.).*

Информационный объект описывается совокупностью заданных элементов данных (характеристик объекта), которым присваиваются конкретные значения (символы, числа, коды).

Элемент данных *– характеристика объекта, которая определяется именем и одним или совокупностью некоторых значений (величин).*

Именами элемента данных могут быть, например, номер работающего, дата рождения, цена, квалификация и т.д. Значение элемента данных определяет эту характеристику для конкретного экземпляра информационного элемента.

Запись – совокупность таких значений элементов данных, которые описывают конкретный экземпляр объекта. Например, если объект ИЗДЕЛИЕ описывается элементами данных: КОД ИЗДЕЛИЯ, НАИМЕНОВАНИЕ ИЗДЕЛИЯ, МАТЕРИАЛ, СТОИМОСТЬ, КОЛИЧЕСТВО, то совокупность значений этих элементов для конкретного изделия и представляет запись.

Набор записей *– это множество записей об объекте для всех экземпляров данного типа объектов, например для всех экземпляров (производимых изделий) объекта ИЗДЕЛИЕ.*

Поиск отдельных записей осуществляется по содержанию идентифицирующего поля (идентификатора).

Идентификатор *– это элемент данных (или совокупность элементов), значение которого используется для определения одного или нескольких значений связанных с ним других элементов.*

Идентификация может быть уникальной (однозначной) или неуникальной (многозначной). Если идентификация однозначна, то говорят, что идентифицируемый элемент (атрибут) является функционально зависимым от ключа (идентифицирующего элемента). Например, можно предположить, что для объекта

ИЗДЕЛИЕ значение элемента КОД ИЗДЕЛИЯ однозначно идентифицирует значения других элементов: НАИМЕНОВАНИЕ ИЗДЕЛИЯ, МАТЕРИАЛ и др.

***Ключ** является идентификатором, уникально определяющим запись об объекте. Ключ в наборе записей может включать в себя один (простой ключ) или более (составной) элементов данных, совокупность которых однозначно идентифицирует запись об объекте.*

Элементы данных, не входящие в состав ключа, называются ***атрибутами***.

В записи об объекте значение атрибутов идентифицируется значением ключа.

В реляционных моделях баз данных набор записей представляется в виде отношения (двумерной таблицы), в котором кортежи (строки таблицы) соответствуют экземплярам объекта, а атрибуты (колонки таблицы) определяют набор и значение его характеристик (элементов данных).

1.1.2. Базы данных и СУБД

Восприятие реального мира можно соотнести с последовательностью разных, хотя иногда и взаимосвязанных явлений. С давних времен люди пытались описать эти явления (даже тогда, когда не могли их понять). Такое описание называют данными.

Традиционно фиксация данных осуществляется с помощью конкретного средства общения (например с помощью естественного языка или изображений) на конкретном носителе (например камне или бумаге). Обычно данные (факты, явления, события, идеи или предметы) и их интерпретация (семантика) фиксируются совместно, так как естественный язык достаточно гибок для представления того и другого. Примером может служить утверждение «Стоимость авиабилета 128». Здесь «128» – данное, а «Стоимость авиабилета» – его семантика.

Нередко данные и интерпретация разделены. Например, «Расписание движения самолетов» может быть представлено в виде таблицы (табл. 1), в верхней части которой (отдельно от данных) приводится их интерпретация. Такое разделение затрудняет работу с данными (попробуйте быстро получить сведения из нижней части таблицы).

Таблица 1. Расписание движения самолетов

Интерпретация							
Номер рейса	**Дни недели**	**Пункт отправления**	**Время вылета**	**Пункт назначения**	**Время прибытия**	**Тип самолета**	**Стоимость билета**
Данные							
138	2_4_7	Баку	21.12	Москва	0.52	ИЛ-86	115.00
57	3_6	Ереван	7.20	Киев	9.25	ТУ-154	92.00
1234	2_6	Казань	22.40	Баку	23.50	ТУ-134	73.50
242	1 по 7	Киев	14.10	Москва	16.15	ТУ-154	57.00
86	2_3_5	Минск	10.50	Сочи	13.06	ИЛ-86	78.50
137	1_3_6	Москва	15.17	Баку	18.44	ИЛ-86	115.00
241	1 по 7	Москва	9.05	Киев	11.05	ТУ-154	57.00
577	1_3_5	Рига	21.53	Таллинн	22.57	АН-24	21.50
78	3_6	Сочи	18.25	Баку	20.12	ТУ-134	44.00
578	2_4_6	Таллинн	6.30	Рига	7.37	АН-24	21.50

Применение ЭВМ для ведения* и обработки данных обычно приводит к еще большему разделению данных и интерпретации. ЭВМ имеет дело главным образом с данными как таковыми. Большая часть интерпретирующей информации вообще не фиксируется в явной форме (ЭВМ не «знает», является ли «21.50» стоимостью авиабилета или временем вылета). Почему же это произошло?

Существуют по крайней мере две исторические причины, по которым применение ЭВМ привело к отделению данных от интерпретации. Во-первых, ЭВМ не обладали достаточными возможностями для обработки текстов на естественном языке – основном языке интерпретации данных. Во-вторых, стоимость памяти ЭВМ была первоначально слишком высока. Память использовалась для хранения самих данных, а интерпретация традиционно возлагалась на пользователя. Пользователь закладывал интерпретацию данных в свою программу, которая «знала», например, что шестое вводимое значение связано с временем прибытия самолета, а четвертое – с временем его вылета. Это существенно повышало роль программы, так как вне интерпретации данные представляют собой не более чем совокупность битов на запоминающем устройстве.

Жесткая зависимость между данными и использующими их программами создает серьезные проблемы в ведении данных и делает их использование менее гибким.

Нередки случаи, когда пользователи одной и той же ЭВМ создают и используют в своих программах разные наборы данных, содержащие сходную информацию. Иногда это связано с тем, что пользователь не знает (либо не захотел узнать), что в соседней комнате или за соседним столом сидит сотрудник, который уже давно ввел в ЭВМ нужные данные. Чаще потому, что при совместном использовании одних и тех же данных возникает масса проблем.

Разработчики прикладных программ (написанных, например, на Бейсике, Паскале или Си) размещают нужные им данные в файлах, организуя их наиболее удобным для себя образом. При этом одни и те же данные могут иметь в разных приложениях совершенно разную организацию (разную последовательность размещения в записи, разные форматы одних и тех же полей и т.п.). Обобщить такие данные чрезвычайно трудно: например, любое изменение структуры записи файла, производимое одним из разработчиков, приводит к необходимости изменения другими разработчиками тех программ, которые используют записи этого файла.

1.1.3. Концепция баз данных

Активная деятельность по отысканию приемлемых способов объединения непрерывно растущего объема информации привела к созданию в начале 60-х годов XX в. специальных программных комплексов, называемых «Системы управления базами данных» (СУБД).

Основная особенность СУБД – это наличие процедур для ввода и хранения не только самих данных, но и описаний их структуры. Файлы, снабженные описанием хранимых в них данных и находящиеся под управлением СУБД, стали называть банками данных, а затем базами данных (БД).

---

* *Ведение (сопровождение, поддержка) данных – термин, объединяющий действия по добавлению, удалению или изменению хранимых данных.*

Пусть, например, требуется хранить расписание движения самолетов (см. табл. 1) и ряд других данных, связанных с организацией работы аэропорта (БД «Аэропорт»). Используя для этого одну из современных «русифицированных» СУБД, можно подготовить следующее описание расписания:

```
СОЗДАТЬ ТАБЛИЦУ Расписание
  (Номер_рейса         Целое
   Дни_недели          Текст (8)
   Пункт_отправления   Текст (24)
   Время_вылета        Время
   Пункт_назначения    Текст (24)
   Время_прибытия      Время
   Тип_самолета        Текст (8)
   Стоимость_билета    Валюта);
```

и ввести его вместе с данными в БД «Аэропорт».

Язык запросов СУБД позволяет обращаться за данными как из программ, так и с терминалов (рис. 1). Сформировав запрос

```
ВЫБРАТЬ  Номер_рейса, Дни_недели, Время_вылета
ИЗ ТАБЛИЦЫ  Расписание
ГДЕ  Пункт_отправления = 'Москва'
  И  Пункт_назначения  = 'Киев'
  И  Время_вылета > 17;
```

получим расписание рейсов «Москва – Киев» на вечернее время, а по запросу

```
ВЫБРАТЬ  КОЛИЧЕСТВО(Номер_рейса)
ИЗ ТАБЛИЦЫ  Расписание
ГДЕ  Пункт_отправления = 'Москва'
  И  Пункт_назначения  = 'Минск';
```

получим количество рейсов «Москва – Минск».

Рис. 1. Связь программ и данных при использовании СУБД

Эти запросы актуальны и при расширении таблицы:

```
ДОБАВИТЬ В ТАБЛИЦУ Расписание
   Длительность_полета  Целое;
```

Однако за все надо расплачиваться: на обмен данными через СУБД требуется большее время, чем на обмен аналогичными данными прямо из файлов, специально созданных для того или иного приложения, поэтому данный способ в настоящее время не используется.

1.1.4. Модели данных

Инфологическая модель отображает реальный мир в некоторые понятные человеку концепции, полностью независимые от параметров среды хранения данных. Существует множество подходов к построению таких моделей: графовые модели, семантические сети, модель «сущность–связь» и т.д. [11]. Наиболее популярной из них оказалась модель «сущность–связь».

Инфологическая модель должна быть отображена в компьютеро-ориентированную даталогическую модель, «понятную» СУБД. В процессе развития теории и практического использования баз данных, а также средств вычислительной техники создавались СУБД, поддерживающие различные даталогические модели [1, 2, 8, 11].

Первоначально использовались иерархические даталогические модели. Простота организации, наличие заранее заданных связей между сущностями, сходство с физическими моделями данных позволяли добиваться приемлемой производительности иерархических СУБД на медленных ЭВМ с весьма ограниченными объемами памяти. Но если данные не имели древовидной структуры, то возникала масса сложностей при построении иерархической модели и стремлении добиться нужной производительности.

Сетевые модели также создавались для малоресурсных ЭВМ. Это достаточно сложные структуры, состоящие из «наборов» – поименованных двухуровневых деревьев. «Наборы» соединяются с помощью «записей-связок», образуя цепочки и т.д. При разработке сетевых моделей было выдумано множество «маленьких хитростей», позволяющих увеличить производительность СУБД, но существенно усложнивших последние. Прикладной программист должен знать массу терминов, изучить несколько внутренних языков СУБД, детально представлять логическую структуру базы данных для осуществления навигации среди различных экземпляров, наборов, записей и т.п. Один из разработчиков операционной системы UNIX сказал: «Сетевая база – это самый верный способ потерять данные».

Сложность практического использования иерархических и сетевых СУБД заставляла искать иные способы представления данных. В конце 60-х годов XX в. появились СУБД на основе инвертированных файлов, отличающиеся простотой организации и наличием весьма удобных языков манипулирования данными. Однако такие СУБД обладают рядом ограничений на количество файлов для хранения данных, количество связей между ними, длину записи и количество ее полей.

Сегодня наиболее распространены реляционные модели, которые будут подробно рассмотрены ниже.

Физическая организация данных оказывает основное влияние на эксплуатационные характеристики БД. Разработчики СУБД пытаются создать наиболее производительные физические модели данных, предлагая пользователям тот или иной инструментарий для поднастройки модели под конкретную БД. Разнообразие способов корректировки физических моделей современных промышленных СУБД не позволяет рассмотреть их в этом разделе.

1.1.5. Реляционная структура данных

В конце 60-х годов появились работы, в которых обсуждались возможности применения различных табличных даталогических моделей данных, т.е. возможности использования привычных и естественных способов представления данных. Наиболее значительной из них была статья сотрудника фирмы IBM д-ра Э. Кодда (Codd, E. F. A Relational Model of Data for Large Shared Data Banks. CACM 13: 6, June 1970), где, вероятно, впервые был использован термин «реляционная модель данных».

Будучи математиком по образованию, Э. Кодд предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение). Он показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как отношение – relation (англ.) [3, 7, 9].

Наименьшая единица данных реляционной модели – это отдельное атомарное (неразложимое) для данной модели значение данных. Так, в одной предметной области фамилия, имя и отчество могут рассматриваться как единое значение, а в другой – как три различных значения.

Доменом называется множество атомарных значений одного и того же типа. Так, в табл. 1 домен пунктов отправления (назначения) – множество названий населенных пунктов, а домен номеров рейса – множество целых положительных чисел.

Смысл доменов состоит в следующем. Если значения двух атрибутов берутся из одного и того же домена, то, вероятно, имеют смысл сравнения, использующие эти два атрибута (например, для организации транзитного рейса можно дать запрос «Выдать рейсы, в которых время вылета из Москвы в Сочи больше времени прибытия из Архангельска в Москву»). Если же значения двух атрибутов берутся из различных доменов, то их сравнение, вероятно, лишено смысла: стоит ли сравнивать номер рейса со стоимостью билета?

Отношение на доменах D1, D2, ..., Dn (не обязательно, чтобы все они были различны) состоит из заголовка и тела. На рис. 2 приведен пример отношения для расписания движения самолетов (см. табл. 1).

Заголовок (в табл. 1. он назывался *интерпретацией*) состоит из такого фиксированного множества атрибутов A1, A2, ..., An, что существует взаимно однозначное соответствие между этими атрибутами Ai и определяющими их доменами Di (i = 1, 2,..., n).

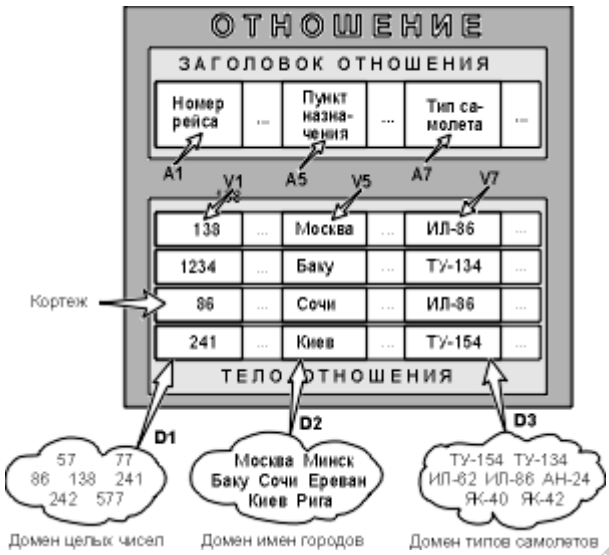


Рис. 2. Отношение с математической точки зрения
(Ai – атрибуты, Vi – значения атрибутов)

Тело состоит из меняющегося во времени множества *кортежей*, где каждый кортеж состоит в свою очередь из множества пар атрибут–значение (Ai:Vi), (i = 1, 2, ..., n), по одной такой паре для каждого атрибута Ai в заголовке. Для любой заданной пары атрибут–значение (Ai:Vi) Vi является значением из единственного домена Di, который связан с атрибутом Ai.

Степень отношения – это число его атрибутов. Отношение степени один называют унарным, степени два – бинарным, степени три – тернарным, а степени n – n-арным. Степень отношения «Рейс» (см. табл. 1) – 8.

Кардинальное число или ***мощность*** отношения – это число его кортежей. Мощность отношения «Рейс» равна 10. Кардинальное число отношения изменяется во времени в отличие от его степени.

Поскольку отношение – это множество, а множества по определению не содержат совпадающих элементов, то никакие два кортежа отношения не могут быть дубликатами друг друга в любой произвольно заданный момент времени. Пусть R – отношение с атрибутами A1, A2, ..., An. Говорят, что множество атрибутов K = (Ai, Aj, ..., Ak) отношения R является возможным ключом R тогда и только тогда, когда удовлетворяются два независимых от времени условия:

1. **Уникальность**: в произвольный заданный момент времени никакие два различных кортежа R не имеют одного и того же значения для Ai, Aj, ..., Ak.

2. **Минимальность**: ни один из атрибутов Ai, Aj, ..., Ak не может быть исключен из K без нарушения уникальности.

Каждое отношение обладает хотя бы одним возможным ключом, поскольку по меньшей мере комбинация всех его атрибутов удовлетворяет условию уникальности. Один из возможных ключей (выбранный произвольным образом) принимается за его первичный ключ. Остальные возможные ключи, если они есть, называются *альтернативными*.

Вышеупомянутые и некоторые другие математические понятия явились теоретической базой для создания реляционных СУБД, разработки соответствующих языковых средств и программных систем, обеспечивающих их высокую производительность, и создания основ теории проектирования баз данных. Однако для массового пользователя реляционных СУБД можно с успехом использовать неформальные эквиваленты этих понятий:

Отношение – Таблица (иногда Файл),

Кортеж – Строка (иногда Запись),

Атрибут – Столбец, Поле.

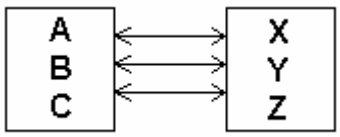
При этом принимается, что «запись» означает «экземпляр записи», а «поле» означает «имя и тип поля».

1.2. Связи данных

Традиционными средствами для представления характера взаимосвязей между парами связанных элементов данных являются ***отображения*** (двусторонние связи) и ***ассоциации*** (односторонние связи) [2].

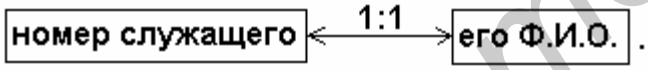
1.2.1. Отображения

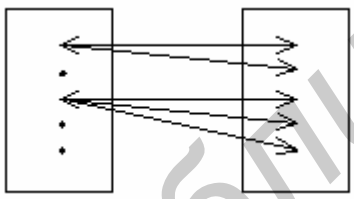
Рассмотрим основные типы отображений и их изображение на диаграммах.



Отображение 1:1. Один экземпляр элемента, от которого направлена связь, идентифицирует один экземпляр элемента, к которому направлена связь. Связь уникальна как слева направо, так и в обратном направлении.

Пример:

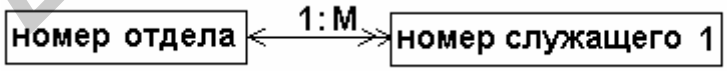


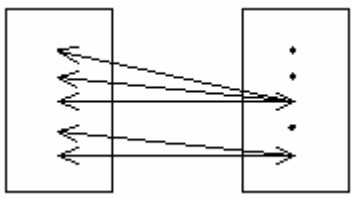


Отображение 1:М. Один экземпляр элемента с выходящей связью идентифицирует несколько экземпляров элемента данных, принимающих эту связь.

Подчеркнем, в направлении слева направо связь не является уникальной. Однако в обратном направлении любой экземпляр элемента, принимающий связь, идентифицирует только один экземпляр объекта, от которого направлена связь, т.е. в этом направлении связь уникальна.

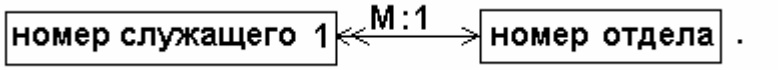
Пример:

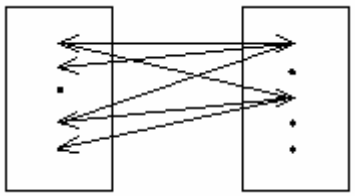




Отображение М:1. Это отображение подобно предыдущему с той лишь разницей, что слева направо связь уникальна, а в обратном направлении не является таковой.

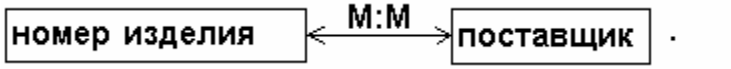
Пример:





Отображение M:M. Каждый экземпляр элемента данных с выходящей связью идентифицирует некоторое число экземпляров элемента данных, принимающих эту связь, и наоборот. Другими словами, идентификация не является уникальной в обоих направлениях.

Пример:



1.2.2. Ассоциации

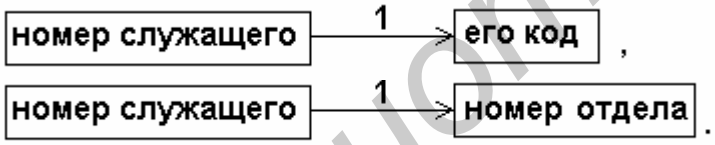
Для определения связи ключ–атрибут чаще всего используются ***ассоциации*** (односторонние связи между парами элементов данных), поскольку в большинстве случаев в БД связи от атрибутов к ключам не определяются. Ассоциации, определенные в обе стороны, представляют собой отображения.

Существует три типа ассоциаций:

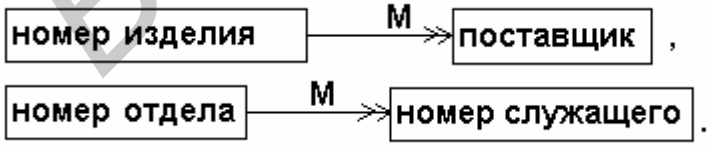
1) простая (тип 1);
2) сложная (тип M);
3) условная (тип C).

Простой называется ассоциация, в которой экземпляр элемента данных с выходящей связью идентифицирует один и только один экземпляр элемента данных, принимающих эту связь.

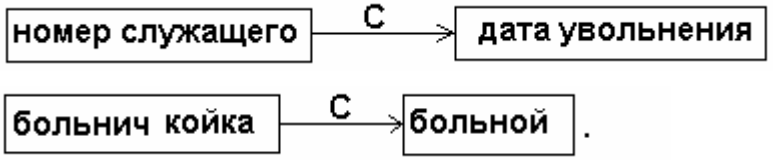
Например:



Сложной (или многозначной) называется ассоциация, в которой экземпляр элементов данных с выходящей связью идентифицирует несколько (два и более) экземпляров элементов данных, к которым эта связь направлена. Идентификация не обязательно является уникальной и может представлять многозначную зависимость. На диаграмме сложную ассоциацию будем отображать двойной стрелочкой и помечать символом M, например:

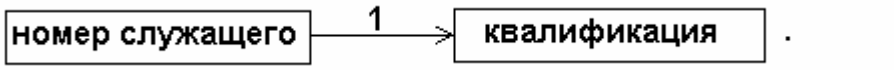


Условной называется ассоциация, в которой для элемента данных с выходящей связью может не существовать соответствующего элемента данных, принимающего эту связь, но если она существует, то относится к единственному экземпляру элементов данных. Идентификация, если она существует, является уникальной, например:



1.2.3. Реляционные ключи

В соответствии с [2] введем определение реляционных ключей. Простым ключом будем называть ключ, состоящий только из одного элемента. Его значение должно быть уникальным в отношении (таблице):

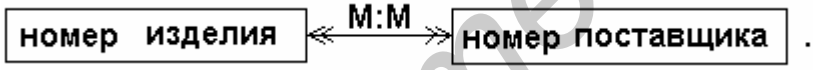


В отношении «номер служащего * квалификация» ключ состоит из единственного поля «номер служащего», а каждое значение данного поля уникальное в отношении и однозначно определяет аргумент «квалификация».

Составной ключ – это ключ, который содержит два или более элемента данных, каждый из которых необходим для однозначной идентификации объекта.

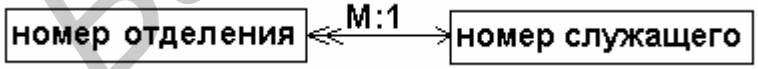
Будем различать:

Полностью составной ключ – это ключ, содержащий несколько атрибутов, между которыми существует отображение M:M, а одиночная ассоциация типа M или ассоциация вообще отсутствует. Атрибуты, составляющие такой ключ, не зависят друг от друга, ни один из них не является дополнительным квалификационным признаком другого атрибута, например:



Полностью составной ключ включает оба атрибута, т.е. ключ состоит из двух полей «номер_изделия» и «номер_поставщика». Этим достигается однозначная идентификация объекта.

Полусоставной ключ – ключ, содержащий несколько атрибутов и построенный с использованием отображения M:1 (которые подразумевают функциональную зависимость). Атрибуты такого ключа можно считать упорядоченными в том смысле, что каждый следующий атрибут в ключе является дополнительным квалификационным признаком предшествующих атрибутов. Например, для структуры



полусоставной ключ будет включать оба атрибута «номер_отделения» и «номер_служащего», что позволяет добиться однозначной идентификации записи в отношении.

1.3. Файловые системы

Переход к использованию централизованных систем управления файлами явился первым шагом в создании СУБД. Такая система берет на себя распределение внешней памяти, связывание имен файлов с соответствующими адресами их размещения во внешней памяти и обеспечение доступа к данным.

Первая распространенная файловая система была разработана фирмой для серии компьютеров IBM-360, в которой поддерживались как чисто последовательные, так и индексно-последовательные файлы. Реализация системы базировалась на возможностях появившихся к этому времени магнитных дисковых устройств и контроллеров управления дисковыми устройствами.

Магнитные диски представляют собой съемные пакеты магнитных пластин (поверхностей), между которыми на одном рычаге двигается пакет магнитных головок. Шаг движения пакета головок является дискретным, и каждому положению пакета головок логически соответствует цилиндр магнитного диска. На каждой поверхности цилиндр формирует дорожку, так что каждая магнитная поверхность содержит число дорожек, равное числу цилиндров. При разметке магнитного диска (действие, предшествующее его использованию) каждая дорожка делится на одно и то же количество блоков таким образом, что в каждый блок можно записать одно и то же максимальное число байтов. Таким образом, для выполнения обмена с магнитным диском на уровне аппаратуры нужно указать номер цилиндра, номер поверхности, номер блока на соответствующей дорожке и число байтов, которое нужно записать или прочитать от начала этого блока.

Распространены два основных подхода.

При первом подходе, применяемом, например, в файловых системах операционных систем фирмы DEC RSX и VMS, пользователи представляют файл как последовательность записей (байтовых последовательностей постоянного или переменного размера). Записи можно читать или записывать последовательно или позиционировать файл на запись с указанным номером.

Второй подход, ставший распространенным вместе с операционной системой UNIX, состоит в том, что любой файл представляется как последовательность байтов. Из файла можно прочитать указанное число байтов либо начиная с его начала, либо предварительно произведя его позиционирование на байт с указанным номером. Аналогично можно записать указанное число байтов в конец файла, либо предварительно выполнив позиционирование файла (базовое блочное представление файла в ОС UNIX скрыто от пользователей).

Все современные файловые системы поддерживают многоуровневую систему именования файлов за счет поддержания во внешней памяти дополнительных файлов со специальной структурой – каталогов. Каждый каталог содержит имена каталогов и/или файлов, содержащихся в данном каталоге. Следовательно, полное имя файла состоит из списка имен каталогов плюс имя файла в каталоге, непосредственно указывающем на данный файл. Разница между способами именования файлов в разных файловых системах состоит в том, с чего начинается эта цепочка имен.

1.3.1. Организация представления данных в файлах

Наиболее простую, широко известную и, видимо, появившуюся исторически первой организацию данных в файловых системах представляет последовательный файл, в котором доступ к записи с номером N возможен только после последовательного считывания и пропуска N–1 предшествующих записей. Примером такой организации, в частности, могут служить файлы операционной системы MS/DOS. Файлы с последовательной организацией статичны, требуют значительных затрат при их изменениях, неэффективны при работе в интерактивных режимах.

Основной недостаток файловых систем – последовательный доступ к записям был устранен с появлением файлов с возможностью произвольного доступа к записям (способ, обеспечивающий прямое обращение к конкретной записи) и особенно с появлением индексно-последовательной организации файлов, поддерживающей и последовательный, и прямой доступ к данным.

Записи в таких файлах упорядочиваются по значению первичного ключа, что позволяет его использовать при обновлении большого их количества последовательно и напрямую, при реализации запросов пользователя.

Индексно-последовательная и прямая (с прямым доступом) организация файлов базируется на физических возможностях МД.

Таким образом, для доступа при считывании или корректировке при первом обращении к диску прочитывается номер цилиндра и дорожки (из таблиц индекса), а затем при втором считывается нужная запись.

Такая организация не могла удовлетворить пользователя, активно работающего с отдельными записями (например с системой бронирования мест).

1.3.2. Файлы с прямым доступом

Для интерактивной работы с данными используется прямой доступ, в котором применяются так называемые функции кэширования вместо поддержки индексов.

Алгоритм кэширования – это процедура вычисления адреса записи на основании некоторого поля записи, обычно ключа.

Память на МД определяется в виде сегмента, состоящего из блоков. Под размещение файла рассчитывается необходимое количество блоков с некоторым запасом, скажем +20 % к нужному объему. Чем больший объем отведен под файл, тем меньше будет возникать конфликтов при размещении записей и быстрее осуществляться доступ. Отношение реально необходимого объема к выделенному называется коэффициентом загрузки.

Для размещения (и в дальнейшем редактирования) записи используются специальные вычислительные итерационные алгоритмы, равномерно распределяющие записи в блоках внутри сегмента. Если для размещения записи получен номер занятого блока, то алгоритм пересчитывается и запись получает новое значение номера блока. Аналогичным образом происходит организация доступа при чтении конкретной записи. Если по рассчитанному номеру блока располагается не искомая запись, то алгоритм выполняется повторно.

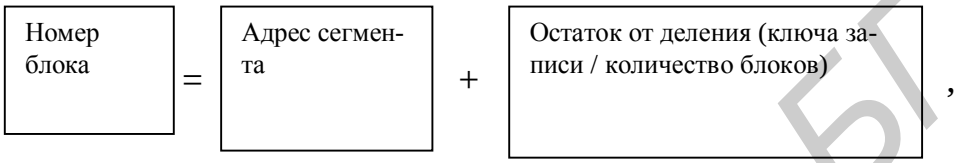
Для разрешения таких конфликтов используется также указатель от первой записи, получаемой при работе алгоритма, к последующей, образуя таким образом цепочки указателей, используя которые можно найти заданную запись.

1.3.3. Пример функции кэширования

Пусть необходимо заполнить диск, который может хранить блоки длиной 1000 байт файлом из 500 записей, каждая по 100 байт. Тогда для хранения необходимо 500*100:1000 = 50 блоков.

Поскольку кэш-функций, которые могли бы решить задачу такого размера, на сегодняшний день нет, то пусть объем будет увеличен на 20 %, т.е. 50*1,2 = 60 блоков. Тогда пусть для сегмента с информацией отведен сегмент диска, например с адреса 3000.

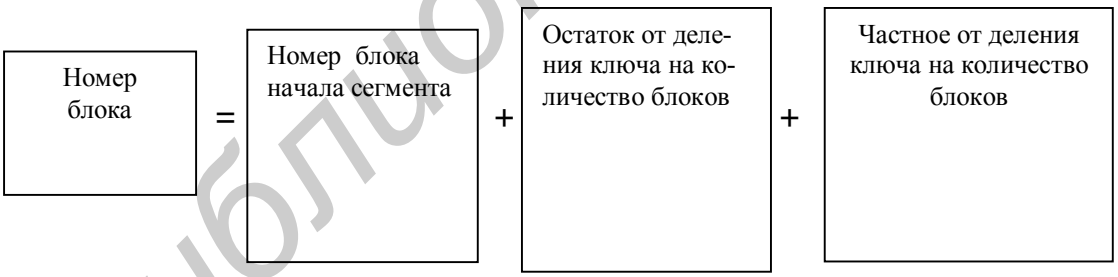
В качестве кэш-функции можно предположить алгоритм



например если ключ записи 1900, то 3000 + остаток (1900/60) = 3000 + 40.

Остаток от деления всегда размещается в интервале от 0 до 59, поэтому вычисляемый адрес всегда попадает в выделенный сегмент.

В случае если полученный номер блока указывает на уже занятый блок, но в нем есть место для записи, то она располагается в блоке. В противном случае алгоритм необходимо доопределить, например, используя метод квадратичных данных, в котором рассматривается следующее вычисление:



* J2 + J,

где J – от 0 до исчезновения конфликта.

Приведенные алгоритмы кэширования эффективны для БД небольшого объема. С увеличением объема возрастают задержки из-за конфликтов.

В настоящее время существуют более эффективные алгоритмы кэширования, используемые для больших БД (с резервированием памяти, выделением дополнительного объема по мере роста БД, с так называемыми динамическими функциями кэширования (например, растяжимым кэшированием, которое сли-

вает вместе или разбивает блоки при расширении базы данных или ее уменьшении)).

1.3.4. Основа представления данных в БД

Основной принцип связывания одной физической записи БД с другой состоит в использовании указателей (здесь указатель понимается как элемент данных записи, который содержит физический адрес связанной с ней другой записи).

Такие цепочки указателей называются связанным списком.

Например: пусть имеются следующие данные:

ФИО	Должность	Должность-указатель
1. Иванов	бухгалтер	4
2. Петров	инженер	7
3. Сидоров	слесарь	0
4. Губанов	бухгалтер	6
5. Карпов	механик	0
6. Смоленский	бухгалтер	0
7. Ершов	инженер	0

Здесь графа «Должность-указатель» содержит указатель на номер следующей записи с таким же значением поля «Должность». Если это последняя запись с таким значением, то в соответствующую ячейку записывается 0.

Чтобы отобрать данные о бухгалтерах БД, необходимо либо смотреть значения всех записей в таблице, либо отсортировать таблицу по полю и должности и выбрать полученный сегмент, связанный список указателей. Используя связанный список указателей, можно отобрать такие записи без дополнительного просмотра. Такие списки можно организовывать для всех полей, используемых в запросах. Однако при значительных объемах файлов проход по цепочке указателей может занимать много времени, особенно поддержание списков на операциях вставки и удаления записей.

Альтернативой является инвертированный список, – где отдельный файл (индекс) состоит из двух типов элементов данных: значение величины и все адреса записей, содержащие это значение.

Связанные и инвертированные списки используются во многих БД.

Системы, основанные на инвертированных списках (индексах), очень близки к реляционной базе данных. Списки в основном используют инвертированные системы для организации работы с данными.

Записи в системах на инвертированных списках упорядочиваются в определенной последовательности в таблицы. Для каждой таблицы определяется произвольное число ключей, для которых строятся индексы. Индексы автома-

тически поддерживаются системой, но явно (в отличие от реляционных БД) используются пользователем при построении запросов.

В запросах используются два класса операторов:

1) операторы, устанавливающие адрес искомой записи (например, первой или искомой относительно предыдущей);

2) операторы над адресуемыми записями (обновить, установить после или перед, добавить, удалить).

Поддержка целостности, как правило, обеспечивается не системой, а прикладными программами.

2. ЛАБОРАТОРНАЯ РАБОТА № 1
«Проектирование модели БД с использованием СУБД Access . Создание запросов с использованием технологии QBE (Query By Example) и команд языка SQL»

Цель работы:

приобрести навыки проектирования физической модели структуры БД и создания SQL-запросов в соответствии с поставленной задачей, используя СУБД Access.

2.1. Общие сведения о MS Access

MS Access представляет собой широко распространенную систему проектирования баз данных, функционирующую в среде Windows. К достоинствам системы можно отнести удобную для пользования графическую оболочку, поддержку высокоэффективной реляционной модели данных, развитые инструментальные средства создания формуляров и отчетов, возможность включения изображений и OLE-объектов, возможность экспорта данных в другие базы данных.

Работу с Access рассмотрим на примере создания базы данных для решения поставленной задачи.

Пример. Создать базу данных «Склад», содержащую сведения о товарах, поставщиках, заказах, а также сведения о предприятии.

Решение. Для того чтобы создать новую базу данных, после запуска СУБД Access в окне начального диалога указать ***новая база данных*** и подтвердить создание новой базы данных нажатием кнопки **Ok**. В результате отобразится окно **Файл новой базы данных,** в котором предлагается указать путь и имя файла новой базы данных. По умолчанию файлу присваивается имя db1, и размещается он в папке «Мои документы», также принятой по умолчанию.

Переименуем файл, указав в поле *Имя файла* новое имя – «Склад». Затем остается подтвердить создание нового файла базы данных, нажав на кнопку

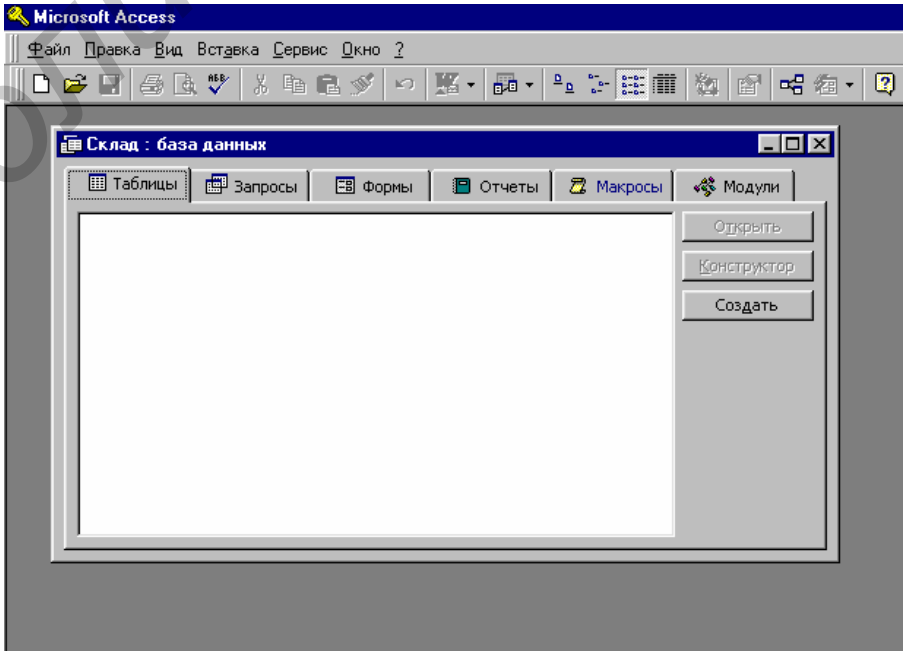


Рис. 3. Окно базы данных

Создать. В результате в указанной папке («**Мои документы**») создан файл БД «Склад». При этом в окне Microsoft Access автоматически появляется окно новой базы данных – **Склад: база данных** (рис. 3).

Это окно предназначено для хранения, создания и редактирования различных объектов базы данных. В окне новой базы данных автоматически выбирается вкладка *Таблицы*. Дальнейшие наши действия будут происходить на этой вкладке окна базы данных.

Для создания новой таблицы воспользуемся кнопкой **Создать** в окне базы данных. В результате выдается окно **Новая таблица** (рис. 4), в котором требуется указать способ создания новой таблицы. Выберем *Конструктор* для создания структуры таблицы.

В режиме конструктора (рис. 5) опишем элементы структуры создаваемой таблицы – присвоим соответствующие имена полей, выберем их типы и, если необходимо, укажем их размеры. Имена полей в Access могут быть длиной до 64 символов и содержать как латинские буквы, так и символы кириллицы.

Поэтому для имен полей можем использовать названия соответствующих характеристик товара. Тип поля будем выбирать в соответствии с информацией, которую предполагается вносить в данное поле. Длину поля выберем, исходя из максимально возможной длины данных, которые могут быть введены в это поле.

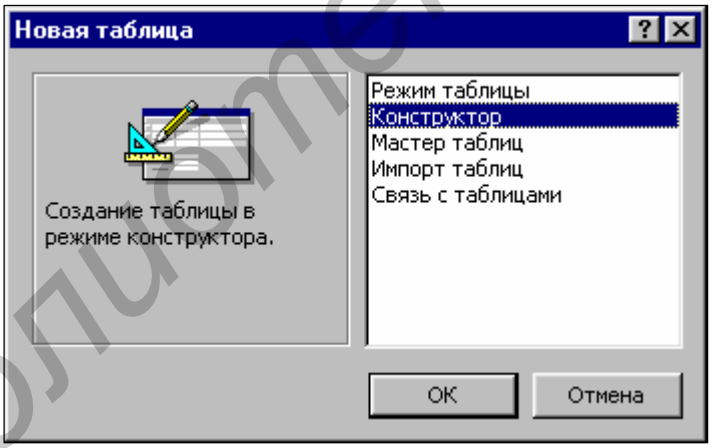


Рис. 4. Выбор режима создания новой таблицы

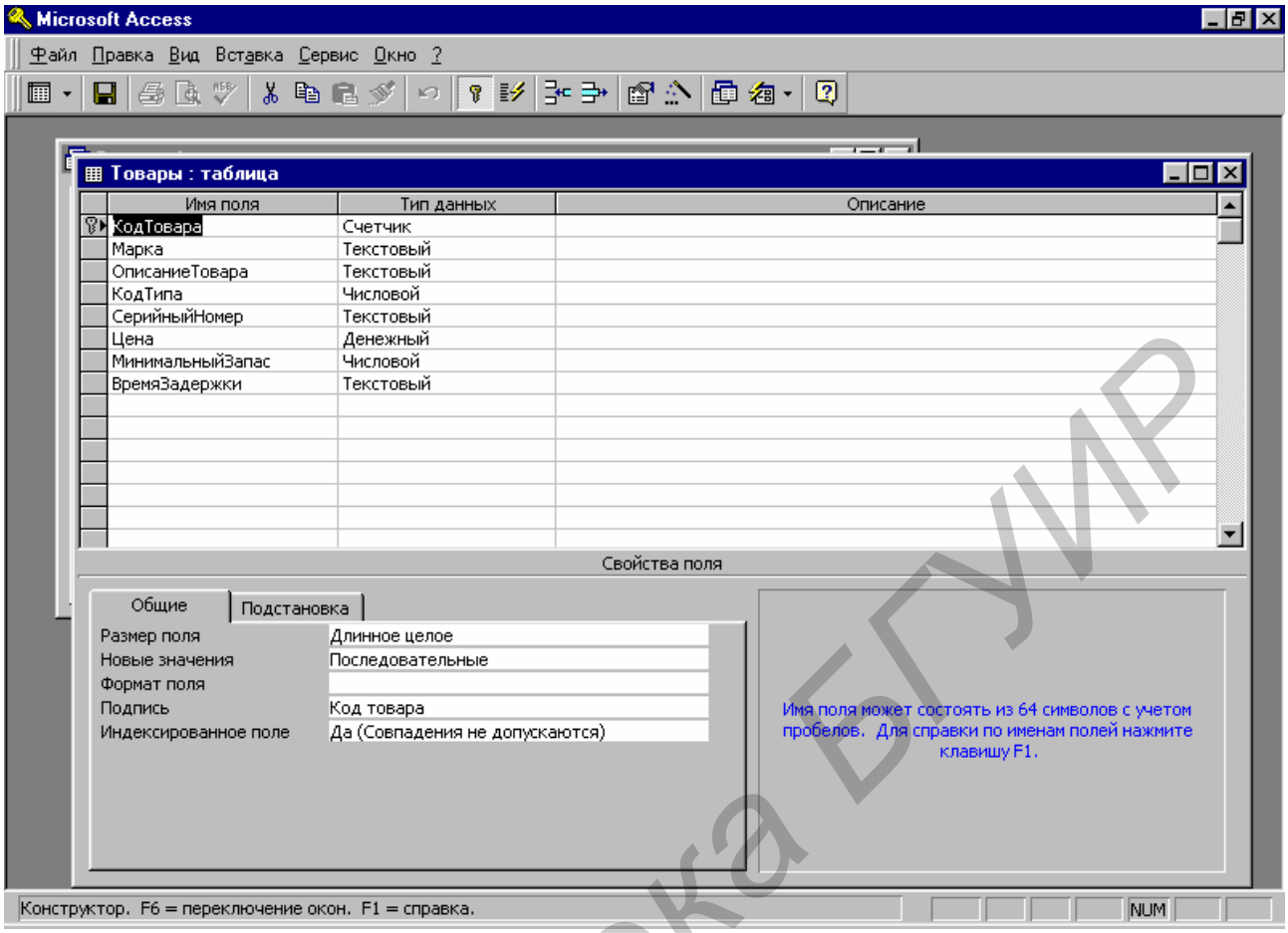


Рис. 5. Окно конструктора таблицы

После завершения работы с конструктором таблиц можно сразу же перейти к вводу информации в созданную таблицу. Для этого можно воспользоваться кнопкой **Вид** панели инструментов **Конструктор таблиц** либо меню **Вид – Режим таблицы**. Можно также закрыть окно Конструктора, сохранив макет таблицы, а затем нажать кнопку **Открыть** в **Окне базы данных** (рис. 3).

Результатом проделанной работы является таблица «Товары», содержащая следующие сведения:

Код товара	Марка	Описание	Код типа	Серийный	Мин. запас	Срок, дней
1	Цейлонский чай		кондитерские		10	10
2	Тибетское ячменное		напитки		25	10
3	Анисовый сироп		приправы		25	10
4	Индейская приправа	Индейская	приправы		0	10
5	Смесь Антона Гумбо	Смесь Гумбо	приправы		0	10

2.2. Создание запросов

В больших базах данных часто возникает проблема поиска необходимой информации (или отбора записей), удовлетворяющей определенным критериям. Задача поиска информации является одной из самых трудоемких и во многих случаях – одной из главных.

Для решения этой задачи предназначен механизм запросов. Этот механизм является стандартным и применяется почти одинаково во всех (или в подавляющем большинстве) СУБД реляционного типа. Он представляет собой

набор команд на языке SQL, определяющих критерий отбора записей в реляционной таблице. Таким образом, чтобы получить необходимую информацию из базы данных, следует записать соответствующие команды на языке SQL или, иначе говоря, – сформировать запрос.

Многие СУБД обладают механизмом автоматизации проектирования запросов. Чаще всего запрос формируется на специальном бланке. Такой метод формирования запроса называется QBE (Query By Example – Запрос по образцу). В MS Access процесс создания запроса подобен процессу создания таблиц. Для того чтобы начать проектирование нового запроса, необходимо перейти на вкладку **Запросы** окна базы данных и нажать кнопку **Создать**. В результате появится окно диалога **Новый Запрос** (рис. 6), аналогичное окну **Новая таблица**. В этом окне будет предложено выбрать один из вариантов создания запроса.

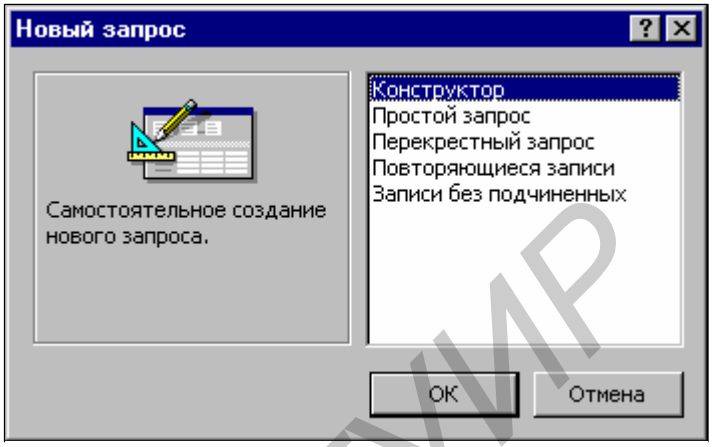


Рис. 6. Окно нового запроса

Запрос можно создать самостоятельно при помощи **Конструктора** или использовать готовый. Назначение каждого из режимов поясняется в левой части окна при указании мышью. Как показывает практика, большинство запросов создается с помощью Конструктора. По этой причине мы более подробно рассмотрим этот способ.

После подтверждения запуска **Конструктора** открывается бланк запроса

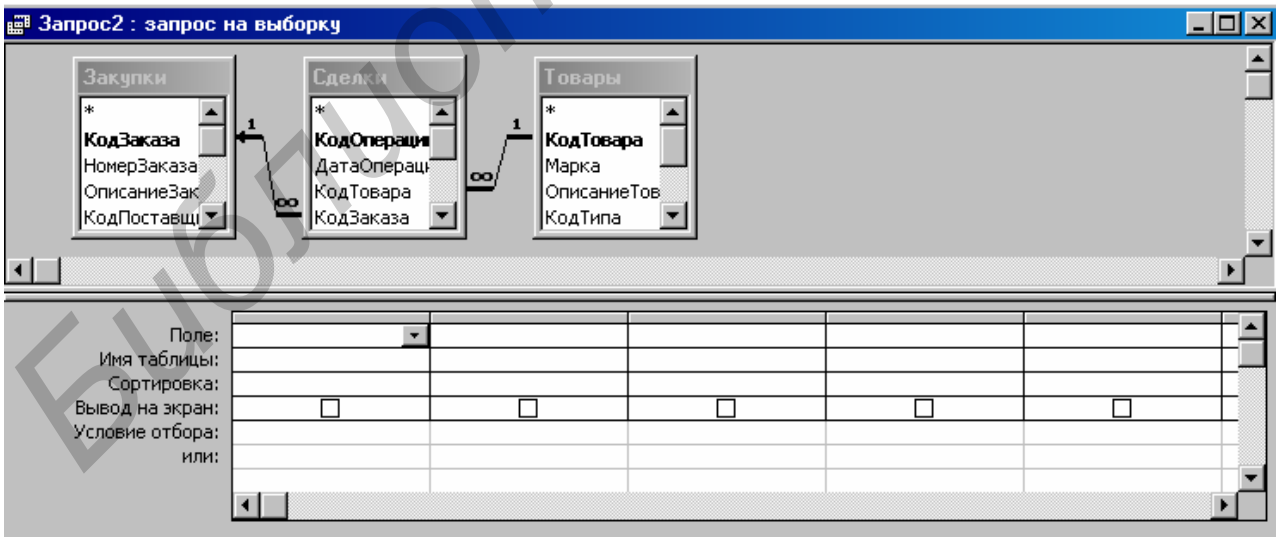


Рис. 7. Конструктор запроса

(рис. 7) и окно **Добавление таблицы** (рис. 8). В этом окне пользователю предоставляется возможность выделить одну или несколько таблиц, участвующих в запросе (выделение таблиц осуществляется аналогично выделению файлов в

операционной системе Windows). Чтобы выделенные таблицы поместить в запрос, следует нажать кнопку **Добавить**. Указанные таблицы отображаются в верхней части окна **Конструктора Запроса** вместе со всеми связями, если они имеются.

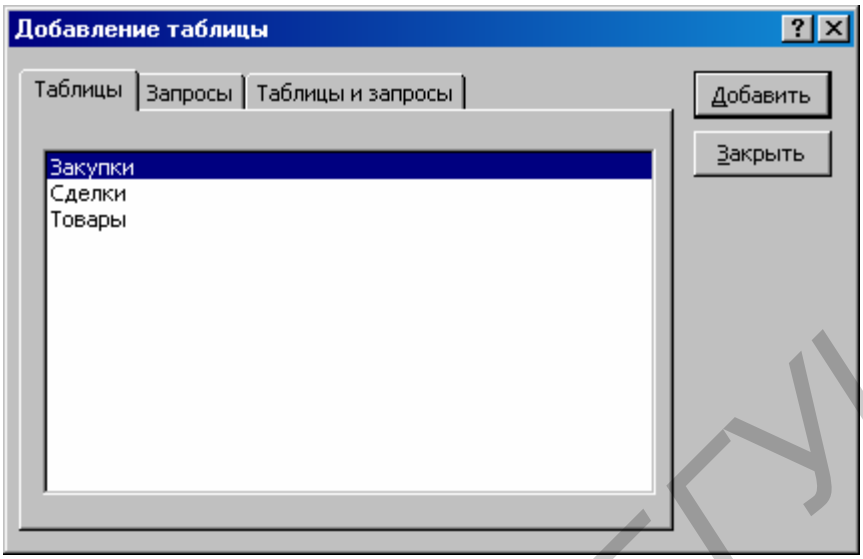


Рис. 8. Окно добавления таблиц

Теперь чтобы сформировать запрос, необходимо в бланке **Конструктора** сформировать образец. Он составляется из полей соответствующих таблиц, ло-

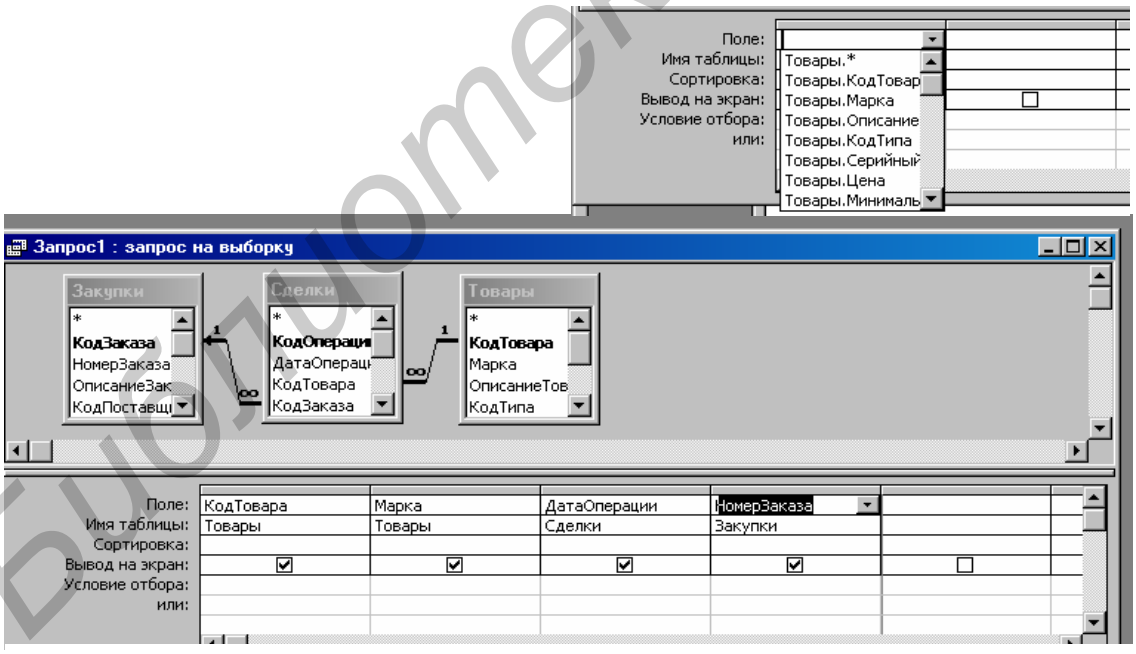


Рис. 9. Работа с бланком запроса в режиме конструктора

гических условий и выражений. В простейшем случае, если в бланк помещаются только некоторые поля из таблиц, мы получаем запрос на выборку определенных полей из одной или нескольких таблиц. Указать наименование поля,

значения которого должны выводиться или участвовать в запросе, можно одним из следующих способов (рис. 9):

- Выбрать поле из списка **Поле** в окне **Конструктора**.
- В таблице из верхней части окна **Конструктора** выделить необходимые поля и перетащить их мышью в ячейку **Поле** бланка запроса в нижней части окна – выбранные поля помещаются в бланке запроса последовательно, начиная с той ячейки, где была освобождена кнопка мыши.
- Если в образец запроса необходимо поместить все поля из таблицы, то можно одним из вышеуказанных способов поместить в бланк символ * («звездочка»).

Варианты заданий

1. Создать базу данных «Сеть магазинов игрушек».

2. Создать базу данных «Покупка автомобиля по паспорту».

3. Создать базу данных «Управление таксопарком».

4. Создать базу данных «Прокат видеокассет».

5. Создать базу данных «Расчет з\п работников».

6. Создать базу данных «Учебный процесс в школе».

7. Создать базу данных «Продажа авиабилетов».

8. Создать базу данных «Сведения о безработных».

9. Создать базу данных «Сведения о студентах».

10. Создать базу данных «Сведения о животных в зоопарке».

11. Создать базу данных «Учет пациентов в поликлинике».

12. Создать базу данных «Сведения о работниках на предприятии».

13. Создать базу данных «Продажа проездных билетов».

14. Создать базу данных «Расчет прибыли на предприятии».

15. Создать базу данных «Учет товаров на складе».

3. Проектирование реляционной базы данных

3.1. Основные этапы проектирования

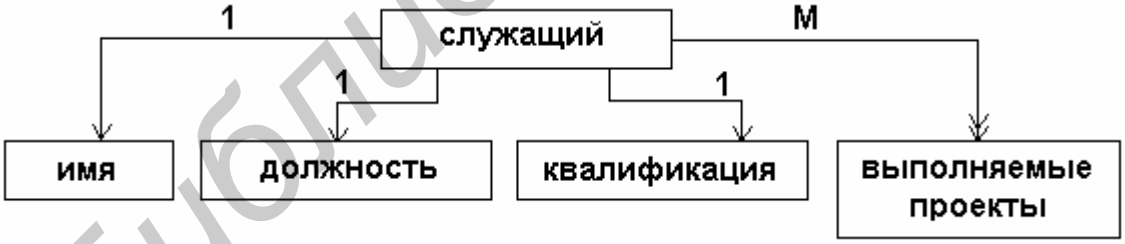
Проектирование базы данных в общем случае включает три самостоятельных этапа [3]: концептуальное, логическое и физическое проектирование.

Концептуальное проектирование. На этом этапе разрабатывается концептуальная модель (модели предметной области). Такая модель служит средством общения между различными пользователями и разрабатывается без учета особенностей физического представления данных и выбора СУБД. С помощью концептуальной модели представляют сущности предметной области и взаимосвязи между ними (используются взаимосвязи 1:1, 1:M, M:M). Проектирование концептуальной модели основывается на анализе решаемых задач по обработке данных.

Задачей концептуального проектирования является определение информационных потребностей, а также процессов и данных, необходимых для решения поставленных задач и достижения цели проектирования. Результатом выполнения является концептуальное представление (модель) данных, которое разрабатывается на основе обобщения пользовательского и локального представлений (моделей).

Пользовательское представление можно понимать как описание совокупности входных и выходных документов с данными, формат которых создает удобную функциональную среду для решения каждой отдельной задачи (функции) с точки зрения пользователя.

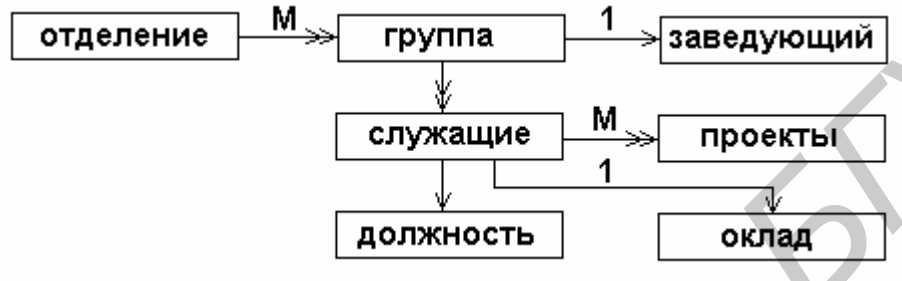
Локальное представление создается на основе пользовательского и описывает функциональную среду решения тех же задач с точки зрения проектировщика БД. Обычно оно определяется множеством бинарных отношений между элементами данных (т.е. пары элементов данных и связывающих их типов ассоциаций), которые могут быть изображены в виде граф, матриц или связанных списков [2], например:



Однозначное соответствие между пользовательскими и локальными представлениями не всегда существует. Так, пользователь может включить в свою модель данные, которые нет необходимости хранить, поскольку они могут быть получены на основании других данных, имеющихся в БД. Например, элемент «заработная плата» нет смысла хранить в БД, т.к. он может быть вычислен на основе оклада и количества отработанных дней. Хранение такого поля в таблицах базы может привести к появлению недостоверных данных (например, если при изменении количества отработанных дней или оклада, поле заработная плата осталось не перерасчитанным). С другой стороны, в БД часто приходится хранить данные, не относящиеся к пользовательскому представле-

нию, а именно: налоговые таблицы, календарь, статистические данные и другие информационно-справочные материалы.

Концептуальная модель создается как обобщение пользовательских и локальных представлений и включает в себя совокупность описания всех данных и требований к ним, полученных на предыдущих этапах. Она не связана структурно с сетевыми иерархическими или реляционными БД и может быть выполнена в виде композиционной модели как объединения совокупности локальных представлений. Композиционная модель служит основой логического проектирования внутренней структуры данных для выбранной СУБД. Кроме того, она используется для оценки возможности и необходимости включения дополнительных внешних представлений. Композиционную модель обычно изображают в следующем виде:



Локальные и композиционные модели, выполненные в виде граф, матриц или связанных списков с описанием элементов, форматов, возможных связей и ключевых полей облегчают взаимодействие пользователя и проектировщика и позволяют проектировщику уточнить, отредактировать, устранить ошибки и дублирование информации.

3.1.1. Этапы концептуального проектирования данных

1. Информационное обследование.
2. Анализ существующих данных:
   2.1. изучение существующих форм документов, отчетов, имеющихся файлов и программ. Опрос служащих реализуется при помощи специальных форм, которые, например, могут содержать следующие поля:
      1) имя и описание объекта данных;
      2) элементы данных:
         a) имя и описание;
         b) источник;
         c) атрибуты (тип, значение);
         d) использование;
         e) ограничение безопасности;
         f) степень важности;
         g) взаимосвязи (использование совместно с другими);
      3) продолжительность хранения.

На этапе концептуального проектирования осуществляется сбор, анализ и упорядочивание (редактирование) требований к данным, построение локальной и композиционной модели данных.

I.**Логическое проектирование.** На этом этапе осуществляется выбор модели СУБД и описание данных при помощи ее логических структур (таблиц, файлов, списков и т.д.).

Процесс логического проектирования БД включает в себя выбор конкретной модели СУБД и отображение концептуального представления в логическую модель, основанную на структурах, характерных для выбранной СУБД. Для реляционных БД это разработка структуры записей данных, организация их в наборы, определение связей между наборами и полей для их реализации (ключевых полей), оптимизация создаваемой модели БД (устранение избыточности и дублирования информации с целью обеспечения достоверности и непротиворечивости данных).

II.**Физическое проектирование.** На этапе физического проектирования решаются вопросы, связанные с производительностью системы, определяются структуры хранения данных на физических носителях, методы доступа.

Физическое проектирование БД заключается в расширении ее логической модели характеристиками, которые необходимы для определения способов физического хранения и использования БД, типа устройств для ее хранения, методов доступа, размеров логических единиц, объема памяти и эффективности, правил обновления и сопровождения БД и т.п. За пользователем остается и право решения вопроса об объединении таблиц.

Основными критериями, которым должна удовлетворять спроектированная БД, будем считать обеспечение функциональных требований приложений, целостности и согласованности информации. Это значит, что методы хранения данных и их использования должны гарантировать защиту данных от потери и некорректных действий, обеспечивать секретность и защиту от несанкционированного доступа. Дублирующиеся данные должны соответствовать одному уровню обновления, чтобы обеспечить достоверность данных для пользователя. База данных должна обладать способностью к расширению и модификации.

3.2. Нормализация отношений

Уточним, что под реляционным отношением (таблицей) будем понимать двумерный массив типа «объект – признак», обладающий следующими свойствами [5]:

- все столбцы (колонки) в таблице однородные, т.е. все элементы одного столбца имеют одинаковую природу;
- столбцы имеют уникальные имена;
- все атрибуты должны быть простыми, неделимыми, т.е. отношения не могут иметь в качестве компонент другие отношения;
- в таблице нет одинаковых строк (каждая строка имеет уникальный идентификатор – набор полей, значения которых однозначно определяют (идентифицируют) данную строку из множества других строк отношения);
- все строки таблицы имеют одну и ту же структуру, т.е. одно и то же количество атрибутов с соответственно одинаковыми именами;

- в операциях с таблицей ее строки и столбцы могут просматриваться в любом порядке и любой последовательности безотносительно к их информационному содержанию и смыслу.

В качестве иллюстрации отношения можно привести список сотрудников кафедры, например следующей структуры:

Фамилия И. О.	Дата рождения	Должность	Оклад
Иванов И. И.	6/6/1966	ст. н. с.	300000
Петров П. П.	11/11/1953	преп.	250000
Сидоров С. С.	8/8/1948	ст. преп.	320000
Ярмолов Я. Я.	10/10/1950	доцент	500000

Исходя из введенного определения отношения, можно сказать, что каждая строка значений (кортеж) в таблице описывает конкретный объект из множества подобных объектов, а каждый столбец таблицы соответствует некоторому атрибуту этого множества подобных объектов.

Рассмотрим, как отобразить результаты концептуального проектирования на множество отношений (таблиц) реляционной СУБД.

Такой переход реализуется как структурное преобразование требований к данным, представленным в виде композиционной модели, в отношения в «третьей нормальной форме», которые должны обеспечить пути доступа к данным, необходимые для выполнения функций обработки. Аппарат преобразований, названный «нормализацией отношений», был разработан Е. Ф. Коддом [4]. В нем определены различные нормальные формы, каждая из которых ограничивает типы допустимых зависимостей отношений. Кодд выделил три нормальные формы:

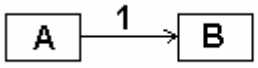
- нормализованные отношения (первая);
- полная функциональная зависимость (вторая);
- взаимная независимость атрибутов (третья).

Позже стали выделять еще две – четвертую и пятую нормальные формы. Они предназначены для устранения независимых многозначных зависимостей [5].

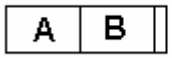
Нормализованные отношения

Отношение называется ***нормализованным****, если каждый из его атрибутов является неделимым (однозначным).*

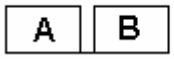
Такому определению удовлетворяет ассоциация типа 1:



Ее можно структурно представить в виде отношения (таблицы) с ключом A и атрибутом B. В схеме обозначения (см. ниже) двойная линия отделяет ключ от атрибутов:

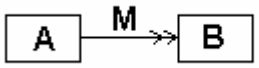


Например, структура отношения, имеющего два поля «код служащего» и



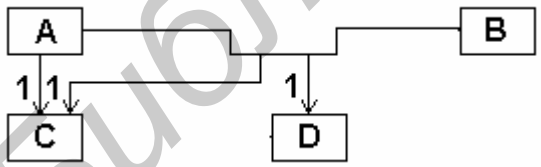
«ФИО», может быть представлена схемой:

Нормализованное отношение, построенное по ассоциации типа M, отображает многозначную связь и структурно может быть представлено в виде отношения с элементами A и B, входящими в полностью составной первичный ключ. Этим достигается обеспечение уникальности значений ключа, т.е. отношение можно представить в виде схемы:



Полная функциональная зависимость достигается устранением зависимости атрибутов составного ключа от какого-либо подмножества этого ключа.

Пример. На представленном рисунке атрибут C зависит и от A и от A*B. В данном случае C необходимо поместить в одно из отношений вместе с идентификатором соответствующего подмножества.

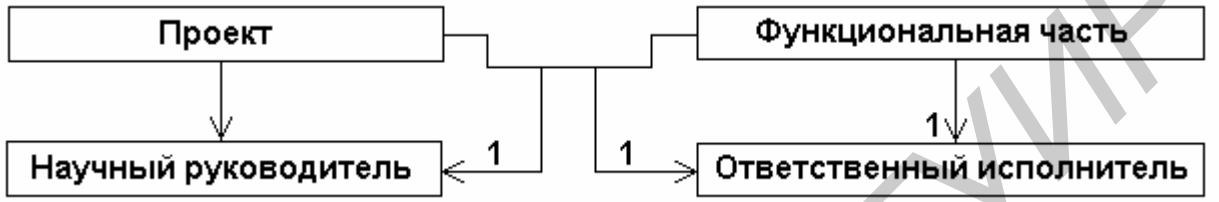


В результате будут созданы два отношения:

Данное решение можно проиллюстрировать на следующем примере. Пусть в некотором проектном институте разрабатываются три проекта. Руководство каждым из проектов осуществляет научный руководитель проекта. Структурно проекты могут состоять из одной или нескольких частей, каждая из которых возглавляется ответственным исполнителем. Пусть пользовательское представление данных описывается следующей таблицей:

Проект	Функцион. часть	Науч. руковод.	Ответств. исполнит.
01	001	Петров П. П.	Ярмохин Я. Я.
01	002	Петров П. П.	Якушев М. М.
02	001	Иванов И. И.	Игнатов И. И.
03	001	Сидоров С. С.	Якунин Я. Я.
03	002	Сидоров С. С.	Бакунин Б. Б.
03	003	Сидоров С. С.	Ганкин Г. Г.

Соответствующее локальное представление можно описать схемой :



В данной схеме в составной ключ входят поля «проект» и «функциональная часть». Причем атрибут «научный руководитель» зависит от подмножества составного ключа (поля «проект»). В соответствии с рассмотренным методом приведения ко второй нормальной форме необходимо устранить частичную зависимость, в результате чего приведенная схема может быть реализована двумя следующими отношениями:

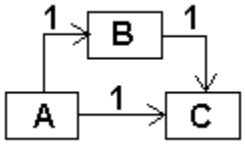
и

Соответствующие им таблицы будут представлены следующим образом:

Проект	**Научный руководитель**
01	**Петров П. П.**
02	**Иванов И. И.**
03	**Сидоров С. С.**

Проект	Функцион. часть	Ответствен. исполнитель
01	001	Ярмохин Я. Я.
01	002	Якушев М. М.
02	001	Игнатов И. И.
03	001	Якунин Я. Я.
03	002	Бакунин Б. Б.
03	003	Ганкин Г. Г.

Взаимная независимость атрибутов достигается устранением транзитивной зависимости в композиционной модели, например:



.

Атрибут С одновременно функционально зависит от В, который в свою очередь функционально зависит от А. Тогда ассоциация от А к С является транзитивной (избыточной) и должна быть удалена. В результате получим отношения:

.

Например, в отношении:

Личный номер	Должность	Оклад
131	Доцент	300
132	Ст. преподаватель	250
133	Ст. преподаватель	250
134	Ст. преподаватель	250

атрибут «оклад» зависит от атрибута «должность», который является зависимым от ключевого поля «личный номер». Устраняя избыточную ассоциацию типа 1 от «личный номер» к «оклад», получим следующие отношения:

Личный номер	Должность
131	Доцент
132	Ст. преподаватель
133	Ст. преподаватель
134	Ст. преподаватель

Должность	Оклад
Доцент	300
Ст. преподаватель	250

3.3. Построение реляционных отношений в третьей нормальной форме

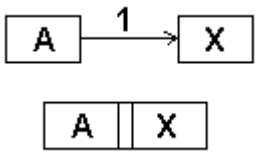
Построение реляционных отношений в третьей нормальной форме можно описать при помощи следующего алгоритма:

1. Композиционная модель анализируется с целью выявления функциональных и транзитивных зависимостей, которые затем устраняются.

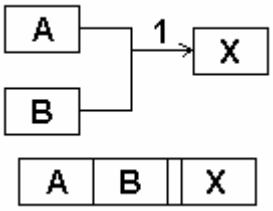
2. В качестве ключей выбираются элементы, из которых направлена хотя бы одна ассоциация типа 1.

3. В каждом отношении, построенном по выбранному ключу, атрибутами становятся элементы, к которым направлены ассоциации типа 1, выходящие из ключа.

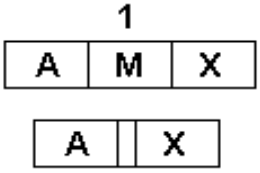
Следующие схемы демонстрируют возможные варианты связей между элементами данных и их реализацию в виде реляционных отношений:



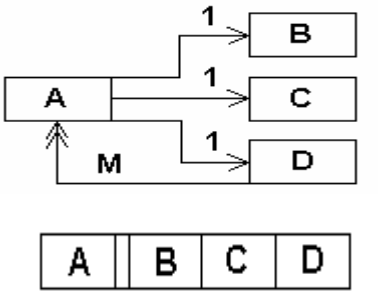
– в качестве ключа в формируемом отношении выбирается элемент A, а атрибутом является элемент данных X, принимающий связь типа 1;



– атрибут X зависит от совокупности полей A и B, образующих составной ключ в отношении;



– в отношении, созданном для реализации указанной схемы, ключом является элемент A, с выходящей ассоциацией типа 1, а атрибутом элемент данных X;

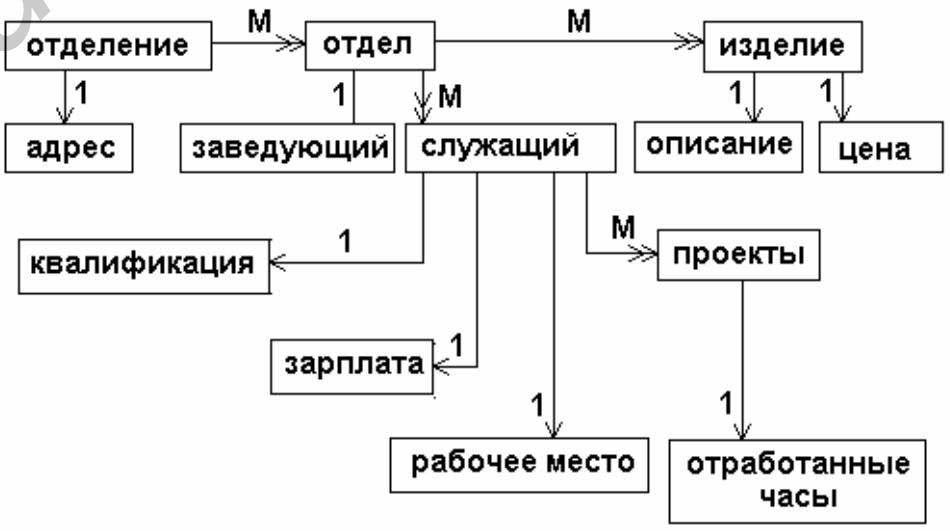


– для данной схемы ключом выбирается элемент данных A, от которого выходят одиночные ассоциации к атрибутам B, C, D .

4. Если между двумя связанными элементами заданы отображения M:M или одиночная ассоциация типа M (см. ниже), то из указанных двух элементов строится ключ. Каждая пара таких элементов определяет отдельные отношения, которые позже по желанию проектировщика могут быть объединены.

5. В окончательный вариант логической модели не включаются те из построенных отношений, которые являются подмножествами (проекциями) других построенных отношений.

Рассмотрим следующий пример.



Пусть в результате концептуального проектирования получена некоторая композиционная модель, представленная на схеме. Анализ полученной модели и применение алгоритма процедуры построения реляционных отношений приводит нас к следующему набору структур:

Полученный набор табличных структур полностью реализует концептуальную модель, выбранную в качестве примера. Данный набор таблиц гарантирует реализацию в дальнейших запросах всех связей между элементами данных, определяемых композиционной моделью. В нем отсутствует избыточное дублирование данных (дублирование используется только на уровне «общих» элементов, реализующих эти связи).

личный номер	должность	оклад

личный номер	должность

должность	оклад

Выделение таких атрибутов позволяет вводить их значения вне зависимости от взаимосвязей, в которых они участвуют.

Приведение отношений к первой, второй и третьей нормальной форме последовательно устраняет аномалии, проявляющиеся при включении, удалении и модификации БД.

3.3.1. Нормальная форма Бойса-Кодда

Реляционное отношение может допускать возможность существования нескольких ключей. Например, отношение:

сотрудник_ номер	сотрудник_ имя	проект_ номер	сотрудник_ задание

сотрудник_ номер	проект_ номер

сотрудник_ имя	проект_ номер

При выборе одного из таких ключей в качестве первичного возможны аномалии при модификации БД (модификация имени сотрудника приведет к модификации всех записей, включающих его номер). В данном случае можно произвести декомпозицию на два отношения:

сотрудник_ номер	сотрудник_ имя

сотрудник_ номер	сотрудник_ проект

либо

сотрудник_ имя	сотрудник_ проект

Поскольку атрибут «сотрудник–номер» является детерминантом атрибута «сотрудник–имя», то здесь определяется полная функциональная зависимость.

Нормальная форма Бойсса-Кодда *– отношение находится в нормальной форме Бойсса-Кодда в том случае, если каждый детерминант является возможным ключом.*

В случае если в отношении имеется только один возможный ключ, то отношение эквивалентно третьей нормальной форме.

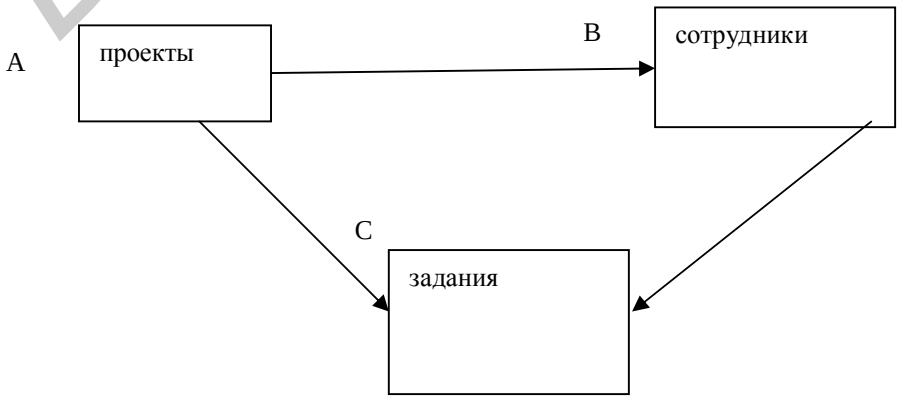
Детерминант – атрибут в левой части функциональной зависимости, который определяет отношение других атрибутов кортежа.

Третья нормальная форма обеспечивает только зависимость неключевых атрибутов от ключевых.

Четвертая нормальная форма

Таблица имеет четвертую нормальную форму, если она имеет ЗНФ и не содержит многозначных зависимостей.

Поскольку эта проблема возникает в связи с многозначными атрибутами, то ее можно решить, поместив каждый многозначный атрибут в свою собственную таблицу вместе с ключом от которого многозначный атрибут зависит. Следовательно, можно представить это отношение диаграммой



В том смысле, что С в каждом i-м проекте участвует Mi сотрудников, Ni заданий и каждый i –й сотрудник выполняет все задания.

Введем определение многозначной зависимости (условие, обеспечивающее взаимную независимость многозначных атрибутов).

В отношении R (A,B,C) существует многозначная зависимость R.A R.B в том и только в том случае, если множество значений B, соответствующих паре значений A и C, зависит только от A и не зависит от C.

Пример. Пусть имеется и реализуется следующая схема отношения (ряд проектов, выполняемых сотрудниками):

про_номер	про_сотр	про_задан

Отношение описывает проекты, на каждом из них могут работать сотрудники, а также список заданий, предусматриваемых проектом.

Предположим, что любой сотрудник, участвующий в проекте, выполняет все задания, предусматриваемые проектом. Тогда единственным возможным ключом является составной атрибут «про_номер*про_сотруд*про_задан».

Следовательно, отношение находится в третьей нормальной форме Бойсса-Кодда. Но при этом такое отношение обладает следующим недостатком.

При присоединении нового сотрудника к проекту необходимо вставить в него столько кортежей, сколько заданий предусмотрено в данном проекте.

В данном отношении одновременно существуют две многозначные зависимости:

про_номер $\xrightarrow{1:M}$ про_сотр

и

про_номер $\xrightarrow{1:N}$ про_задан.

Для дальнейшей нормализации используется ***теорема Фейджина***.

Отношение R (A,B,C) можно спроецировать без потерь в отношения R1 (A,B) и R2 (A,C) в том и только в том случае, когда существует многозначная зависимость

$A \rightarrow\!\!\!\rightarrow B \mid C \Rightarrow A \rightarrow\!\!\!\rightarrow B \;\&\; A \rightarrow\!\!\!\rightarrow C.$

Следовательно, мы можем представить наше отношение в виде декомпозиции

про_номер	про_сотрудник

про_номер	про_задание

.

Соединение этих отношений дает исходное. Оба эти отношения находятся в четвертой нормальной форме, если в случае многозначной зависимости между атрибутами A и B все остальные атрибуты функционально зависят от A.

Теория реляционного похода к разработке БД оказалась плодотворной и достаточной и для информационного моделирования предметных областей.

Однако ее недостатки (сложность, отсутствие выразительных средств представления семантики (смысла) реальной предметной области, отсутствие средств учета зависимостей и аппарата разделения сущностей и связей, сложность и др.) привели к появлению семантического моделирования данных.

4. ЛАБОРАТОРНАЯ РАБОТА № 2
«Проектирование БД в среде СУБД Sybase SQL Anywhere»

Цель работы.

Приобрести навыки проектирования структуры баз данных в среде корпоративных СУБД.

Одним из важных этапов работы с базой данных является ее создание и определение ее объектов. Для создания воспользуемся утилитой SQL Central, предоставленной нам SYBASE SQL Anywhere.

Ход работы.

После запуска SQL Central раскроем папку Utilities и выберем мастер Create Database для создания базы данных (рис. 10).

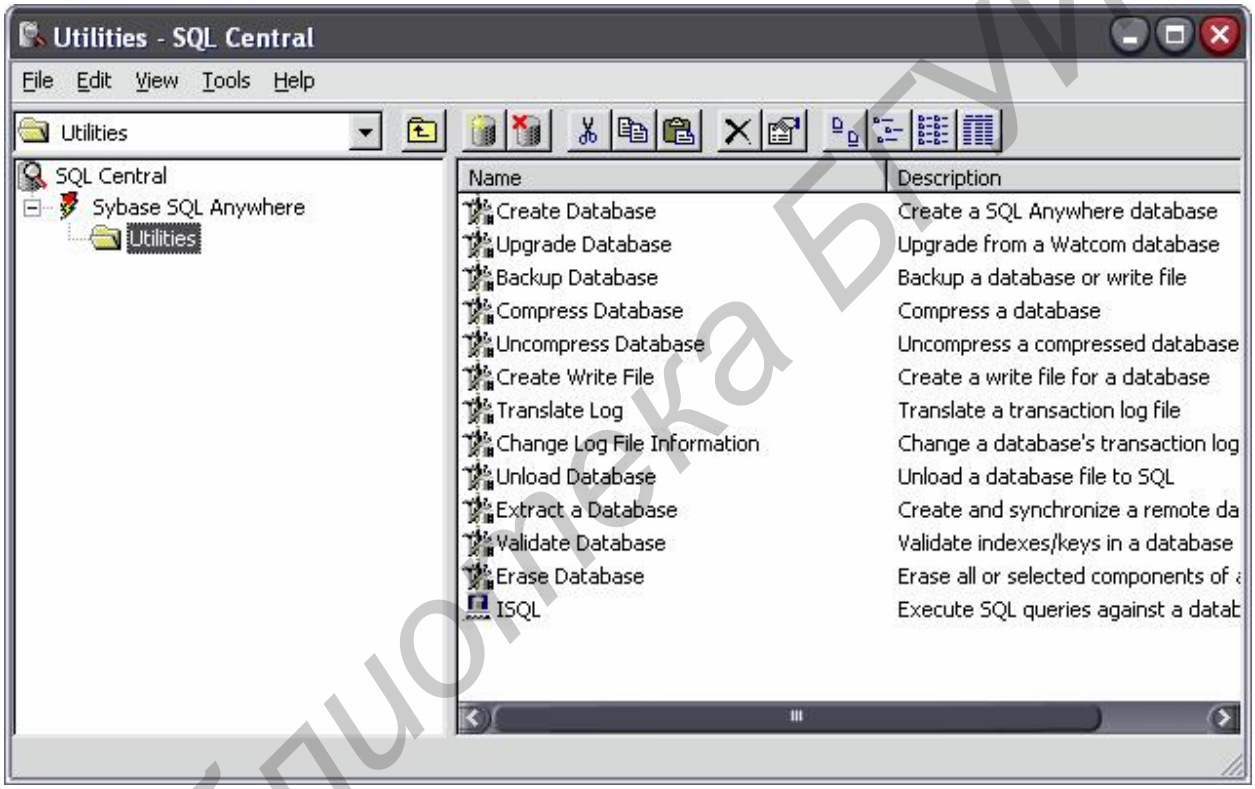


Рис. 10. Выбор мастера Create Database для создания базы данных

Нам будет предложен ряд шагов, каждый из которых находится в отдельном окне. В верхней части окна прилагается пояснение к каждому шагу, после чего идет вопрос с выбором вариантов ответа. Далее приведены шаги работы с мастером и основные действия, предпринимаемые пользователем.

Шаг 1. Начало создания БД (рис. 10).

Рис. 11.Создание файла БД в выбранной директории

В поле ввода мы должны ввести путь, который будет содержать файл базы данных. Также можно указать этот путь с помощью кнопки **Browse**. Укажем путь D:\database\Passenger.db для создаваемой базы **Passenger**.db.

Для перехода к следующему шагу нажмем кнопку **Далее**.

Шаг 2. Параметры журнала изменения базы данных (рис. 12).

Рис. 12. Выбор параметра журнала изменения БД

Если вы выберете поддержку журнала изменения базы данных, вы можете выбрать директорию для ведения данного журнала.

Если указать пункт No translation log, журнал изменения не будет создан. Если же отметить пункт Maintain a log file in this path and file, БД будет использовать журнал изменений в указанной директории, которую можно указать самому либо воспользоваться пунктом **Browse**, также мастер предлагает свой вариант имени файла по умолчанию в той же директории, где хранится база данных.

После этого переходим к третьему шагу.

Шаг 3. Параметры зеркального файла для журнала изменений базы данных (рис. 13).

Рис. 13. Параметры для задания зеркального файла

Вы можете использовать зеркалирование журнала изменений с таким же именем, только находящемся на другом устройстве. Это защитит от сбоя винчестера, но может понизить производительность работы базы данных.

По умолчанию флажок не установлен, и значит, зеркалирование не будет поддерживаться. Если поставить флажок напротив пункта Maintain a mirror log file, в поле окна можно будет ввести имя и путь зеркального файла, для чего также можно воспользоваться кнопкой **Browse**.

Для перехода к следующему пункту нажмем кнопку **Далее**.

Шаг 4. Выбор атрибутов базы данных (рис. 14).

Рис. 14. Выбор атрибутов базы данных

Encrypt the Database – шифрование базы данных;

Ignore Trailing Blanks in Comparisons – игнорирование конечных пробелов при сравнении полей;

Case sensitivity for all names and values – чувствительность к написанию имен или значений верхним или нижним регистром.

Не будем указывать ни одного флажка для большей производительности.

Будет создан доступ DBO (Database Object). Ему принадлежит несколько системных объектов (таблиц). Вы можете изменить его.

Изменять ничего не будем и сразу перейдем к следующему шагу.

Шаг 5. Выбор размера страницы базы данных (рис. 15).

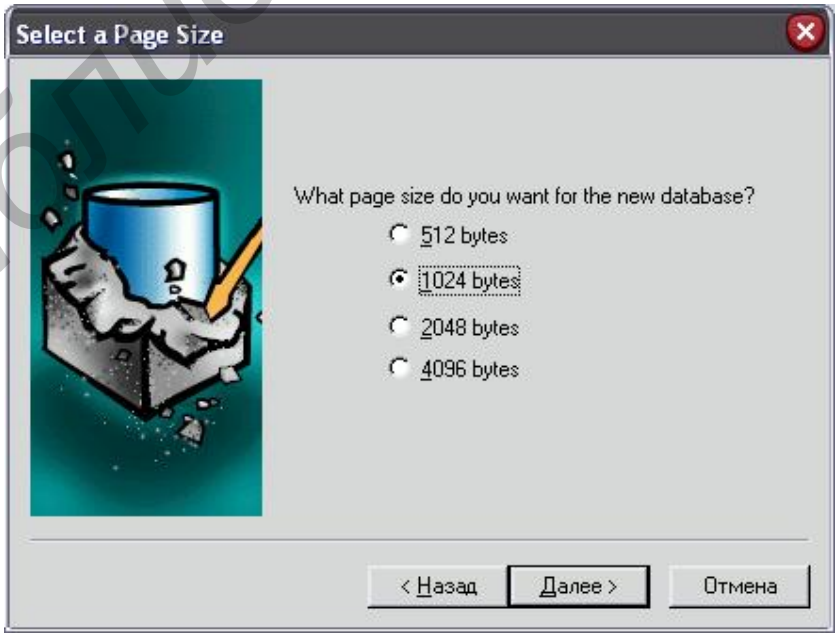


Рис. 15. Выбор размера страницы базы данных

Для указания размера страницы обмена с базой данных следует выбрать один из четырех вариантов. По умолчанию устанавливается размер 1024 байта.

Следует отметить, что лучше выбрать размер, кратный размеру кластера, для ускорения работы с базой данных.

Переходим к следующему пункту нажатием кнопки **Далее**.

Шаг 6. Выбор кодовой таблицы базы данных (рис. 16).

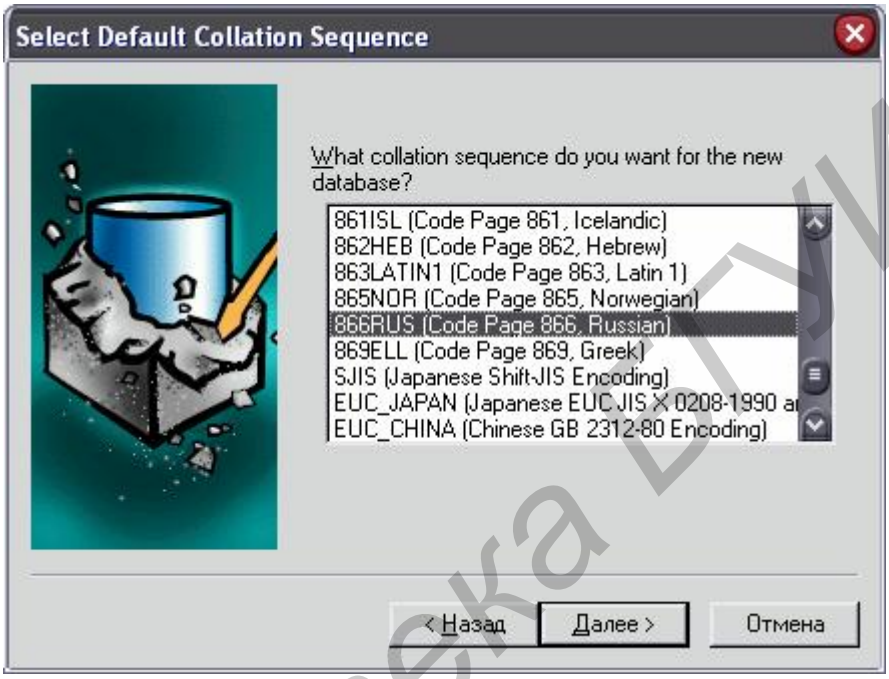


Рис. 16. Выбор кодовой таблицы базы данных

Ниже идет список всех возможных кодовых таблиц. Для предоставления текста на русском языке следует выбрать кодовую таблицу 866RUS.

Переходим к следующему шагу.

Шаг 7. Готовность к созданию базы данных (рис. 17).

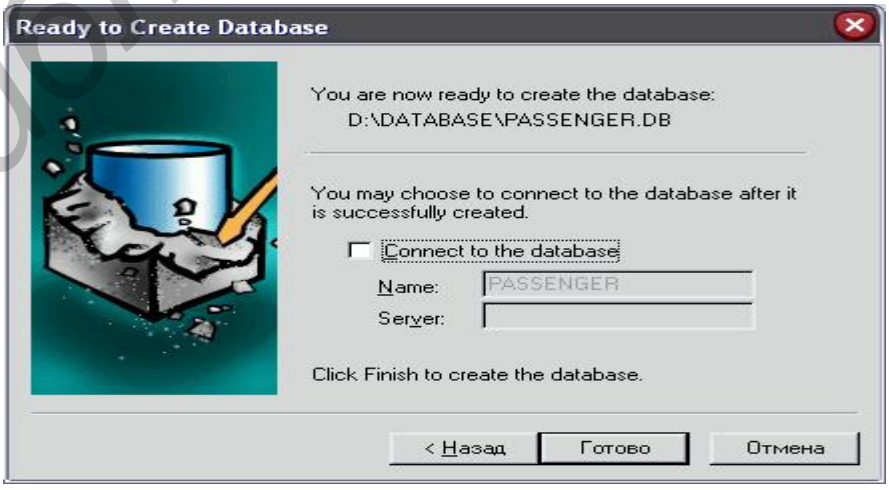


Рис. 17. Готовность к созданию БД

Теперь мы готовы к созданию базы данных в указанной на шаге 1 директории.

Дальше указывается директория расположения файла базы данных.

Вы можете выбрать возможность подключения к базе данных после ее успешного создания.

Установленный флажок Connect to the database означает, что мы поддерживаем это предложение.

В нашем случае мы этого делать не будем, а подключимся к ней попозже.

Следует нажать кнопку **Готово** для окончания создания БД.

Теперь происходит непосредственное создание базы данных, что отображается в следующем окне мастера Create Database (рис. 18).

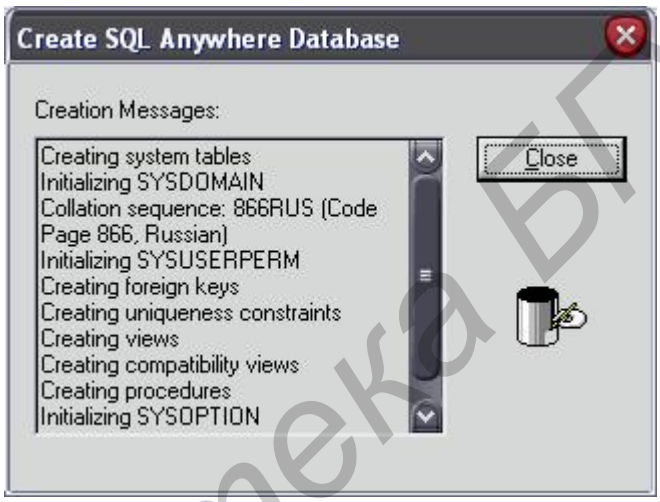


Рис.18. Окно создания БД

В конце протокол создания БД оповещает нас об успешном либо неуспешном создании базы данных.

Так происходит процесс создания базы данных средствами SQL Central.

Однако базу данных можно также создать утилитой iSQL с помощью языка SQL (окно Command).

Для этого вводим в окно текст :

```
//создание базы данных в утилите iSQL

DBTOOL CREATE DATABASE
'D:\database\Passenger.db'              //Полное имя базового файла БД
TRANSACTION LOG              //полное имя файла для журнала изменений
TO
'D:\database\Passenger.log'
IGNORE CASE                  /* не делать разницы между строчными и прописными
                             буквами */
//ENCRYPT                    // шифрование
PAGE SIZE 1024               // размер страницы
```

```
COLLATION '866rus'          // тип кодовой страницы
TRAILING SPACES             /* признак игнорирования конечных пробелов при
                               сравнениях строк */
```

Результат будет выведен в окно утилиты iSQL Statistics.

При создании БД в ней зарегистрирован пользователь dba с паролем sql, имеющий класс полномочий DBA (права администратора БД).

Теперь в созданной нами базе данных можно создавать объекты и работать с ними.

5. Введение в SQL

5.1. История языка SQL

История наиболее распространенного в настоящее время языка реляционных баз данных SQL насчитывает уже более 25 лет. Первый, достаточно полный функционально, но не полностью синтаксически и семантически определенный вариант языка SQL (его исходным названием было SEQUEL – Structured English Query Language) был разработан и частично реализован в рамках проекта экспериментальной реляционной СУБД SystemR (проект выполнялся с 1974 по 1979 г. в научно-исследовательской лаборатории компании IBM в г. Сан-Хосе, Калифорния).

Название языка SQL (Structured Query Language – структурированный язык запросов) только частично отражает его суть. Конечно, язык всегда был главным образом ориентирован на удобную и понятную пользователям формулировку запросов к реляционной БД, но на самом деле с самого начала задумывался как полный язык БД. Под этим мы понимаем то, что (по крайней мере, теоретически) знание SQL полностью достаточно для выполнения любых осмысленных действий с базой данных, управляемой SQL-ориентированной СУБД. Помимо операторов формулирования запросов и манипулирования данными язык содержит:

- средства определения схемы БД и манипулирования схемой;
- операторы для определения ограничений целостности и триггеров;
- средства определения представлений БД;
- средства авторизации доступа к отношениям и их полям;
- средства управления транзакциями.

Другими словами, язык SQL претендует на то, что он способен полностью представить реляционную модель данных, т.е. его средств достаточно для представления всех аспектов реляционных баз данных в терминах Кодда.

После появления в 1989 г. первого международного стандарта языка SQL (SQL-89) и в особенности после принятия в 1992 г. второго международного стандарта SQL-92 стало возможным говорить про стандартную среду SQL-ориентированной СУБД. Для грамотного использования любой SQL-ориентированной реляционной СУБД знание стандартов языка кажется необходимым.

5.2. Язык баз данных SQL-89

SQL-89 стал первым международным принятым стандартом языка SQL. У этого языка имеется масса недостатков: многие важные понятия не определены, много отдано на откуп реализациям и т.д., но тем не менее этот стандарт сыграл свою роль в становлении действительно стандартизованных реляционных систем управления базами данных (на самом деле значимость SQL-89 сохраняется и сегодня, поскольку большинство производителей современных СУБД под-

держивает именно этот стандарт). Более того, с появлением стандарта SQL-89 стало реально возможно проектировать, разрабатывать и сопровождать информационные системы, не слишком привязанные к конкретному производителю СУБД. В некотором смысле появление SQL-89 явилось продвижением технологии баз данных в сторону открытых систем (мобильность, интероперабельность и т.д.).

5.2.1. Типы данных

В языке SQL-89 поддерживаются следующие типы данных: CHARACTER, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION. Эти типы данных классифицируются на типы строк символов, точных чисел и приблизительных чисел.

К *первому* классу относится CHARACTER. Спецификатор типа имеет вид CHARACTER (length), где length задает длину строк данного типа. Заметим, что в SQL-89 нет типа строк переменного размера, хотя во многих реализациях они допускаются. Литеральные строки символов изображаются в виде 'последовательность-символов' (например, 'example').

Представителями *второго* класса типов являются NUMERIC, DECIMAL (или DEC), INTEGER (или INT) и SMALLINT. Спецификатор типа NUMERIC имеет вид NUMERIC [(precision [, scale]). Специфицируются точные числа, представляемые с точностью precision и масштабом scale. Здесь и далее, если опущен масштаб, то он полагается равным 0, а если опущена точность, то ее значение по умолчанию определяется в реализации.

Спецификатор типа DECIMAL (или DEC) имеет вид NUMERIC [(precision [, scale]). Специфицируются точные числа, представленные с масштабом scale и точностью, равной или большей значения precision.

INTEGER специфицирует тип данных точных чисел с масштабом 0 и определяемой в реализации точностью. SMALLINT специфицирует тип данных точных чисел с масштабом 0 и определяемой в реализации точностью не большей, чем точность чисел типа INTEGER.

Литеральные значения точных чисел в общем случае представляются в форме

[+ | -] <целое-без-знака> [.<целое-без-знака>].

Наконец, к классу типов данных приблизительных чисел относятся типы FLOAT, REAL и DOUBLE PRECISION. Спецификатор типа FLOAT имеет вид FLOAT [(precision)]. Специфицируются приблизительные числа с двоичной точностью, равной или большей значения precision.

REAL специфицирует тип данных приблизительных чисел с точностью, определенной в реализации. DOUBLE PRECISION специфицирует тип данных приблизительных чисел с точностью, определенной в реализации, большей, чем точность типа REAL.

Литеральные значения приблизительных чисел в общем случае представляются в виде

<литеральное-значение-точного-числа>E<целое-со-знаком>.

Заметим еще, что в большинстве реализаций SQL поддерживаются некоторые дополнительные типы данных, например DATE, TIME, INTERVAL, MONEY. Некоторые из этих типов специфицированы в стандарте SQL-92, но в текущих реализациях синтаксические и семантические свойства таких типов могут различаться.

5.2.2. Структура запросов

Для того чтобы можно было более или менее точно рассказать про структуру запросов в стандарте SQL-89, необходимо начать со сводки синтаксических правил:

```
<cursor specification> ::=
        <query expression> [<order by clause>]
<query expression> ::=
        <query term>
        <query expression> UNION [ALL] <query term>
<query term> ::=
        <query specification>
        | (<query expression>)
<query specification> ::=
        (SELECT [ALL DISTINCT] <select list> <table
expression>)
<select statement> ::=
        SELECT [ALL DISTINCT] <select list> INTO
<select target
        list> <table expression>
<subquery> ::=
        (SELECT [ALL DISTINCT] <result specification>
        <table expression>
<table expression> ::=
        <from clause>
        [<where clause>] [<group by clause>] [<having
clause>]
```

Язык допускает три типа синтаксических конструкций, начинающихся с ключевого слова SELECT: спецификация курсора (cursor specification), оператор выборки (select statement) и подзапрос (sub query). Основой для всех этих конструкций является синтаксическая конструкция «табличное выражение» (table expression). Семантика табличного выражения состоит в том, что на основе последовательного применения разделов from, where, groupby и having из заданных в разделе from таблиц строится некоторая новая результирующая таблица, порядок следования строк которой неопределен и среди строк которой могут находиться дубликаты (т.е. в общем случае таблица-результат табличного выражения является мультимножеством строк). На самом деле именно структура табличного выражения наибольшим образом характеризует структуру запросов языка SQL-89. Мы рассмотрим ниже струк-

туру и смысл разделов табличного выражения, но до этого немного подробнее обсудим три упомянутые конструкции, включающие табличные выражения.

5.2.3. Спецификация курсора

Наиболее общей является конструкция «спецификация курсора». Курсор – это понятие языка SQL, позволяющее с помощью набора специальных операторов получить построчный доступ к результату запроса к БД. К табличным выражениям, участвующим в спецификации курсора, не предъявляются какие-либо ограничения. Как видно из сводки синтаксических правил, при определении спецификации курсора используются три дополнительных конструкции: спецификация запроса, выражение запросов и раздел ORDER BY.

5.2.4. Спецификация запроса

В спецификации запроса задается список выборки (список арифметических выражений над значениями столбцов результата табличного выражения и констант). В результате применения списка выборки к результату табличного выражения производится построение новой таблицы, содержащей то же число строк, но, вообще говоря, другое число столбцов, содержащих результаты вычисления соответствующих арифметических выражений из списка выборки. Кроме того, в спецификации запроса могут содержаться ключевые слова ALL или DISTINCT. При наличии ключевого слова DISTINCT из таблицы, полученной применением списка выборки к результату табличного выражения, удаляются строки-дубликаты; при указании ALL (или просто при отсутствии DISTINCT) удаление строк-дубликатов не производится.

5.2.5. Выражение запросов

Выражение запросов – это выражение, строящееся по указанным синтаксическим правилам на основе спецификаций запросов. Единственной операцией, которую разрешается использовать в выражениях запросов, является операция UNION (объединение таблиц) с возможной разновидностью UNION ALL. К таблицам-операндам выражения запросов предъявляется то требование, что все они должны содержать одно и то же число столбцов и соответствующие столбцы всех операндов должны быть одного и того же типа. Выражение запросов вычисляется слева направо с учетом скобок. При выполнении операции UNION производится обычное теоретико-множественное объединение операндов, т.е. из результирующей таблицы удаляются дубликаты. При выполнении операции UNION ALL образуется результирующая таблица, в которой могут содержаться строки-дубликаты.

5.2.6. Раздел ORDER BY

Наконец, раздел ORDER BY позволяет установить желаемый порядок просмотра результата выражения запросов. Синтаксис ORDER BY следующий:

```
<order by clause> ::=
        ORDER BY <sort specification> [{,<sort
specification>}...]
<sort specification> ::=
        {<unsigned integer> <column specification>} [ASC
DESC]
```

Как видно из этих синтаксических правил, фактически задается список столбцов результата выражения запросов, и для каждого столбца указывается порядок просмотра строк результата в зависимости от значений этого столбца (ASC – по возрастанию, этот режим используется по умолчанию, DESC – по убыванию). Столбцы можно задавать их именами в том и только в том случае, когда (1) выражение запросов не содержит операций UNION или UNION ALL и (2) в списке выборки спецификации запроса этому столбцу соответствует арифметическое выражение, состоящее только из имени столбца. Во всех остальных случаях в разделе ORDER BY должен указываться порядковый номер столбца в таблице-результате выражения запросов.

5.2.7. Оператор выборки

Оператор выборки – это отдельный оператор языка SQL-89, позволяющий получить результат запроса в прикладной программе без привлечения курсора. Поэтому оператор выборки имеет синтаксис, отличающийся от синтаксиса спецификации курсора, и при его выполнении возникают ограничения на результат табличного выражения. Фактически и то, и другое диктуется спецификой оператора выборки как одиночного оператора SQL: при его выполнении результат должен быть помещен в переменные прикладной программы. Поэтому в операторе появляется раздел INTO, содержащий список переменных прикладной программы, и возникает то ограничение, что результирующая таблица должна содержать не более одной строки. Соответственно результат базового табличного выражения должен содержать не более одной строки, если оператор выборки не содержит спецификации DISTINCT, и таблица, полученная применением списка выборки к результату табличного выражения, не должна содержать несколько несовпадающих строк, если спецификация DISTINCT задана.

5.2.8. Подзапрос

Наконец, последняя конструкция SQL-89, которая может содержать табличные выражения, – это подзапрос, т.е. запрос, который может входить в предикат условия выборки оператора SQL. В SQL-89 к подзапросам применяется то ограничение, что результирующая таблица должна содержать в точности один столбец. Поэтому в синтаксических правилах, определяющих подзапрос, вместо списка выборки указано «выражение, вычисляющее значение», т.е. арифметическое выражение. Заметим еще, что поскольку подзапрос всегда

вложен в некоторый другой оператор SQL, то в качестве констант в арифметическом выражении выборки и логических выражениях разделов WHERE и HAVING можно использовать значения столбцов текущих строк таблиц, участвующих в (под)запросах более внешнего уровня. Более подробно об этом см. ниже, при описании семантики табличных выражений.

5.2.9. Табличное выражение

Стандарт SQL-89 рекомендует рассматривать вычисление табличного выражения как последовательное применение разделов FROM, WHERE, GROUP BY и HAVING к таблицам, заданным в списке FROM.

5.2.10. Раздел FROM

Раздел FROM имеет следующий синтаксис:

```
<from clause> ::=
       FROM <table reference> ({,<table reference>}...]
<table reference> ::=
       <table name> [<correlation name>]
```

Результатом выполнения раздела FROM является расширенное декартово произведение таблиц, заданных списком таблиц раздела FROM. Расширенное декартово произведение (расширенное, потому что в качестве операндов и результата допускаются мультимножества) в стандарте определяется следующим образом: *«Расширенное произведение R есть мультимножество всех строк r таких, что r является конкатенацией строк из всех идентифицированных таблиц в том порядке, в котором они идентифицированы. Мощность R есть произведение мощностей идентифицированных таблиц. Порядковый номер столбца в R есть n+s, где n – порядковый номер порождающего столбца в именованной таблице T, а s – сумма степеней всех таблиц, идентифицированных до T в разделе FROM»*.

Как видно из синтаксиса, рядом с именем таблицы можно указывать еще одно имя «correlation name». Фактически это некоторый синоним имени таблицы, который можно использовать в других разделах табличного выражения для ссылки на строки именно этого вхождения таблицы.

Если табличное выражение содержит только раздел FROM (это единственный обязательный раздел табличного выражения), то результат табличного выражения совпадает с результатом раздела FROM.

5.2.11. Раздел WHERE

Если в табличном выражении присутствует раздел WHERE, то следующим вычисляется он. Синтаксис раздела WHERE следующий:

```
<where clause> ::= WHERE <search condition>
<search condition> ::=
     <boolean term>
     <search condition> OR <boolean term>
<Boolean term> ::=
     <boolean factor>
```

```
     <boolean term> AND <boolean factor>
<boolean factor> ::= [NOT] <boolean primary>
<boolean primary> ::= <predicate> (<search condition>)
```

Вычисление раздела WHERE производится по следующим правилам: Пусть R – результат вычисления раздела FROM. Тогда условие поиска применяется ко всем строкам R, и результатом раздела WHERE является таблица, состоящая из тех строк R, для которых результатом вычисления условия поиска является TRUE. Если условие выборки включает подзапросы, то каждый подзапрос вычисляется для каждого кортежа таблицы R (в стандарте используется термин «effectively» в том смысле, что результат должен быть таким, как если бы каждый подзапрос действительно вычислялся заново для каждого кортежа R). Заметим, что поскольку SQL-89 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.

Среди предикатов условия поиска в соответствии с SQL-89 могут находиться следующие предикаты: предикат сравнения, предикат BETWEEN, предикат IN, предикат LIKE, предикат NULL, предикат с квантором и предикат EXISTS. Сразу заметим, что во всех реализациях SQL на эффективность выполнения запроса существенно влияет наличие в условии поиска простых предикатов сравнения (предикатов, задающих сравнение столбца таблицы с константой). Наличие таких предикатов позволяет СУБД использовать индексы при выполнении запроса, т.е. избегать полного просмотра таблицы. Хотя в принципе язык SQL позволяет пользователям не заботиться о конкретном наборе предикатов в условиях выборки (лишь бы они были синтаксически и семантически правильны), при реальном использовании SQL-ориентированных СУБД такие технические детали стоит иметь в виду.

5.2.12. Предикат сравнения

Синтаксис предиката сравнения определяется следующими правилами:

```
<comparison predicate> ::=
     <value expression> <comp op>
    {<value expression> <sub query>}
<comp op> ::=
    = <> < > <= >=
```

Через «<>» обозначается операция «неравенства». Арифметические выражения левой и правой частей предиката сравнения строятся по общим правилам построения арифметических выражений и могут включать в общем случае имена столбцов таблиц из раздела FROM и константы. Типы данных арифметических выражений должны быть сравнимыми (например, если тип столбца a таблицы A является типом символьных строк, то предикат «a = 5» недопустим).

Если правый операнд операции сравнения задается подзапросом, то дополнительным ограничением является то, что мощность результата подзапроса должна быть не более единицы. Если хотя бы один из операндов операции сравнения имеет неопределенное значение или если правый операнд является подзапросом с пустым результатом, то значение предиката сравнения равно UNKNOWN.

Заметим, что значение арифметического выражения не определено, если в его вычислении участвует хотя бы одно неопределенное значение. Еще одно важное замечание из стандарта SQL-89: в контексте GROUP BY, DISTINCT и ORDER BY неопределенное значение выступает как специальный вид определенного значения, т.е. возможно, например, образование группы строк, значение указанного столбца которых является неопределенным. Для обеспечения переносимости прикладных программ нужно внимательно оценивать специфику работы с неопределенными значениями в конкретной СУБД.

5.2.13. Предикат BETWEEN

Предикат BETWEEN имеет следующий синтаксис:

```
<between predicate> ::=
       <value expression>
       [NOT] BETWEEN <value expression> AND <value
expression>
```

Результат «x BETWEEN y AND z» тот же самый, что результат «x >= y AND x <= z». Результат «x NOT BETWEEN y AND z» тот же самый, что результат «NOT (x BETWEEN y AND z)».

5.2.14. Предикат IN

Предикат IN определяется следующими синтаксическими правилами:

```
<in predicate> ::=
    <value expression> [NOT] IN
    {<subquery> (<in value list>)}
<in value list> ::=
       <value specification>
       {,<value specification>}...
```

Типы левого операнда и значений из списка правого операнда (напомним, что результирующая таблица подзапроса должна содержать ровно один столбец) должны быть сравнимыми.

Значение предиката равно TRUE в том и только в том случае, когда значение левого операнда совпадает хотя бы с одним значением списка правого опе-

ранда. Если список правого операнда пуст (так может быть, если правый операнд задается подзапросом) или значение «подразумеваемого» предиката сравнения x = y (где x – значение арифметического выражения левого операнда) равно FALSE для каждого элемента y списка правого операнда, то значение предиката IN равно FALSE. В противном случае значение предиката IN равно UNKNOWN. По определению значение предиката «x NOT IN S» равно значению предиката «NOT (x IN S)».

5.2.15. Предикат LIKE

Предикат LIKE имеет следующий синтаксис:

```
<like predicate> ::=
      <column specification>
      [NOT] LIKE <pattern> [ESCAPE <escape character>]
<pattern> ::= <value specification>
<escape character> ::= <value specification>
```

Типы данных столбца левого операнда и образца должны быть типами символьных строк. В разделе ESCAPE должен специфицироваться одиночный символ.

Значение предиката равно TRUE, если pattern является подстрокой заданного столбца. При этом если раздел ESCAPE отсутствует, то при сопоставлении шаблона со строкой производится специальная интерпретация двух символов шаблона: символ подчеркивания («_») обозначает любой одиночный символ; символ процента («%») обозначает последовательность произвольных символов произвольной длины (может быть, нулевой).

Если же раздел ESCAPE присутствует и специфицирует некоторый одиночный символ x, то пары символов «x_» и «x%» представляют одиночные символы «_» и «%» соответственно.

Значение предиката LIKE есть UNKNOWN, если значение столбца либо шаблона неопределено.

Значение предиката «x NOT LIKE y ESCAPE z» совпадает со значением «NOT x LIKE y ESCAPE z».

5.2.16. Предикат NULL

Предикат NULL описывается синтаксическим правилом:

```
<null predicate> ::=
      <column specification> IS [NOT] NULL
```

Этот предикат всегда принимает значения TRUE или FALSE. При этом значение «x IS NULL» равно TRUE тогда и только тогда, когда значение x неопределено. Значение предиката «x NOT IS NULL» равно значению «NOT x IS NULL».

5.2.17. Предикат с квантором

Предикат с квантором имеет следующий синтаксис:

```
<quantified predicate> ::=
```

```
      <value expression> <comp op> <quantifier>
<subquery>
<quantifier> ::=
      <all> <some>
<all> ::= ALL
<some> ::= SOME ANY
```

Обозначим через x результат вычисления арифметического выражения левой части предиката, а через S – результат вычисления подзапроса.

Предикат «x <comp op> ALL S» имеет значение TRUE, если S пусто или значение предиката «x <comp op> s» равно TRUE для каждого s, входящего в S. Предикат «x <comp op> ALL S» имеет значение FALSE, если значение предиката «x <comp op> s» равно FALSE хотя бы для одного s, входящего в S. В остальных случаях значение предиката «x <comp op> ALL S» равно UNKNOWN.

Предикат «x <comp op> SOME S» имеет значение FALSE, если S пусто или значение предиката «x <comp op> s» равно FALSE для каждого s, входящего в S. Предикат «x <comp op> SOME S» имеет значение TRUE, если значение предиката «x <comp op> s» равно TRUE хотя бы для одного s, входящего в S. В остальных случаях значение предиката «x <comp op> SOME S» равно UNKNOWN.

5.2.18. Предикат EXISTS

Предикат EXISTS имеет следующий синтаксис:

```
<exists predicate> ::=
      EXISTS <subquery>
```

Значением этого предиката всегда является TRUE или FALSE, и это значение равно TRUE тогда и только тогда, когда результат вычисления подзапроса не пуст.

5.2.19. Раздел GROUP BY

Если в табличном выражении присутствует раздел GROUP BY, то следующим выполняется он. Синтаксис раздела GROUP BY следующий:

```
<group by clause> ::=
      GROUP BY <column specification>
      [{,<column specification>}...]
```

Если обозначить через R таблицу, являющуюся результатом предыдущего раздела (FROM или WHERE), то результатом раздела GROUP BY является разбиение R на множество групп строк, состоящего из минимального числа групп таких, что для каждого столбца из списка столбцов раздела GROUP BY во всех строках каждой группы, включающей более одной строки, значения этого столбца равны. Для обозначения результата раздела GROUP BY в стандарте используется термин «сгруппированная таблица».

5.2.20. Раздел HAVING

Наконец, последним при вычислении табличного выражения используется раздел HAVING (если он присутствует). Синтаксис этого раздела следующий:

```
<having clause> ::=
      HAVING <search condition>
```

Раздел HAVING может осмысленно появиться в табличном выражении только в том случае, когда в нем присутствует раздел GROUP BY. 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.

Результатом выполнения раздела HAVING является сгруппированная таблица, содержащая только те группы строк, для которых результат вычисления условия поиска есть TRUE. В частности, если раздел HAVING присутствует в табличном выражении, не содержащем GROUP BY, то результатом его выполнения будет либо пустая таблица, либо результат выполнения предыдущих разделов табличного выражения, рассматриваемый как одна группа без столбцов группирования.

5.2.21. Агрегатные функции и результаты запросов

Агрегатные функции (в стандарте SQL-89 они называются функциями над множествами) определяются в SQL-89 следующими синтаксическими правилами:

```
<set function specification> ::=
      COUNT(*)
      |<distinct set function>
      |<all set function>
<distinct set function> ::=
      { AVG MAX MIN SUM COUNT }
      (DISTNICT <column specification>)
```

```
<all set function> ::=
      { AVG MAX MIN SUM } ([ALL] <value expression>)
```

Как видно из этих правил, в стандарте SQL-89 определены пять стандартных агрегатных функций: COUNT – число строк или значений, MAX – максимальное значение, MIN – минимальное значение, SUM – суммарное значение и AVG – среднее значение.

5.2.22. Семантика агрегатных функций

Агрегатные функции предназначены для того, чтобы вычислять некоторое значение для заданного множества строк. Таким множеством строк может быть группа строк, если агрегатная функция применяется к сгруппированной таблице, или вся таблица. Для всех агрегатных функций, кроме COUNT(*), фактический (т.е. требуемый семантикой) порядок вычислений следующий: на основании параметров агрегатной функции из заданного множества строк производится список значений. Затем по этому списку значений производится вычисление функции. Если список оказался пустым, то значение функции COUNT для него есть 0, а значение всех остальных функций – NULL.

Пусть T обозначает тип значений из этого списка. Тогда результат вычисления функции COUNT – точное число с масштабом и точностью, определяемыми в реализации. Тип результата значений функций MAX и MIN совпадает с T. При вычислении функций SUM и AVG тип T не должен быть типом символьных строк, а тип результата функции – это тип точных чисел с определяемыми в реализации масштабом и точностью, если T – тип точных чисел, и тип приблизительных чисел с определяемой в реализации точностью, если T – тип приблизительных чисел.

Вычисление функции COUNT(*) производится путем подсчета числа строк в заданном множестве. Все строки считаются различными, даже если они состоят из одного столбца со значением NULL во всех строках.

Если агрегатная функция специфицирована с ключевым словом DISTINCT, то список значений строится из значений указанного столбца. (Подчеркнем, что в этом случае не допускается вычисление арифметических выражений!) Далее из этого списка удаляются неопределенные значения и в нем устраняются значения-дубликаты. Затем вычисляется указанная функция.

Если агрегатная функция специфицирована без ключевого слова DISTINCT (или с ключевым словом ALL), то список значений формируется из значений арифметического выражения, вычисляемого для каждой строки заданного множества. Далее из списка удаляются неопределенные значения и производится вычисление агрегатной функции. Обратите внимание, что в этом случае не допускается применение функции COUNT.

Замечание: оба ограничения, указанные в двух предыдущих абзацах, являются более техническими, чем принципиальными, и могут отсутствовать в конкретных реализациях. Тем не менее это ограничения стандарта SQL-89, и их нужно придерживаться при мобильном программировании.

5.2.23. Одиночные операторы манипулирования данными

Каждый из операторов этой группы является абсолютно независимым от какого бы то ни было другого оператора.

5.2.24. Оператор выборки

Для удобства мы повторяем синтаксис этого оператора еще раз:

```
<select statement> ::=
      SELECT [ALL DISTINCT] <select name> INTO <select
target
      list> <table expression>
<select target list>::=
      <target specification> [{,<target
specification>}...]
```

Поскольку, как мы уже объясняли, результатом одиночного оператора выборки является таблица, состоящая не более чем из одной строки, список целей специфицируется в самом операторе.

5.2 25. Оператор поискового удаления

Оператор описывается следующим синтаксическим правилом:

```
<delete statement: searched> ::=
      DELETE FROM <table name> WHERE [<search condition>]
```

Таблица T, указанная в разделе FROM оператора DELETE, должна быть обновляемой. На вид условия поиска накладывается то ограничение, что на таблицу T не должны содержаться ссылки ни в каком вложенном подзапросе предикатов раздела WHERE.

Фактически оператор выполняется следующим образом: последовательно просматриваются все строки таблицы T, и те строки, для которых результатом вычисления условия выборки является TRUE, удаляются из таблицы T. При отсутствии раздела WHERE удаляются все строки таблицы T.

5.2.26. Оператор поисковой модификации

Оператор обладает следующим синтаксисом:

```
<update statement: searched> ::=
      UPDATE <table name> SET <set clause: searched>
      [{,<set clause: searched>}...] [WHERE <search
conditions>]
<set clause: searched> ::=
      <object column: searched> =
      { <value expression> NULL }
<object column: searched> ::= <column name>
```

Таблица T, указанная в операторе UPDATE, должна быть обновляемой. На условие поиска накладывается то условие, что на таблицу T не должны содержаться ссылки ни в каком вложенном подзапросе предикатов раздела WHERE.

Оператор фактически выполняется следующим образом: таблица T последовательно просматривается, и каждая строка, для которой результатом вычис-

ления условия поиска является TRUE, обновляется в соответствии с разделом SET. Если арифметическое выражение в разделе SET содержит ссылки на столбцы таблицы T, то при вычислении арифметического выражения используются значения столбцов текущей строки до их модификации.

5.2.27. Оператор вставки

Оператор служит для создания новых строк в таблице и обладает следующим синтаксисом:

```
<insert statement> ::=
      INSERT INTO <table name> [(<insert column list>)]
      {VALUES (<insert value list>) <query
specification>}
<insert column list> ::=
      <column name> [{,<column name>}...]
<insert value list> ::=
      <insert value> [{,<insert value>...]
<insert value> ::=
      <value specification> NULL
```

Таблица T, указанная в операторе INSERT, должна быть обновляемой и не должна указываться в разделе FROM спецификации запроса или подзапроса, используемого в разделе VALUES.

Выполнение оператора INSERT происходит следующим образом: создается возможная (candidate) строка, содержащая столько же столбцов, сколько их в таблице T (если T – представление, то возможная строка содержит столько столбцов, сколько их в таблице, порождающей T); для каждого объектного столбца возможной строки его значение заменяется на вставляемое значение; полученная строка заносится в T.

Если в разделе VALUES указывается спецификация запроса, то пусть R обозначает ее результат. Если R пуст, то параметру SQLCODE присваивается значение 100 и никакая строка не вставляется. Число созданных возможных строк равно мощности R. Вставляемые значения одной возможной строки являются значениями одной строки R, и значения в одной строке R являются вставляемыми значениями одной возможной строки.

6. ЛАБОРАТОРНАЯ РАБОТА № 3
«Проектирование процедур и функций на языке SQL»

Цель работы:

Приобрести навыки решения практических задач по обработке данных в БД с использованием языка SQL.

Ход работы.

Хранимые процедуры – это подпрограммы на языке SQL, хранящиеся в базах данных и представляющие собой один из видов их общих ресурсов. Тело любой хранимой процедуры представляет последовательность SQL-операторов, например таких, как выборка данных (SELECT), их модификация (UPDATE), удаление данных (DELETE), операторы цикла (LOOP), условные операторы (IF, CASE) и ряд других. Процедуры вызываются оператором CALL и могут использовать как входные параметры (передающие значения в процедуру), так и выходные параметры (возвращающие результаты процедуры вызывающему программному объекту). Процедуры могут вызываться из процедур, функций и других типов программных объектов.

Хранимые процедуры, создаются оператором CREATE PROCEDURE. Модификация тела хранимой процедуры осуществляется оператором ALTER PROCEDURE.

Итак, рассмотрим несколько примеров создания и использования процедур. Обратим внимание на следующие моменты.

Тело хранимой процедуры является составным оператором, т.е. совокупностью операторов, заключенных между служебными словами BEGIN и END. Наряду с SQL-операторами в составном операторе могут быть определены локальные переменные, курсоры, временные таблицы данных и исключительные ситуации. Они доступны только в пределах составного оператора и невидимы за его пределами. Время их существования ограничено периодом исполнения составного оператора. Локальные определения широко используются при разработке программных объектов. Будут они применяться и в данной главе.

Описание каждого формального параметра в процедуре начинается с одного из служебных слов IN, OUT или INOUT. Они предназначены для указания типа формального параметра. Приведем пояснения к типам формальных параметров:

- **IN** – обозначает, что формальный параметр является входным, т. е. передающим значение процедуре;
- **OUT** – обозначает, что формальный параметр является выходным, т.е. посредством его осуществляется передача одного из результатов работы хранимой процедуры вызывающему программному объекту;
- **INOUT** – обозначает, что формальный параметр процедуры выполняет роль как входного, так выходного параметра.

Вызов хранимых процедур производится оператором CALL с соответствующими фактическими параметрами.

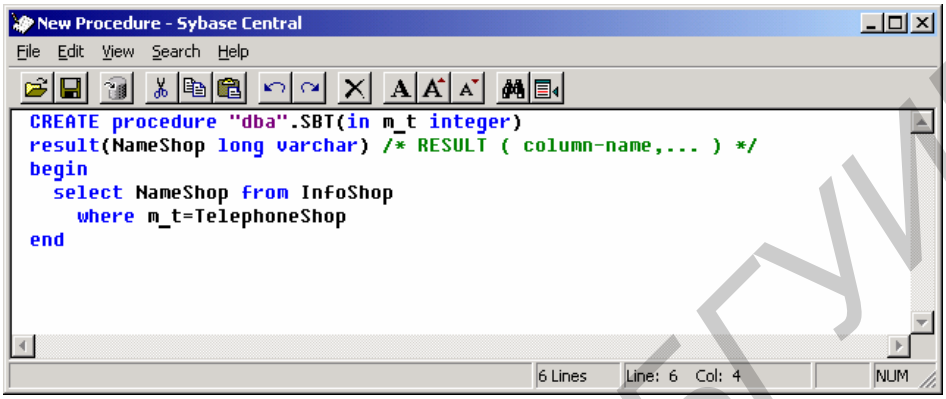
Варианты заданий

Вариант 1

Задача процедуры – вывод записей из БД по заданной переменной.

Первоначальные манипуляции:

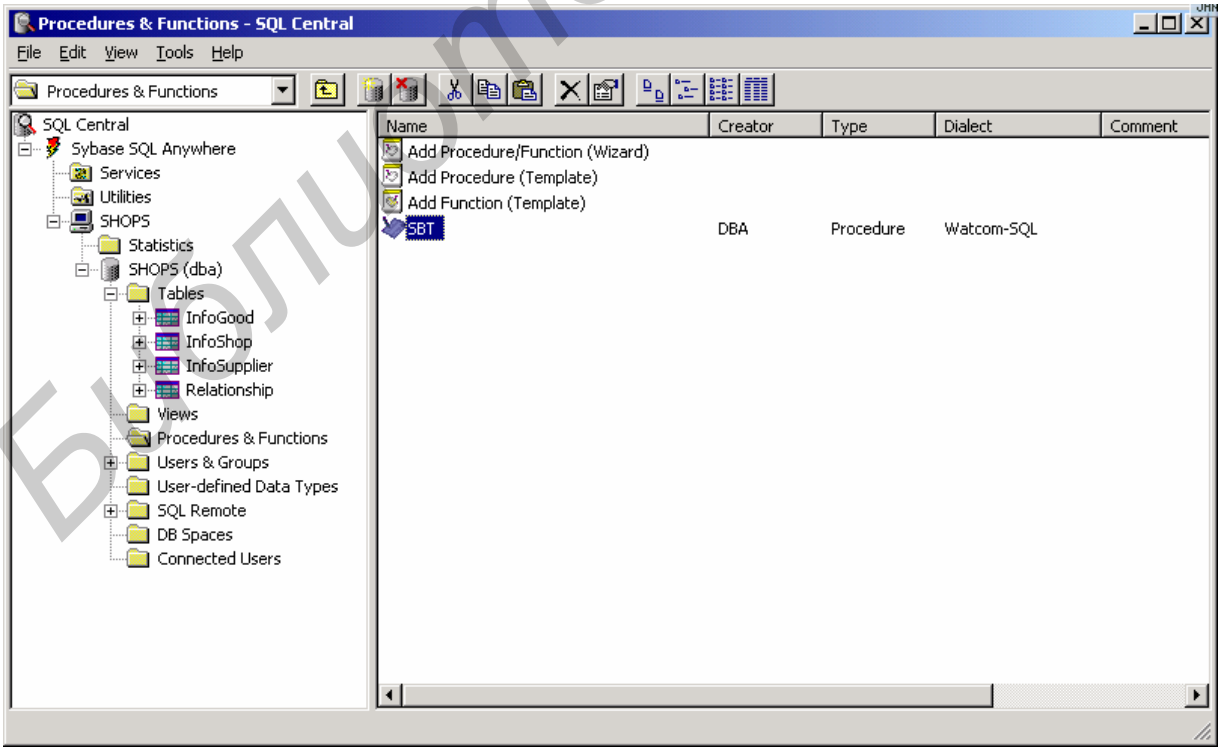
1) запуск SQL Central;
2) подсоединение к БД;
3) выбор вкладки Procedurs&Functions\Add Procedure(Template).



Перед вами шаблонная заготовка для описания процедуры.

Заполнив ее, мы получаем процедуру под именем SBT (Select By Telephone), которая выводит название магазина по введенному номеру телефона.

Необходимо произвести внедрение процедуры: Ctrl+S или File->Execute Script. Вы можете увидеть, что процедура SBT была добавлена в список процедур.



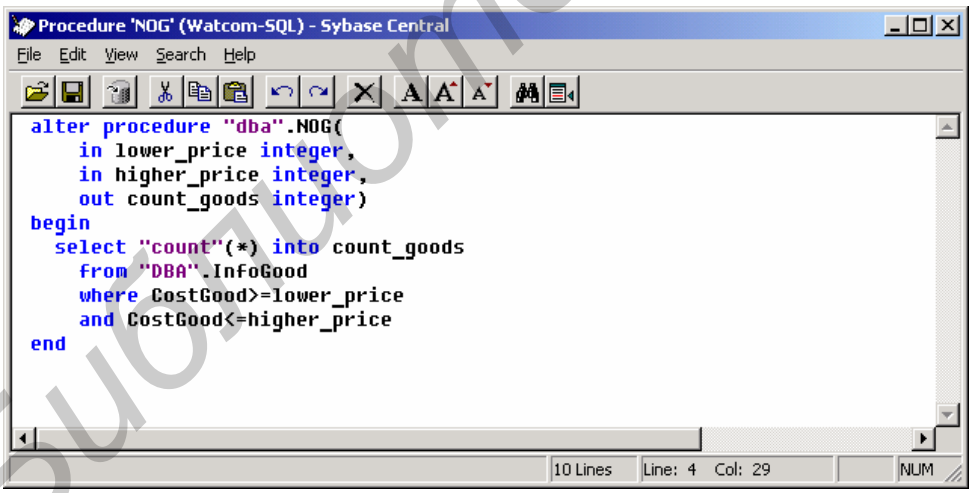
Вызов процедуры (ее тестирование).

Как уже отмечалось, вызов процедуры осуществляется посредством оператора CALL. Синтаксис и результат работы оператора можно увидеть на скриншоте.

Вариант 2

Процедуры могут возвращать результаты вызывающим их программным объектам. Пример будет построен на использовании параметров IN и OUT.

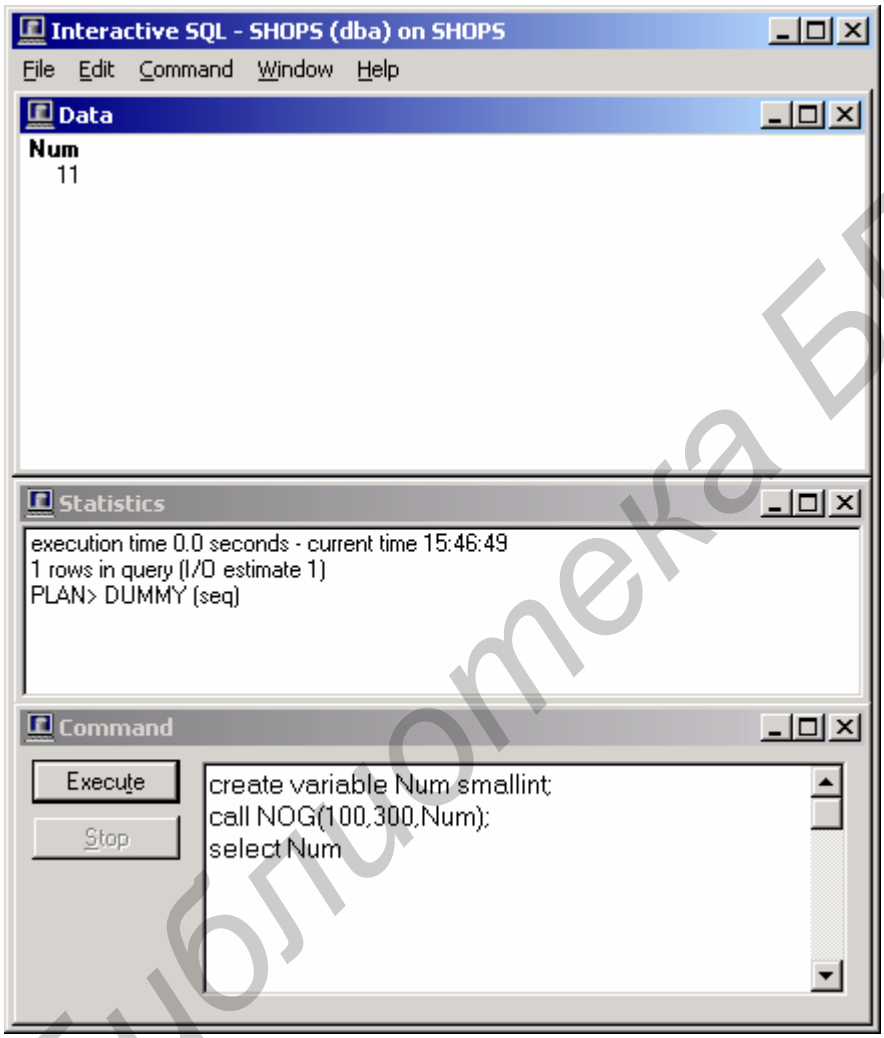
Задача процедуры: подсчитать число товаров, цена которых лежит в заданном диапазоне (NOG – Number Of Goods).




```
alter procedure "dba".NOG (
   in lower_price integer,
//нижний ценовой предел
   in higher_price integer,
//верхний ценовой предел
   out count_goods integer
//переменная, в которую будет помещено число товаров, соответствующих заданным условиям , //посредством которой это значение будет выведено в свет
)
begin
```

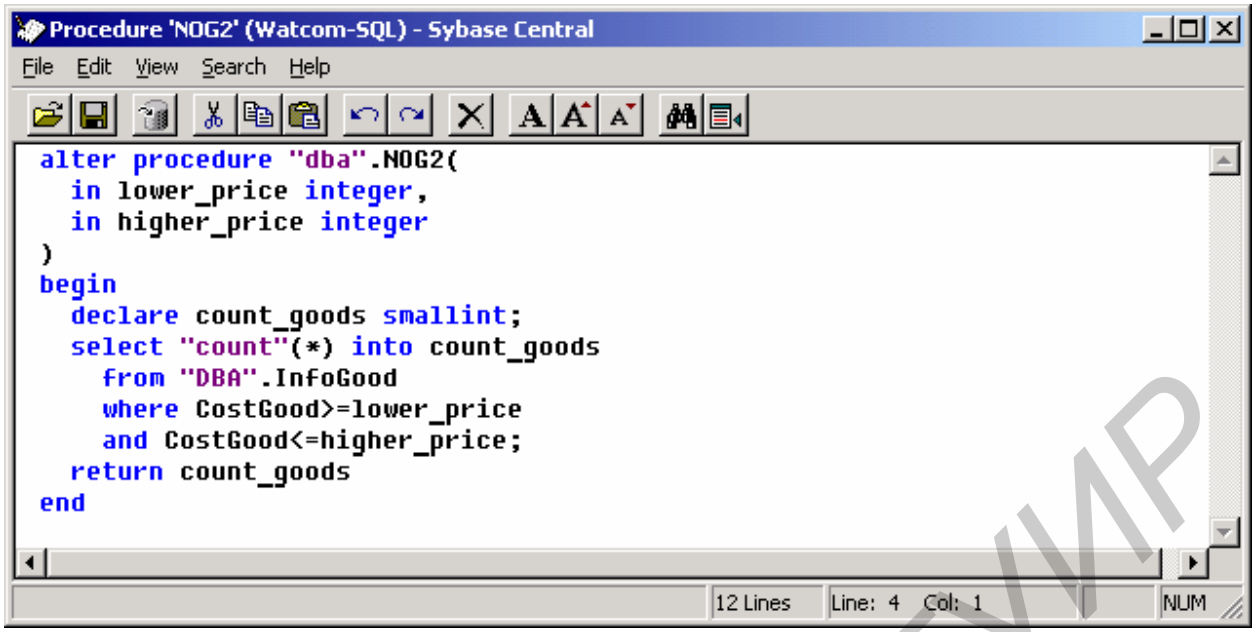
```
  select "count"(*) //подсчет числа товаров
                                into count_goods //результат записывается в count_goods
    from "DBA".InfoGood //указание используемой таблицы
    where CostGood>=lower_price //условия выбора записей
    and CostGood<=higher_price
end
```

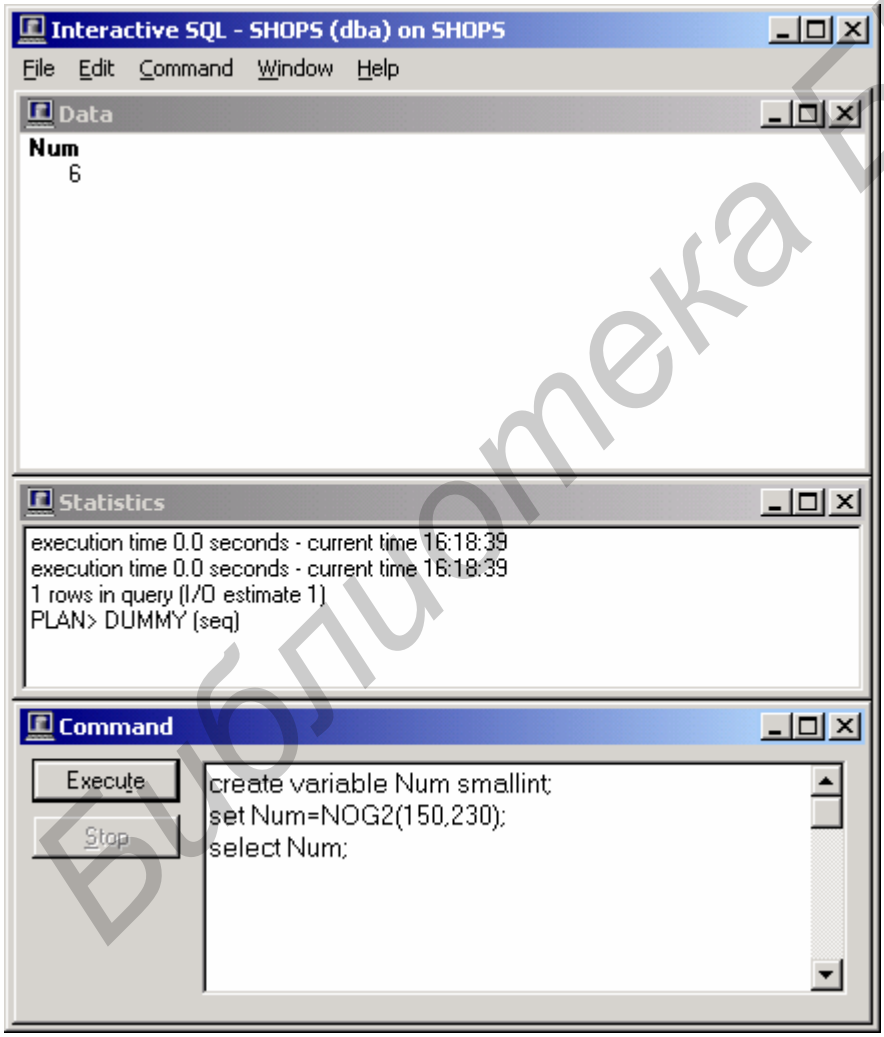
Созданная процедура



Для того чтобы проверить работу процедуры, необходимо создать переменную. Она будет существовать только во время текущего соединения с БД (create variable Num integer). Значение перекидывается в переменную посредством вызова процедуры (call NOG (100, 300, Num)). Последующий вывод значения Num в окно Data утилиты ISQL.

Результат, формируемый процедурой NOG, можно получить и с использованием оператора RETURN.

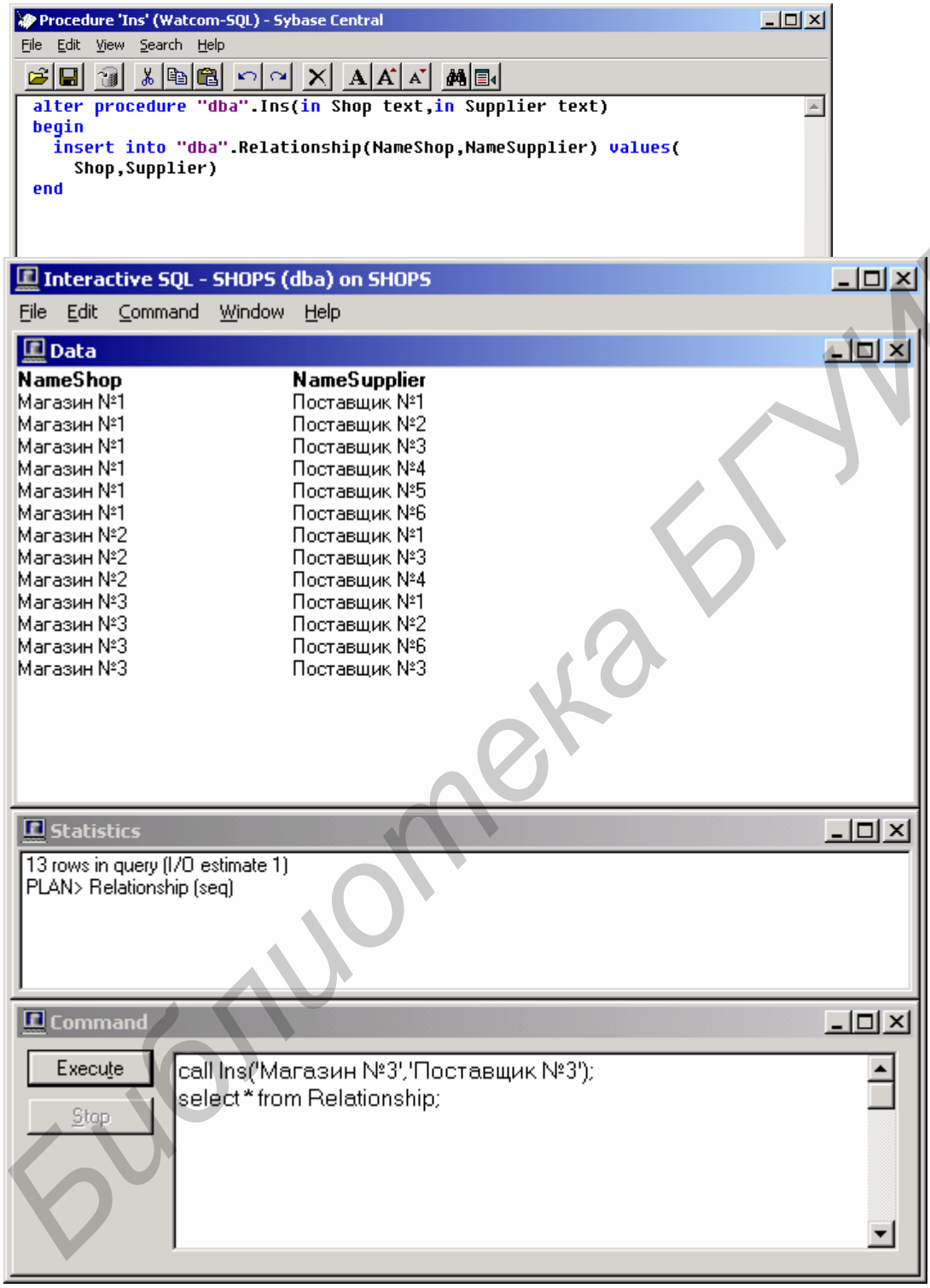




Механизм работы тот же, за исключением некоторых изменений, связанных с использованием оператора RETURN.

Вариант 3

Задача процедуры: использование процедуры для добавления записи.



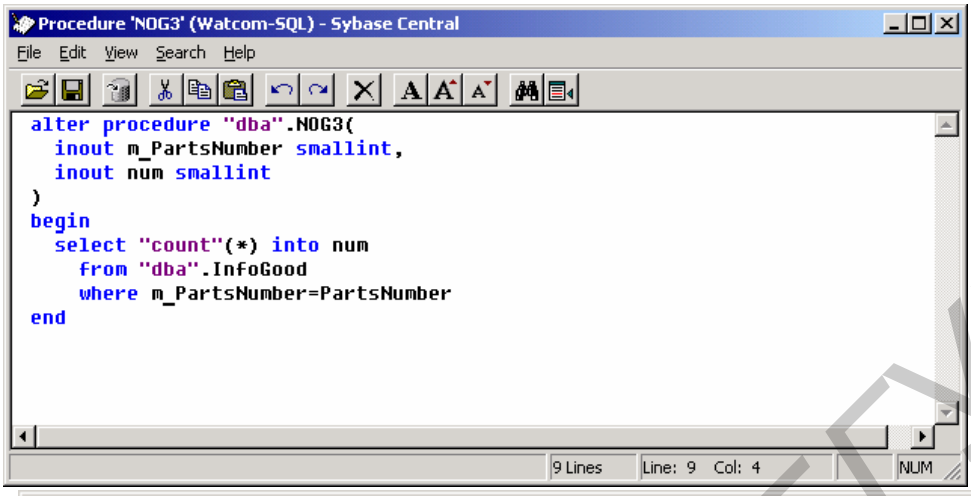
Данная процедура добавляет запись в таблицу Relationship (название магазина Shop в NameShop и Supplier в NameSupplier).

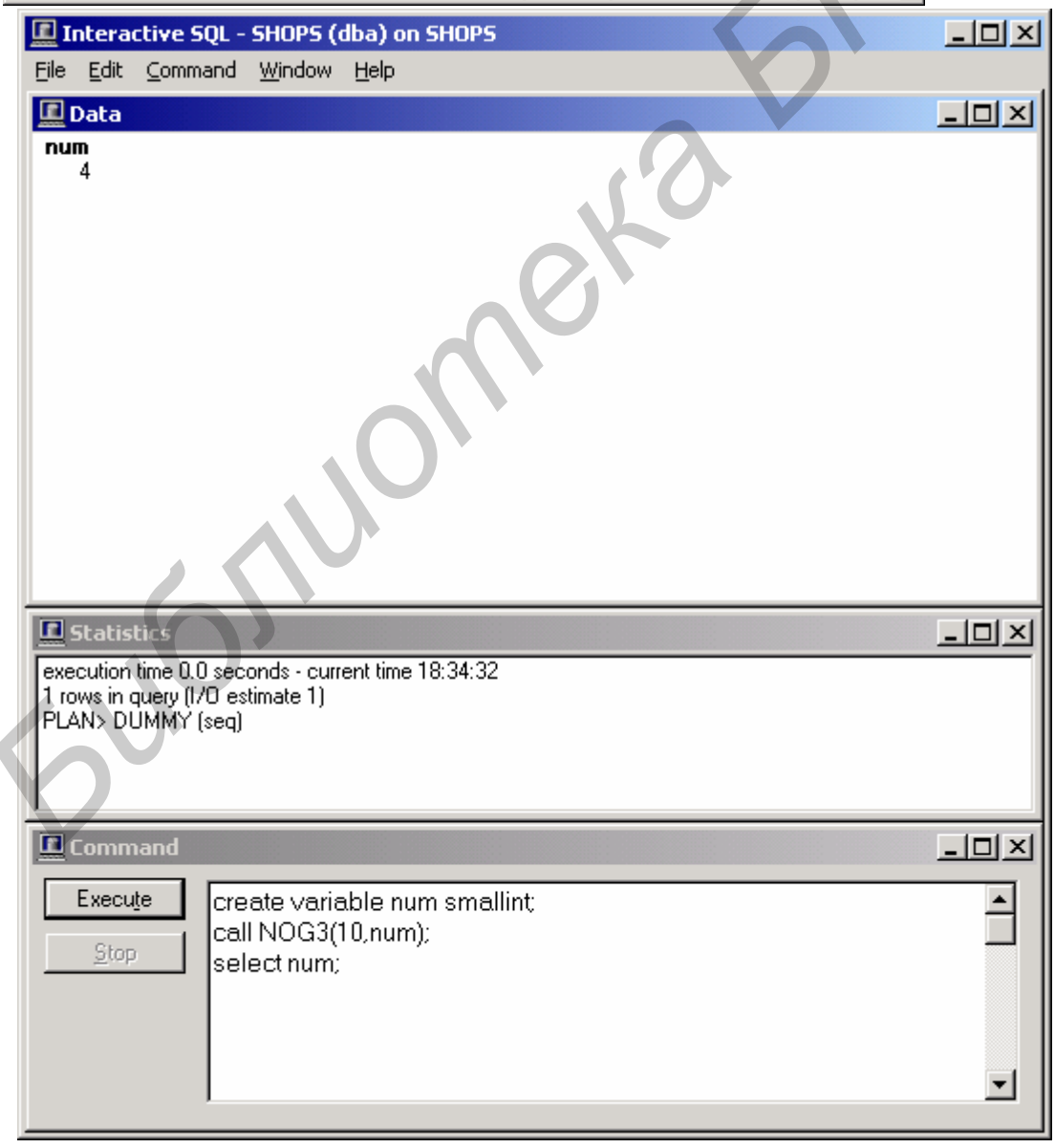
Была вызвана процедура Ins. В результате просматриваем в окне Data утилиты ISQL набор всех записей в таблице Relationship.

Вариант 4

Использование оператора INOUT.

Задача процедуры: вывести количество записей, в которых содержатся сведения о товарах, партии которых равны указанной параметром величине





Подсчитаем число товаров, число партий которых одинаково и равно введенному значению (m_PartsNumber).

Как видим, в окне Data ISQL отображается результат работы процедуры, из которого следует, что в таблице InfoGood находится 4 наименования товаров, число партий которых равно 10.

Св. план 2005, поз. 117

Учебное издание

Орешко Игорь Георгиевич
Комличенко Виталий Николаевич
Бутов Алексей Александрович и др.

ОСНОВЫ И ЛИНГВИСТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ БАЗ ДАННЫХ

Учебно-методическое пособие
для студентов специальности 40 01 02-02
«Информационные системы и технологии в экономике»
всех форм обучения

Редактор Т. П. Андрейченко
Корректор М. В. Тезина

Подписано в печать 3.03.07. Формат 60х84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л.
Уч.-изд. л. 4,0. Тираж 150 экз. Заказ 205.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6