Федеральное агентство по образованию РФ
Государственное образовательное учреждение высшего профессионального образования
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ГОУВПО «АМГУ»)

УТВЕРЖДАЮ
Проректор по учебной работе
_____ В.В. Проказин
«___»_____ 201__г.

Энергетический факультет

кафедра «Автоматизация производственных процессов и электротехники»

Учебно-методический комплекс дисциплины

БАЗЫ ДАННЫХ

для специальности
22.03.01 – автоматизация технологических процессов и производств

Составитель: А.А.Степанова
Под редакцией: А.Н. Рыбалева

Благовещенск 2007 – 2010

Печатается по решению редакционно-издательского совета энергетического факультета Амурского государственного университета

«Базы данных» для специальности 22.03.01 «Автоматизация технологических процессов и производств»: учебно-методический комплекс дисциплины./ Степанова А.А. – Благовещенск. Издательство Амурского гос. ун-та, 2007-2010 г. 78 с.

Учебно-методический комплекс дисциплины «Базы данных» представляет собой совокупность учебно-методических документов, призванных обеспечить организацию и содержательную целостность системы методов и средств обучения. Основной целью данного комплекса является систематизация содержания дисциплины, улучшение её методического обеспечения, правильное планирование и организация работы и контроля знаний студентов.

Содержание

1. Предисловие

Настоящий УМКД предназначен в помощь студентам очной формы обучения специальности 220301 «Автоматизация технологических процессов и производств» при изучении дисциплины «Базы данных».

При его написании учитывались рекомендации из положения «Об учебно-методическом комплексе дисциплины». УМКД разрабатывался на основе утвержденных в установленном порядке Государственного образовательного стандарта, типовых учебных планов и рабочей программы дисциплины, а также нормативных документов Министерства образования и науки Российской Федерации по вопросам организации учебно-воспитательного процесса.

обсужден на заседании кафедры автоматизации производственных процессов и электротехники

«_____»_____200_ г., протокол №_____

Заведующий кафедрой_____ А.Н. Рыбалев

СОГЛАСОВАНО

Начальник УМУ

Г.Н. Торопчина

(подпись, И.О.Ф.)

«___»_____200__г.

СОГЛАСОВАНО

Председатель УМС факультета

Ю.В. Мясоедов

(подпись, И.О.Ф.)

«___»_____200__г.

СОГЛАСОВАНО

Заведующий выпускающей кафедрой

А.Н. Рыбалёв

(подпись, И.О.Ф.)

«___»_____200__г.

2. Рабочая программа дисциплины.

Федеральное агентство по образованию Российской Федерации

Амурский государственный университет

УТВЕРЖДАЮ
Проректор по учебной работе

_____В.В. Проказин

«____»_____2010 г.

РАБОЧАЯ ПРОГРАММА

по дисциплине «Базы данных»

для специальности 22.03.01 «Автоматизация технологических процессов и производств»

Курс 2 Семестр 4

Лекции 18 (час.)

Лабораторные занятия 36 (час.) Зачет 4

Самостоятельная работа 21 (час.)

Всего часов 75

Составитель А.А.Степанова, старший преподаватель кафедры автоматизации производственных процессов и электротехники (2007 г.)
(И.О.Ф., должность, ученое звание)

Факультет Энергетический

Кафедра автоматизации производственных процессов и электротехники

2010 г.

Рабочая программа составлена на основании Государственного образовательного стандарта ВПО 220300 «Автоматизированные технологии и производства» и учебного плана специальности 22.03.01 «Автоматизация технологических процессов и производств»: блок специальных дисциплин, «Базы данных»

Рабочая программа обсуждена на заседании кафедры автоматизации производственных процессов и электротехники

«__»_____200__г., протокол №_____

Заведующий кафедрой_____А.Н. Рыбалев

Рабочая программа одобрена УМС 22.03.01 «Автоматизация технологических процессов и производств»

«__»_____200__г., протокол №_____

Председатель_____А.Н. Рыбалев

СОГЛАСОВАНО
Начальник УМУ
_____Г.Н. Торопчина
(подпись, И.О.Ф)
«__»_____200__г.

СОГЛАСОВАНО
Председатель УМС факультета
_____Ю.В. Мясоедов
(подпись, И.О.Ф)
«__»_____200__г.

СОГЛАСОВАНО
Заведующий выпускающей кафедрой
_____А.Н. Рыбалев
(подпись, И.О.Ф)
«__»_____200__г.

ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ, ЕЕ МЕСТО В УЧЕБНОМ ПРОЦЕССЕ

Цель преподавания дисциплины – сформировать у студентов знания основ проектирования баз данных, способов их реализации, разработки систем управления баз данных.

Задачи изучения дисциплины – освоение студентами современных средств проектирования и реализации баз данных, построения моделей данных.

В результате изучения дисциплины студент должен

знать:

– классификацию баз данных;

– методологию построения моделей данных;

- основы реляционной алгебры и языков запросов;

– программные средства проектирования баз данных и систем управления базами данных;

уметь:

– проектировать базы данных.

СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

1. ЛЕКЦИОННЫЕ ЗАНЯТИЯ (18 час.)

ТЕМА 1. ОСНОВЫ ПОСТРОЕНИЯ БАЗ ДАННЫХ – 2 часа.

Базы данных и информационные системы. Банки данных. База данных. Система управления базами данных. Словарь данных. Администратор базы данных.

Архитектура информационной системы. Сервер БД, клиент. Файл-сервер, SQL-сервер.

Классификация СУБД. полнофункциональные СУБД; серверы БД; клиенты БД; средства разработки программ работы с БД. Персональные СУБД, многопользовательские СУБД.

Способы разработки и выполнения приложений. Схема обмена данными при работе с БД

ТЕМА 2. МОДЕЛИ И ТИПЫ ДАННЫХ – 4 часа.

Модель представления данных

Иерархическая модель. Достоинства и недостатки модели.

Сетевая модель. Достоинства и недостатки модели.

Реляционная модель. Понятие отношения. Достоинства и недостатки модели. Постреляционная модель

Многомерная модель. Измерения, ячейки. Достоинства и недостатки модели.

Объектно-ориентированная модель. Инкапсуляция, Наследование, Полиморфизм.

Основные типы данных СУБД.

ТЕМА 3. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ – 4 часа.

Определение реляционной модели. Отношение, сущность, атрибуты, домен.

Схема отношения. Первичный ключ, ссылочная целостность. Индекси-

рование. Индекс, методы поиска.

Связывание таблиц. Основные виды связи таблиц: Связь вида 1:1, связь вида 1:M, связь вида M:1, связь вида M:M. Контроль целостности связей.

Теоретические языки запросов. Реляционная алгебра. Языки исчислений. Структурированный язык запросов SQL. Основные операторы языка.

ТЕМА 4. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ – 4 часа.

Проблемы проектирования. Логическое проектирование, проектирование структур. Избыточное дублирование данных и аномалии

Формирование исходного отношения. Метод нормальных форм. Зависимости между атрибутами. Функциональная взаимозависимость. Частичная зависимость. Транзитивная зависимость

Первая нормальная форма (1НФ); Вторая нормальная форма (2НФ); третья нормальная форма (ЗНФ); усиленная третья нормальная форма, или нормальная форма Бойса -Кодда(БКНФ); четвертая нормальная форма (4НФ); пятая нормальная форма (5НФ).

Рекомендации по разработке структур. Организация связи сущностей. Обеспечение целостности

ТЕМА 5. МЕТОД СУЩНОСТЬ-СВЯЗЬ – 2 часа.

Основные понятия метода: сущность, атрибут сущности, ключ сущности, связь между сущностями, степень связи, класс принадлежности экземпляров сущности, диаграммы ER-экземпляров, диаграммы ER-типа.

Этапы проектирования.

Правила формирования отношений. Формирование отношений для связи 1:1. Формирование отношений для связи 1:M. Формирование отношений для связи M:M

ТЕМА 6. ИНФОРМАЦИОННЫЕ СИСТЕМЫ В СЕТЯХ – 2 часа.

Модели архитектуры клиент-сервер. Двухзвенные модели распределения функций. Модели удаленного доступа к данным. Модель сервера БД. Модель распределенной БД.

Трехзвенная модель распределения функций. Модель монитора транзакций. Мониторы обработки транзакций. Тупики. Протоколы фиксации транзакций. Откат транзакции. Протокол трехфазной фиксации транзакций.

Информационные системы в локальных сетях. Сетевые СУБД.

2. ЛАБОРАТОРНЫЕ ЗАНЯТИЯ (36 часов)

1. Знакомство со средой Access (4 ч)
2. Конструктор БД. Создание таблиц. (2 ч)
3. Конструктор БД. Создание форм (2 ч)
4. Конструктор БД. Создание отчетов (4 ч)
5. Запросы. Простые запросы на выборку(2 ч)
6. Запросы. Запросы на выборку из 2 и более таблиц (2 ч)
7. Запросы. Запросы с группировкой (2 ч)
8. Запросы. Подзапросы (2 ч)
9. Запросы. Модификация таблиц БД (4 ч)
10. Создание БД. Разработка структуры БД (2 ч)
11. Создание БД. Разработка сложных форм (2 ч)
12. Создание БД. Разработка отчетов на основе запросов (2 ч)
13. Создание БД. Макросы (6 ч)

Все лабораторные работы проводятся в компьютерном классе.

3. САМОСТОЯТЕЛЬНАЯ РАБОТА (68 часов)

Самостоятельная работа студентов по дисциплине предусматривает выполнение индивидуального задания.

Индивидуальное задание (21 час)

1. Необходимо в соответствии со своей предметной областью спроектировать БД.

2. Создать базу данных.

3.Занести в нее данные.

4.Организовать постоянные связи между таблицами для обеспечения целостности своей БД при: изменении записей, добавлении записей, удалении записей.

5.Убедиться, что:

• данные, внесенные в таблицы, непротиворечивы;

• система поддержки целостности БД функционирует. Для этого попытаться изменить, ввести и удалить данные в таблицах с нарушением правил поддержания целостности БД.

6. Организовать запросы к БД, которые позволяли бы продемонстрировать:

7. Оформить отчет, используя Конструктор отчетов.

Предметные области:

1. Учебный центр
2. Автовокзал
3. Магазин
4. Кулинарная книга
5. Библиотека
6. Агентство недвижимости
7. Поликлиника
8. Такси

Формы контроля самостоятельной работы:

– промежуточные аттестации;

– зачет;

4. ПЕРЕЧЕНЬ И ТЕМЫ ПРОМЕЖУТОЧНЫХ ФОРМ КОНТРОЛЯ ЗНАНИЙ

Промежуточный контроль знаний студентов по дисциплине преду-

сматривает две контрольные точки в 4 семестре, по которым выставляются оценки на основе информации о выполнении лабораторных работ и сдаче зачета по каждой из 4 тем лабораторных работ.

5. ЗАЧЕТ

Зачет сдается по теоретическим вопросам и задачам на следующие темы:

1. Конструктор БД.
2. Запросы.
3. Создание БД.

Теоретические вопросы:

1. Базы данных и информационные системы.
2. Банки данных. Базы данных.
3. Система управления базами данных.
4. Словарь данных. Администратор базы данных.
5. Архитектура информационной системы.
6. Сервер БД, клиент. Файл-сервер, SQL-сервер.
7. Классификация СУБД.
8. Средства разработки программ работы с БД.
9. Способы разработки и выполнения приложений. Схема обмена данными при работе с БД
10. Модель представления данных
11. Иерархическая модель. Достоинства и недостатки модели.
12. Сетевая модель. Достоинства и недостатки модели.
13. Реляционная модель. Понятие отношения. Достоинства и недостатки модели.
14. Многомерная модель. Измерения, ячейки. Достоинства и недостатки модели.
15. Объектно-ориентированная модель.

16. Основные типы данных СУБД.

17. Определение реляционной модели. Отношение, сущность, атрибуты, домен.

18. Схема отношения. Первичный ключ, ссылочная целостность.

19. Индексирование. Индекс, методы поиска

20. Связывание таблиц. Основные виды связи таблиц

21. Контроль целостности связей

22. Теоретические языки запросов. Реляционная алгебра.

23. Языки исчислений.

24. Структурированный язык запросов SQL. Основные операторы языка.

25. Логическое проектирование, проектирование структур.

26. Избыточное дублирование данных и аномалии

27. Формирование исходного отношения.

28. Метод нормальных форм.

29. Зависимости между атрибутами. Функциональная взаимозависимость.

30. Первая нормальная форма (1НФ);

31. Вторая нормальная форма (2НФ);

32. Третья нормальная форма (3НФ); усиленная третья нормальная форма, или нормальная форма Бойса -Кодда(БКНФ);

33. Четвертая нормальная форма (4НФ);

34. Пятая нормальная форма (5НФ).

35. Организация связи сущностей. Обеспечение целостности

36. Основные понятия метода «сущность-связь»

37. Этапы проектирования.

38. Правила формирования отношений.

39. Модели архитектуры клиент-сервер.

40. Двухзвенные модели распределения функций.

41. Трехзвенная модель распределения функций.

42. Сетевые СУБД.

УЧЕБНО-МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ПО ДИСЦИПЛИНЕ

1.ПЕРЕЧЕНЬ ОБЯЗАТЕЛЬНОЙ (ОСНОВНОЙ) ЛИТЕРАТУРЫ

1.1. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных. Учебник для высших учебных заведений / М.:Бином-Пресс, 2006 – 736 с.

1.2. Советов Б.Я., Цехановский В.В., Чертовский В.Д. Базы данных. Теория и практика. – М.: Высшая школа, 2005.

1.3. Михеева В., Харитонова И. Microsoft Access 2002, — СПб.: БХВ-Петербург, 2002.-1040 с.

1.4. Бородаев В. А., Кустов В. Н. Банки и базы данных. Уч. пособие. Л.: ВИКИ, 1989.

1.5. ЗамулинА.В. Системы программирования баз данных и знаний. Новосибирск.: Наука. Сиб. отд., 1990.

2.ПЕРЕЧЕНЬ ДОПОЛНИТЕЛЬНОЙ ЛИТЕРАТУРЫ

2.1. ВейскасД, Эффективная работа с Microsoft Access 97. — СПб.: Питер Ком, 1999. -976 с.

2.2. Мартин Дж. Организация баз данных в вычислительных системах. / Пер. с англ. М.: Мир, 1980.

2.3. Романов Б. А., Кушниренко А. С. dBase IV. Назначение, функции, применение. М.: Радио и связь, 1991.

2.4. Ульман Дж. Основы систем баз данных. М.: Финансы и статистика, 1983.

2.5. Кузнецов С. Д. Введение в СУБД // Системы Управления Базами Данных, № 4, 1995. - С. 114-122.

.

УЧЕБНО-МЕТОДИЧЕСКАЯ (ТЕХНОЛОГИЧЕСКАЯ) КАРТА ДИСЦИПЛИНЫ

Номер недели	Номер темы	Вопросы, изучаемые на лекции	Лабораторные занятия (номера)	Используемые наглядные и методические пособия	Самостоятельная работа студентов		Формы контроля
					содержание	час.	
1	2	3	4	5	6	7	8
1	1	Базы данных и информационные системы. Банки данных. База данных. Способы разработки и выполнения приложений. Схема обмена данными при работе с БД	Среда разработки Access				Блиц-опрос. Контрольная точка. Защита лабораторной работы.
2			Среда разработки Access		Подготовка к зачету по теме «Среда разработки».	2	Защита лабораторной работы..Зачет по теме «Среда разработки».
3	2	Модель представления данных. Иерархическая модель. Сетевая модель. Реляционная модель. Понятие отношения.	Конструктор БД. Создание таблиц.				Блиц-опрос. Контрольная точка. Защита лабораторной работы.
4			Конструктор БД. Создание форм				Контрольная точка. Защита лабораторной работы.
5	2	Постреляционная модель Многомерная модель. Измерения, ячейки. Объектно-ориентированная модель. Инкапсуляция, Наследование, Полиморфизм. Основные типы данных СУБД.	Конструктор БД. Создание отчетов		Подготовка к контрольной работе по теме 2.	2	Блиц-опрос. Контрольная точка. Защита лабораторной работе. Контрольная работа по теме 2
6			Конструктор БД. Создание отчетов		Подготовка к зачету по теме по теме «Конструктор БД»	2	Защита лабораторной работы . Зачет по теме «Конструктор БД»

1	2	3	4	5	6	7	8
7	3	Определение реляционной модели. Отношение, сущность, атрибуты, домен. Схема отношения. Первичный ключ, ссылочная целостность. Связывание таблиц. Основные виды связи таблиц: Связь вида 1:1, связь вида 1:M, связь вида M:1, связь вида M:M. Контроль целостности связей	Запросы. Простые запросы на выборку				Блиц-опрос. Контрольная точка. Защита лабораторной работы.
8			Запросы. Запросы на выборку из 2 и более таблиц				Блиц-опрос. Контрольная точка. Защита лабораторной работы.
9	3	Теоретические языки запросов. Реляционная алгебра. Языки исчислений. Структурированный язык запросов SQL. Основные операторы языка.	Запросы. Запросы с группировкой		Подготовка к контрольной работе по теме 3	2	Блиц-опрос. Контрольная точка. Защита лабораторной работы. Контрольная работа по теме 3
10			Запросы. Подзапросы				Блиц-опрос. Контрольная точка. Защита лабораторной работы.
11	4	Проблемы проектирования. Логическое проектирование, проектирование структур. Избыточное дублирование данных и аномалии Формирование исходного отношения. Метод нормальных форм. Зависимости между атрибутами.	Запросы. Модификация таблиц БД				Блиц-опрос. Контрольная точка. Защита лабораторной работы.
12			Запросы. Модификация таблиц БД		Подготовка к зачету по теме 3.	2	Защита лабораторной работы. Зачет по теме 3.
13	4	Первая нормальная форма (1НФ); Вторая нормальная форма (2НФ); третья нормальная форма (3НФ); усиленная третья нормальная	Создание БД. Разработка структуры БД		Решение индивидуального задания.	2	Блиц-опрос. Контрольная точка. Защита лабораторной работы. Проверка выполнения индивидуального задания.

1	2	3	4	5	6	7	8
		форма, или нормальная форма Бойса -Кодда(БКНФ); четвертая нормальная форма (4НФ); пятая нормальная форма (5НФ).					
14			Создание БД. Разработка сложных форм		Решение индивидуального задания.	2	Блиц-опрос. Контрольная точка. Защита лабораторной работы. Проверка выполнения индивидуального задания.
15	5	Основные понятия метода сущность-связь: сущность, атрибут сущности, Правила формирования отношений. Формирование отношений для связи 1:1. Формирование отношений для связи 1:М. Формирование отношений для связи М:М	Создание БД. Разработка отчетов на основе запросов		Решение индивидуального задания.	2	Блиц-опрос. Контрольная точка. Защита лабораторной работы. Проверка выполнения индивидуального задания.
16			Создание БД. Макросы		Решение индивидуального задания.	2	Блиц-опрос. Контрольная точка. Защита лабораторной работы. Проверка выполнения индивидуального задания.
17	6	Модели архитектуры клиент-сервер. Двухзвенные модели распределения функций. Трехзвенная модель распределения функций. Модель монитора транзакций. Тупики. Протоколы фиксации транзакций. Информационные системы в локальных сетях. Сетевые СУБД.			Решение индивидуального задания.	2	Блиц-опрос. Контрольная точка. Защита лабораторной работы. Проверка выполнения индивидуального задания.
18			Создание БД. Макросы		Подготовка к зачету по теме 4.	1	Защита лабораторной работы . Зачет по теме 4

3. График самостоятельной работы студентов по дисциплине на каждый семестр с указанием ее содержания, объема в часах, сроков и форм контроля.

№ темы	Содержание	Самостоятельная работа студентов		Формы контроля
		Срок, уч. недели	Час	
1	Среда разработки Access	2	2	Подготовка к зачету по теме «Среда разработки
2	Модели данных	5	2	Контрольная работа по теме
3	Подготовка к зачету по теме по теме «Конструктор БД»	6	2	Зачет по теме «Конструктор БД»
4	Теоретические языки запросов. Реляционная алгебра. Структурированный язык запросов SQL.	9	2	Контрольная работа по теме
5	Модификация таблиц БД	12	2	Зачет по теме
6	Разработка БД	17	10	Проверка выполнения индивидуального задания
7	Подготовка к зачету по теме 4.	18	1	Зачет по теме

4. Методические рекомендации по проведению лабораторных занятий.

Лабораторные занятия предусмотрены в рабочей программе в объеме 36 часов. Тематика лабораторных занятий представлена в таблице.

№ темы	Название темы	Кол-во часов
1	Среда разработки Access	4
2	Конструктор БД. Создание таблиц.	2
3	Конструктор БД. Создание форм	2
4	Конструктор БД. Создание отчетов	4
5	Запросы. Простые запросы на выборку	2
6	Запросы. Запросы на выборку из 2 и более таблиц	2

7	Запросы. Запросы с группировкой	2
8	Запросы. Подзапросы	2
9	Запросы. Модификация таблиц БД	4
10	Создание БД. Разработка структуры БД	2
11	Создание БД. Разработка сложных форм	2
12	Создание БД. Разработка отчетов на основе запросов	2
13	Создание БД. Макросы	6

Цель лабораторных занятий – научить студентов работать в среде Access, разрабатывать базы данных, создавать таблицы, формы, простые и сложные запросы, а также отчеты на основе запросов.

5. Конспект лекций

ТЕМА 1. ОСНОВЫ ПОСТРОЕНИЯ БАЗ ДАННЫХ

1.1. Базы данных и информационные системы

В основе решения многих задач лежит обработка информации. Для облегчения обработки информации создаются информационные системы (ИС). Автоматизированными называют ИС, в которых применяют технические средства, в частности ЭВМ. Большинство существующих ИС являются автоматизированными, поэтому для краткости просто будем называть их ИС.

В *широком понимании* под определение ИС подпадает любая система обработки информации. По *области применения* ИС можно разделить на системы, используемые в производстве, образовании, здравоохранении, науке, военном деле, социальной сфере, торговле и других отраслях. По *целевой функции* ИС можно условно разделить на следующие основные категории: управляющие, информационно-справочные, поддержки принятия решений.

Заметим, что иногда используется более *узкая трактовка понятия ИС* как совокупности аппаратно-программных средств, задействованных для решения некоторой прикладной задачи. В организации, например, могут существовать информационные системы, на которых соответственно возложены следующие задачи: учет кадров и материально-технических средств, расчете поставщиками и заказчиками, бухгалтерский учет и т. п.

Банк данных является разновидностью ИС, в которой реализованы функции централизованного хранения и накопления обрабатываемой информации, организованной в одну или несколько баз данных.

Банк данных (БнД) в общем случае состоит из следующих компонентов: базы (нескольких баз) данных, системы управления базами данных, словаря данных, администратора, вычислительной системы и обслуживающего персонала. Вкратце рассмотрим названные компоненты и некоторые связанные с ними важные понятия.

База данных (БД) представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

БД бывают *централизованными* (хранятся на одном компьютере) и *распределенными* (хранятся на нескольких компьютерах некоторой сети).

Логическую структуру хранимых в базе данных называют ***моделью представления данных.*** К основным моделям представления данных (моделям данных) относятся следующие: иерархическая, сетевая, реляционная, постреляционная, многомерная и объектно-ориентированная.

Система управления базами данных (СУБД) — это комплекс языковых и программных средств, предназначенный для создания, ведения и

совместного использования БД многими пользователями. Обычно СУБД различают по используемой модели данных. Так, СУБД, основанные на использовании реляционной модели данных, называют реляционными СУБД.

Приложение представляет собой программу или комплекс программ, обеспечивающих автоматизацию обработки информации для прикладной задачи. Нами рассматриваются приложения, использующие БД. Приложения могут создаваться в среде или вне среды СУБД — с помощью системы программирования, использующей средства доступа к БД, к примеру Delphi или C++ Builder. Приложения, разработанные в среде СУБД, часто называют *приложениями СУБД*, а приложения, разработанные вне СУБД, — *внешними приложениями*.

Для работы с базой данных зачастую достаточно средств СУБД и не нужно использовать приложения, создание которых обычно требует программирования. Приложения разрабатывают главным образом в случаях, когда требуется сделать работу пользователей более удобной или автоматизировать рутинные операции с БД.

Словарь данных (СД) представляет собой подсистему БнД, предназначенную для централизованного хранения информации о структурах данных, взаимосвязях файлов БД друг с другом, типах данных и форматах их представления, принадлежности данных пользователям, кодах защиты и разграничения доступа и т. п.

Словарь данных, иначе называемый системным каталогом, как следует из определения, является хранилищем служебной информации о данных в базе («данных о данных», или метаданных).

Функционально СД присутствует во всех БнД, но не всегда выполняющий эти функции компонент имеет именно такое название. Чаще всего функции СД выполняются СУБД и вызываются из основного меню системы или реализуются с помощью ее утилит.

Если СД является частью БД, то его называют *интегрированным СД*, в противном случае СД является *автономным*. Автономные словари данных обычно используют не только в интересах собственно данных базы, но и в целях управления другими информационными ресурсами организаций при разработке структур баз данных на этапе проектирования, для ведения документации, управления проектами и т. д.

Стандартизация интерфейса СД привела к разработке службы словаря информационных ресурсов (Information Resource Dictionary System - IRDS). Служба IRDS имеет четыре интерфейса: графический, командный язык, экспорта/импорта и прикладных программ. Реализация IRDS представляет собой программный инструмент для унифицированного управления различны: ми информационными ресурсами организации группами пользователей и приложениями. Введение IRDS может быть целесообразно на ранних этапах проектирования БД организации, когда необходимо отложить привязку БД к конкретной СУБД. Кроме того, с помощью служб IRDS можно переносить информацию между IRDS-совместимыми СД раз-

личных СУБД (независимо от используемой в них модели данных).

Администратор базы данных (АБД) есть лицо или группа лиц, отвечающих за выработку требований к БД, ее проектирование, создание, эффективное использование и сопровождение. В процессе эксплуатации АБД обычно следит за функционированием информационной системы, обеспечивает защиту от несанкционированного доступа, контролирует избыточность, непротиворечивость, сохранность и достоверность хранимой в БД информации. Для однопользовательских информационных систем функции АБД обычно возлагаются на лиц, непосредственно работающих с приложением БД.

В вычислительной сети АБД, как правило, взаимодействует с *администратором сети.* В обязанности последнего входят контроль за функционированием аппаратно-программных средств сети, реконфигурация сети, восстановление программного обеспечения после сбоев и отказов оборудования, профилактические мероприятия и обеспечение разграничения доступа.

Вычислительная система (ВС) представляет собой совокупность взаимосвязанных и согласованно действующих ЭВМ или процессоров и других устройств, обеспечивающих автоматизацию процессов приема, обработки и выдачи информации потребителям. Поскольку основными функциями БнД являются хранение и обработка данных, то используемая ВС, наряду с приемлемой мощностью центральных процессоров (ЦП) должна иметь достаточный объем оперативной и внешней памяти прямого доступа.

Обслуживающий персонал выполняет функции поддержания технических и программных средств в работоспособном состоянии. Он проводит профилактические, регламентные, восстановительные и другие работы по планам, а также по мере необходимости.

1.2. Архитектура информационной системы

Эффективность функционирования информационной системы (ИС) во многом зависит от ее архитектуры. В настоящее время перспективной является архитектура клиент-сервер. В достаточно распространенном варианте она предполагает наличие компьютерной сети и распределенной базы данных, включающей корпоративную базу данных (КБД) и персональные базы данных (ПБД). КБД размещается на компьютере-сервере, ПБД размещаются на компьютерах сотрудников подразделений, являющихся клиентами корпоративной БД.

Сервером определенного ресурса в компьютерной сети называется компьютер (программа), управляющий этим ресурсом, ***клиентом*** — компьютер (программа), использующий этот ресурс. В качестве ресурса компьютерной сети могут выступать, к примеру, базы данных, файловые системы, службы печати, почтовые службы. Тип сервера определяется видом ресурса, которым он управляет. Например, если управляемым ресурсом является база данных, то соответствующий сервер называется *сервером базы, данных.*

Достоинством организации информационной системы по архитектуре клиент-сервер является удачное сочетание *централизованного* хранения,

обслуживания и коллективного доступа к общей корпоративной информации с индивидуальной работой пользователей над персональной информацией. Архитектура клиент-сервер допускает различные варианты реализации.

Исторически первыми появились распределенные ИС с применением ***файл-сервера*** (рис. 1.1). В таких ИС по запросам пользователей файлы базы данных передаются на персональные компьютеры (ПК), где и производится их обработка. *Недостатком* такого варианта архитектуры является высокая интенсивность передачи обрабатываемых данных. Причем зачастую передаются избыточные данные: вне зависимости от того сколько записей из базы данных требуется пользователю, файлы базы данных передаются целиком.

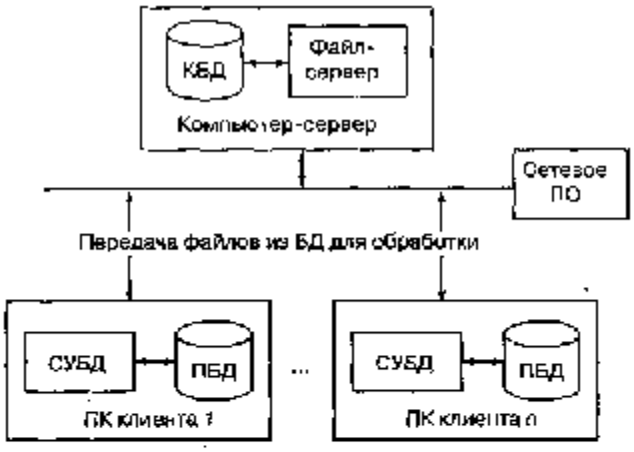
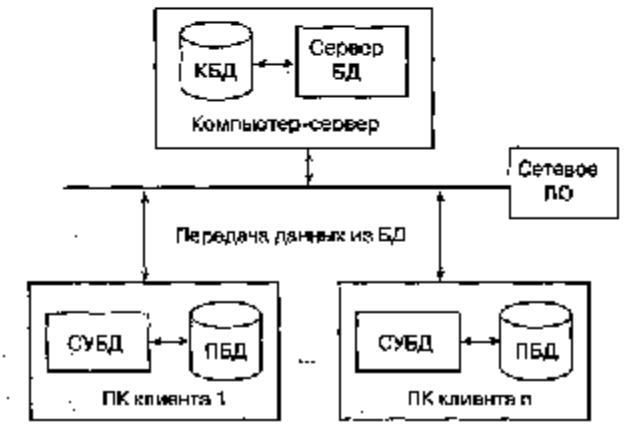


Рис. 1.1. Структура ИС с файл-сервером



Структура распределенной ИС, построенной по архитектуре клиент-сервер с использованием сервера баз данных, показана на рис. 1.2. При такой архитектуре сервер базы данных обеспечивает выполнение основного объема обработки данных. Формируемые пользователем или приложением запросы поступают к серверу БД в виде инструкций языка SQL. Сервер базы данных выполняет поиск и извлечение нужных данных, которые затем передаются на компьютер пользователя. *Достоинством* такого подхода в сравнении предыдущим является заметно меньший объем передаваемых

данных.

Для создания и управления персональными БД и приложений, работающих с ними, используются СУБД, такие как Access и Visual FoxPro фирмы Microsoft, Paradox фирмы Borland.

Корпоративная БД создается, поддерживается и функционирует под управлением сервера БД, например Microsoft SQL Server или Oracle Server.

В зависимости от размеров организации и особенностей решаемых задач информационная система может иметь одну из следующих конфигураций:

- компьютер-сервер, содержащий корпоративную и персональные базы;
- компьютер-сервер и персональные компьютеры с ПБД;
- несколько компьютеров-серверов и персональных компьютеров с ПБД.

Использование архитектуры клиент-сервер дает возможность постепенного наращивания информационной системы предприятия, во-первых, по мере развития предприятия; во-вторых, по мере развития самой информационной системы.

Разделение общей БД на корпоративную БД и персональные БД позволяет уменьшить сложность проектирования БД по сравнению с централизованным вариантом, а значит, снизить вероятность ошибок при проектировании и стоимость проектирования.

Важнейшим *достоинством* применения БД в информационных системах является обеспечение независимости данных от прикладных программ. Это дает возможность пользователям не заниматься проблемами представления данных на физическом уровне: размещения данных в памяти, методов доступа к ним и т. д.

Такая независимость достигается поддерживаемым СУБД многоуровневым представлением данных в БД на логическом (пользовательском) и физическом уровнях. Благодаря СУБД и наличию логического уровня представления данных обеспечивается отделение концептуальной (понятийной) модели БД от ее физического представления в памяти ЭВМ.

1.3. Системы управления базами данных

В этом подразделе приводится классификация СУБД и рассматриваются основные их функции. В качестве основных классификационных признаков можно использовать следующие: вид программы, характер использования, модель данных. Названные признаки существенно влияют на целевой выбор СУБД и эффективность использования разрабатываемой информационной системы.

Классификация СУБД. В общем случае под СУБД можно понимать любой программный продукт, поддерживающий процессы создания, ведения и использования БД. Рассмотрим, какие из имеющихся на рынке программ имеют отношение к БД и в какой мере они связаны с базами данных.

К СУБД относятся следующие основные виды программ:

- полнофункциональные СУБД;

• серверы БД;
• клиенты БД;
• средства разработки программ работы с БД.

Полнофункциональные ***СУБД*** (ПФСУБД) представляют собой традиционные СУБД, которые сначала появились для больших машин, затем для мини-машин и для ПЭВМ. Из числа всех СУБД современные ПФСУБД являются наиболее многочисленными и мощными по своим возможностям. К ПФСУБД относятся, например, такие пакеты, как Clarion Database Developer, DataEase, DataFlex, dBase IV, Microsoft Access, Microsoft FoxPro и Paradox R:BASE.

Обычно ПФСУБД имеют развитый интерфейс, позволяющий с помощью команд меню выполнять основные действия с БД: создавать и модифицировать структуры таблиц, вводить данные, формировать запросы, разрабатывать отчеты, выводить их на печать и т. п. Для создания запросов и отчетов не обязательно программирование, а удобно пользоваться языком QBE (Query By Example — формулировки запросов по образцу). Многие ПФСУБД включают средства программирования для профессиональных разработчиков.

Некоторые системы имеют в качестве вспомогательных и дополнительные средства проектирования схем БД или CASE-подсистемы. Для обеспечения доступа к другим БД или к данным SQL-серверов полнофункциональные СУБД имеют факультативные модули.

Серверы БД предназначены для организации центров обработки данных в сетях ЭВМ. Эта группа БД в настоящее время менее многочисленна, но их количество постепенно растет. Серверы БД реализуют функции управления базами данных, запрашиваемые другими (клиентскими) программами обычно с помощью операторов SQL.

Примерами серверов БД являются следующие программы: NetWare SQL (Novell), MS SQL Server (Microsoft), InterBase (Borland), SQLBase Server (Gupta), Intelligent Database (Ingress).

В роли ***клиентских программ*** для серверов БД в общем случае могут использоваться различные программы: ПФСУБД, электронные таблицы, текстовые процессоры, программы электронной почты и т. д. При этом элементы пары «клиент — сервер» могут принадлежать одному или разным производителям программного обеспечения.

В случае, когда клиентская и серверная части выполнены одной фирмой, естественно ожидать, что распределение функций между ними выполнено рационально. В остальных случаях обычно преследуется цель обеспечения доступа к данным «любой ценой». Примером такого соединения является случай, когда одна из полнофункциональных СУБД играет роль сервера, а вторая СУБД (другого производителя) — роль клиента. Так, для сервера БД SQL Server (Microsoft) в роли клиентских (фронтальных) программ могут выступать многие СУБД, такие как dBASE IV, Blyth Software, Paradox, DataEase, Focus, 1-2-3, MDBS III, Revelation и другие.

Средства разработки программ работы с **БД** могут использовать-

ся для создания разновидностей следующих программ:

- клиентских программ;
- серверов БД и их отдельных компонентов;
- пользовательских приложений.

Программы первого и второго вида довольно малочисленны, так как предназначены, главным образом, для системных программистов. Пакетов третьего вида гораздо больше, но меньше, чем полнофункциональных СУБД.

К средствам разработки пользовательских приложений относятся системы программирования, например Clipper, разнообразные библиотеки программ для различных языков программирования, а также пакеты автоматизации разработок (в том числе систем типа клиент-сервер). В числе наиболее распространенных можно назвать следующие инструментальные системы: Delphi и Power Builder (Borland), Visual Studio (Microsoft), SILVERRUN (Computer Advisers Inc.), S-Designor (SDP и Powersoft) и ERwin (LogicWorks).

Если говорить о конкретных системах программирования (для языков C++, C#, Visual Basic, Java и др.), то все они содержат некоторые средства доступа к наиболее широко используемым БД.

Кроме перечисленных средств, для управления данными и организации обслуживания БД используются различные дополнительные средства, к примеру мониторы транзакций (см. подраздел 4.2).

По характеру использования СУБД делят на *персональные* и *многопользовательские*.

Персональные СУБД обычно обеспечивают возможность создания персональных БД и недорогих приложений, работающих с ними. Персональные СУБД или разработанные с их помощью приложения зачастую могут выступать в роли клиентской части многопользовательской СУБД. К персональным СУБД, например, относятся Visual FoxPro, Paradox, Clipper, dBase, Access и др.

Многопользовательские СУБД включают в себя сервер БД и клиентскую часть и, как правило, могут работать в неоднородной вычислительной среде (с разными типами ЭВМ и операционными системами). К многопользовательским СУБД относятся, например, СУБД Oracle и Informix.

В зависимости от способа хранения и обработки БД (централизованного или децентрализованного) СУБД можно разделить на два класса: *централизованные* (или обычные) СУБД и *децентрализованные* (или распределенные) СУБД. В обычных СУБД данные хранятся в том же месте, где и программы их управления. В распределенных СУБД как программное обеспечение, так и данные распределены по узлам сети. Распределенные СУБД могут быть *однородными* или *неоднородными*. Неоднородность СУБД может проявляться в отличии поддерживаемых модели данных, типов данных, языков запросов, фирм-разработчиков и т. д.

Одной из разновидностей распределенных СУБД являются *мультибазовые системы,* в которых управление каждым из узлов осуществляется автономно. В мультибазовых СУБД производится такая интеграция локальных систем, при которой не требуется изменение существующих СУБД и в то же время конечным пользователям предоставляется доступ к совместно используемым данным. Пользователи локальных СУБД получают возможность управлять данными собственных узлов без централизованного контроля, который присутствует в обычных распределенных СУБД. Примером мультибазовой СУБД является система UniSQL компании Cincom Corporation.

В зависимости от возможности распараллеливания процесса обработки данных выделяют *СУБД с последовательной и параллельной обработкой (параллельные СУБД).* Параллельные СУБД функционируют в многопроцессорной вычислительной системе (как правило, со множеством устройств хранения данных) или в сети компьютеров.

По используемой модели данных СУБД (как и БД), разделяют на иерархические, сетевые, реляционные, объектно-ориентированные и другие типы. Некоторые СУБД могут одновременно поддерживать несколько моделей данных.

С точки зрения пользователя, СУБД реализует ***функции*** хранения, изменения (пополнения, редактирования и удаления) и обработки информации, а также разработки и получения различных выходных документов.

Для работы с хранящейся в базе данных информацией СУБД предоставляет программам и пользователям следующие два типа *языков:*

- язык описания данных — высокоуровневый непроцедурный язык декларативного типа, предназначенный для описания логической структуры данных;
- язык манипулирования данными — совокупность конструкций, обеспечивающих выполнение основных операций по работе с данными: ввод, модификацию и выборку данных по запросам.

Названные языки в различных СУБД могут иметь отличия. Наибольшее распространение получили два стандартизованных языка: QBE (Query By Example) — язык, запросов по образцу и SQL (Structured Query Language) — структурированный язык запросов. QBE в основном обладает свойствами языка *манипулирования* данными, SQL сочетает в себе свойства языков обоих типов — *описания* и *манипулирования* данными.

Перечисленные выше функции СУБД, в свою очередь, используют следующие основные функции более низкого уровня, которые назовем ***низкоуровневыми:***

- управление данными во внешней памяти;
- управление буферами оперативной памяти;
- управление транзакциями;
- ведение журнала изменений в БД;
- обеспечение целостности и безопасности БД.

Реализация функции *управления данными во внешней памяти* в раз-

ных системах может различаться и на уровне управления ресурсами (используя файловые системы ОС или непосредственное управление устройствами ПЭВМ), и по логике самих алгоритмов управления данными. В основном методы и алгоритмы управления данными являются «внутренним делом» СУБД и прямого отношения к пользователю не имеют. Качество реализации этой функции наиболее сильно влияет на эффективность работы специфических ИС, например, с огромными БД, со сложными запросами, большим объемом обработки данных.

Необходимость буферизации данных и как следствие реализации функции *управления буферами* оперативной памяти обусловлено тем, что объем оперативной памяти меньше объема внешней памяти.

Буферы представляют собой области оперативной памяти, предназначенные для ускорения обмена между внешней и оперативной памятью. В буферах временно хранятся фрагменты БД, данные из которых предполагается использовать при обращении к СУБД или планируется записать в базу после обработки.

Механизм транзакций используется в СУБД для поддержания целостности данных в базе. ***Транзакцией*** называется некоторая неделимая последовательность операций над данными БД, которая отслеживается СУБД от начала и до завершения. Если по каким-либо причинам (сбои и отказы оборудования, ошибки в программном обеспечении, включая приложение) транзакция остается незавершенной, то она отменяется.

В зависимости от времени, требуемого для выполнения, выделяют *обычные и продолжительные транзакции*. Продолжительные транзакции могут охватывать часы, дни и даже месяцы. Такие транзакции могут возникать в процессе проектирования и разработки сложных систем крупным коллективом людей. Кроме того, помимо обычных *плоских транзакций*, используется модель *вложенных транзакций*. В последнем случае транзакция рассматривается как набор взаимосвязанных подзадач (субтранзакций), каждая из которых также может состоять из произвольного количества субтранзакций.

Говорят, что транзакции присущи три основных свойства:

- атомарность (выполняются все входящие в транзакцию операции или ни одна);
- серализуемость (отсутствует взаимное влияние выполняемых в одно и то же время транзакций);
- долговечность (даже крах системы не приводит к утрате результатов зафиксированной транзакции).

Примером транзакции является операция перевода денег с одного счета на другой в банковской системе. Здесь необходим, по крайней мере, двухшаговый процесс. Сначала снимают деньги с одного счета, затем добавляют их к другому счету. Если хотя бы одно из действий не выполнится успешно, результат операции окажется неверным и будет нарушен баланс между счетами.

Контроль транзакций важен в однопользовательских и в многополь-

зовательских СУБД, где транзакции могут быть запущены параллельно. В последнем случае говорят о сериализуемости транзакций. Под *серилизацией* параллельно выполняемых транзакций понимается составление такого плана их выполнения (сериального плана), при котором суммарный эффект реализации транзакций эквивалентен эффекту их последовательного выполнения.

При параллельном выполнении смеси транзакций возможно возникновение конфликтов (блокировок), разрешение которых является функцией СУБД. При обнаружении таких случаев обычно производится «откат» путем отмены изменений, произведенных одной или несколькими транзакциями.

Ведение журнала изменений в БД (журнализация изменений) выполняется СУБД для обеспечения надежности хранения данных в базе при наличии аппаратных сбоев и отказов, а также ошибок в программном обеспечении.

Журнал СУБД — это особая БД или часть основной БД, непосредственно недоступная пользователю и используемая для записи информации обо всех изменениях базы данных. В различных СУБД в журнал могут заноситься записи, соответствующие изменениям в СУБД на разных уровнях: от минимальной внутренней операции модификации страницы внешней памяти до логической операции модификации БД (например, вставки записи, удаления столбца, изменения значения в поле) и даже транзакции.

Для эффективной реализации функции ведения журнала изменений в БД необходимо обеспечить повышенную надежность хранения и поддержания в рабочем состоянии самого журнала. Иногда для этого в системе хранят несколько копий журнала.

Обеспечение целостности БД составляет необходимое условие успешного функционирования БД, особенно для случая использования БД в сетях. ***Целостность БД*** *есть* свойство базы данных, означающее, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную область информация. Поддержание целостности БД включает проверку целостности и ее восстановление в случае обнаружения противоречий в базе данных. Целостное состояние БД описывается с помощью *ограничений целостности* в виде условий, которым должны удовлетворять хранимые в базе данные. Примером таких условий может служить ограничение диапазонов возможных значений атрибутов объектов, сведения о которых хранятся в БД, или отсутствие повторяющихся записей в таблицах реляционных БД.

Обеспечение безопасности достигается в СУБД шифрованием прикладных программ, данных, защиты паролем, поддержкой уровней доступа к базе данных и к отдельным ее элементам (таблицам, формам, отчетам и т. д.).

1.4. Локальные информационные системы

Функциональные части информационной системы могут размещаться на одном или на нескольких компьютерах. Рассмотрим варианты орга-

низации ИС на одном ПК. Соответствующую ИС обычно называют локальной или однопользовательской (хотя последнее не совсем строго, поскольку на одном компьютере поочередно могут работать несколько пользователей). Более сложные варианты организации И С рассматриваются в разделе 4.

Организация функционирования локальной ИС на одном компьютере в среде некоторой операционной системы (ОС) возможна с помощью следующих вариантов использования программных средств:

- «полной» СУБД;
- приложения и «усеченной» (ядра) СУБД;
- независимого приложения.'

Первый способ обычно применяется в случаях, когда в дисковой памяти компьютера помещается вся СУБД и она часто используется для доработки приложения (рис. 1.3).

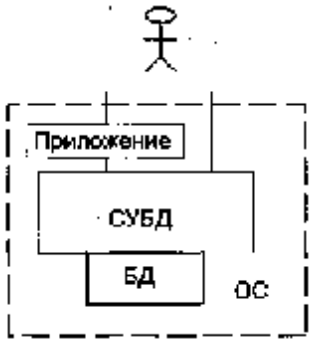


Рис. 1.3. Использование приложения и СУБД

Взаимодействие пользователя с СУБД происходит напрямую через пользовательский (терминальный) интерфейс СУБД, либо с помощью приложения.

Основное *достоинство* схемы — простота разработки и сопровождения БД и приложений при наличии развитых соответствующих средств разработки и сервисных средств. *Недостатком* этой схемы являются затраты дисковой памяти на хранение программы СУБД и оперативной памяти для исполняемого кода.

Приложение с ядром СУБД (рис. 1.4) используют для достижения следующих целей:

- уменьшения объема занимаемого СУБД пространства жесткого диска и оперативной памяти;
- повышения скорости работы приложения;
- защиты приложения от модификации со стороны пользователя (обычно ядро не содержит средств разработки приложений).

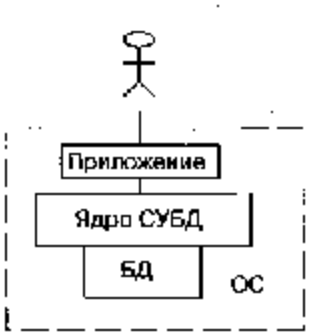


Рис. 1.4. Использование приложения и ядра СУБД

Примером такого подхода является использование модуля FoxRun системы FoxBase+. Из современных СУБД отметим Microsoft Access, включающую дополнительный пакет Microsoft Access Developer's Toolkit. С его помощью можно создавать переносимую на дискетах «укороченную» (run-time) версию Microsoft Access, не содержащую инструментов разработки.

Достоинствами использования ядра СУБД по сравнению с использованием полной версии СУБД являются: меньшее потребление ресурсов памяти компьютера, ускорение работы приложения и возможность защиты приложения от модификации. К основным *недостаткам* можно отнести все еще значительный объем дисковой памяти, необходимой для хранения ядра СУБД, и недостаточно высокое быстродействие работы приложений (выполнение приложения по-прежнему происходит путем *интерпретации).*

При третьем способе организации ИС исходная программа предварительно *компилируется* — преобразуется в последовательность исполняемых машинных команд. В результате получается готовая к выполнению независимая программа, не требующая для своей работы ни всей СУБД, ни ее ядра (рис. 1.5). Заметим, что с точки зрения выполнения основных функций хранения и обработки данных такая программа мало отличается от приложения, работающего под управлением СУБД или ее ядра.

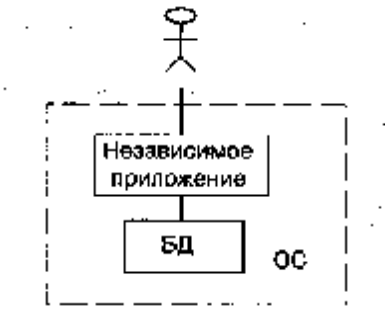


Рис. 1.5. Использование независимого приложения

Основными *достоинствами* этого варианта по сравнению с двумя предыдущими являются: экономия внешней и оперативной памяти компьютера, ускорение выполнения приложения и полная защита приложения от модификации (случай дизассемблирования и вставки своего кода и ему подобные в расчет не берутся). К *недостаткам* можно отнести трудоем-

кость доработки приложений и отсутствие возможности использовать стандартные средства СУБД по обслуживанию БД.

1.5. Способы разработки и выполнения приложений

Современные СУБД позволяют решать широкий круг задач по работе с базами данных без разработки приложения. Тем не менее есть случаи, когда целесообразно разработать приложение. Например, если требуется автоматизация манипуляций с данными, терминальный интерфейс СУБД недостаточно развит, либо имеющиеся в СУБД стандартные функции по обработке информации не устраивают пользователя. Для разработки приложений СУБД должна иметь программный интерфейс, основу которого составляют функции и/или процедуры соответствующего языка программирования.

Существующие СУБД поддерживают следующие технологии (и их комбинации) разработки приложений:

- ручное кодирование программ (Clipper, FoxPro, Paradox);
- создание текстов приложений с помощью генераторов (FoxApp в FoxPro, Personal Programmer в Paradox);
- автоматическая генерация готового приложения методами визуального программирования (Delphi, Access, Paradox for Windows).

При ручном кодировании программисты вручную набирают текст программ приложений, после чего выполняют их отладку.

Использование *генераторов* упрощает разработку приложений, поскольку при этом можно получать программный код без ручного набора. Генераторы приложений облегчают разработку основных элементов приложений (меню, экранных форм, запросов и т. д.), но зачастую не могут полностью исключить ручное кодирование.

Средства *визуального программирования* приложений являются дальнейшим развитием идеи использования генераторов приложений. Приложение при этом строится из готовых «строительных блоков» с помощью удобной интегрированной среды. При необходимости разработчик легко может вставить в приложение свой код. Интегрированная среда, как правило, предоставляет мощные средства создания, отладки и модификации приложений. Использование средств визуального программирования позволяет в кратчайшие сроки создавать более надежные, привлекательные и эффективные приложения по сравнению с приложениями, полученными первыми двумя способами.

Разработанное приложение обычно состоит из одного или нескольких файлов операционной системы.

Если основным файлом приложения является исполняемый файл (например, exe-файл), то это приложение, скорее всего, является *независимым приложением*, которое выполняется автономно от среды СУБД. Получение независимого приложения на практике осуществляется путем *компиляции* исходных текстов программ, полученных различными способами: путем набора текста вручную, а также полученных с помощью генератора приложения или среды визуального программирования.

Независимые приложения позволяют получать, например, СУБД FoxPro и система визуального программирования Delphi. Отметим, что с помощью средств Delphi обычно независимые приложения не разрабатывают, так как это достаточно трудоемкий процесс, а привлекают процессор баз данных BDE (Borland DataBase Engine), играющий роль ядра СУБД. Одним из первых средств разработки приложений для персональных ЭВМ является система Clipper, представляющая собой «чистый компилятор».

Во многих случаях приложение не может исполняться без среды СУБД. Выполнение приложения состоит в том, что СУБД анализирует содержимое файлов приложения (в частном случае — это текст исходной программы) и автоматически строит необходимые исполняемые машинные команды. Другими словами, приложение выполняется методом *интерпретации*.

Режим интерпретации реализован во многих современных СУБД, например Access, Visual FoxPro и Paradox, а также в СУБД недавнего прошлого, к примеру FoxBase и FoxPro.

Кроме этого, существуют системы, использующие промежуточный вариант между компиляцией и интерпретацией — так называемую *псевдокомпиляцию*. В таких системах исходная программа путем компиляции преобразуется в промежуточный код (псевдокод) и записывается на диск. В этом виде ее в некоторых системах разрешается даже редактировать, но главная цель псевдокомпиляции — преобразовать программу к виду, ускоряющему процесс ее интерпретации. Такой прием широко применялся в СУБД, работающих под управлением DOS, например Foxbase+ и Paradox 4.0/4.5 for DOS.

В СУБД, работающих под управлением Windows, псевдокод чаще используют для того, чтобы запретить модифицировать приложение. Это полезно для защиты от случайной или преднамеренной порчи работающей программы. Например, такой прием применен в СУБД Paradox for Windows, где допускается разработанные экранные формы и отчеты преобразовывать в соответствующие объекты, не поддающиеся редактированию.

Некоторые СУБД предоставляют пользователю возможность выбора варианта разработки приложения: как интерпретируемого СУБД программного кода или как независимой программы.

«*Достоинством* применения *независимых приложений* является то, что время выполнения машинной программы обычно меньше, чем при интерпретации. Такие приложения целесообразно использовать на слабых машинах и в случае установки систем «под ключ», когда необходимо закрыть приложение от доработок со стороны пользователей.

Важным *достоинством* применения *интерпретируемых* приложений является легкость их модификации. Если готовая программа подвергается частым изменениям, то для их внесения нужна инструментальная система, то есть СУБД или аналогичная среда. Для интерпретируемых приложений такой инструмент всегда под рукой, что очень удобно.

Другим серьезным *достоинством систем* с интерпретацией является

то, что хорошие СУБД обычно имеют мощные сервисные средства (контроль целостности данных, защита от несанкционированного доступа, динамическая оптимизация выполнения запросов, архивация данных и прочее). В последних упомянутые функции приходится программировать вручную либо оставлять на совести администраторов.

При *выборе средств для разработки приложения* следует учитывать три основных фактора: ресурсы компьютера, особенности приложения (потребность в модификации функций программы, время на разработку, необходимость использования сервисных функций) и цель разработки (отчуждаемый программный продукт или система автоматизации своей повседневной деятельности).

Для пользователя, имеющего современный компьютер и планирующего создать несложное приложение, по всей видимости, больше подойдет СУБД интерпретирующего типа. Напомним, что такие системы достаточно мощны, имеют высокоуровневые средства, удобны для разработки и отладки, позволяют быстро выполнить разработку и обеспечивают удобное сопровождение и модификацию приложения.

При использовании компьютера со слабыми характеристиками лучше остановить свой выбор на системе со средствами разработки независимых приложений. При этом следует иметь в виду, что малейшее изменение в приложении влечет за собой циклическое повторение этапов программирования, компиляции и отладки программы. Разница в выполнении независимого приложения и выполнения приложения в режиме интерпретации колеблется в пределах миллисекунд в пользу независимого приложения. В то же время разница во времени подготовки приложения к его использованию обычно составляет величины порядка минуты—часы в пользу систем с интерпретацией.

1.6. Схема обмена данными при работе с БД

Пользователю любой категории (администратору БД, разработчику приложения, обычному пользователю) для грамотного решения задач полезно представлять вычислительный процесс, происходящий в ОС при работе с БД.

При работе пользователя с базой данных над ее содержимым выполняются следующие основные операции: выбор, добавление, модификация (замена) и удаление данных. Рассмотрим, как происходит обмен данными между отдельным пользователем и персональной СУБД при выполнении наиболее часто используемой операции *выбора данных*. Обмен данными между пользователем и БД для других операций отличается несущественно.

Схематично обмен данными при работе пользователя с БД можно представить так, как показано на рис. 1.6, где обычными стрелками обозначены связи по управлению, утолщенными — связи по информации.

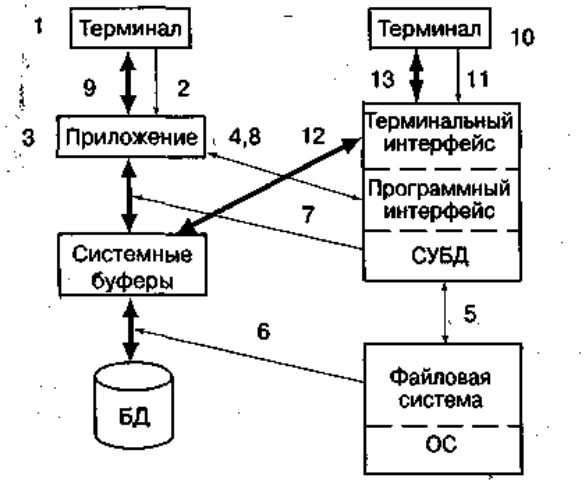


Рис. 1.6. Схема обмена данными при работе с БД

Цикл взаимодействия пользователя с БД с помощью приложения можно разделить на следующие основные этапы:

1. Пользователь терминала (1) в процессе диалога с приложением формулирует запрос (2) на некоторые данные из БД.

2. Приложение (3) на программном уровне средствами языка манипулирования данными формулирует запрос (4), с которым обращается к СУБД.

3. Используя свои системные управляющие блоки и таблицы, СУБД с помощью *словаря данных* определяет местоположение требуемых данных и обращается (5) за ними к ОС. ,

4. Программы методов доступа файловой системы ОС считывают (6) из внешней памяти искомые данные и помещают их в системные буферы СУБД.

5. Преобразуя полученные данные к требуемому формату, СУБД пересылает их (7) в соответствующую область программы и сигнализирует (8) о завершении операции каким-либо образом (например, кодом возврата).

6. Результаты выбора данных из базы приложение (3) отображает (9) на терминале пользователя (1).

В случае работы пользователя в диалоговом режиме с СУБД (без приложения) цикл взаимодействия пользователя с БД упрощается. Его можно представить следующими этапами:

1. Пользователь терминала (10) формулирует на языке запросов СУБД, например QBE, по связи (11) требование на выборку некоторых данных из базы.

2. СУБД определяет местоположение требуемых данных и обращается (5) за ними к ОС, которая считывает (6) из внешней памяти искомые

данные и помещает их в системные буферы СУБД.

3.Информация из системных буферов преобразуется (12) к требуемому формату, после чего отображается (13) на терминале пользователя (10).

Напомним, что описанная схема поясняет, как функционирует СУБД с одним пользователем на отдельной ПЭВМ.

Если компьютер и ОС поддерживают многопользовательский режим работы, то в такой вычислительной системе может функционировать *многопользовательская СУБД*. Последняя, в общем случае, позволяет одновременно обслуживать нескольких пользователей, работающих непосредственно с СУБД или с приложениями (каждое из которых может поддерживать работу с одним или несколькими пользователями).

Иногда к вычислительной системе подключается так называемый «удаленный пользователь», находящийся на некотором удалении от ЭВМ и соединенный с ней при помощи какой-либо передающей среды (интерфейс ЭВМ, телефонный канал связи, радиоканал, оптико-волоконная линия и т. д.). Чаще всего такой пользователь программным способом эмулируется под обычного локального пользователя. СУБД, как правило, этой подмены «не замечает» и работает по обслуживанию запросов обычным образом.

В многопользовательских СУБД при выполнении различных операций параллельно проистекают процессы, подобные описанным выше и показанным на рис. 1.6.

При обслуживании нескольких параллельных источников запросов (от пользователей и приложений) СУБД так планирует использование своих ресурсов и ресурсов ЭВМ, чтобы обеспечить независимое или почти независимое выполнение операций, порождаемых запросами.

Многопользовательские СУБД часто применяются на больших и средних ЭВМ, где основным режимом использования ресурсов является коллективный доступ.

На персональных ЭВМ пользователь обычно работает один, но с различными программами, в том числе и одновременно (точнее, попеременно). Иногда такими программами оказываются СУБД: различные программы или разные копии одной и той же СУБД. Последняя ситуация возникает, например, при работе с различными базами данных с помощью СУБД Access.

Технология одновременной работы пользователя с несколькими программами неплохо реализована в Windows. Здесь каждая выполняемая программа имеет свое окно взаимодействия с пользователем и имеются удобные средства переключения между программами. При работе в Windows СУБД избавлена от необходимости поддержания нескольких сеансов работы с пользователями.

ТЕМА 2. МОДЕЛИ И ТИПЫ ДАННЫХ

Хранимые в базе данные имеют определенную логическую структуру — иными словами, описываются некоторой ***моделью представления данных*** (моделью данных), поддерживаемой СУБД. К числу классических относятся следующие модели данных:

- иерархическая,
- сетевая,
- реляционная.

Кроме того, в последние годы появились и стали более активно внедряться на практике следующие модели данных:

- постреляционная,
- многомерная,
- объектно-ориентированная.

2.1. Иерархическая модель

В иерархической модели связи между данными можно описать с помощью упорядоченного графа (или дерева). Упрощенно представление связей между данными в иерархической модели показано на рис. 2.1.

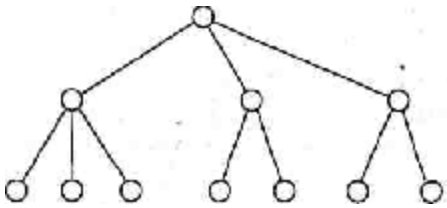


Рис. 2.1. Представление связей в иерархической модели

Для описания структуры (схемы) иерархической БД на некотором языке программирования используется тип данных «дерево». Тип «дерево» схож с типами данных «структура» языков программирования ПЛ/1 и С и «запись» языка Паскаль. В них допускается вложенность типов, каждый из которых находится на некотором уровне. Тип «дерево» является составным. Он включает в себя подтипы («поддеревья»), каждый из которых, в свою очередь, является типом «дерево». Каждый из типов «дерево» состоит из одного «корневого» типа и упорядоченного набора (возможно, пустого) подчиненных типов. Каждый из элементарных типов, включенных в тип «дерево», является простым или составным типом «запись». Простая «запись» состоит из одного типа, например числового, составная «запись» объединяет некоторую совокупность типов, например, целое, строку символов и указатель (ссылку). Пример типа «дерево» как совокупности типов показан на рис. 2.2.

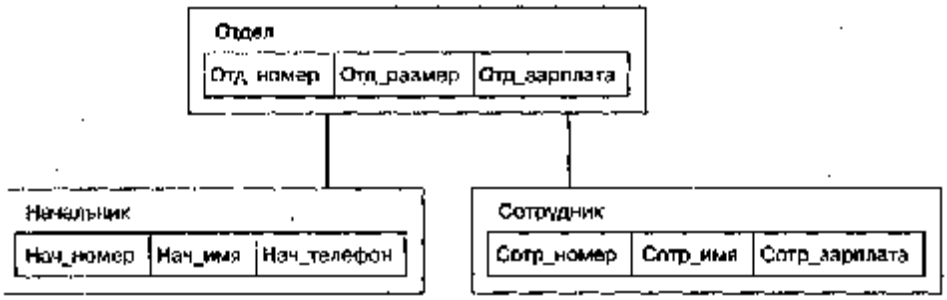


Рис. 2.2. Пример типа «дерево»

Корневым называется тип, который имеет подчиненные типы и сам не является подтипом. *Подчиненный* тип (подтип) является *потомком* по отношено к типу, который выступает для него в роли предка (родителя). Потомки одного и того же типа являются *близнецами* по отношению друг к другу.

В целом тип «дерево» представляет собой иерархически организованный набор типов «запись».

Иерархическая БД представляет собой упорядоченную совокупность экземпляров данных типа «дерево» (деревьев), содержащих экземпляры типа «запись» (записи). Часто отношения родства между типами переносят а отношения между самими записями. Поля записей хранят собственно числовые или символьные значения, составляющие основное содержание БД. Обход всех элементов иерархической БД обычно производится сверху вниз слева направо.

В иерархических СУБД может использоваться терминология, отличающаяся от приведенной.

Данные в базе с приведенной схемой (рис. 2.2) могут выглядеть, например, как показано на рис. 2.3.

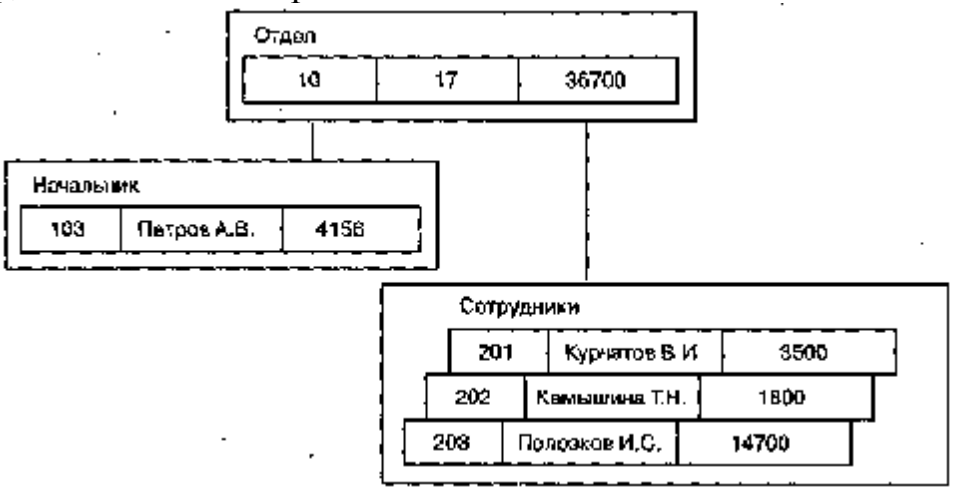


Рис. 2.3. Данные в иерархической базе

Для организации *физического* размещения иерархических данных в памяти ЭВМ могут использоваться следующие группы методов:

• представление линейным списком с последовательным распределением памяти (адресная арифметика, левосписковые структуры);

• представление связными линейными списками (методы, использующие указатели и справочники).

К основным операциям манипулирования иерархически организованными данными относятся следующие:

• поиск указанного экземпляра БД (например, дерева со значением 10 в поле Отд_номер);

• переход от одного дерева к другому;

• переход от одной записи к другой внутри дерева (например, к следующей записи типа Сотрудники);

• вставка новой записи в указанную позицию;

• удаление текущей записи и т. д.

В соответствии с определением типа «дерево», можно заключить, что между предками и потомками автоматически поддерживается контроль целостности связей. Основное правило контроля целостности формулируется слет дующим образом: потомок не может существовать без родителя, а у некоторых родителей может не быть потомков. Механизмы поддержания целостности связей между записями различных деревьев отсутствуют.

К *достоинствам* иерархической модели данных относятся эффективное использование памяти ЭВМ и неплохие показатели времени выполнения основных операций над данными. Иерархическая модель данных удобна для работы с иерархически упорядоченной информацией.

Недостатком иерархической модели является ее громоздкость для обработки информации с достаточно сложными логическими связями, а также сложность понимания для обычного пользователя.

На иерархической модели данных основано сравнительно ограниченное количество СУБД, в числе которых можно назвать зарубежные системы IMS, PC/Focus, Team-Up и Data Edge, а также отечественные системы Ока, ИНЭС и МИРИС.

2.2. Сетевая модель

Сетевая модель данных позволяет отображать разнообразные взаимосвязи элементов данных в виде произвольного графа, обобщая тем самым иерархическую модель данных (рис. 2.4). Наиболее полно концепция сетевых БД впервые была изложена в Предложениях группы КОДАСИЛ (KODASYL).

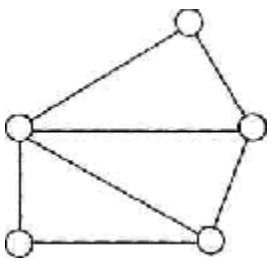


Рис. 2.4. Представление связей в сетевой модели

Для описания схемы сетевой БД используется две группы типов: «запись» и «связь». Тип «связь» определяется для двух типов «запись»: предка и потомка. Переменные типа «связь» являются экземплярами свя-

зей.

Сетевая БД состоит из набора записей и набора соответствующих связей. На формирование связи особых ограничений не накладывается. Если в иерархических структурах запись-потомок могла, иметь только одну запись-предка, то в сетевой модели данных запись-потомок может иметь произвольное число записей-предков (сводных родителей). Пример схемы простейшей сетевой БД показан на рис. 2.5. Типы связей здесь обозначены надписями на соединяющих типы записей линиях.

Рис. 2.5. Пример схемы сетевой БД

В различных СУБД сетевого типа для обозначения одинаковых по сути понятий зачастую используются различные термины. Например,- такие как элементы и агрегаты данных, записи, наборы, области и т. д.

Физическое размещение данных в базах сетевого типа может быть организовано практически теми же методами, что и в иерархических базах данных.

К числу важнейших операций манипулирования данными баз сетевого типа можно отнести следующие:

- поиск записи в БД;
- переход от предка к первому потомку;
- переход от потомка к предку;
- создание новой записи;
- удаление текущей записи;
- обновление текущей записи;
- включение записи в связь;
- исключение записи из связи;
- изменение связей и т. д.

Достоинством сетевой модели данных является возможность эффективной реализации по показателям затрат памяти и оперативности. В сравнении с иерархической моделью сетевая модель предоставляет большие возможности в смысле допустимости образования произвольных связей.

Недостатком сетевой модели данных является высокая сложность и жесткость схемы БД, построенной на ее основе, а также сложность для понимания и выполнения обработки информации в БД обычным пользователем. Кроме того, в сетевой модели данных ослаблен контроль целостности связей вследствие допустимости установления произвольных связей между записями.

Системы на основе сетевой модели не получили широкого распространения на практике. Наиболее известными сетевыми СУБД являются следующие: IDMS, db_VistaIII, СЕТЬ, СЕТОР и КОМПАС.

2.3. Реляционная модель

Реляционная модель данных предложена сотрудником фирмы IBM Эдгаром Коддом и основывается на понятии отношение (relation).

Отношение представляет собой множество элементов, называемых кортежами. Подробно теоретическая основа реляционной модели данных рассматривается в следующем разделе. Наглядной формой представления отношения является привычная для человеческого восприятия двумерная таблица.

Таблица имеет строки (записи) и столбцы (колонки) Каждая строка таблицы имеет одинаковую структуру и состоит полей. Строкам таблицы соответствуют кортежи, а столбцам — атрибуты отношения.

С помощью одной таблицы удобно описывать простейший вид связей между данными, а именно деление одного объекта (явления, сущности, системы и проч.), информация о котором хранится в таблице, на множество подобъектов, каждому из которых соответствует строка или запись таблицы. При этом каждый из подобъектов имеет одинаковую структуру или свойства, описываемые соответствующими значениями полей записей. Например, таблица может содержать сведения о группе обучаемых, о каждом из которых известны следующие характеристики: фамилия, имя и отчество, пол, возраст и образование. Поскольку в рамках одной таблицы не удается описать более сложные логические структуры данных из предметной области, применяют *связывание* таблиц.

Физическое размещение данных в реляционных базах на внешних носителях легко осуществляется с помощью обычных файлов.

Достоинство реляционной модели данных заключается в простоте, понятности и удобстве физической реализации на ЭВМ. Именно простота и понятность для пользователя явились основной причиной их широкого использования. Проблемы же эффективности обработки данных этого типа оказались технически вполне разрешимыми.

Основными *недостатками* реляционной модели являются следующие: отсутствие стандартных средств идентификации отдельных записей и сложность описания иерархических и сетевых связей.

Примерами зарубежных реляционных СУБД для ПЭВМ являются следующие: dBaselll Plus и dBase IY (фирма Ashton-Tate), DB2 (IBM), R:BASE (Microrim), FoxPro ранних версий и FoxBase (Fox Software), Paradox и dBASE for Windows (Borland), FoxPro более поздних версий, Visual FoxPro и Access (Microsoft), Clarion (Clarion Software), Ingres (ASK Computer Systems) и Oracle (Oracle).

К отечественным СУБД реляционного типа относятся системы: ПАЛЬМА (ИК АН УССР), а также система НуТech (МИФИ).

Заметим, что последние версии реляционных СУБД имеют некоторые свойства объектно-ориентированных систем. Такие СУБД часто назы-

вают объектно-реляционными. Примером такой системы можно считать продукты Oracle 8.x. Системы предыдущих версий вплоть до Oracle 7.x считаются «чисто» реляционными.

2.4. Постреляционная модель

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

Постреляционная модель данных представляет собой расширенную реляционную модель, снимающую ограничение неделимости данных, хранящихся в записях таблиц. Постреляционная модель данных допускает многозначные поля — поля, значения которых состоят из подзначений. Набор значений многозначных полей считается самостоятельной таблицей, встроенной в основную таблицу.

На рис. 2.6 на примере информации о накладных и товарах для сравнения приведено представление одних и тех же данных с помощью реляционной (я) и постреляционной (б) моделей. Таблица INVOICES (накладные) содержит данные о номерах накладных (INVNO) и номерах покупателей (CUSTNO). И таблице INVOICE.ITEMS (накладные-товары) содержатся данные о каждой из накладных: номер накладной (INVNO), название товара (GOODS) и количество товара (QTY). Таблица INVOICES связана с таблицей INVOICE.ITEMS по полю INVNO.

Как видно из рисунка, по сравнению с реляционной моделью в постреляционной модели данные хранятся более эффективно, а при обработке не требуется выполнять операцию соединения данных из двух таблиц.

Помимо обеспечения вложенности полей постреляционная модель поддерживает ассоциированные многозначные поля (множественные группы). Совокупность ассоциированных полей называется ***ассоциацией.*** При этом в строке первое значение одного столбца ассоциации соответствует первым значениям всех других столбцов ассоциации. Аналогичным образом связаны все вторые значения столбцов и т. д.

На длину полей и количество полей в записях таблицы не накладывается требование постоянства. Это означает, что структура данных и таблиц имеет большую гибкость.

Поскольку постреляционная модель допускает хранение в таблицах ненормализованных данных, возникает проблема обеспечения целостности и непротиворечивости данных. Эта проблема решается включением в СУБД механизмов, подобных хранимым процедурам в клиент-серверных системах.

Для описания функций контроля значений в полях имеется возможность создавать процедуры (коды конверсии и коды корреляции), автоматически вызываемые до или после обращения к данным. Коды корреляции выполняются сразу после чтения данных, перед их обработкой. Коды конверсии, наоборот, выполняются после обработки данных.

а) INVOICES

INVNO	CUSTNO
0373	8723
8374	8232
7364	8723

INVOICE.ITEMS

INVNO	GOODS	QTY
0373	Сыр	3
0373	Рыба	2
8374	Лимонад	1
8374	Сок	6
8374	Печенье	2
7364	Йогурт	1

б)

INVOICES

INVNO	CUSTNO	GOODS	QTY
0373	8723	Сыр	3
		Рыба	2
8374	8232	Лимонад	1
		Сок	6
		Печенье	2
7364	8723	Йогурт	**1**

Рис. 2.6. Структуры данных реляционной и постреляционной моделей

Достоинством постреляционной модели является возможность представления совокупности связанных реляционных таблиц одной постреляционной таблицей. Это обеспечивает высокую наглядность представления информации и повышение эффективности ее обработки.

Недостатком постреляционной модели является сложность решения проблемы обеспечения целостности и непротиворечивости хранимых данных.

Рассмотренная нами постреляционная модель данных поддерживается СУБД uniVers. К числу других СУБД, основанных на постреляционной модели данных, относятся также системы Bubba и Dasdb.

2.5. Многомерная модель

Многомерный подход к представлению данных в базе появился практически одновременно с реляционным, но реально работающих многомерных СУБД (МСУБД) до настоящего времени было очень мало. С середины 90-х годов интерес к ним стал приобретать массовый характер.

Толчком послужила в 1993 году программная статья одного из основоположников реляционного подхода Э. Кодда. В ней сформулированы 12

основных требований к системам класса OLAP (OnLine Analytical Processing — оперативная аналитическая обработка), важнейшие из которых связаны с возможностями концептуального представления и обработки многомерных данных. Многомерные системы позволяют оперативно обрабатывать информацию для проведения анализа и принятия решения.

В развитии концепций ИС можно выделить следующие два направления:

- системы оперативной (транзакционной) обработки;
- системы аналитической обработки (системы поддержки принятия решений).

Реляционные СУБД предназначались для информационных систем *оперативной* обработки информации и в этой области были весьма эффективны. В системах аналитической обработки они показали себя несколько неповоротливыми и недостаточно гибкими. Более эффективными здесь оказываются многомерные СУБД (МСУБД).

Многомерные СУБД являются узкоспециализированными ('У1>Д. предназначенными для интерактивной аналитической обработки информации. Раскроем основные понятия, используемые в этих СУБД: агрегируемость, историчность и прогнозируемость данных.

Агрегируемость данных означает рассмотрение информации на различных уровнях ее обобщения. В информационных системах степень детальности представления информации для пользователя зависит от его уровня: аналитик, пользователь-оператор, управляющий, руководитель.

Историчность данных предполагает обеспечение высокого уровня статичности (неизменности) собственно данных и их взаимосвязей, а также обязательность привязки данных ко времени.

Статичность данных позволяет использовать при их обработке специализированные методы загрузки, хранения, индексации и выборки.

Временная привязка данных необходима для частого выполнения запросов, имеющих значения времени и даты в составе выборки. Необходимость упорядочения данных по времени в процессе обработки и представления данных пользователю накладывает требования на механизмы хранения и доступа к информации. Так, для уменьшения времени обработки запросов желательно, чтобы данные всегда были отсортированы в том порядке, в котором они наиболее часто запрашиваются.

Прогнозируемость данных подразумевает задание функций прогнозирования и применение их к различным временным интервалам.

Многомерность модели данных означает не многомерность визуализации цифровых данных, а многомерное логическое представление структуры информации при описании и в операциях манипулирования данными.

По сравнению с реляционной моделью многомерная организация данных обладает более высокой *наглядностью* и *информативностью*. Для иллюстрации на рис. 2.8 приведены реляционное (а) и многомерное *{б)* представления одних и тех же данных об объемах продаж автомобилей.

Модель	Месяц	Объем
«Жигули»	июнь	12
«Жигули»	июль	24
«Жигули»	август	5
«Москвич»	июнь	2
«Москвич»	июль	18
«Волга»	июль	19

б)

Модель	Июнь	Июль	Август
«Жигули»	12	24	5
«Москвич»	2	18	No
«Волга»	No	19	No

Рис. 2.8. Реляционное и многомерное представление данных

Если речь идет о многомерной модели с мерностью больше двух, то не обязательно визуально информация представляется в виде многомерных объектов (трех-, четырех- и более мерных гиперкубов). Пользователю и в этих случаях более удобно иметь дело с двухмерными таблицами или графиками. Данные при этом представляют собой «вырезки» (точнее, «срезы») из многомерного хранилища данных, выполненные с разной степенью детализации.

Рассмотрим основные понятия многомерных моделей данных, к числу которых относятся измерение и ячейка.

Измерение (Dimension) — это множество однотипных данных, образующих одну из граней гиперкуба. Примерами наиболее часто используемых временных измерений являются Дни, Месяцы, Кварталы и Годы. В качестве географических измерений широко употребляются Города, Районы, Регионы и Страны. В многомерной модели данных измерения играют роль индексов, служащих для идентификации конкретных значений в ячейках гиперкуба.

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

В примере на рис. 2.8, б каждое значение ячейки Объем продаж однозначно определяется комбинацией временного измерения (Месяц продаж) и модели автомобиля. На практике зачастую требуется большее количество измерений. Пример трехмерной модели данных приведен на рис. 2.9.

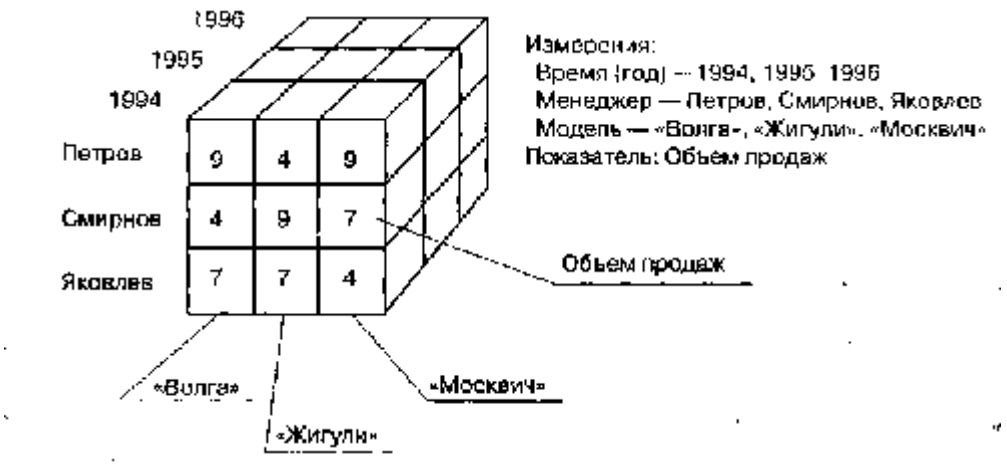


Рис. 2.9. Пример трехмерной модели

В существующих МСУБД используются два основных варианта (схемы) организации данных: гиперкубическая и поликубическая.

В *поликубической* схеме предполагается, что в БД может быть определено несколько гиперкубов с различной размерностью и с различными измерениями в качестве граней. Примером системы, поддерживающей поликубический вариант БД, является сервер Oracle Express Server.

В случае *гиперкубической* схемы предполагается, что все показатели определяются одним и тем же набором измерений. Это означает, что при наличии нескольких гиперкубов БД все они имеют одинаковую размерность и совпадающие измерения. Очевидно, в некоторых случаях информация в БД может быть избыточной (если требовать обязательное заполнение ячеек).

В случае многомерной модели данных применяется ряд специальных операций, к которым относятся: формирование «среза», «вращение», агрегация и детализация.

«Срез» (Slice) представляет собой подмножество гиперкуба, полученное в результате фиксации одного или нескольких измерений. Формирование «срезов» выполняется для ограничения используемых пользователем значений, так как все значения гиперкуба практически никогда одновременно не используются. Например, если ограничить значения измерения Модель автомобиля в гиперкубе (рис. 2.9) маркой «Жигули», то получится двухмерная таблица продаж этой марки автомобиля различными менеджерами по годам.

Операция «вращение» (Rotate) применяется при двухмерном представлении данных. Суть ее заключается в изменении порядка измерений при визуальном представлении данных. Так, «вращение» двухмерной таблицы, показанной на рис. 2.8 б, приведет к изменению ее вида таким образом, что по оси X будет марка автомобиля, а по оси Y — время.

Операцию «вращение» можно обобщить и на многомерный случай, если под ней понимать процедуру изменения порядка следования измерений.

В простейшем случае, например, это может быть взаимная пере-

становка двух произвольных измерений.

Операции «агрегация» (Drill Up) и «детализация» (Drill Down) означают соответственно переход к более общему и к более детальному представлению информации пользователю из гиперкуба.

Для иллюстрации смысла операции «агрегация» предположим, что у нас имеется гиперкуб, в котором помимо измерений гиперкуба, приведенного на рис. 2.9, имеются еще измерения: Подразделение, Регион, Фирма, Страна. Заметим, что в этом случае в гиперкубе существует иерархия (снизу вверх) отношений между измерениями: Менеджер, Подразделение, Регион, Фирма, Страна.

Пусть в описанном гиперкубе определено, насколько успешно в 1995 году менеджер Петров продавал автомобили «Жигули» и «Волга». Тогда, поднимаясь на уровень выше по иерархии, с помощью операции «агрегация» можно выяснить, как выглядит соотношение продаж этих же моделей на уровне подразделения, где работает Петров.

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

Недостатком многомерной модели данных является ее громоздкость для простейших задач обычной оперативной обработки информации.

Примерами систем, поддерживающих многомерные модели данных, являются Essbase (Arbor Software), Media Multi-matrix (Speedware), Oracle I xpress Server (Oracle) и Cache (InterSystems). Некоторые программные про-i у кты, например Media/MR (Speedware), позволяют одновременно работать с мно-i ((.мерными и с реляционными БД. В СУБД Cache, в которой внутренней моделью iai шых является многомерная модель, реализованы три способа доступа к данным: прямой (на уровне узлов многомерных массивов), объектный и реляционный.

2.6. Объектно-ориентированная модель

В объектно-ориентированной модели при представлении данных имеется возможность идентифицировать отдельные записи базы. Между записями базы данных и функциями их обработки устанавливаются взаимосвязи с помощью механизмов, подобных соответствующим средствам в объектно-ориентированных языках программирования.

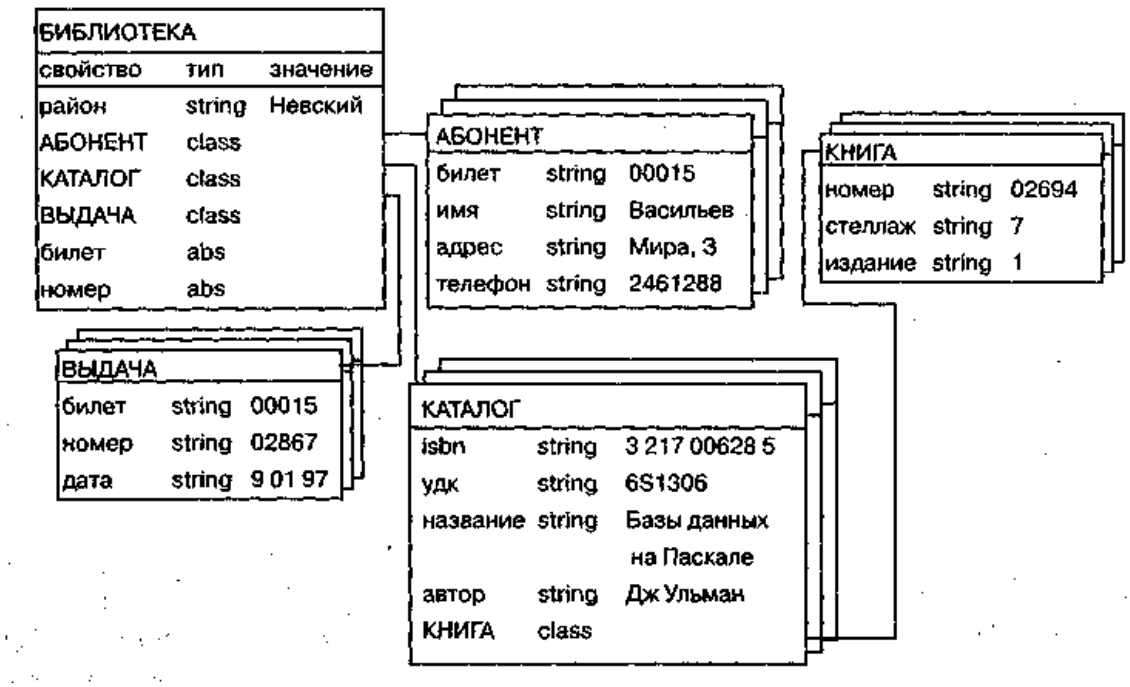
Стандартизованная объектно-ориентированной модель описана в рекомендациях стандарта ODMG-93 (Object Database Management Group — группа управления объектно-ориентированными базами данных). Реализовать в полном объеме рекомендации QDMG-93 пока не удастся. Для иллюстрации ключевых идеи рассмотрим несколько упрощенную модель объектно-ориентированной БД.

Структура объектно-ориентированной БД графически представим в виде дерева, узлами которого являются объекты. Свойства объектов опи-

сываются некоторым стандартным типом (например, строковым — string) или типом, конструируемым пользователем (определяется как class).

Значением свойства типа string является строка символов. Значение свойства типа class есть объект, являющийся экземпляром соответствующего класса. Каждый объект-экземпляр класса считается потомком объекта, в котором он определен как свойство. Объект-экземпляр класса принадлежит своему классу и имеет одного родителя. Родовые отношения в БД образуют связную иерархию объектов.

Пример логической структуры объектно-ориентированной БД библиотечного дела приведен на рис. 2.10.



Здесь объект типа БИБЛИОТЕКА является родительским для объектов-экземпляров классов АБОНЕНТ, КАТАЛОГИ ВЫДАЧА. Различные объекты типа КНИГА могут иметь одного или разных родителей. Объекты типа КНИГА, имеющие одного и того же родителя, должны различаться по крайней мере инвентарным номером (уникален для каждого экземпляра книги), но имеют одинаковые значения свойств *isbn*, *удк*, *название* и *автор*.

Логическая структура объектно-ориентированной БД внешне похожа на структуру иерархической БД. Основное отличие между ними состоит в методах манипулирования данными.

Для выполнения действий над данными и рассматриваемой модели БД применяются логические операции, усиленные объектно-ориентированными механизмами инкапсуляции, наследования и полиморфизма. Ограниченно могут примениться операции, подобные командам SQL (например, для создания БД).

Создание и модификация БД сопровождается автоматическим формированием и последующей корректировкой индексов (индексных таблиц), содержащих информацию для быстрого поиска данных.

Рассмотрим кратко понятия инкапсуляции, наследования и полиморфизма применительно к объектно-ориентированной модели БД.

Инкапсуляция ограничивает область видимости имени свойства пределами того объекта, в котором оно определено. Так, если в объект типа КАТАЛОГ добавить свойство, задающее телефон автора книги и имеющее название *телефон,* то мы получим одноименные свойства у объектов АБОНЕНТ и КАТАЛОГ. Смысл такого свойства будет определяться тем объектом, в который оно инкапсулировано.

Наследование, наоборот, распространяет область видимости свойства на всех потомков объекта. Так, всем объектам типа КНИГА, являющимся потомками объекта типа КАТАЛОГ, можно приписать свойства объекта-родителя: *isbn, удк, название* и *автор*. Если необходимо расширить действие механизма наследования на объекты, не являющиеся непосредственными родственниками (например, между двумя потомками одного родителя), то в их общем предке определяется абстрактное свойство типа abs. Так, определение абстрактных свойств *билет* и *номер* в объекте БИБЛИОТЕКА приводит к наследованию этих свойств всеми дочерними объектами АБОНЕНТ, КНИГА и ВЫДАЧА. Не случайно поэтому значения свойства *билет* классов АБОНЕНТ и ВЫДАЧА, показанных на рисунке, будут одинаковыми — 00015.

Полиморфизм в объектно-ориентированных языках программирования означает способность одного и того же программного кода работать с разнотипными данными. Другими словами, он означает допустимость в объектах разных типов иметь методы (процедуры или функции) с одинаковыми именами. Во время выполнения объектной программы одни и те же методы оперируют с разными объектами в зависимости от типа аргумента. Применительно к нашей объектно-ориентированной БД полиморфизм означает, что объекты класса КНИГА, имеющие разных родителей из класса КАТАЛОГ, могут иметь разный набор свойств. Следовательно, программы работы с объектами класса КНИГА могут содержать полиморфный код.

Поиск в объектно-ориентированной БД состоит в выяснении сходства между объектом, задаваемым пользователем, и объектами, хранящимися в БД. Определяемый пользователем объект, называемый объектом-целью (свойство объекта имеет тип *goal),* в общем случае может представлять собой подмножество всей хранимой в БД иерархии объектов. Объект-цель, а также результат выполнения запроса могут храниться в самой базе. 11ример запроса о номерах читательских билетов и именах абонентов, получавших в библиотеке хотя бы одну книгу, показан на рис. 2.11.

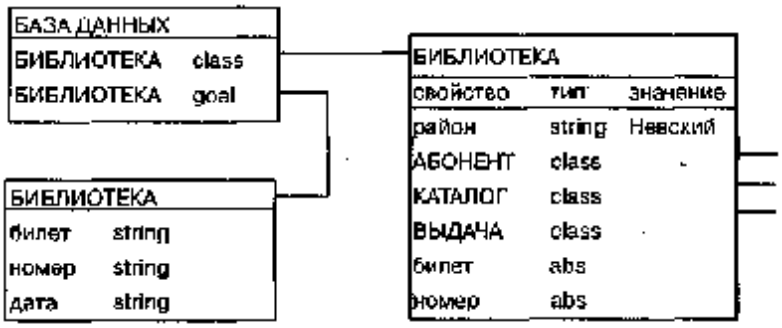


Рис. 2.11. Фрагмент БД с объектом-целью

Основным *достоинством* объектно-ориентированной модели данных в сравнении с реляционной является возможность отображения информации о сложных взаимосвязях объектов. Объектно-ориентированная модель данных позволяет идентифицировать отдельную запись базы данных и определять функции их обработки.

Недостатками объектно-ориентированной модели являются высокая понятийная сложность, неудобство обработки данных и низкая скорость выполнения запросов.

В 90-е годы существовали экспериментальные прототипы объектно-ориентированных систем управления базами данных. В настоящее время такие системы получили достаточно широкое распространение, в частности, к ним относятся следующие СУБД: G-Base (Grapael), GemStone (Servio-Logic совместно с OGI), Statice (Symbolics), ObjectStore (Object Design), Objectivity /DB (Objectivity), Versant (Versant Technologies), 02 (Ardent Software), ODB-Jupiter (научно-производственный центр «Интелтек Плюс»), а также Iris, Orion и Postgres.

2.7. Типы данных

Основные типы данных СУБД

Первоначально СУБД применялись преимущественно для решения финансово-экономических задач. При этом, независимо от модели представления, в базах данных использовались следующие основные типы данных:

- числовые. В качестве подтипов числовых данных часто используются целочисленные, денежные (финансовые) и обычные вещественные. Примеры значений данных: 0.43, 328, 2E+5;
- символьные (алфавитно-цифровые). Примеры значений данных: «пятница», «строка», «программист»;
- логические, принимающие значения «истина» (true) и «ложь» (false);
- даты, задаваемые с помощью специального типа «Дата» или как обычные символьные данные. Примеры значений данных: 1.12.97, 2/23/1999.

В разных СУБД эти типы могли несущественно отличаться друг от друга но названию, диапазону значений и виду представления. С расширением области применения персональных компьютеров стали появляться

специализированные системы обработки данных, например, геоинформационные, обработки видеоизображений и т. д. В ответ на это разработчики СУБД стали и водить в них поддержку новых типов данных. К числу сравнительно новых типов данных можно отнести следующие:

• временные и дата-временные, предназначенные для хранения информации о времени и/или дате. Примеры значений данных: 31.01.85 (дата), 9:10:03 (время), 6.03.1960 12:00 (дата и время);

• символьные переменной длины, предназначенные для хранения текстовой информации большой длины, например, документа;

• двоичные, предназначенные для хранения графических объектов, аудио-11 видеоинформации, пространственной, хронологической и другой специальной информации. Двоичные данные часто называют *мультимедиа-данными*. Например, в MS Access таким типом является тип данных «Поле объекта ()LE», который позволяет хранить в БД графические данные в формате BMP (I bitmap) и автоматически их отображать при работе с БД;

- гиперссылки (hyperlinks), предназначенные для хранения ссылок на различные ресурсы (узлы, файлы, документы и т. д.), находящиеся вне базы данных, например, в сети Internet, корпоративной сети intranet или на жестком диске компьютера. Примеры значений данных: http:\\www.chat.ru, ftp:\\chance4u.teens.com;
- данные в XML формате. ,

Технология OLE (Object Linking and Flmbedding) реализует такой механизм связывания и встраивания объектов, при котором для обработки объекта вызывается приложение, в котором этот объект создавался.

Мультимедиа -данные

Слово мультимедиа (multimedia) стало популярным в компьютерной обмети в 90-х годах. Точного перевода его с английского языка на русский не существует, сравнительно близки следующие варианты перевода: «многосредность» или «множество сред». Под средами здесь понимаются данные различной природы: звуковые, видео-, графические, текстовые, с различными эффектами отображения на экране (анимацией) и т. д.

В широком смысле термин мультимедиа означает совокупность технологий производства и применения различных аппаратных и программных средств для ПЭВМ, позволяющих поддерживать работу компьютера с перечисленными видами информации.

Мультимедиа-средства нужны, чтобы существенно оживить процедуру общения пользователя с компьютером. Использование технологии мультимедиа позволяет расширить область применения ПЭВМ и поднять на более высокий качественный уровень решение традиционных задач.

Говоря о мультимедиа-возможностях реляционных систем, как получивших наибольшее распространение, отметим, что основная их роль - хранить мультимедиа-данные. Порождать и видоизменять сами данные в настоящее время могут только мультимедиа-программы, каковыми СУБД не являются.

В реляционных системах основным местом хранения данных, в том числе и мультимедиа-данных, является таблица. Для обеспечения хранения мультимедиа-данных в структуре таблицы должны быть предусмотрены соответствующие поля. Кроме того, мультимедиа-данные могут храниться в экранных формах и отчетах.

Главное отличие названных способов состоит в том, что в первом случае мультимедиа-данные связываются с каждой записью базы, а во втором случае мультимедиа-данные включаются в экранную форму или отчет один раз. Так, при просмотре БД с помощью экранной формы связанные с записями мультимедиа-данные при изменении текущей записи изменяются, а мультимедиа-данные самой экранной формы - остаются без

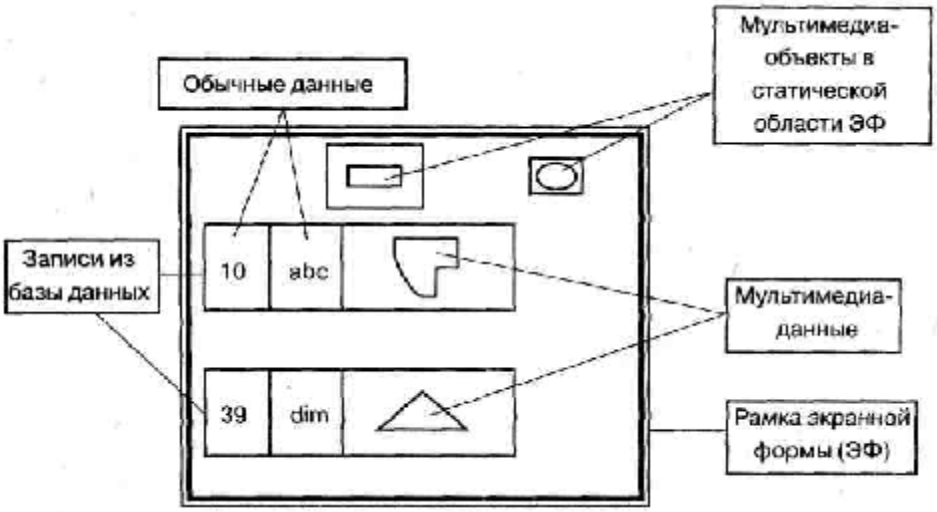


Рис. 2.12. Мультимедиа-данные в экранной форме

изменения. Мультимедиа-данные включают в экранные формы и отчеты для повышении их наглядности при отображении на экране (рис. 2.12).

В различных СУБД применяются разные механизмы поддержания мультимедиа- данных. Чаще всего для их размещения хранения используются поля хранения двоичных объектов или так называемые BLOB-поля (Binary Large Object - большие двоичные объекты). Поскольку мультимедиа-данные могут иметь различные виды (аудио-, видео-, графическая и т. д. информация), а в рамках каждого вида - разные форматы (например, для хранения графической информации используются файлы с расширениями: bmp, pcx, tif, gif и т. д.), удобным способом привязки их к средствам обработки оказывается и упоминавшийся ранее механизм OLE. В связи с этим одним из наиболее распространенных типов BLOB-полей являются OLE-поля.

К примеру, в MS Access поддерживается OLE-поле, в системе Paradox можно создавать также поля типа graphic и binary (в поле типа graphic информация только интерпретируется и отсутствует связь с приложением; информация из в поля типа binary не отображается на экране, так как СУБД ее не интерпретирует).

Поскольку для работы с мультимедиа-информацией в общем случае могут использоваться как специализированные средства ее обработки и

хранения, так и реляционные системы, то перед пользователем может встать вопрос: «А что же лучше?». Существенную роль в ответе на этот вопрос, по всей видимости, будет играть специфика решаемой задачи.

В качестве общей рекомендации при выборе того или иного программного продукта можно принять следующее. Если в прикладной задаче используется разного рода информация и доля обычной символьно-числовой информации велика, то лучше остановить свой выбор на реляционной системе (при наличии в ней достаточно развитых средств поддержки мультимедиа-данных). Причина такого решения очевидна - реляционные системы сейчас являются достаточно проработанными и эффективными в применении. В случае преобладания потребностей в обработке двоичной информации с использованием специальных методик, алгоритмов и интерфейсов, скорее всего следует остановить свой выбор на специализированных средствах работы с мультимедиа-данными.

ТЕМА 3. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

3.1. Определение реляционной модели

Реляционная модель данных (РМД) некоторой предметной области представляет собой набор отношений, изменяющихся во времени. При создании информационной системы совокупность отношений позволяет хранить данные об объектах предметной области и моделировать связи между ними. Элементы РМД и формы их представления приведены в табл. 3.1.

Таблица 3.1 Элементы реляционной модели

Элемент реляционной модели	Форма представления
Отношение	Таблица
Схема отношения	Строка заголовков столбцов таблицы (заголовок таблицы)
Кортеж	Строка таблицы
Сущность	Описание свойств объекта
Атрибут	Заголовок столбца таблицы
Домен	Множество допустимых значений атрибута
Значение атрибута	Значение поля в записи
Первичный ключ	Один или несколько атрибутов
Тип данных	Тип значений элементов таблицы

Отношение является важнейшим понятием и представляет собой

двумерную таблицу, содержащую некоторые данные.

Сущность есть объект любой природы, данные о котором хранятся в базе данных. Данные о сущности хранятся в отношении.

Атрибуты представляют собой свойства, характеризующие сущность. В структуре таблицы каждый атрибут именуется и ему соответствует заголовок некоторого столбца таблицы.

Математически отношение можно описать следующим образом. Пусть даны n множеств Dl, D2, D3,..., Dn, тогда отношение R есть множество упорядоченных ***кортежей*** **<dl, d2, d3 ,..., dn>,** где **dk Є Dk, dk — *атрибут*, a Dk — *домен*** отношения ***R.***

На рис. 3.1 приведен пример представления отношения СОТРУДНИК.

В общем случае порядок кортежей в отношении, как и в любом множестве, не определен. Однако в реляционных СУБД для удобства кортежи все же упорядочивают. Чаще всего для этого выбирают некоторый атрибут, по которому система автоматически сортирует кортежи по возрастанию или убыванию. Если пользователь не назначает атрибута упорядочения, система автоматически присваивает номер кортежам в порядке их ввода.

Рис. 3.1. Представление отношения СОТРУДНИК

Формально, если переставить атрибуты в отношении, то получается новое отношение. Однако в реляционных БД перестановка атрибутов не приводит к образованию нового отношения.

Домен представляет собой множество всех возможных значений определенного атрибута отношения. Отношение СОТРУДНИК включает 4 домена. *Домен 1* содержит фамилии всех сотрудников, *домен 2* — номера всех отделов фирмы, *домен 3* - названия всех должностей, *домен 4* — даты рождения всех сотрудников. Каждый домен образует значения одного типа данных, например, числовые или символьные.

Отношение СОТРУДНИК содержит 3 кортежа. Кортеж рассматриваемого отношения состоит из 4 элементов, каждый из которых выбирается из соответствующего домена. Каждому кортежу соответствует строка таблицы (рис. 3.1).

Схема отношения (заголовок отношения) представляет собой список имен атрибутов. Например, для приведенного примера схема отношения имеет вид СОТРУДНИК(ФИО, Отдел, Должность, Д_Рождения). Множество собственно кортежей отношения часто называют *содержимым (телом) отношения.*

Первичным ***ключом (ключом отношения, ключевым атрибутом)*** называется атрибут отношения, однозначно идентифицирующий каждый из его кортежей. Например, в отношении СОТРУДНИК(ФИО, Отдел, Должность, Д_Рождения) ключевым является атрибут «ФИО». *Ключ* может быть *составным (сложным),* то есть состоять из нескольких атрибутов.

Каждое отношение обязательно имеет комбинацию атрибутов, которая может служить ключом. Ее существование гарантируется тем, что отношение — это множество, которое не содержит одинаковых элементов — кортежей. То есть в отношении нет повторяющихся кортежей, а это значит, что по крайней мере вся совокупность атрибутов обладает свойством однозначной идентификации кортежей отношения. Во многих СУБД допускается создавать отношения, не определяя ключи.

Возможны случаи, когда отношение имеет несколько комбинаций атрибутов, каждая из которых однозначно определяет все кортежи отношения. Все эти комбинации атрибутов являются ***возможными*** *ключами* отношения. Любой из возможных ключей может быть выбран как *первичный.*

Если выбранный первичный ключ состоит из минимально необходимого набора атрибутов, говорят, что он является ***не избыточным.***

Ключи обычно используют для достижения следующих целей:

1) исключения дублирования значений в ключевых атрибутах (остальные атрибуты в расчет не принимаются);

2)упорядочения кортежей. Возможно упорядочение по возрастанию или убыванию значений всех ключевых атрибутов, а также смешанное упорядочение (по одним — возрастание, а по другим — убывание);

3)ускорения работы к кортежами отношения (подраздел 3.2);

4)организации связывания таблиц (подраздел 3.3).

Пусть в отношении R1 имеется не *ключевой* атрибут А, значения которого являются значениями *ключевого* атрибута В другого отношения R2. Тогда говорят, что атрибут А отношения R1 есть ***внешний ключ.***

С помощью внешних ключей устанавливаются связи между оптик ним ми. Например, имеются два отношения СТУДЕНТУФИО, Группа, (специальность) и ПРЕДМЕТ(Назв.Пр, Часы), которые связаны отношением СТУДЕНТ_ПРЕДМЕТ(ФИО. Назв.Пр. Оценка) (рис. 3.2). В связующем отношении атрибуты ФИО и Назв.Пр образуют составной ключ. Эти атрибуты представляют собой внешние ключи, являющиеся первичными ключами других отношений.

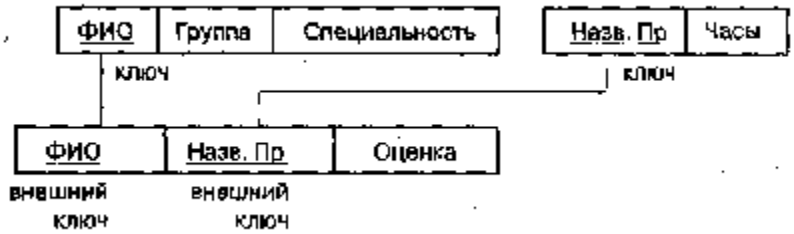


Рис. 3.2. Связь отношений

Реляционная модель накладывает на внешние ключи ограничение для обеспечения целостности данных, называемое *ссылочной целостностью*. Это означает, что каждому значению внешнего ключа должны соответствовать строки в связываемых отношениях.

Поскольку не всякой таблице можно поставить в соответствие отношение, приведем условия, выполнение которых позволяет таблицу считать отношением.

1. Все строки таблицы должны быть уникальны, то есть не может быть строк с одинаковыми первичными ключами.
2. Имена столбцов таблицы должны быть различны, а значения их простыми, то есть недопустима группа значений в одном столбце одной строки.
3. Все строки одной таблицы должны иметь одну структуру, соответствующую именам и типам столбцов.
4. Порядок размещения строк в таблице может быть произвольным.

Наиболее часто таблица с отношением размещается в отдельном файле.

В некоторых СУБД одна отдельная таблица (отношение) считается базой данных. В других СУБД база данных может содержать несколько таблиц.

В общем случае можно считать, что БД включает одну или несколько таблиц, объединенных смысловым содержанием, а также процедурами контроля целостности и обработки информации в интересах решения некоторой прикладной задачи. Например, при использовании СУБД Microsoft Access в файле БД наряду с таблицами хранятся и другие объекты базы: запросы, отчеты, формы, макросы и модули.

Таблица данных обычно хранится на магнитном диске в отдельном файле операционной системы, поэтому по ее именованию могут существовать ограничения. Имена нолей хранятся внутри таблиц. Правила их формирования определяются СУБД, которые, как правило, на длину полей и используемый алфавит серьезных ограничений не накладывают.

Вели задаваемое таблицей отношение имеет ключ, то считается, что таблица гоже имеет ключ, и ее называют ***ключевой*** или ***таблицей с ключевыми полями.***

У большинства СУБД файл таблицы включает управляющую часть (описание типов полей, имена полей и другая информация) и область раз-

мещения записей.

К отношениям можно применять систему операций, позволяющую получить одни отношения из других. Например, результатом запроса к реляционной БД может быть новое отношение, вычисленное на основе имеющихся отношений. Поэтому можно разделить обрабатываемые данные на хранимую и вычисляемую части.

Основной единицей обработки данных в реляционных БД является отношение, а не отдельные его кортежи (записи).

3.2. Индексирование

Как отмечалось выше, определение ключа для таблицы означает автоматическую сортировку записей, контроль отсутствия повторений значений в ключевых полях записей и повышение скорости выполнения операций поиска в таблице. Для реализации этих функций в СУБД применяют *индексирование*.

Термин «индекс» тесно связан с понятием «ключ», хотя между ними есть и некоторое отличие.

Под ***индексом*** понимают средство *ускорения* операции поиска записей в таблице, а следовательно, и других операций, использующих поиск: извлечение, модификация, сортировка и т. д. Таблицу, для которой используется индекс, называют *индексированной*.

Индекс выполняет роль *оглавления* таблицы, просмотр которого предшествует обращению к записям таблицы. В некоторых системах, например, Paradox, индексы хранятся в индексных файлах, хранимых отдельно от табличных файлов.

Варианты решения проблемы организации физического доступа к информации зависят в основном от следующих факторов:

- вида содержимого в поле ключа записей индексного файла;
- типа используемых ссылок (указателей) на запись основной таблицы;
- метода поиска нужных записей.

В поле *ключа индексного файла* можно хранить значения ключевых полей индексируемой таблицы либо свертку ключа (так называемый хеш-код). Преимущество хранения хеш-кода вместо значения состоит в том, что длина свертки независимо от длины исходного значения ключевого поля всегда имеет некоторую постоянную и достаточно малую величину (например, 4 байта), что существенно снижает время поисковых операций. Недостатком хеширования является необходимость выполнения операции свертки (требует определенного времени), а также борьба с возникновением коллизий (свертка различных значений может дать одинаковый хеш-код).

Для организации *ссылки* на запись таблицы могут использоваться три типа адресов: абсолютный (действительный), относительный и символический (идентификатор).

На практике чаще всего используются *два метода поиска:* последовательный и бинарный (основан на делении интервала поиска пополам).

Проиллюстрируем организацию индексирования таблиц двумя схемами: одноуровневой и двухуровневой. При этом примем ряд предположений, обычно выполняемых в современных вычислительных системах. Пусть ОС поддерживает прямую организацию данных на магнитных дисках, основные таблицы и индексные файлы хранятся в отдельных файлах. Информация файлов хранится в виде совокупности блоков фиксированного размера, например целого числа кластеров.

При ***одноуровневой схеме*** в индексном файле хранятся короткие записи, имеющие два поля: поле содержимого старшего ключа (хеш-кода ключа) адресуемого блока и поле адреса начала этого блока (рис. 3.3)

В каждом блоке записи располагаются в порядке возрастания значения ключа или свертки. Старшим ключом каждого блока является ключ его последней записи.

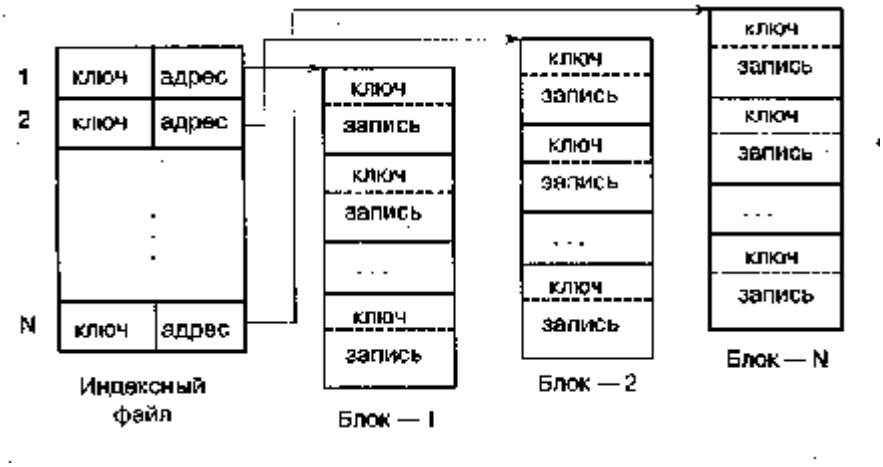


Рис. 3.3. Одноуровневая схема индексации

Если в индексном файле хранятся хеш-коды ключевых полей индексированной таблицы, то алгоритм поиска нужной записи (с указанным ключом) и таблице включает в себя следующие три этапа.

1.Образование свертки значения ключевого поля искомой записи.

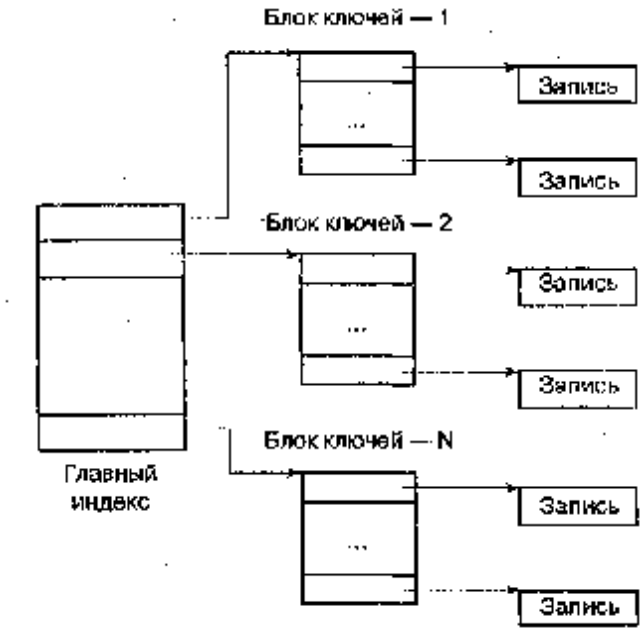
2.Поиск в индексном файле записи о блоке, значение первого поля которого больше полученной свертки (это гарантирует нахождение искомой свертки в этом блоке).

3.Последовательный просмотр записей блока до совпадения сверток искомой записи и записи блока файла. В случае коллизий сверток ищется запись, значение ключа которой совпадает со значением ключа искомой записи.

Основным *недостатком* одноуровневой схемы является то, что ключи (свертки) записей хранятся вместе с записями. Это приводит к увеличению времени поиска записей из-за большой длины просмотра (значения данных в записях приходится пропускать).

Двухуровневая схема в ряде случаев оказывается более рациональной, в ней ключи (свертки) записей отделены от содержимого записей (рис. 3.4).

В этой схеме индекс основной таблицы распределены по совокупности файлов: одному файлу главного индекса и множеству файлов с блоками ключей.



На практике для создания индекса для некоторой таблицы БД пользователь указывает поле таблицы, которое требует индексации. Ключевые поля таблицы во многих СУБД как правило индексируются автоматически. Индексные файлы, создаваемые по ключевым полям таблицы, часто называются *файлами первичных индексов.*

Индексы, создаваемые пользователем для не ключевых полей, иногда называют ***вторичными (пользовательскими) индексами.*** Введение таких индексов не изменяет физического расположения записей таблицы, но влияет на последовательность просмотра записей. Индексные файлы, создаваемые для поддержания вторичных индексов таблицы, обычно называются *файлами вторичных индексов.*

Связь вторичного индекса с элементами данных базы может быть установлена различными способами. Один из них — использование вторичного индекса как входа для получения первичного ключа, по которому затем с использованием первичного индекса производится поиск необходимых записей (рис. 3.5).

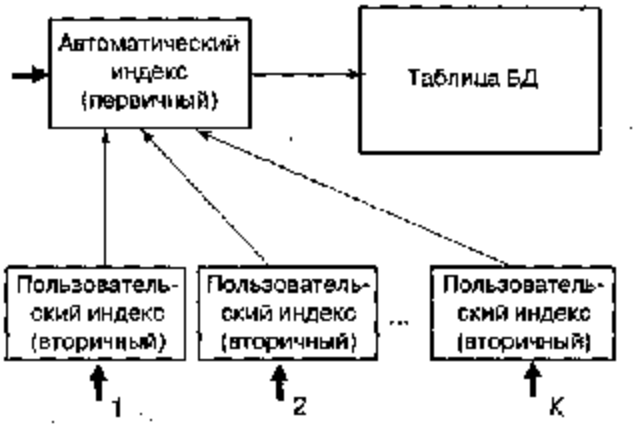


Рис. 3.5. Способ использования вторичных индексов

Некоторыми СУБД, например Access, деление индексов на первичные и вторичные не производится. В этом случае используются автоматически создаваемые индексы и индексы, определяемые пользователем по любому из не ключевых полей.

Главная причина повышения скорости выполнения различных операций в индексированных таблицах состоит в том, что основная часть работы производится с небольшими индексными файлами, а не с самими таблицами. Наибольший эффект повышения производительности работы с индексированными таблицами достигается для значительных по объему таблиц. Индексирование требует небольшого дополнительного места на диске и незначительных затрат процессора на изменение индексов в процессе работы. Индексы в общем случае могут изменяться перед выполнением запросов к БД, после выполнения запросов к БД, по специальным командам пользования или программным вызовам приложений.

3.3. Связывание таблиц

При проектировании реальных БД информацию обычно размещают в нескольких таблицах. Таблицы при этом связаны семантикой информации. В реляционных СУБД для указания связей таблиц производят операцию их *связывания*.

Укажем выигрыш, обеспечиваемый в результате связывания таблиц. Многие СУБД при связывании таблиц автоматически выполняют контроль целостности вводимых в базу данных в соответствии с установленными связями. В конечном итоге это *повышает достоверность* хранимой в БД информации.

Кроме того, установление связи между таблицами *облегчает доступ* к данным. Связывание таблиц при выполнении таких операций, как поиск, просмотр, редактирование, выборка и подготовка отчетов, обычно обеспечивает возможность обращения к произвольным полям связанных записей. Это уменьшает количество явных обращений к таблицам данных и число манипуляций в каждой из них.

Основные виды связи таблиц

Между таблицами могут устанавливаться бинарные (между двумя таблицами), тернарные (между тремя таблицами) и, в общем случае, n-

арные связи. Рассмотрим наиболее часто встречающиеся *бинарные* связи.

При связывании двух таблиц выделяют основную и дополнительную (подчиненную) таблицы. Логическое связывание таблиц производится с помощью ***ключа связи.***

Ключ связи, по аналогии с обычным ключом таблицы, состоит из одного или нескольких полей, которые в данном случае называют ***полями связи*** (ПС).

Суть связывания состоит в установлении соответствия полей связи основной и дополнительной таблиц. Поля связи основной таблицы могут быть обычными и ключевыми. В качестве полей связи подчиненной таблицы чаще всего используют ключевые поля.

В зависимости от того, как определены ноля сняли основной и дополнительной таблиц (как соотносятся ключевые поля с полями сняли), между двумя таблицами в общем случае могут устанавливаться следующие четыре основных вида связи (табл. 3.2):

- один — один (1:1);
- один — много (1:M);
- много — один (M:1);
- много — много (M:M или M:N).
-

Таблица 3.2 Характеристика видов связей таблиц

Характеристика полей связи по видам	1:1	1:M	M:1	M:M
Поля связи основной таблицы	являются ключом	являются ключом	не являются ключом	не являются ключом
Поля связи дополнительной таблицы	являются ключом	не являются ключом	являются ключом	не являются ключом

Связь вида 1:1

Связь вида 1:1 образуется в случае, когда все поля связи основной и дополнительной таблиц являются ключевыми. Поскольку значения в ключевых полях обеих таблиц не повторяются, обеспечивается взаимнооднозначное соответствие записей из этих таблиц. Сами таблицы, по сути, здесь становятся равноправными.

Пример 1.

Пусть имеются основная Ol и дополнительная Д1 таблицы. Ключевые поля обозначим символом «*», используемые для связи поля обозначим символом «+».

Таблица. О1

* +

Поле11	Поле12
а	10
б	40
в	3

Таблица. Д1

* +

Поле21	Поле22
а	стол
в	книга

В приведенных таблицах установлена связь между записью (а, 10) таблицы О1 и записью (а, стол) таблицы Д1. Основанием этого является совпадение значений в полях связи. Аналогичная связь существует и между записями (в, 3) и (в, книга) этих же таблиц. В таблицах записи отсортированы по значениям в ключевых полях.

Сопоставление записей двух таблиц по существу означает образование новых «виртуальных записей» (псевдозаписей). Так, первую пару записей логически можно считать новой псевдозаписью вида (а, 10, стол), а вторую пару псевдозаписью вида (в, 3, книга).

На практике связи вида 1:1 используются сравнительно редко, так как хранимую в двух таблицах информацию легко объединить в одну таблицу, которая нанимает гораздо меньше места в памяти ЭВМ. Возможны случаи, когда удобнее иметь не одну, а две и более таблицы. Причинами этого может быть необходимость ускорить обработку, повысить удобство работы нескольких пользователей с общей информацией, обеспечить более высокую степень защиты информации и т. д. Приведем пример, иллюстрирующий последнюю из приведенных причин.

Пример 2.

Пусть имеются сведения о выполняемых в некоторой организации научно-исследовательских работах. Эти данные включают в себя следующую информацию по каждой из работ: тему (девиз и полное наименование работ), шифр (код), даты начала и завершения работы, количество этапов, головного исполнителя и другую дополнительную информацию. Все работы имеют гриф «Для служебного пользования» или «секретно».

В такой ситуации всю информацию целесообразно хранить в двух таблицах: в одной из них — всю секретную информацию (например, шифр, полное наименование работы и головной исполнитель), а в другой — всю оставшуюся несекретную информацию. Обе таблицы можно связать по шифру работы. Первую из таблиц целесообразно защитить от несанкцио-

нированного доступа.

Связь вида 1:М

Связь 1:М имеет место в случае, когда одной записи основной таблицы соответствует несколько записей вспомогательной таблицы. Пример 3.

Пусть имеются две связанные таблицы О2 и Д2. В таблице О2 содержится информация о видах мультимедиа-устройств ПЭВМ, а и таблице Д2 — сведения о фирмах-производителях этих устройств, а также о наличии на складе хотя бы одного устройства.

Таблица 02

*** +**

Код	Вид устройства
а	CD-ROM
б	CD-Recorder
в	Sound Blaster

Таблица Д2

* + *

Код	Фирма-	Наличие
а	Acer	да
а	Mitsumi	нет
а	NEC	да
а	Panasonic	да
а	Sony	да
б	Philips	нет
б	Sony	нет
б	Yamaha	да
в	Creative Labs	да

Таблица Д2 имеет два ключевых поля, так как одна и та же фирма может производить устройства различных видов. В примере фирма Sony производит устройства считывания и перезаписи с компакт-дисков.

Сопоставление записей обеих таблиц по полю «Код» порождает псевдозаписи вида: (а, CD-ROM, Acer, да), (а, CD-ROM, Mitsumi, нет), (а, CD-ROM, NEC, да), (а, CD-ROM, Panasonic, да), (а, CD-ROM, Sony, да), (б, CD-Recorder, Philips, нет), (б, CD-Recorder, Sony, да) и т. д.

Если свести псевдозаписи в новую таблицу, то получим полную информацию обо всех видах мультимедиа-устройств ПЭВМ, фирмах, их производящих, а также сведения о наличии конкретных видов устройств на складе.

Связь вида М:1

Связь М:l имеет место в случае, когда одной или нескольким записям основной таблицы ставится в соответствие одна запись дополнительной таблицы.

Пример 4.

Рассмотрим связь таблиц О3 и Д3. В основной таблице О3 содержится информация о названиях деталей (Поле11), видах материалов, из которого детали можно изготовить (Поле12), и марках материала (Поле13). В дополнительной таблице Д3 содержатся сведения о названиях деталей (Поле21), планируемых сроках изготовления (Поле22) и стоимости заказов (Поле23).

Таблица 03

+

Поле11	Поле12	Поле13
деталь1	чугун	марка1
деталь1	чугун	марка2
деталь2	сталь	марка1
деталь2	сталь	марка2
деталь2	сталь	марка3
деталь3	алюминий	-
деталь4	чугун	марка2

Таблица Д3

* +

Поле21	Поле22	Поле23
деталь1	4.03.98	90
деталь2	3.01.98	35
Деталь 3	17.02.98	90
деталь4	6.05.98	240

Связывание этих таблиц обеспечивает такое установление соответствия между записями, которое эквивалентно образованию следующих псевдозаписей: (деталь1, чугун, марка1, 4.03.98, 90), (деталь1, чугун, марка2, 4.03.98, 90), (деталь2, сталь, марка1, 3.01.98, 35), (деталь2, сталь, марка2, 3.01.98, 35), (деталь2, сталь, марка3, 3.01.98,35), (деталь3, алюминий, —, 17.02.98,90), (деталь4, чугун, марка2, 6.05.98, 240).

Полученная псевдотаблица может быть полезна при планировании или принятии управленческих решений, когда необходимо иметь все возможные варианты исполнения заказов по каждому изделию. Таблица О3 не имеет ключей ив ней возможно повторение записей. Если таблицу Д3 сделать основной, а таблицу О3 —дополнительной, получим связь вида 1:М. Поступив аналогично с таблицами 02 и Д2, можно получить связь вида М:1. Отсюда следует, что вид связи (1:М или М:1) зависит от того, какая таблица является главной, а какая дополнительной.

Связь вида М:М

Самый общий вид связи М:М возникает в случаях, когда нескольким записям основной таблицы соответствует несколько записей дополнительной таблицы.

Пример 5.

Пусть в основной таблице 04 содержится информация о том, на каких станках могут работать рабочие некоторой бригады. Таблица Д4 содержит сведения о том, кто из бригады ремонтников какие станки обслуживает.

Таблица 04

* * +

Работает	На станке
Иванов А.В.	станок1
Иванов А.В.	станок2
Петров Н.Г.	станок1
Петров Н.Г.	станокЗ
Сидоров В.К.	станок2

Таблица Д4

* * +

Обслуживает	Станок
Голубев Б.С.	станок1
Голубев Б.С.	станокЗ
Зыков А.Ф.	станок2
Зыков А.Ф.	станокЗ

Первой и третьей записям таблицы 04 соответствует первая запись таблицы Д4 (у всех этих записей значение второго поля — «станок!»). Четвертой записи таблицы 04 соответствуют вторая и четвертая записи таблицы Д4 (во втором поле этих записей содержится «станокЗ»).

Исходя из определения полей связи этих таблиц можно составить новую таблицу с именем «О4+Д4», записями которой будут псевдозаписи. Записям полученной таблицы можно придать смысл возможных смен, составляемых при планировании работы. Для удобства, поля новой таблицы переименованы (кстати, такую операцию предлагают многие из современных иных СУБД).

Таблица «04+Д4»

Работа	Станок	Обслуживание
Инанов	станок1	Голубев В.С.
Иванов	станок2	Зыков А.Ф.
Петров	станок1	Голубев В.С.
Петров	станокЗ	Голубев В.С.
Петров	станокЗ	Зыков А.Ф.
Сидоров	станок2	Зыков А.Ф.

Приведенную таблицу можно использовать, например, для получения ответа на вопрос: «Кто обслуживает станки, на которых трудится Петров Н.Г.?».

Очевидно, аналогично связи 1:1, связь *M:M* не устанавливает подчинение таблиц.

3.4. Контроль целостности связей

Из перечисленных видов связи чаще используется связь вида 1:M. Связь вида 1:1 можно считать частным случаем связи 1:M, когда одной записи главной таблицы соответствует одна запись вспомогательной таблицы. Связь M:1, по сути, является «зеркальным отображением» связи 1:M. Оставшийся вид связи M:M характеризуется как слабый вид связи или даже как отсутствие связи. Поэтому в дальнейшем рассматривается связь вида 1:M.

При образовании связи вида **1:M одна** запись главной таблицы (главная, родительская запись) оказывается связанном с несколькими записями дополнительной (дополнительные, подчиненные записи) и имеет место схема, показанная на рис. 3.6.

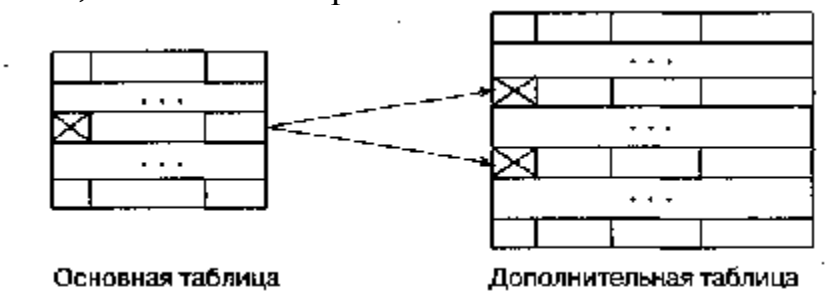


Рис. 3.6. Связь 1:M записей двух таблиц

Контроль целостности связей обычно означает анализ содержимого двух таблиц на соблюдение следующих правил:

- каждой записи основной таблицы соответствует нуль или более записей дополнительной таблицы;
- в дополнительной таблице нет записей, которые не имеют родительских записей в основной таблице;
- каждая запись дополнительной таблицы имеет только одну родительскую запись основной таблицы.

Опишем действие контроля целостности при манипулировании данными в таблицах. Рассмотрим три основные операции над данными двух таблиц:

- ввод новых записей,
- модификацию записей,
- удаление записей.

При рассмотрении попытаемся охватить все возможные методы организации контроля целостности. В реальных СУБД могут применяться собственные методы, подобные описываемым.

При ***вводе новых записей*** возникает вопрос определения последовательности ввода записей в таблицы такой, чтобы не допустить нарушение целостности. Исходя из приведенных правил, логичной является схема, при которой данные сначала вводятся в основную таблицу, а по-

том — в дополнительную. Очередность ввода может быть установлена на уровне целых таблиц или отдельных записей (случай одновременного ввода в несколько открытых таблиц).

В процессе заполнения *основной* таблицы контроль значений полей связи ведется как контроль обычного ключа (на совпадение со значениями тех же полей других записей). Заполнение но ни связи *дополнительной* таблицы контролируется на предмет совпадения со значениями полей связи основной таблицы. Если вновь вводимое значение в поле связи дополнительной таблицы не совпадает ни с одним соответствующим значением в записях основной таблицы, то ввод такого значения должен блокироваться.

Модификация записей. Изменение содержимого полей связанных записей, не относящихся к полям связи, очевидно, должно происходить обычным образом. Нас будет интересовать механизм изменения полей связи.

При редактировании полей связи дополнительной таблицы очевидным требованием является то, чтобы новое значение поля связи *совпадало* с соответствующим значением какой-либо записи основной таблицы. То есть дополнительная запись может сменить *родителя,* но остаться без него не должна.

Редактирование поля связи основной таблицы разумно подчинить одному из следующих правил:

- редактировать записи, у которых нет подчиненных записей. Если есть подчиненные записи, то блокировать модификацию полей связи;
- изменения в полях связи основной записи мгновенно передавать во все поля связи всех записей дополнительной таблицы (каскадное обновление).

В операциях *удаления записей* связанных таблиц большую свободу, очевидно, имеют записи дополнительной таблицы. Удаление их должно происходить практически бесконтрольно.

Удаление записей основной таблицы логично подчинить одному из следующих правил:

- удалять можно запись, которая не имеет подчиненных записей;
- запретить (блокировать) удаление записи при наличии подчиненных записей, либо удалять ее вместе со всеми подчиненными записями (каскадное удаление).

3.5. Теоретические языки запросов

Операции, выполняемые над отношениями, можно разделить на две группы. Первую группу составляют операции над множествами, к которым относятся операции: объединения, пересечения, разности, деления и декартова произведения, Вторую группу составляют специальные операции над отношениями, к которым, в частности, относятся операции: проекции, соединения, выбора.

В различных СУБД реализована некоторая часть операций над от-

ношениями, определяющая в какой-то мере возможности данной СУБД и сложность реализации запросов к БД.

В реляционных СУБД для выполнения операций над отношениями используются две группы языков, имеющие в качестве своей математической основы теоретические языки запросов, предложенные Э.Коддом:

- реляционная алгебра;
- реляционное исчисление.

Эти языки представляют минимальные возможности реальных языков манипулирования данными в соответствии с реляционной моделью и эквивалентны друг другу по своим выразительным возможностям. Существуют не очень сложные правила преобразования запросов между ними.

В реляционной алгебре операнды и результаты всех действий являются отношениями. Языки реляционной алгебры являются *процедурными*, так как отношение, являющееся результатом запроса к реляционной БД, вычисляется при выполнении последовательности реляционных операторов, применяемых к отношениям. Операторы состоят из операндов, в роли которых выступают отношения, и реляционных операций. Результатом реляционной операции является отношение.

Языки *исчислений*, в отличие от реляционной алгебры, являются *непроцедурными* (описательными, или декларативными) и позволяют выражать запросы с помощью предиката первого порядка (высказывания в виде функции), которому должны удовлетворять кортежи или домены отношений. Запрос к БД, выполненный с использованием подобного языка, содержит лишь информацию о желаемом результате. Для этих языков характерно наличие наборов правил для записи запросов. В частности, к языкам этой группы относится SQL.

При рассмотрении языков реляционной алгебры и исчислений будем использовать базу данных, включающую в себя следующие таблицы:

- S (поставщики);
- P (детали);
- SP (поставки).

Первичными ключами этих таблиц являются соответственно: П# (код поставщика), Д# (код детали) и составной ключ (П#, Д#). Содержимое таблиц приведено на рис. 3.7. Для удобства изложения предположим, что в рассматриваемых языках запросов нет ограничений на употребление символов русского алфавита в именах атрибутов. Каждое из полей П# и Д# таблицы SP в отдельности является внешним ключом по отношению к таблице S и P соответственно.

Предположим, что имена доменов (множеств допустимых значений) совпадают с именами атрибутов. Исключение составляют атрибуты Город_П (город, в котором находится поставщик) и Город_Д (город, в котором выпускается деталь), которые имеют общий домен: множество названий городов. Имя этого домена может быть, например, просто Город. Характеристики доменов как типов данных следующие: Д# — строка символов длиной 5, Имя — строка символов длиной 20, Статус — цифровое длиной 5,

Город — строка символов длиной 15, Д# - строка символов длиной 6, Тип - строка символов длиной 6, Вес — цифровое длиной 5, Количество - цифровое длиной 5.

S

П #	Имя	Статус	ГородП
S1	Сергей	20	Москва
S2	Иван	10	Киев
S3	Борис	30	Киев <
S4	Николай	20	Москва
S5	Андрей	30	Минск

P

Д#	Название	Тип	Вес	Город_Д
P1	гайка	каленый	12	Москва
P2	болт	мягкий	17	Киев
P3	винт	твердый	17	Ростов
P4	винт	каленый	14	Москва
P5	палец	твердый	12	Киев
P6	шпилька	каленый	19	Москва

SP

П#	Д#	Количество
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

Рис. 3.7. Таблицы поставщиков, деталей и поставок

3.6. Реляционная алгебра

Реляционная алгебра как теоретический язык запросов по сравнению с реляционным исчислением более наглядно описывает выполняемые над отношениями действия.

Примером языка запросов, основанного на реляционной алгебре, яв-

ляется ISBL (Information System Base Language — базовый язык информационных систем). Языки запросов, построенные на основе реляционной алгебры, в современных СУБД широкого распространения не получили. Однако знакомство с ней полезно для понимания сути реляционных операций, выражаемых другими используемыми языками.

Вариант реляционной алгебры, предложенный Коддом, включает в себя следующие *основные операции:* объединение, разность (вычитание), пересечение, декартово (прямое) произведение (или произведение), выборка (селекция, ограничение), проекция, деление и соединение. Упрощенное графическое представление этих операций приведено на рис. 3.8.

По справедливому замечанию Дейта, реляционная алгебра Кодда обладает несколькими недостатками. Во-первых, восемь перечисленных операций по охвату своих функций, с одной стороны, избыточны, так как минимально необходимый набор составляют пять операций: объединение, вычитание, произведение, проекция и выборка. Три другие операции (пересечение, соединение и деление) можно определить через пять минимально необходимых. Так. например, соединение — это проекция выборки произведения.

Во-вторых, этих восьми операций недостаточно для построения реальной СУБД на принципах реляционной алгебры. Требуются расширения, включающие операции: переименования атрибутов, образования новых вычисляемых атрибутов, вычисления итоговых функций, построения сложных алгебраических выражений, присвоения, сравнения и т. д.

Рассмотрим перечисленные операции более подробно, сначала — операции реляционной алгебры Кодда, а затем — дополнительные операции, введенные Дейтом.

Операции реляционной алгебры Кодда можно разделить на *две* группы: *базовые теоретико-множественные* и *специальные реляционные.* Первая группа операций включает в себя классические операции теории множеств: объединение, разность, пересечение и произведение. Вторая группа представляет собой развитие обычных теоретико-множественных операций в направлении к реальным задачам манипулирования данными, в ее состав входят следующие операции: проекция, селекция, деление и соединение.

Операции реляционной алгебры могут выполняться над одним отношением (например, проекция) или над двумя отношениями (например, объединение). В первом случае операция называется унарной, а во втором — бинарной. При выполнении бинарной операции участвующие в операциях отношения должны быть совместимы по структуре.

Рис. 3.8. Основные операции реляционной алгебры

Совместимость структур отношений означает совместимость имен атрибутов и типов соответствующих доменов. Частным случаем совместимости является идентичность (совпадение). Для устранения конфликтов имен атрибутов в исходных отношениях (когда совпадение имен недопустимо), а также для построения произвольных имен атрибутов результирующего отношения применяется операция переименования атрибутов. Структура результирующего отношения по определенным правилам наследует свойства структур исходных отношений. В большинстве рассматриваемых бинарных реляционных операций будем считать, что заголовки исходных отношений идентичны, так как в этом случае не возникает проблем с заголовком результирующего отношения (в общем случае, заголовки могут не совпадать, тогда нужно оговаривать правила формирования заголовка отношения-результата).

Объединением двух совместимых отношений R1 и R2 одинаковой размерности (Rl UNION R2) является отношение R, содержащее все элементы исходных отношений (с исключением повторений).

Пример 1. Объединение отношений.

Пусть отношение обозначает R1 множество поставщиков из Лондона, а отношение R2 — множество поставщиков, которые поставляют деталь P1. Тогда отношение R обозначает поставщиков, находящихся в Лондоне, или поставщиков, выпускающих деталь P1, либо тех и других.

R1

п#	Имя	Статус	Город_П
S1	Сергей	20	Москва
S4	Николай	20	Москва

R2

п#	Имя	Статус	Город_П
S1	Сергей	20	Москва
S2	Иван	10	Киев

R(R1 UNION R2)

п#	Имя	Статус	Город П
S1	Сергей	20	Москва
S2	Иван	10	Киев
S4	Николай	20	Москва

Вычитание совместимых отношений R1 и R2 одинаковой размерности (R1 MINUS R2) есть отношение, тело которого состоит из множества кортежей, принадлежащих R1, но не принадлежащих отношению R2. Для тех же отношений К I и R2 из предыдущего примера отношение R будет представлять собой множество поставщиков, находящихся в Лондоне, но не выпускающих деталь P1, то есть R={(S4, Николай, 20, Москва)}.

Заметим, что результат операции вычитания зависит от порядка следования операндов, то есть Rl MINUS R2 и R2 MINUS R1 — не одно и то же.

Пересечение двух совместимых отношений R1 и R2 одинаковой размерит in (Rl INTERSECT R2) порождает отношение R с телом, включающим и себя кортежи, одновременно принадлежащие обоим исходным отношением. Для отношений R1 и R2 результирующее отношение R будет означать тех производителей из Лондона, выпускающих деталь P1. Тело отношения N состоит из единственного элемента (S1, Сергей, 20, Москва).

Произведение отношения R1 степени к1 и отношения R2 степени к2 (R1 TIMES R2), которые не имеют одинаковых имен атрибутов, есть такое отношение R степени (к1+к2), заголовок которого представляет сцепление заголовков отношений R1 и R2, а тело имеет кортежи такие, что первые к1 элементов кортежей принадлежат множеству R1, а последние к2 элементов — множеству R2. При необходимости получить произведение двух отношений, имеющих одинаковые имена одного или нескольких атрибутов, применяется ни рация переименования RENAME, рассматриваемая далее.

Пример 2. Произведение отношений.

Пусть отношение R1 представляет собой множество номеров всех

текущих поставщиков {SI, S2, S3, S4, S5}, а отношение R2 — множество номеров тех текущих деталей (P1, P2, P3, P4, P5, P6}.. Результатом операции Rl TIMES R*2* является множество всех пар типа «поставщик — деталь», то есть {(S1,P1), i S l,P2), (S1.P3), (S1.P4), (S1JP5), (S1.P6), (S2.P1),..., (S5.P6)}.

В теории множеств результатом операции прямого произведения является множество, каждый элемент которого является парой элементов, первый из которых принадлежит R1, а второй — принадлежит R2. Поэтому кортежами декартова произведения бинарных отношений будут кортежи вида: ((а, б), (в, г)), где кортеж (а, б) принадлежит отношению RI, а кортеж (в, г) — принадлежит отношению R2. В реляционной алгебре применяется расширенный вариант прямого произведения, при котором элементы кортежей двух исходных отношений сливаются, что при записи кортежей результирующего отношения означает удаление лишних скобок, то есть (а, б, в, г).

Выборка (R WHERE f) отношения R по формуле f представляет собой повое отношение с таким же заголовком и телом, состоящим из таких кортежей отношения R, которые удовлетворяют истинности логического выражения, заданного формулой f. Для записи формулы используются операнды — имена атрибутов (или номера столбцов), константы, логические операции (AND — И, OR — ИЛИ, NOT — НЕ), операции сравнения и скобки.

Примеры 3. Выборки. P WHERE Вес < 14

д#	Название	Тип	Вес	Город_Д
P1	гайка	каленый	12	Москва
P5	палец	твердый	12	Киев

SP WHERE П# = “SI” AND Д# = “P1”

п#	**д#**	Количество
S1	P1	300

Проекция отношения А на атрибуты X, Y,..., Z (A [X, Y,..., Z]), где множество {X, Y,..., Z) является подмножеством полного списка атрибутов заголовка отношения А, представляет собой отношение с заголовком X, Y,..., Z и телом, содержащим кортежи отношения А, за исключением повторяющихся кортежей. Повторение одинаковых атрибутов в списке X, Y,..., Z запрещается.

Операция проекции допускает следующие дополнительные варианты записи:

- отсутствие списка атрибутов подразумевает указание всех атрибутов (операция тождественной проекции);
- выражение вида R[ ] означает *пустую* проекцию, результатом которой *является* пустое множество;
- операция проекции может применяться к произвольному отношению, в том числе и к результату выборки.

Примеры 4. Проекции.

P [Тип, Город_Д]

Тип	Город_Д
каленый	Москва
мягкий	Киев
твердый	Ростов
твердый	Киев

(S WHERE Город_П= «Киев») [П#]	
П#	Город_П
S2	Киев
S3	Киев

Результатом *деления* отношения R1 с атрибутами A и B на отношение R2 с атрибутом B (Rl DIVIDEBY R2), где A и B простые или составные атрибуты, причем атрибут B — общий атрибут, определенный на одном и *том* же домене (множестве доменов составного атрибута), является отношение R с заголовком A и телом, состоящим из кортежей r таких, что в отношении R1 имеются кортежи (r, s), причем множество значений s включает множество значений атрибута B отношения R2.

Пример 5. Деление отношения.

Пусть R1- проекция SP [П#, Д#], а R2 — отношение с заголовком Д# и телом {P1, P4}, тогда результатом деления R1 на R2 будет отношение R с заголовком П # и телом { S1.S4}.

R1

П#	Д#
S1	P1
S1	P2
S1	P3
S1	P4
S1	P5
S1	P6
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4
S4	P5

R2

Д#
P2
P4

R1 divideby R2

П#
S1
S4

Соединение C_f(R1, R2) (Q-соединение) отношений R1 и R2 по условию, заданному формулой представляет собой отношение R, которое можно получить путем Декартова произведения отношений R1 и R2 с последующим применением к результату операции выборки по формуле f. Правила записи формулы f такие же, как и для операции селекции.

Важными с практической точки зрения частными случаями соединения являются эквисоединение и естественное соединение.

Операция *эквисоединения* характеризуется тем, что формула задает равенство операндов. Приведенный выше пример демонстрирует частный случай операции эквисоединения по одному столбцу. Иногда эквисоединение двух отношений выполняется но таким столбцам, атрибуты которых в обоих отношениях имеют соответственно одинаковые имена и домены. В этом случае говорят об эквисоединении по общему атрибуту.

Операция *естественного соединения* (операция JOIN) применяется к двум отношениям, имеющим общий атрибут (простой или составной). Этот атрибут в отношениях имеет одно и то же имя (совокупность имен) и определен на одном и том же домене (доменах).

Результатом операции естественного соединения является отношение R, которое представляет собой проекцию эквисоединения отношений R1 и R2 по общему атрибуту на объединенную совокупность атрибутов обоих отношений.

Пример 6. Q-соединение.

Необходимо найти соединение отношений S и P по атрибутам Город_П и Город_Д соответственно, причем кортежи результирующего отношения должны удовлетворять отношению «больше» (в смысле лексикографического порядка — по алфавиту).

(S TIMES P) WHERE Город_П > Город_Д

Пример 7. Эквисоединение.

Пусть необходимо найти естественное соединение отношений S и P по общему атрибуту, характеризующему город (в отношении S — это Город_П, а в отношении P — Город_Д). Поскольку условие операции требует одинаковости имен атрибутов, по которым выполняется соединение, то применяется операция RENAME переименования атрибутов.

п#	Имя	Статус	Город_П	Д#	Название	Тип	Вес	Город_Д
S2	Иван	10	Киев	P1	гайка	каленый	12	Москва
S2	Иван	10	Киев	P4	винт	каленый	14	Москва
S2	Иван	10	Киев	P6	шпилька	каленый	19	Москва
S3	Борис	30	Киев	P1	гайка	каленый	12	Москва
S3	Борис	30	Киев	P4	винт	каленый	14	Москва
S3	Борис	30	Киев	P6	шпилька	каленый	19	Москва

(S RENAME Город_П AS Город) JOIN (P RENAME Город_Д AS Город)

П#	Имя	Статус	Город	Д#	Название	Тип	Вес
S1	Сергей	20	Москва	P1	гайка	каленый	12
S1	Сергей	20	Москва	P4	винт	каленый	14
S1	Сергей	20	Москва	P6	шпилька	каленый	19
S2	Иван	10	Киев	P2	болт	мягкий	17
S2	Иван	10	Киев	P5	палец	твердый	12
S3	Борис	30	Киев	P2	болт	мягкий	17
S3	Борис	30	Киев	P5	палец	твердый	12
S4	Николай	20	Москва	P1	гайка	каленый	12
S4	Николай	20	Москва	P6	шпилька	каленый	19

Дополиителъные операции реляционной алгебры *включают* следующие операции.

Переименование позволяет изменить имя атрибута отношения и имеет вид:

RENAME <исходное отношение> <старое имя атрибута> AS <новое имя атрибута>,

где <исходное отношение> задается именем отношения либо выражением реляционной алгебры. В последнем случае выражение заключают в круглые скобки.

Расширение порождает новое отношение, похожее на исходное, но отличающееся наличием добавленного атрибута, значения которого поручаются путем некоторых скалярных вычислений. Операция расширения имеет вид:

EXTEND <исходное отношение> ADD <выражение> AS <новый атрибут>,

где к исходному отношению добавляется (ключевое слово ADD) <новый атрибут>, подсчитываемый по правилам, заданным <выражением>. Исходное отношение может быть задано именем отношения и с помощью выражения реляционной алгебры, заключенного в круглые скобки. При этом имя нового атрибута не должно входить в заголовок исходного отношения и не может использоваться в <выражении>. Помимо обычных арифметических операций и операций сравнения, в выражении можно ис-

пользовать различные функции, называемые итоговыми, такие как: COUNT (количество), SUM (сумма), AVG (среднее), MAX (максимальное), MIN (минимальное). Например,

EXTEND (P JOIN SP) ADD (Вес * Количество) AS Общий_Вес.

Операция ***подведения итогов*** SUMMARIZE выполняет «вертикальные» или групповые вычисления и имеет следующий формат:

SUMMARIZE <исх. отн.> BY (<список атрибутов>) ADD <выр.> AS <новый атрибут>,

где исходное отношение задается именем отношения либо заключенным в круглые скобки выражением реляционной алгебры, <список атрибутов> представляет собой разделенные запятыми имена атрибутов исходного отношения Al, A2,..., AN, <выр.> — скалярное выражение, аналогичное выражению операции EXTEND, а <новый атрибут> — имя формируемого атрибута. В списке атрибутов и в выражении не должен использоваться <новый атрибут>.

Результатом операции SUMMARIZE является отношение R с заголовком, состоящим из атрибутов списка, расширенного новым атрибутом. Для получения тела отношения R сначала выполняется проецирование (назовем проекцию R1) исходного отношения на атрибуты Al, A2,..., AN, после чего каждый кортеж проекции расширяется новым (N+1)-м атрибутом. Поскольку проецирование, как правило, приводит к сокращению количества кортежей по отношению к исходному отношению (удаляются одинаковые кортежи), то можно считать, что происходит своеобразное группирование кортежей исходного отношения: одному кортежу отношения R1 соответствует один или более (если было дублирование при проецировании) кортежей исходного отношения. Значение (N+l)-го атрибута каждого кортежа отношения R формируется путем вычисления выражения над соответствующей этому кортежу группой кортежей исходного отношения.

Пример 8. Подведение итогов.

Пусть требуется вычислить количество поставок по каждому из поставщиков.

П#	Количество_поставок
S1	6
S2	2
S3	1
S4	3

Отметим, что функция COUNT определяет количество кортежей в каждой из групп исходного отношения.

Операция *множественного подведения итогов*, подобно соответствующим операциям переименования и расширения, выполняет одновременно несколько «вертикальных» вычислений и записывает результаты отдельные новые атрибуты.

Простейшим примером такой операции может служить следующая запись:

SUMMARIZE SP BY (Д#) ADD SUM Количество AS Общее_число_поставок AVG Количество AS Среднее_число_поставок.

К основным операторам, позволяющим изменять тело существующего отношения, отнесем операции реляционного *присвоения*, *вставки*, *обновления и удаления*. Операцию *присвоения* можно представить следующим образом:

выражение-цель> := <выражение-источник>,

где оба выражения задают совместимые (точнее, эквивалентные) по структуре отношения. Типичный случай выражений: в левой части — имя отношения, а в правой — некоторое выражение реляционной алгебры. Выполнение и рации присвоения сводится к замене предыдущего значения отношения на новое (начальное значение, если тело отношения было пустым), определенное выражением-источником.

С помощью операции присвоения можно не только полностью заменить значения отношения-цели, но и добавить или удалить кортежи. Приведем пример, в котором в отношение S дописывается один кортеж:

S:= S UNION {{< П# : 'S6' >, < Имя : 'Борис' > < Статус : '50' >, <Город_П : 'Мадрид' >}}.

Более удобными операциями изменения тела отношения являются операции вставки, обновления и удаления.

Операция *вставки* INSERT имеет следующий вид:

INSERT <выражение-источник> INTO <выражение-цель>,

Где оба выражения должны быть совместимы по структуре. Выполнение операции (водится к вычислению <выражение-источник> и вставке полученных кортежей в отношение, заданное <выражение-цель>.

Пример.

INSERT (S WHERE Город_П= «Москва») INTO Temp

Операция *обновления* UPDATE имеет следующим вид

UPDATE <выражение-цель> <список элементов>,

где <список элементов> представляет собой последовательность разделенных запятыми операций присвоения <атрибут> := <скалярное выражение>| Результатом выполнения операции обновления является отношение, полученное после присвоения соответствующих значений атрибутам отношения, заданного целевым выражением.

Например,

UPDATE P WHERE Тип='каленый' Город := 'Киев'. Эта операция предписывает изменить значение атрибута Город (независимо от того, каким оно было) на новое значение — 'Киев' таких кортежей отношения P, атрибут Тип которых имеет значение 'каленый'.

Операция *удаления* DELETE имеет следующий вид:

DELETE <выражение-цель>,

где <выражение-цель> представляет собой реляционное выражение, описывающее удаляемые кортежи.

Например, DELETE S WHERE Статус < 20.

Операция *реляционного сравнения* может использоваться для прямого сравнения двух отношений. Она имеет синтаксис:

<выражение1> θ <выражение2>,

где оба выражения задают совместимые по структуре отношения, а знак θ — один из следующих операторов сравнения: = (равно), ≠(не равно), < (собственное подмножество), < (подмножество), > (надмножество), > (собственное надмножество).

Например, сравнение: «Совпадают ли города поставщиков и города хранения деталей?» можно записать так: S [Город_П] = P [Город_Д].

Основные правила записи выражений. Как отмечалось, результатом произвольной реляционной операции является отношение, которое, в свою очередь, может участвовать в другой реляционной операции. Это свойство реляционной алгебры называется *свойством замкнутости.*

Свойство замкнутости позволяет записывать *вложенные выражения* реляционной алгебры, основой которых выступают рассмотренные ранее элементарные операции: объединение, проекция, пересечение, выборка и т. д.

При записи произвольного выражения реляционной алгебры надо принимать во внимание следующее: В реляционной алгебре должен быть определен приоритет выполнения операций (например, операция пересечение более приоритетна чем операция объединение), который нужно учитывать при записи выражений. Для изменения порядка выполнения операций в выражениях можно использовать круглые скобки.

3.7. Структурированный язык запросов SQL

Структурированный язык запросов SQL основан на реляционном исчислении с *переменными кортежами.*

Общая характеристика языка

Язык SQL предназначен для выполнения операций над таблицами (создание, удаление, изменение структуры) и над данными таблиц (выборка, изменение, добавление и удаление), а также некоторых сопутствующих операций. SQL является *непроцедурным* языком и не содержит операторов управления, инициализации подпрограмм, ввода-вывода и т. п. В связи с этим SQL автономно не используется, обычно он погружен в среду встроенного языка программирования СУБД (например, FoxPro СУБД Visual FoxPro, ObjectPAL СУБД Paradox, Visual Basic for Applications СУБД

Access).

В современных СУБД с интерактивным интерфейсом можно создавать запросы, используя другие средства, например QBE. Однако применение SQL зачастую позволяет повысить эффективность обработки данных в базе. Например, при подготовке запроса в среде Access можно перейти из окна Конструктора запросов (формулировки запроса по образцу на языке QBE) в окно с эквивалентным оператором SQL. Подготовку нового запроса путем редактирования уже имеющегося в ряде случае проще выполнить путем изменения ми ритора SQL. В различных СУБД состав операторов SQL может несколько отличаться.

Различают два основных метода использования встроенного SQL: статический и динамический.

При *статическом* использовании языка *(статический SQL)* в тексте программы имеются вызовы функций языка SQL, которые жестко включаются в исполняемый модуль после компиляции. Изменения в вызываемых функциях могут быть на уровне отдельных параметров вызовов с помощью переменных языка программирования.

При *динамическом* использовании языка *(динамический SQL)* предполагается динамическое построение вызовов SQL-функций и интерпретация этих вызовов, например, обращение к данным удаленной базы, в ходе выполнения программы. Динамический метод обычно применяется в случаях, когда в приложении заранее неизвестен вид SQL-вызова и он строится в диалоге с пользователем.

Основным назначением языка SQL является подготовка и выполнение запросов. И результате выборки данных из одной или нескольких таблиц может быть получено множество записей, называемое *представлением*.

Представление по существу является таблицей, формируемой в результате выполнения запроса. Можно сказать, что оно является разновидностью хранимого запроса. По одним и тем же таблицам можно построить несколько представлений. Само представление описывается путем указания идентификатора представления и запроса, который должен быть выполнен для его получения.

Для удобства работы с представлениями в язык SQL введено понятие курсора. ***Курсор*** представляет собой своеобразный указатель, используемый для перемещения по наборам записей при их обработке.

Описание и использование курсора в языке SQL выполняется следующим образом. В описательной части программы выполняют связывание переменной типа курсор (CURSOR) с оператором SQL (обычно с оператором SELECT). В выполняемой части программы производится открытие курсора (OPEN <имя курсора>), перемещение курсора по записям (FETCH <имя курсора>...), сопровождаемое соответствующей обработкой, и, наконец, закрытие курсора (CLOSE <имя курсора>).

Основные операторы языка

Опишем минимальное подмножество языка SQL, опираясь на его

реализацию в стандартном интерфейсе ODBC (Open Database Connectivity — совместимость открытых баз данных) фирмы Microsoft.

Операторы языка SQL можно условно разделить на два подъязыка: язык определения данных (Data Definition Language — DDL) и язык манипулирования данными (Data Manipulation Language — DML). Основные операторы языка SQL представлены в табл. 3.3.

Рассмотрим формат и основные возможности важнейших операторов, за исключением специфических операторов, отмеченных в таблице символом «*». Несущественные операнды и элементы синтаксиса (например, принятое во многих системах программирования правило ставить «;» в конце оператора) будем опускать.

*1. Оператор **создания таблицы** имеет формат вида:*

CREATE TABLE <имя таблицы>

(<имя столбца> <тип данных> [NOT NULL] [,<имя столбца> <тип данных> [NOT NULL]]...)

Обязательными операндами оператора являются имя создаваемой таблицы и имя хотя бы одного столбца (поля) с указанием типа данных, хранимых в этом столбце.

При создании таблицы для отдельных полей могут указываться некоторые дополнительные правила контроля вводимых в них значений. Конструкция NOT NULL (не пустое) служит именно таким целям и для столбца таблицы означает, что в этом столбце должно быть определено значение.

Таблица 3.3 Операторы языка SQL

Вид	Название	Назначение
DDL	CREATE TABLE	создание таблицы
	DROP TABLE	удаление таблицы
	ALTER TABLE	изменение структуры таблицы создание индекса
	CREATE INDEX	
	DROP INDEX	удаление индекса
	CREATE VIEW	создание представления
	DROP VIEW	удаление представления
	GRAND*	назначение привилегий
	REVOKE*	удаление привилегий
DML	SELECT	выборка записей
	UPDATE	изменение записей
	INSERT	вставка новых записей удаление записей
	DELETE	

Пример 1. Создание таблицы.

Пусть требуется создать таблицу goods описания товаров, имеющую поля: type — вид товара, comp_id — идентификатор компании-производителя, name — название товара и price — цена товара. Оператор определения таблицы может иметь следующий вид:

CREATE TABLE goods (type SQL_CHAR(8) NOT NULL,

comp_id SQLCHAR(IO) NOT NULL, name SQL_VARCHAR(20), price SQL_DECIMAL(8,2)).

*2. Оператор **изменения структуры таблицы** имеет формат вида:*

ALTER TABLE <имя таблицы>

({ADD, MODIFY, DROP} <имя столбца> [<тип данных>] [NOT NULL]

[,{ADD, MODIFY, DROP} <имя столбца> [<тип данных>] [NOT NULL]]...)

Изменение структуры таблицы можп■ состоял, и доо.ш кипи (Л1)1)), иимс нении (MODIFY) или удалении (DROP) одного или исско пжих столбцов табли* цы. Правила записи оператора ALTER TABLE такие же, как и оператора C-RliA'l'li TABLE. При удалении столбца указывать <тип данных> не нужно.

Пример 2. Добавление поля таблицы.

Пусть в созданной ранее таблице goods необходимо добавить поле number, отводимое для хранения величины запаса товара. Для этого следует записать оператор вида:

ALTER TABLE goods (ADD number SQLJTNTEGER).

*3. Оператор **удаления таблицы** имеет формат вида:*

DROP TABLE <имя таблицы>

Оператор позволяет удалить имеющуюся таблицу. Например, для удаления таблицы с именем items достаточно записать оператор вида:

DROP TABLE items.

*4. Оператор **создания индекса** имеет формат вида:*

CREATE [UNIQUE] INDEX <имя индекса> ON <имя таблицы> (<имя столбца> [ASC | DESC] [,<имя столбца> [ASC | DESC] ...)

Оператор позволяет создать индекс для одного или нескольких столбцов заданной таблицы с целью ускорения выполнение запросных и поисковых операций с таблицей. Для одной таблицы можно создать несколько индексов.

Задав необязательную опцию UNIQUE, можно обеспечить уникальность значений во всех указанных в операторе столбцах. По существу, создание индекса с указанием признака UNIQUE означает определение ключа в созданной ранее таблице.

При создании индекса можно задать порядок автоматической сортировки значений в столбцах — в порядке возрастания ASC (по умолчанию), или в порядке убывания DESC. Для разных столбцов можно задавать различный порядок сортировки.

Пример 3. Создание индекса.

Пусть для таблицы EMP, имеющей поля: NAME (имя), SAL (зарплата), MGR (руководитель) и DEPT (отдел), нужно создать индекс main_indx для сортировки имен в алфавитном порядке и убыванию размеров зарплаты. Оператор создания индекса может иметь вид:

CREATE INDEX main_indx ON emp (name, sal DESC).

*5. Оператор **удаления индекса** имеет формат вида:*

DROP INDEX <имя индекса>

Позволяет удалять созданный ранее индекс с соответствующем именем. Так, например, для уничтожения индекса main_indx к таблице nip достаточно записать оператор DROP INDEX main_indx.

6. Оператор ***создания представления*** имеет формат вида:

CREATE VIEW <имя представления>

[(<имя столбца> [,<имя столбца>]…)] AS <оператор SELECT>

Данный оператор позволяет создать представление. Если имена столбцов в представлении не указываются, то будут использоваться имена столбцов из запроса, описываемого соответствующим оператором SELECT.

Пример 4. Создание представления.

Пусть имеется таблица companies описания производителей товаров с полями: comp_id (идентификатор компании), comp_name (название организации), comp_address (адрес) и phone (телефон), а также таблица goods производимых товаров с полями: type (вид товара), comp_id (идентификатор компании), name (название товара) и price (цена товара). Таблицы связаны между собой по полю comp_id. Требуется создать представление repr с краткой информацией о товарах 11 их производителях: вид товара, название производителя и цена товара. Оператор определения представления может иметь следующий вид:

```
CREATE VIEW
repr
AS
SELECT
goods.type, companies.comp_name, goods.price
FROM
goods, companies
WHERE
goods.comp_id = companies.comp_id
```

*7. Оператор **удаления представления** имеет формат вида:*

DROP VIEW <имя представления>

Оператор позволяет удалить созданное ранее представление. Заметим, что при удалении представления таблицы, участвующие в запросе, удалению не подлежат. Удаление представления repr производится оператором.вида:

DROP VIEW repr.

*8. Оператор **выборки записей** имеет формат вида:*

SELECT [ALL | DISTINCT] <список данных>

FROM <список таблиц>

[WHERE <условие выборки>]|

[GROUP BY <имя столбца> |,<имя столбца>]|

[HAVING <условие иоиска>|

[ORDER BY <спецификация> [,<спецификация>] ..]

Это наиболее важный оператор из всех операторов SQL. Функциональные возможности его огромны. Рассмотрим основные из них.

Оператор SELECT позволяет производить выборку и вычисления над данными из одной или нескольких таблиц. Результатом выполнения оператора является ответная таблица, которая может иметь (ALL), или не иметь (DISTINCT) повторяющиеся строки. По умолчанию в ответную таблицу включаются все строки, в том числе и повторяющиеся. В отборе данных участвуют записи одной или нескольких таблиц, перечисленных в списке операнда FROM.

Список данных может содержать имена столбцов, участвующих в запросе, а также выражения над столбцами.. В простейшем случае в выражениях можно записывать имена столбцов, знаки арифметических операций (+, — >*>/), константы и круглые скобки. Если в списке данных записано выражение, то наряду с выборкой данных выполняются вычисления, результаты которого попадают в новый (создаваемый) столбец ответной таблицы.

При использовании в списках данных имен столбцов нескольких таблиц для указания принадлежности столбца некоторой таблице применяют конструкцию вида: <имя таблицы>.<имя столбца>.

Операнд WHERE задает условия, которым должны удовлетворять записи в результирующей таблице. Выражение <условие выборки> является логическим. Его элементами могут быть имена столбцов, операции сравнения, арифметические операции, логические связки (И, ИЛИ, НЕТ), скобки, специальные функции LIKE, NULL, IN и т. д.

Операнд GROUP BY позволяет выделять в результирующем множестве записей группы. ***Группой*** являются записи с совпадающими значениями в столбцах, перечисленных за ключевыми словами GROUP BY. Выделение групп требуется для использования в логических выражениях операндов WHERE и HAVING, а также для выполнения операций (вычислений) над группами.

В логических и арифметических выражениях можно использовать следующие групповые операции (функции): AVG (среднее значение в группе), MAX (максимальное значение в группе), MIN (минимальное значение в группе), SUM (сумма значений в группе), COUNT (число значений в группе).

Операнд HAVING действует совместно с операндом GROUP BY и используется для дополнительной селекции записей во время определения групп. Правила записи <условия поиска> аналогичны правилам формирования <условия выборки> операнда WHERE.

Операнд ORDER BY задает порядок сортировки результирующего множества. Обычно каждая «<спецификация> аналогична соответствующей конструкции оператора CREATE INDEX и представляет собой пару вида: <имя столбца >[ASC| DESC].

Замечание.

Оператор SELECT может иметь и другие более сложные синтаксические конструкции.

Одной из таких конструкций, например, являются так называемые ***подзапросы.*** Они позволяют формулировать *вложенные запросы*, когда результаты одного оператора SELECT используются в логическом выражении условия выборки операнда WHERE другого оператора SELECT.

Вторым примером более сложной формы оператора SELECT является оператор, в котором отобранные записи в дальнейшем предполагается модифицировать (конструкция FOR UPDATE OF). СУБД после выполнения такого оператора обычно блокирует (защищает), отобранные записи от модификации их другими пользователями.

Еще один случай специфического использования оператора SELECT — выполнение объединений результирующих таблиц при выполнении нескольких операторов SELECT (операнд UNION).

Пример 5. Выбор записей.

Для таблицы EMP, имеющей поля: NAME (имя), SAL (зарплата), MGR (руководитель) и DEPT (отдел), требуется вывести имена сотрудников и размер их зарплаты, увеличенный на 100 единиц. Оператор выбора можно записать следующим образом:

SELECT name, sal+100 FROM emp.

Пример 6. Выбор с условием.

Вывести названия таких отделов таблицы EMP, в которых в данный момент отсутствуют руководители. Оператор SELECT для этого запроса можно записать так:

SELECT dept FROM emp WHERE mgr is NULL.

Пример 7. Выбор с группированием.

Пусть требуется найти минимальную и максимальную зарплаты для каждого из отделов (по таблице EMP). Оператор SELECT для этого запроса имеет вид:

SELECT dept, MIN(sal), MAX(sal) FROM emp GROUP BY dept.

9. Оператор изменения записей имеет формат вида:

UPDATE <имя таблицы>

SET <имя столбца> = {<выражение> , NULL }

[, SET <имя столбца> = {<выражение> , NULL } ...]

[WHERE <условие>]

Выполнение оператора UPDATE состоит в изменении значений в опреде-1 ленных операндом SET столбцах таблицы для тех записей, которые удовлетворяют условию, заданному операндом WHERE.

Новые значения полей в записях могут быть пустыми (NULL), либо вычисляться в соответствии с арифметическим выражением. Правила запи-1

си арифметических и логических выражений аналогичны соответствующим правилам оператора SELECT.

Пример 8. Изменение записей.

Пусть необходимо увеличить на 500 единиц зарплату тем служащим, которые получают не более 6000 (по таблице EMP). Запрос, сформулированный с помощью оператора SELECT, может выглядеть так:

UPDATE emp

SET sal = 6500 WHERE sal <= 6000.

*10. Оператор **вставки новых** записей имеет форматы двух видов:*

INSERT INTO <имя таблицы> [(<список столбцов>)] VALUES (<список значений>)

и

INSERT INTO <имя таблицы> [(<список столбцов>)] <предложение SELECT>

В первом формате оператор INSERT предназначен для ввода новых записей с 1 заданными значениями в столбцах. Порядок перечисления имен столбцов должен соответствовать порядку значений, перечисленных в списке операнда VALUES. Если <список столбцов> опущен, то в <списке значений> должны быть перечне- I лены все значения в порядке столбцов структуры таблицы.

Во втором формате оператор INSERT предназначен для ввода в заданную 1 таблицу новых строк, отобранных из другой таблицы с помощью предложения SELECT.

Пример 9. Ввод записей.

Ввести в таблицу EMP запись о новом сотруднике. Для этого можно записать такой оператор вида:

INSERT INTO emp

VALUES («Ivanov», 7500, «Lee», «cosmetics»).

11. Оператор *удаления записей* имеет формат вида:

DELETE EROM <имя таблицы> [WHERE <условие>]

Результатом выполнения оператора DELETE является удаление из ука-
111 ной таблицы строк, которые удовлетворяют условию, определенному операндом WHERE. Если необязательный операнд WHERE опущен, то есть условие отбора удаляемых записей отсутствует, удалению подлежат все записи.

Пример 10. Удаление записей.

15 связи с ликвидацией отдела игрушек (toy), требуется удалить из таблицы EMP всех сотрудников этого отдела. Оператор DELETE для этой задачи будет выглядеть так:

DELETE FROM emp

WHERE dept - «toy».

4. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

4.1. Проблемы проектирования

Проектирование информационных систем, включающих в себя базы данных, осуществляется на физическом и логическом уровнях. Решение проблем проектирования на *физическом уровне* во многом зависит от используемой СУВД, зачастую автоматизировано и скрыто от пользователя. В ряде случаев пользователю предоставляется возможность настройки отдельных параметров системы, которая не составляет большой проблемы..

Логическое проектирование заключается в определении числа и структуры таблиц, формировании запросов к БД, определении типов отчетных документов, разработке алгоритмов обработки информации, создании форм для ввода и редактирования данных в базе и решении ряда других задач.

Решение задач логического проектирования БД в основном определяется спецификой задач предметной области. Наиболее важной здесь является проблема структуризации данных, на ней мы сосредоточим основное внимание.

При проектировании *структур данных* для автоматизированных систем можно выделить три основных подхода:

1. Сбор информации об объектах решаемой задачи в рамках одной таблицы (одного отношения) и последующая декомпозиция ее на несколько взаимосвязанных таблиц на основе процедуры нормализации отношений.

2. Формулирование знаний о системе (определение типов исходных данных и их взаимосвязей) и требований к обработке данных, получение с помощью CASE-системы (системы автоматизации проектирования и разработки баз данных) готовой схемы БД или даже готовой прикладной информационной системы.

3. Структурирование информации для использования в информационной системе в процессе проведения системного анализа на основе совокупности правил и рекомендаций.

Основные проблемы, имеющие место при определении структур данных в отношения реляционной модели.

Избыточное дублирование данных и аномалии

Следует различать простое (неизбыточное) и избыточное дублирование данных. Наличие первого из них допускается в базах данных, а избыточное дублирование данных может приводить к проблемам при обработке данных. Приведем примеры обоих вариантов дублирования.

Пример *неизбыточного дублирования* данных представляет приведенное на рис. 4.1 отношение С__Т с атрибутами Сотрудник и Телефон. Для сотрудников, находящихся в одном помещении, номера телефонов совпадают. Номер телефона 4328 встречается несколько раз, хотя для каждого служащего номер телефона уникален. Поэтому ни один из номеров не является избыточным. Действительно, при удалении одного из номеров телефонов будет утеряна информация о том, по какому номеру можно дозвониться до одного из служащих.

с_т

Сотрудник	Телефон
Иванов И.М.	3721
Петров М.И.	4328
Сидоров Н.Г.	4328
Егоров В.В.	4328

Рис. 4.1. Неизбыточное дублирование

Пример *избыточного дублирования (избыточности)* представляет приведенное на рис. 4.2а отношение С_Т_Н, которое, в отличие от отношения С_Т, дополнено атрибутом Н_комн (номер комнаты сотрудника). Естественно предположить, что все служащие в одной комнате имеют один и тот же телефон. Следовательно, в рассматриваемом отношении имеется избыточное дублирование данных. Так, в связи с тем, что Сидоров и Егоров находятся в той же комнате, что и Петров, их номера молено узнать из кортежа со сведениями о Петрове.

На рис. 4.26 приведен пример неудачного отношения С_Т_Н, в котором вместо телефонов Сидорова и Егорова поставлены прочерки (неопределенные значения).

Сотрудник	Телефон	Н_комн
Иванов И.М.	3721	109
Петров М.И.	4328	111
Сидоров Н.Г.	4328	111
Егоров В. В.	4328	111

Сотрудник	Телефон	Н_комн
Иванов И.М.	3721	109
Петров М.И.	4328	111
Сидоров Н.Г.	—	111
Егоров В.В.	—	111

Рис. 4.2. Избыточное дублирование

Неудачность подобного способа исключения избыточности заключается в следующем. Во-первых, при программировании придется потратить дополнительные усилия на создание механизма поиска информации прочерков таблицы. Во-вторых, память все равно выделяется под атрибуты с прочерками, хотя дублирование данных и исключено. В-третьих, что особенно важно, при исключении из коллектива Петрова кортеж со сведениями о нем будет исключен из отношения, а значит, уничтожена информация о телефоне 111-й комнаты, что недопустимо.

Возможный способ выхода из данной ситуации приведен на рис. 5.3. Здесь показаны два отношения С_Н и Н_Т, полученные путем декомпозиции исходного отношения С_Т_Н. Первое из них содержит информацию о номе-

рах комнат, в которых располагаются сотрудники, а второе - информацию о номерах телефонов в каждой из комнат. Теперь, если Петрова и уволят из учреждения и, как следствие этого, удалят всякую информацию о нем из базы данных учреждения, это не приведет к утере информации о номере телефона и 111-й комнате.

Т_Н

Телефон	Н_ком
3721	109
4328	111

С_Н

Сотрудник	Н_комн
Иванов И.М.	109
Петров М.И.	111
Сидоров Н.Г.	111
Егоров В.В.	111

Рис. 4.3. Исключение избыточного дублирования

Процедура декомпозиции отношения С_Т_Н на два отношения С_Н и Н_Т является основной процедурой нормализации отношений.

Аномалиями будем начинать такую ситуацию в таблицах БД, которая при водит к противоречиям в БД либо существенно усложняет обработку данных

Выделяют три основные вида аномалий: аномалии модификации (или редактирования), аномалии удаления и аномалии добавления.

Аномалии модификации проявляются в том, что Изменение значения один го данного может повлечь за собой просмотр всей таблицы и соответствующее изменение некоторых других записей таблицы.

Так, например, изменение номера телефона в комнате 111 (рис. 5.2а), ЧТО представляет собой один единственный факт, потребует просмотра всей таблицы С_Т__Н и изменения поля Н_комн согласно текущему содержимому таблицы в записях, относящихся к Петрову, Сидорову и Егорову.

Аномалии удаления состоят в том, что при удалении какого-либо данного *m* таблицы может пропасть и другая информация, которая не связана напрямую удаляемым данным.

В той же таблице С_Т_Н удаление записи о сотруднике Иванове (напри мер, по причине увольнения или ухода на заслуженный отдых) приводит к исчезновению информации о номере телефона, установленного в 109-й комнате

Аномалии добавления возникают в случаях, когда информацию в таблицу нельзя поместить до тех пор, пока она неполная, либо вставка новой записи требует дополнительного просмотра таблицы.

Примером может служить операция добавления нового сотрудника все в ту же таблицу С_Т_Н. Очевидно, будет противоестественным хранение све-

дений в этой таблице только о комнате и номере телефона в ней, пока никто из сотрудников не помещен в нее. Более того, если в таблице G_T_H поле Служащий является ключевым, то хранение в ней неполных записей с отсутствующей фамилией служащего просто недопустимо из-за неопределенности значения ключевого поля.

Вторым примером возникновения аномалии добавления может быть ситуация включения в таблицу нового сотрудника. При добавлении таких записей для исключения противоречий желательно проверить номер телефона и соответствующий номер комнаты хотя бы с одним из сотрудников, сидящих с новым сотрудником в той же комнате. Если же окажется, что у нескольких сотрудников, сидящих в одной комнате, имеются разные телефоны, то вообще не ясно, что делать (то ли в комнате несколько телефонов, то ли какой-то из номеров ошибочный).

Формирование исходного отношения

Проектирование БД начинается с определения всех объектов, сведения о которых будут включены в базу, и определения их атрибутов. Затем атрибуты сводятся в одну таблицу - исходное отношение.

Пример. Формирование исходного отношения.

Предположим, что для учебной части факультета создается БД о преподавателях. На первом этапе проектирования БД в результате общения с завучем (заведующим учебной частью) должны быть определены содержащиеся в базе сведения о том, как она должна использоваться и какую информацию заказчик хочет получать в процессе ее эксплуатации. В результате устанавливаются атрибуты, которые должны содержаться в отношениях БД, и связи между ними. Перечислим имена выделенных атрибутов и краткие характеристики:

ФИО - фамилия и инициалы преподавателя. Исключаем возможность совпадния фамилии и инициалов у преподавателей.

Должн - должность, занимаемая преподавателем.

Оклад - оклад преподавателя.

Стаж - преподавательский стаж.

*Н*Стаж - надбавка за стаж.

Каф - номер кафедры, на которой числится преподаватель.

Предм - название предмета (дисциплины), читаемого преподавателем.

Группа - номер группы, в которой преподаватель проводит занятия.

ВидЗан - вид занятий, проводимых преподавателем в учебной группе.

Одно из требований к отношениям заключается в том, чтобы все атрибуты ношения имели атомарные (простые) значения. В исходном отношении каждый атрибут кортежа также должен быть простым. Пример исходного шипения ПРЕПОДАВАТЕЛЬ приведен на рис, 5.4.

ПРЕПОДАВАТЕЛЬ

ФИО	Должн	Оклад	Стаж	Д_Стаж	Каф	Предм	Группа	ВидЗан
Иванов	преп	500	5	100	25	СУБД	256	Практ

Иванов	преп	500	5	100	25	ПЛ/1	123	Практ
Петров	ст.преп	800	7	100	25	СУБД	256	Лекция
Петров	ст. преп	800	7	100	25	Пас-	256	Практ
Сидоров	преп	500	10	150	25	ПЛ/1	123	Лекция
Сидоров	преп	500	10	150	25	Паскаль	256	Лекция
Егоров	преп	500	5	100	24	ПЭВМ	244	Лекция

Рис. 5.4. Исходное отношение ПРЕПОДАВАТЕЛЬ

Указанное отношение имеет следующую схему ПРЕПОДАВАТЕЛЬ (ФИО, Должн, Оклад, Стаж, Д_Стаж, Каф, Предм, Группа, ВидЗан).

Исходное отношение ПРЕПОДАВАТЕЛЬ содержит избыточное дублирование данных, которое и является причиной аномалий редактирования. Различают избыточность явную и неявную.

Явная избыточность заключается в том, что в отношении ПРЕПОДАВАТЕЛЬ строки с данными о преподавателях, проводящих занятия в нескольких группах, повторяются соответствующее число раз. Например, в отношении ПРЕПОДАВАТЕЛЬ все данные по Иванову повторяются дважды. Поэтому, если Иванов И.М. станет старшим преподавателем, то этот факт должен быть отражен в обеих строках. В противном случае будет иметь место противоречие в данных, что является примером аномалии редактировании обусловленной явной избыточностью данных в отношении.

Неявная избыточность в отношении ПРЕПОДАВАТЕЛЬ проявляется *в* одинаковых окладах у всех преподавателей и в одинаковых добавках к окладу за одинаковый стаж. Поэтому, если при изменении окладов за должность с 500 на 510 это значение изменят у всех преподавателей, кроме, например, Сидорова, то база станет противоречивой. Это пример аномалии редактирования для варианта с неявной избыточностью.

Средством исключения избыточности в отношениях и, как следствие, аномалий является нормализация отношений, рассмотрим ее более подробно.

5.2. Метод нормальных форм

Проектирование БД является одним из этапов жизненного цикла информационной системы. Основной задачей, решаемой в процессе проектирования БД, является задача нормализации ее отношений. Рассматриваемый ниже метод нормальных форм является классическим методом проектировании реляционных БД. Этот метод основан на фундаментальном в теории реляционных баз данных понятии зависимости между атрибутами отношений.

Зависимости между атрибутами

Рассмотрим основные виды зависимостей между атрибутами отношении функциональные, транзитивные и многозначные.

Понятие функциональной зависимости является базовым, так как на его основе формулируются определения всех остальных видов зависимостей.

Атрибут В ***функционально зависит*** от атрибута А, если каждому значению А соответствует в точности одно значение В. Математически функции начальная зависимость В от А обозначается записью А—>В. Это означает, что ни всех кортежах с одинаковым значением атрибута А атрибут В будет иметь

также одно и то же значение. Отметим, что А и В могут быть составными состоять из двух и более атрибутов.

В отношении на рис. 4.4 можно выделить функциональные зависимости между атрибутами ФИО->Каф, ФИО->Должн, Должн->Оклад и другие Наличие функциональной зависимости в отношении определяется природой вещей, информация о которых представлена кортежами отношения. В отношении на рис. 4.4 ключ является составным и состоит из атрибутов ФИО, Предмет, Группа.

Функцикциональиая взаимозависимость. Если существует функциональная зависимость вида А->В и В->А, то между А и В имеется взаимно однозначное соответствие, или функциональная взаимозависимость. Наличие функциональной взаимозависимости между атрибутами А и В обозначим как А<->В или В<->А.

Пример. Пусть имеется некоторое отношение, включающее два атрибута, функциоонально зависящие друг от друга. Это серия и номер паспорта (N) и Фамилия, имя и отчество владельца (ФИО). Наличие функциональной зависимости поля ФИО от N означает не только тот факт, что значение поля N однозначно определяет значение поля ФИО, но и то, что одному и тому же значению поля N соответствует только единственное значение поля ФИО. Понятно, что в данном случае действует и обратная ФЗ: каждому значению ФИО соответствует только одно значение поля N. В данном примере предполагается, что ситуация наличия полного совпадения фамилий, имен и отчеств двух людей исключена.

Если отношение находится в 1НФ, то все неключевые атрибуты функционально зависят от ключа с различной степенью зависимости. *Частичной зависимостью {частичной функциональной зависимостью)* называется зависимость неключевого атрибута от части составного ключа. В рассматриваемом отношении атрибут Должн находится в функциональной зависимости от атрибута ФИО, являющегося частью ключа. Тем самым атрибут Должн находится в частичной зависимости от ключа отношения.

Альтернативным вариантом является *полная функциональная зависимость* неключевого атрибута от всего составного ключа. В нашем примере атрибут ВидЗан находится в полной функциональной зависимости от составного ключа.

Атрибут С зависит от атрибута А *транзитивно*. Существует *транзитивная зависимость*, если для атрибутов А, В, С выполняются условия А->В и И- >С, но обратная зависимость отсутствует. В отношении на рис. 4.4 транзитивной зависимостью связаны атрибуты:

Ф И О-> Должн-> Оклад

Между атрибутами может иметь место многозначная зависимость.

В отношении R атрибут В *многозначно зависит* от атрибута А если каждому значению А соответствует множество значений В, не связанных с другими атрибутами из R.

Многозначные зависимости могут быть «один ко многим» (1:М), «многие к одному» (М:1) или «многие ко многим» (М:М), обозначаемые соответст-

венно: A:=>B,A<=:B.

Например, пусть преподаватель ведет несколько предметов, а каждый предмет может вестись несколькими преподавателями, тогда имеет место зависимость ФИО => Предмет. Так, из таблицы, приведенной на рис 4.1 видно, что преподаватель Иванов И.М. ведет занятия по двум предметам, а дисциплина СУБД - читается двумя преподавателями: Ивановым И.М. и Петровым М.И.

Замечание. В общем случае между двумя атрибутами одного отношения могут существовать зависимости: 1:1, 1:M, M:1 и M:M. Поскольку зависимой». между атрибутами является причиной аномалий, стараются расчленить отношения с зависимостями атрибутов на несколько отношений. В результате образуется совокупность связанных отношений (таблиц) со связями вида 1:1, 1: M. M: 1 и M:M (подраздел 3.3). Связи между таблицами отражают зависимости между атрибутами различных отношений.

Взаимно независимые атрибуты. Два или более атрибута называются взаимно независимыми, если ни один из этих атрибутов не является функционально зависимым от других атрибутов.

В случае двух атрибутов отсутствие зависимости атрибута A от атрибута B можно обозначить так: A^—>B. Случай, когда A-.—>B и B-.->A, можно обо значить A-.=B.

Выявление зависимостей между атрибутами

Выявление зависимостей между атрибутами необходимо для выполнения проектирования БД методом нормальных форм, рассматриваемого далее.

Основной способ определения наличия функциональных зависимостей внимательный анализ семантики атрибутов. Для каждого отношения существует, но не всегда, определенное множество функциональных зависимостей между атрибутами. Причем если в некотором отношении существует одна или несколько функциональных зависимостей, можно вывести другие функциональные зависимости, существующие в этом отношении.

Пример. Пусть задано отношение R со схемой R(A1, A2, A3) и числовыми значениями, приведенными в следующей таблице:

A1	A2	A3
12	21	34
17	21	34
11	24	33
13	25	31
15	23	35
14	22	32

Априори известно, что в ***R*** существуют функциональные зависимости:
A1 - > A2 и A2-> A3.

Анализируя это отношение, можно увидеть, что в нем существуют еще зависимости:

A1->A3, A1A2->A3,A1A2A3->A1A2,

A1A2 - > A2A3 ит. п.

Но то же время в отношении нет других функциональных зависимостей, что во введенных нами обозначениях можно отразить следующим образом:

A2.-->A1, A3.-->A1 и т. д.

Отсутствие зависимости A1 от A2 объясняется тем, что одному и тому же значению атрибута A2 (21) соответствуют разные значения атрибута A1 (12 и 17). Другими словами, имеет место многозначность, а не функциональность.

Перечислив все существующие функциональные зависимости в отношении R, получим полное множество функциональных зависимостей, которое обозначим F^+.

Таким образом, для последнего примера исходное множество F = (A1--> A2, A2 > A3), а полное множество F^+ = (Al-> A2, A2н> A3, A1-> A3, A1A2—> A3, A2A3-» A1A2, A1A2- A2A3,...).

Для построения F^+ из F необходимо знать ряд правил (или аксиом) вывода одних функциональных зависимостей из других.

Существует 8 основных аксиом вывода: рефлексивности, пополнения, транзитивности, расширения, продолжения, псевдотранзитивности, объединения и декомпозиции. Перечисленные аксиомы обеспечивают получение тех ФЗ, т. е. их совокупность применительно к процедуре вывода можно считать «функционально полной». Содержание аксиом и соответствующие примеры приведены в Приложении 1.

Ввыявим зависимости между атрибутами отношения ПРЕПОДАВАТЕЛЬ, приведенного на рис. 4.4. При этом учтем следующее условие, которое выполняется в данном отношении: один преподаватель в одной группе может доводить один вид занятий (лекции или практические занятия).

В результате анализа отношения получаем зависимости между атрибутами, показанные на рис. 4.5.

К выделению этих ФЗ для рассматриваемого примера приводят следующее соображения.

Фамилия, имя и отчество у преподавателей факультета уникальны. Каждому преподавателю однозначно соответствует его стаж, т. е. имеет место функциональная зависимость ФИО->Стаж. Обратное утверждение неверно, так как одинаковый стаж может быть у разных преподавателей.

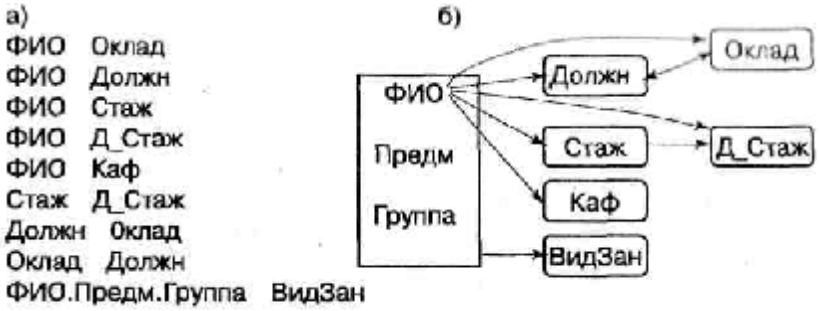


Рис. 5.5. Зависимости между атрибутами

Каждый преподаватель имеет определенную добавку за стаж, т. е. имеет место функциональная зависимость ФИО—>Д__Стаж, но обратная функциональная зависимость отсутствует, так как одну и ту же надбавку могут иметь несколько преподавателей.

Каждый преподаватель имеет определенную должность (преп., ст.преп., доцент, профессор), но одну и ту же должность могут иметь несколько преподавателей, т. е. имеет место функциональная зависимость ФИО-»Должн, а обратная функциональная зависимость отсутствует.

Каждый преподаватель является сотрудником одной и только одной кафедры. Поэтому функциональная зависимость ФИО—>Каф имеет место. С другой стороны, на каждой кафедре много преподавателей, поэтому обратной функциональной зависимости нет.

Каждому преподавателю соответствует конкретный оклад, который одинаков для всех педагогов с одинаковыми должностями, что учитывается зависимостями ФИО-»Оклад и Должн —Оклад. Нет одинаковых окладов для разных должностей, поэтому имеет место функциональная зависимость Оклад—>Должн.

Один и тот же преподаватель в одной группе по разным предметам может проводить разные виды занятий. Определение вида занятий, которые проводит преподаватель, невозможно без указания предмета и группы, поэтому имеет место функциональная зависимость ФИО, Предм, Группа—>-ВидЗан. Действительно, Петров М.И. в 256-й группе читает лекции и проводит практические занятия. Но лекции он читает по СУБД, а практику проводит по Паскалю.

Нами не были выделены зависимости между атрибутами ФИО, Предм и Группа, поскольку они образуют составной ключ и не учитываются в процессе нормализации исходного отношения.

После того, как выделены все функциональные зависимости, следует проверить их согласованность с данными исходного отношения ПРЕПОДАВАТЕЛЬ (рис. 4.4).

Например, Должн.-«преп» и **Оклад-500** всегда соответствуют друг другу во всех кортежах, т. е. подтверждается функциональная зависимость Долж->Оклад. Так же следует верифицировать и остальные функциональные зависимости, не забывая об ограниченности имеющихся в отношении данных.

Нормальные формы

Процесс проектирования БД с использованием метода нормальных форм

является итерационным и заключается в последовательном переводе отношений из первой нормальной формы в нормальные формы более высокого порядка по определенным правилам. Каждая следующая нормальная форма ограничивает определенный тип функциональных зависимостей, устраняет соответствующие аномалии при выполнении операций над отношениями БД и сохраняет свойства предшествующих нормальных форм.

Выделяют следующую последовательность нормальных форм:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);
- третья нормальная форма (ЗНФ);
- усиленная третья нормальная форма, или нормальная форма Бойса - Кодда(БКНФ);
- четвертая нормальная форма (4НФ);
- пятая нормальная форма (5НФ).

Первая нормальная форма. Отношение находится в 1НФ, если все его атрибуты являются простыми (имеют единственное значение). Исходное отношение строится таким образом, чтобы оно было в 1НФ.

Перевод отношения в следующую нормальную форму осуществляется методом «декомпозиции без потерь». Такая декомпозиция должна обеспечить то, что запросы (выборка данных по условию) к исходному отношению и к отношениям, получаемым в результате декомпозиции, дадут одинаковый результат.

Основной операцией метода является операция проекции. Поясним ее на примере. Предположим, что в отношении R(A,B,C,D,E,...) устранение функциональной зависимости C—>D позволит перевести его в следующую нормальную форму. Для решения этой задачи выполним декомпозицию отношения R на два новых отношения R1(A,B,C,E,...) и R2(C,D). Отношение R2 является проекцией отношения R на атрибуты C и D.

Исходное отношение ПРЕПОДАВАТЕЛЬ, используемое для иллюстрации метода, имеет составной ключ ФИО. Предм. Группа и находится в 1НФ, поскольку все его атрибуты простые.

В этом отношении в соответствии с рис. 4.5 б можно выделить частичную зависимость атрибутов Стаж, Д_Стаж, Каф, Должн, Оклад от ключа - указанные атрибуты находятся в функциональной зависимости от атрибута ФИО, являющегося частью составного ключа.

Эта частичная зависимость от ключа приводит к следующему:

1. В отношении присутствует явное и неявное избыточное дублировать данных, например:

• повторение сведений о стаже, должности и окладе преподавателей, про водящих занятия в нескольких группах и/или по разным предметам;

• повторение сведений об окладах для одной и той же должности или в надбавках за одинаковый стаж.

2. Следствием избыточного дублирования данных является проблема их редактирования. Например, изменение должности у преподавателя Иванова И.М. потребует просмотра всех кортежей отношения и внесения изменении в те из них, которые содержат сведения о данном преподавателе.

Часть избыточности устраняется при переводе отношения в 2НФ.

Вторая нормальная форма. Отношение находится в 2НФ, если оно находится в 1НФ и каждый неключевой атрибут функционально полно зависит от первичного ключа (составного).

Для устранения частичной зависимости и перевода отношения в 2НФ не обходимо, используя операцию проекции, разложить его на несколько отношений следующим образом:

• построить проекцию без атрибутов, находящихся в частичной функциональной зависимости от первичного ключа;

• построить проекции на части составного первичного ключа и атрибуты зависящие от этих частей.

В результате получим два отношения R1 и R2 в 2НФ (рис. 5.6).

В отношении R1 первичный ключ является составным и состоит из атрибутов ФИО. Предм. Группа. Напомним, что данный ключ в отношении RI получен в предположении, что каждый преподаватель в одной группе по од ному предмету может либо читать лекции, либо проводить практические занятия. В отношении R2 ключ ФИО.

Исследование отношений R1 и R2 показывает, что переход к 2НФ позволил исключить явную избыточность данных в таблице R2 - повторение строк со сведениями о преподавателях. В R2 по-прежнему имеет место неявное дублирование данных.

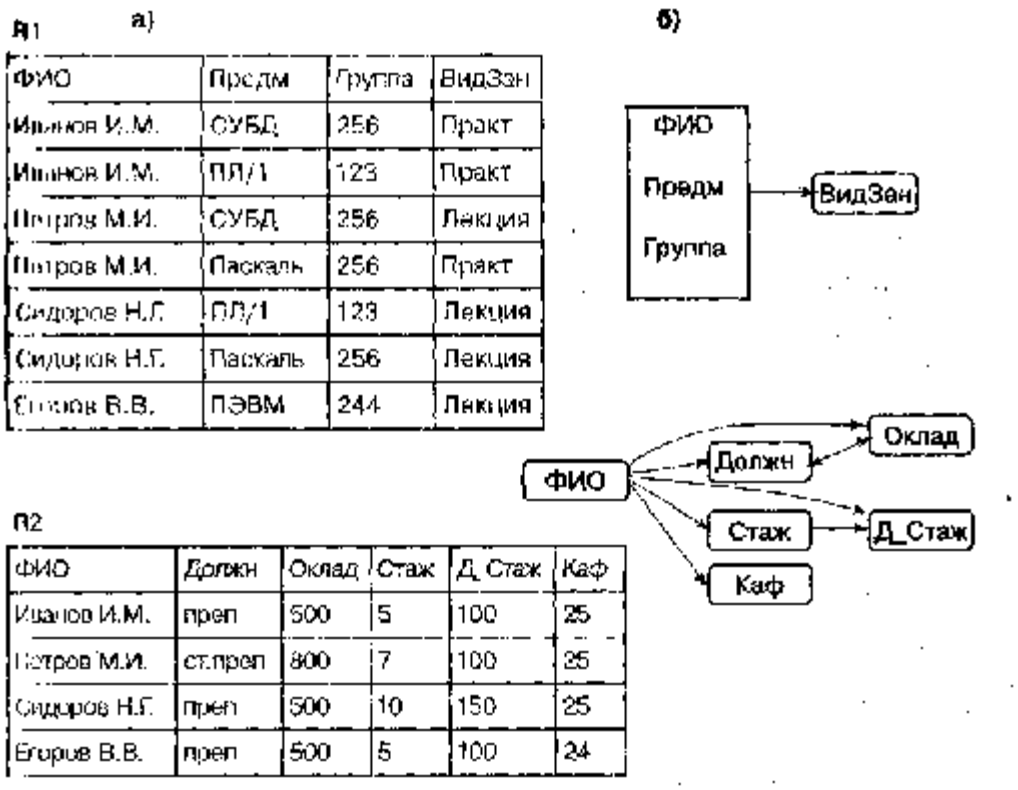
Для дальнейшего совершенствования отношения необходимо преобразовать его в ЗНФ.

Третья нормальная форма.

Определение 1. Отношение находится в ЗНФ, если оно находится в 2НФ и каждый неключевой атрибут нетранзитивно зависит от первичного ключ;!

Существует и альтернативное определение.

Определение 2. Отношение находится в ЗНФ в том и только в том случае если все неключевые атрибуты отношения взаимно независимы и Полностью зависят от первичного ключа.



ФИО	Предм	Группа	ВидЗан
Иванов И.М.	СУБД	256	Практ
Иванов И.М.	ПЛ/1	123	Практ
Петров М.И.	СУБД	256	Лекция
Петров М.И.	Паскаль	256	Практ
Сидоров Н.Г.	ПЛ/1	123	Лекция
Сидоров Н.Г.	Паскаль	256	Лекция
Егоров В.В.	ПЭВМ	244	Лекция

ФИО	Должн	Оклад	Стаж	Д_Стаж	Каф
Иванов И.М.	преп	500	5	100	25
Петров М.И.	ст.преп	800	7	100	25
Сидоров Н.Г.	преп	500	10	150	25
Егоров В.В.	преп	500	5	100	24

Рис. 4.6. Отношения БД в 2НФ

Доказать справедливость этого утверждения несложно. Действительно, то, что неключевые атрибуты полностью зависят от первичного ключа, означает, что данное отношение находится в форме 2НФ. Взаимная независимость атрибутов (определение приведено выше) означает отсутствие всякой зависимости между атрибутами отношения, в том числе и транзитивной зависимости между ними. Таким образом, второе определение ЗНФ сводится к первому определению.

Если в отношении R1 транзитивные зависимости отсутствуют, то в отношении R2 они есть:

ФИО->Должн->Оклад, Ф И 0-> Оклад-> Должн, ФИО-» Стаж-* Д_Стаж

Транзитивные зависимости также порождают избыточное дублирование информации в отношении. Устраним их. Для этого используя операцию проекции на атрибуты, являющиеся причиной транзитивных зависимостей, преобразуем отношение R2, получив при этом отношения К3, *R4* и R5, каждое из которых находится в ЗНФ (рис. 5.7а). Графически эти отношения представлены на рис. 4.76. Заметим, что отношение R2 можно преобразовать подругому, а именно: в отношении R3 вместо атрибута Должн взят атрибут Оклад.

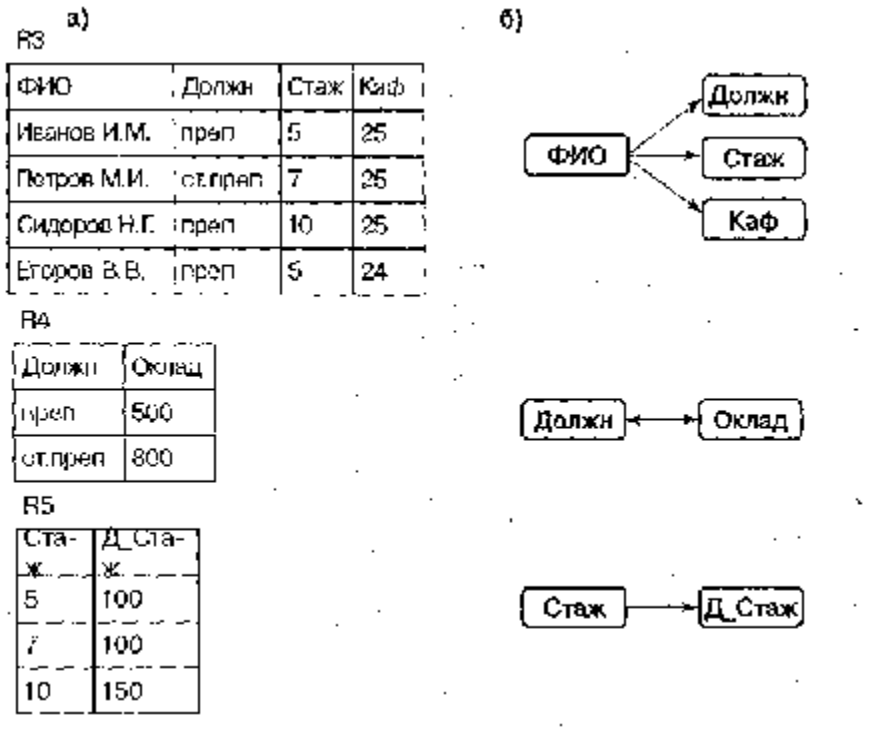


Рис. 4.7. Отношения БД в ЗНФ

На практике построение ЗНФ схем отношений в большинстве случаев является достаточным и приведением к ним процесс проектирования реляционной БД заканчивается. Действительно, приведение отношений к ЗНФ в нашем примере, привело к устранению избыточного дублирования.

Если в отношении имеется зависимость атрибутов составного ключа от неключевых атрибутов, то необходимо перейти к усиленной ЗНФ.

Усиленная ЗНФ или нормальная форма Бойса - Кодда (БКНФ).

Отношение находится в БКНФ, если оно находится в ЗНФ и в нем отсутствуют зависимости ключей (атрибутов составного ключа) от неключевых атрибутов.

У нас подобной зависимости нет, поэтому процесс проектирования на этом заканчивается. Результатом проектирования является БД, состоящая из следующих таблиц: Rl, R3, R4, R5. В полученной БД имеет место необходимое дублирование данных, но отсутствует избыточное.

Четвертая нормальная форма.

Рассмотрим пример нового отношения ПРОЕКТЫ, схема которого выглядит следующим образом: ПРОЕКТЫ (Номер_проекта, Код_сотрудника, Задание_сотрудника). Первичным ключом отношения является вся совокупность атрибутов: Номер_проекта, Код_сотрудника и Задание_сотрудника.

В отношении содержатся номера проектов, для каждого проекта - список кодов сотрудников-исполнителей, а также список заданий, предусмотренных каждым проектом. Сотрудники могут участвовать в нескольких проектах, и разные проекты могут содержать одинаковые задания. Предполагается, что каждый сотрудник, участвующий в некотором проекте, выполняет все задания по этому проекту (предположение не всегда справедливо, но желательно для нашего примера).

При такой постановке вопроса единственным возможным ключом отно-

шения является составной атрибут Номер_проекта, Код_сотрудника, Задание_сотрудника. Он, естественно, и стал первичным ключом отношения. Отсюда следует, что отношение ПРОЕКТЫ, находится в форме БКНФ.

Пусть исходная информация в этом отношении выглядит следующим образом:

ПРОЕКТЫ

Номер_проекта	Код_сотрудника	Задание_сотрудника
001	05	1
001	05	2
001	05	3
004	02'	1
004	02	2
004	03	1
004	03	2
004	05	1
004	05	2
007	06	1

Главный недостаток отношения ПРОЕКТЫ состоит в том, что при подключении/отстранении от проекта некоторого сотрудника приходится добавлять/исключать из отношения столько кортежей, сколько заданий имеется в проекте. Внесение или исключение и отношении одного факта о некотором сотруднике требует серии элементарных операций из-за дублирования значений в кортежах.

Отсюда возникают вопросы: зачем хранить в кортежах повторяющиеся значения кодов сотрудников? Нужно ли перечислять все задания по каждом) проекту, да еще для каждого сотрудника-исполнителя этого проекта? Нельзя ли информацию о привязке заданий к проектам поместить в отдельную таблицу и исключить повторения в основной таблице?

Заметим, что косвенный признак аномалии, как и ранее, - дублирование информации в таблице. Выскажем предположение, что причиной аномалии является наличие некоторой зависимости между атрибутами отношения (как увидим далее - многозначной зависимости).

Действительно, в отношении ПРОЕКТЫ существуют следующие две многозначные зависимости:

Номер_проекта=>Код_сотрудника Номер_проекта=>Задание_сотрудника

В произвольном отношении R(A, B, C) может одновременно существовать многозначная зависимость A=»B и A= >C. Это обстоятельство обозначим как A=>B |C.

Дальнейшая нормализация отношений, схожих с отношением Проекты. основывается на следующей теореме.

Теорема Фейджина (Fagin R.). Отношение R(A, B, C) молено спроецировать без потерь в отношения R1(A, B) и R2(A, C) в том и только том случае, когда существует зависимость A=>B |C.

Под проецированием без потерь здесь понимается такой способ декомпозиции отношения, при котором исходное отношение полностью и без избыточности восстанавливается путем *естественного соединения* полученных отношений (см. подраздел 3.6).

Поясним проецирование без потерь на примере.

Пусть имеется простейшее отношение R(A, B, C), имеющее вид:

R

А	**в**	**С**
К	**15**	**1**
К	**15**	**2**
Л	**10**	**1**
м	**20**	**1**
м	**20**	**2**
м	**20**	**3**

Построим проекции KI и R2 на атрибуты A, B и A, C соответственно. Они будут выглядеть так:

R1

А	**в**
К	15
л	10
м	20

R2

А	**с**
К	1
К	2
Л	1
м	1
м	2
м	3

Результатом *операции соединения* бинарных отношений R1(A, B) и R2(A, C) по атрибуту A является тернарное отношение с атрибутами A, B и C, кортежи которого получаются путем связывания отношений R1 и R2 по типу 1:M на основе совпадения значений атрибута A (подраздел 3.3).

Так, связывание кортежей (к, 15)и{(к, 1), (к, 2)} дает кортежи {(к, 15,1),(к, 15,2)}.

Нетрудно видеть, что связывание R1(A, B) и R2(A, C) в точности порождает исходное отношение R(A, B, C). В отношении R нет лишних кортежей, нет и потерь.

Определение четвертой нормальной формы. Отношение R находится в чет-«гртой нормальной форме (4НФ) в том и только в том случае, когда существует многозначная зависимость A=>B, а все остальные атрибуты R функционально зависят от A.

Приведенное выше отношение ПРОЕКТЫ можно представить в виде двух отношений: ПРОЕКТЫ-СОТРУДНИКИ и ПРОЕКТЫ-ЗАДАНИЯ. Структура этих отношений и содержимое соответствующих таблиц выглядит следующим образом:

ПРОЕКТЫ-СОТРУДНИКИ (Номер_проекта, Код_сотрудника).

Первичный ключ отношения: Номер_проекта, Код_сотрудника.

ПРОЕКТЫ-СОТРУДНИКИ

Номер_проекта	Код_сотрудника
001	05
004	02
004	1
004	05
007	06

ПРОЕКТЫ-ЗАДАНИЯ (Номер приема, Задание сотрудника).

Первичный ключ отношения: Помер_ проекта, Задание_сотрудника.

ПРОЕКТЫ-ЗАДАНИЯ

Номер_проекта	Задание_сотрудника
001	1
001	2
001	3
004	1
004	2
007	1

Как легко увидеть, оба этих отношения находятся в 4НФ и свободны от замеченных недостатков. Дублирование значений атрибутов кодов сотрудников пропало. Попробуйте самостоятельно соединить эти отношения и убедиться в том, что в точности получится отношение ПРОЕКТЫ.

В общем случае не всякое отношение можно восстановить к исходному. В нашем случае восстановление возможно потому, что каждый сотрудник, участвующий в некотором проекте, выполнял все задания по этому проекту

(именно это укладывается в принцип 1:M соединения отношений). Сами же сотрудники участвовали в нескольких проектах, и разные проекты могли содержать одинаковые задания.

Пятая нормальная форма.

Результатом нормализации всех предыдущих схем отношений были два новых отношения. Иногда это сделать не удается, либо получаемые отношения заведомо имеют нежелательные свойства. В этом случае выполняют декомпозицию исходного отношения на отношения, количество которых превышает два.

Рассмотрим отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ, которое имеет заголовок СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ (Код_сотрудника, Код_ отдела, Номер_проекта). Первичный ключ отношения включает все атрибуты: Код_сотрудника, Код_отдела и Номер_проекта. Пусть в этом отношении один сотрудник может работать в нескольких отделах, причем в каждом отделе он может принимать участие в нескольких проектах. В одном отделе могут работать несколько сотрудников, но каждый проект выполняет только один сотрудник. Функциональных и многозначных зависимостей между атрибутами не существует.

Это отношение является частью базы данных вымышленного научного подразделения НИИЧАВО - Научно-Исследовательского Института ЧАродей-ства и ВОлшебства из повести А. и Б. Стругацких «Понедельник начинается в субботу». Коды отделов здесь обозначают: АД - Администрация,

СОТРУДНИКИ ОТДЕЛЫ ПРОЕКТЫ

Код сотрудника	Код_отдела	Номер_проекта
01	РД	036
02	АД	004
03	УП	004
04	АД	019
05	ЛС	001
05	ЛС	004
06	УП	007
08	ВЦ	013
09	ВЦ	014
10	**сж**	013

ВЦ - Вычислительный центр, ЛС - Линейного счастья, РД - Родильный дом, СЖ. - Смысла жизни, УП - Универсальных превращений.

Исходя из структуры отношения СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ можно заключить, что оно находится в форме 4НФ. Тем не менее в отношении могут быть аномалии, связанные с возможностью повторения значений атрибутов в нескольких кортежах. Например, то, что сотрудник может работать в нескольких отделах, при увольнении сотрудника требует отыскания и последующего удаления из исходной таблицы нескольких записей.

Введем определение *зависимости соединения.* Отношение R(X, Y, ... , Z) удовлетворяет *зависимости соединения,* которую обозначим как *(X, Y,..., Z), в том и только в том случае, если R восстанавливается без потерь путем со-

единения своих проекций на X, Y, ... , Z. Зависимость соединения является обобщением функциональной и многозначной зависимостей.

Определение пятой нормальной формы. Отношение R находится в 5НФ (или нормальной форме проекции-соединения - PJ/NF) в том и только том случае, когда любая зависимость соединения в R следует из существования некоторого возможного ключа в R.

Образуем составные атрибуты отношения СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ:

СО={ Код_сотрудника, Код_отдела } СП={ Код_сотрудника, Номер_проекта} ОП={ Код_отдела, Номер_проекта}.

Покажем, что если отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ спроецировать на составные атрибуты СО, СП и ОП, то соединение этих проекций дает исходное отношение. Это значит, что в нашем отношении существовала зависимость соединения *(СО, СП, ОН). Проекции на составные атрибуты назовем соответственно СОТРУДНИКИ-ОТДЕЛЫ, СОТРУДНИКИ-ПРОЕКТЫ и ОТДЕЛЫ-ПРОЕКТЫ.

Ранее мы выполняли соединение двух проекций и сразу получали искомый результат. Для восстановлении отношения из трех (или нескольких) проекций надо получить все попарные соединения (так как информация о том, какое из них «лучше», отсутствует), над которыми затем выполнить операцию пересечения множеств. Проверим, так ли это.

СОТРУДНИКИ-ОТДЕЛЫ

Код_ сотрудника	Код_ отдела
01	РД
02	АД
03	УП
04	АД
05	ЛС
05	ЛС
06	УП
08	ВЦ
09	ВЦ
10	**СЖ**

СОТРУДНИКИ-ПРОЕКТЫ

Код_ сотрудника	Номер_ проекта
01	036
02	004
03	004
04	019
05	001
05	004
06	007
08	013
09	014
10	013

ОТДЕЛЫ-ПРОЕКТЫ

Код_ отдела	Номер
АД	004
АД	019
ВЦ	013
ВЦ	014
ЛС	001
ЛС	004
СЖ	013
РД	036
УП	004
УП	007

Получим попарные соединения трех приведенных выше отношений, которые будут иметь вид:

*(СО, СП)

Код_ сотрудника	Код_ отде-	Номер
01	РД	036
02	АД	004
03	УП	004
04	АД	019
05	ЛС	001
(.)')	**лс**	004
06	УП	007
08	ВЦ	013
09	ВЦ	014
10	**еж**	013

*(СП, ОП)

Код_ сотрудника	Код_ отде-	Номер
01	РД	036
02	АД	004
02	АД	019
03	УП	004
03	УП	007
04	АД	004
04	АД	019
05	ЛС	001
05	ЛС	004
06	УП	004
06	УП	007
08	ВЦ	013
08	ВЦ	014
09	ВЦ	013
09	ВЦ	014
10	**еж**	013

*(СО, ОП) v

Код_ сотрудника	Код_ отдела	Номер
01	РД	036
02	АД	004
02	ЛС	004
02	УП	004
03	АД	004
03	ЛС	004
03	УП	004
04	АД	019
05	ЛС	001
05	АД	004
05	ЛС	004
05	УП	004
06	УП	004
06	УП	007
08	ВЦ	013
08	ВЦ	014
09	ВЦ	013
09	ВЦ	014
10	**еж**	013

Нетрудно увидеть, что пересечение всех отношений дает исходное отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ.

Замечание.

Существуют и другие способы восстановления исходного отношения и . его проекций. Так, для восстановления отношения СОТРУД11И КИ-ОТД1: ЛЫ-ПРОЕКТЫ можно соединить отношения СОТРУДНИКИ-ОТДЕЛЫ и СОТРУДНИКИ-ПРОЕКТЫ по атрибуту Код_сотрудника, после чего полу ценное отношение соединить с отношением ОТДЕЛЫ-ПРОЕКТЫ по составному атрибуту (Код_отдела, Номер_проекта).

Отношения СОТРУДНИКИ-ОТДЕЛЫ, СОТРУДНИКИ-ПРОЕКТЫ и ОТ ДЕЛЫ-ПРОЕКТЫ находятся в 5НФ. Эта форма является последней из известных. Условия ее получения довольно нетривиальны и поэтому она почти не используется на практике. Более того, она имеет определенные недостатки!

Предположим, необходимо узнать, где и какие проекты исполняет сотрудник с кодом 02. Для этого в отношении СОТРУДНИКИ-ОТДЕЛЫ найдем, что сотрудник с кодом 02 работает в отделе АД, а из отношения ОТДЕЛЫ-ПРОЕКТЫ найдем, что в отделе АД выполняются проекты 004 и 019. А это значит, что сотрудник 02 должен выполнять проекты 004 и 019. Увы, информация о том, что сотрудник с кодом 02 выполняет проект 019, ошибочна.

Такие противоречия можно устранить только путем совместного рассмотрения всех проекций основного отношения. Именно поэтому после соединения проекций и выполнялась операция их пересечения. Безусловно, это - недостаток. Отметим, что наличие недостатков не требует обязательного отказа от определенных видов нормальных форм. Надо учитывать недостатки и условия их проявления. В некоторых постановках задач недостатки не проявляются.

На практике обычно ограничиваются структурой БД, соответствующей ЗНФ или БКНФ. Поэтому процесс нормализации отношений методом нормальных форм предполагает последовательное удаление из исходного отношения следующих межатрибутных зависимостей:

- частичных зависимостей неключевых атрибутов от ключа (удовлетворение требований 2НФ);
- транзитивных зависимостей неключевых атрибутов от ключа (удовлетворение требований ЗНФ);
- зависимости ключей (атрибутов составных ключей) от неключевых атрибутов (удовлетворение требований БКНФ).

Кроме метода нормальных форм Кодда, используемого для проектирования небольших БД, применяют и другие методы, например, метод ER-диаграмм (метод «Сущность-связь»). Этот метод используется при проектировании больших БД, на нем основан ряд средств проектирования БД. Метод ER-диаграмм рассматривается в следующем разделе.

На последнем этапе метода ER-диаграмм отношения, полученные в результате проектирования, проверяются на принадлежность их к БКНФ. Этот этап может выполняться уже с использованием метода нормальных форм. После завершения проектирования создается БД с помощью СУБД.

4.3. Рекомендации по разработке структур

Какими должны быть таблицы сущностей

Основное правило при создании таблиц сущностей - это «каждой сущности -отдельную таблицу».

Поля таблиц сущностей могут быть двух видов: ключевые и неключевые. Введение ключей в таблице практически во всех реляционных СУБД позволяет обеспечить уникальность значений в записях таблицы по ключу, ускорять обработку записей таблицы, а также выполнить автоматическую сортировку записей по значениям в ключевых полях.

Обычно достаточно определения простого ключа, реже - вводят составной ключ. Таблицей с составным ключом может быть, например, таблица хранения списка сотрудников (фамилия, имя и отчество), в котором встречаются однофамильцы. В некоторых СУБД пользователям предлагается определить автоматически создаваемое ключевое поле нумерации (в Access - это поле типа «счетчик»), которое упрощает решение проблемы уникальности записей таблицы.

Иногда в таблицах сущностей имеются поля описания свойств или характеристик объектов. Если в таблице есть значительное число повторений по этим полям и па информация имеет существенный объем, то .лучше их выделить в отдельную таблицу (придерживаясь правила: «каждой сущности - отдельную таблицу»). Тем не менее, следует образовать дополнительную таблицу, если свойства взаимосвязаны.

В более общем виде последние рекомендации можно сформулировать так: информацию о сущностях следует представить таким образом, чтобы неключевые поля в таблицах были взаимно независимыми и полностью зависели от ключа (см. определение третьей нормальной формы).

В процессе обработки таблиц сущностей надо иметь в виду следующее. Новую сущность легко добавить и изменить, но при удалении следует уничтожить все ссылки на нее из таблиц связей, иначе таблицы связей будут некорректными. Многие современные СУБД блокируют некорректные действия в подобных операциях

Организация связи сущностей

Записи таблицы связей предназначены для отображения связей между сущностями, информация о которых находится в соответствующих таблицах сущностей.

Обычно одна таблица связей описывает взаимосвязь двух сущностей. Поскольку таблицы сущностей в простейшем случае имеют по одному ключевому полю, то таблица связей двух таблиц для обеспечения уникальности записей о связях должна иметь дна ключа. Можно создать таблицу связей, как и таблицу объектов, и без ключей, но тогда функции контроля за уникальностью записей ложатся на пользователя.

Более сложные связи (не бинарные) следует сводить к бинарным. Для описания взаимосвязей N объектов требуется N-1 таблиц связей. Транзитивных связей не должно быть. Избыток связей приводит к противоречиям (см. пример отношений СОТРУДНИКИ-ОТДЕЛЫ, СОТРУДНИКИ-ПРОЕКТЫ,

ОТДЕЛЫ-ПРОЕКТЫ предыдущего подраздела).

При работе с таблицами связей следует иметь в виду, что любая запись таблицы связей легко может быть удалена, поскольку сущности некоторое вы мя могут обойтись, и без связей. При добавлении или изменении содержимомзаписей таблицы надо контролировать правильность ссылок на существующие объекты, так как связь без объектов существовать не может. Большинство современных СУБД контролируют правильность ссылок на объекты.

4.4. Обеспечение целостности

Под *целостностью* понимают свойство базы данных, означающее, что она содержит полную, непротиворечивую и адекватно отражающую предметную область информацию.

Различают *физическую алогическую* целостность. Физическая целостность означает наличие физического доступа к данным и то, что данные не утрачены. Логическая целостность означает отсутствие логических ошибок в базе данных, к которым относятся нарушение структуры БД или ее объектов, удаление или изменение установленных связей между объектами и т. д. В дальнейшем речь будем вести о логической целостности.

Поддержание целостности БД включает проверку (контроль) целостности и ее восстановление в случае обнаружения противоречий в базе. Целостное состояние БД задается с помощью *ограничений целостности* в виде условий, которым должны удовлетворять хранимые в базе данные.

Среди ограничений целостности можно выделить два основных типа ограничений: *ограничения значений* атрибутов отношений и *структурные ограничения* на кортежи отношений.

Примером *ограничений значений* атрибутов отношений является требование недопустимости пустых или повторяющихся значений в атрибутах, а также контроль принадлежности значений атрибутов заданному диапазону. Так, в записях отношений о кадрах значения атрибута Дата_рождения не могут превышать значений атрибута Дата_приема.

Наиболее гибким средством реализации контроля значений атрибутов являются *хранимые процедуры* и *триггеры,* имеющиеся в некоторых СУБД.

Структурные ограничения определяют требования *целостности сущностей* и *целостности ссылок.* Каждому экземпляру сущности, представленному в отношении, соответствует только один его кортеж. Требование *целостности сущностей* состоит в том, что любой кортеж отношения должен быть отличим от любого другого кортежа этого отношения, т. е., иными словами, любое отношение должно обладать первичным ключом.

Формулировка требования целостности ссылок тесно связана с понятием *внешнего ключа.* Напомним, что внешние ключи служат для связи отношений (таблиц БД) между собой. При этом атрибут одного отношения (родительского) называется *внешним ключом* данного отношения, если он является *первичным* ключом другого отношения (дочернего). Говорят, что отношение, в котором определен внешний ключ, ссылается на отношение, в котором этот лее атрибут является первичным ключом.

Требование целостности ссылок состоит в том, что для каждого значе-

ния внешнего ключа родительской таблицы должна найтись строка в дочерней таблице с таким же значением первичного ключа. Например, если в отношении R1 (рис. 4.8) содержатся сведения о сотрудниках кафедры, а атрибут этого отношения Должн является первичным ключом отношения R2, то в этом отношении для каждой должности из R1 должна быть строка с соответствующим ей окладом.

Во многих современных СУБД имеются средства обеспечения контроля целостности БД.

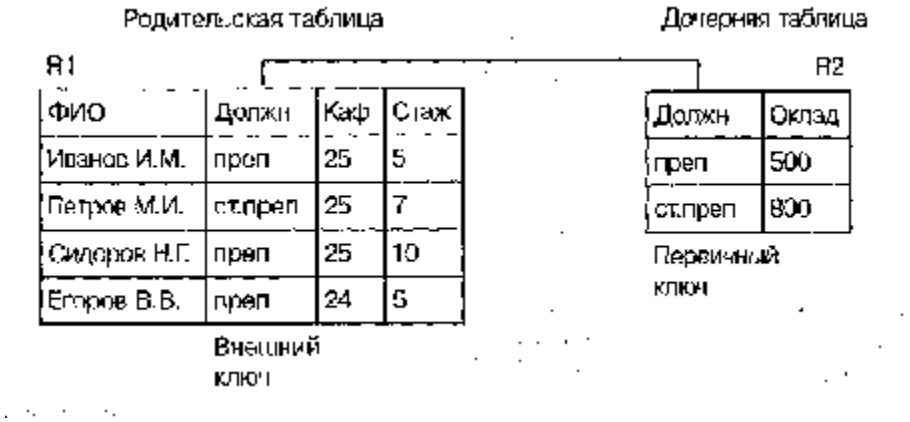


Рис. 4.8. Связь отношений с помощью внешнего ключа

ТЕМА 5. МЕТОД СУЩНОСТЬ-СВЯЗЬ

Метод сущность-связь называют также методом «ER-диаграмм»: во-первых, ER аббревиатура от слов *Essence (сущность)* и *Relation (связь)*, во-вторых, метод основан на использовании диаграмм, называемых соответственно диаграммами ER-экземпляров и диаграммами ER-типа.

5.1. Основные понятия метода

Основными понятиями метода сущность-связь являются следующие:

- сущность
- атрибут сущности,
- ключ сущности,
- связь между сущностями,
- степень связи,
- класс принадлежности экземпляров сущности,
- диаграммы ER-экземпляров,
- диаграммы ER-типа.

Сущность представляет собой объект, информация о котором хранится в 1НФ. Экземпляры сущности отличаются друг от друга и однозначно идентифицируются.

Атрибут представляет собой свойство сущности. Это понятие аналогично понятию атрибута в отношении. Так, атрибутами сущности ПРЕПОДАВАТЕЛЬ может быть его Фамилия, Должность, Стаж (преподавательский) и т. д.

Ключ сущности - атрибут или набор атрибутов, используемый для иден-

тификации экземпляра сущности. Как видно из определения, понятие ключа сущности аналогично понятию ключа отношения.

Связь двух или более сущностей - предполагает зависимость между атрибутами и этих сущностей. Название связи обычно представляется *глаголом*. Примерами связей между сущностями являются следующие: ПРЕПОДАВАТЕЛЬ *ВЕДЕТ* ДИСЦИПЛИНУ (Иванов ВЕДЕТ «Базы данных»),ПРЕПОДАВАТЕЛЬ *ПРЕПОДАЕТ-В*-ГРУППЕ (Иванов ПРЕПОДАЕТ-В 256 группе), ПРЕПОДАВАЛТЕЛЬ *РАБОТАЕТ-НА* КАФЕДРЕ (Иванов РАБОТАЕТ-НА25 кафедре).

Приведенные определения сущности и связи не полностью формализованы, но приемлемы для практики. Следует иметь в виду, что в результате проектирования могут быть получены несколько вариантов одной БД. Так, два разных проектировщика, рассматривая одну и ту же проблему с разных точек зрения, могут получить различные наборы сущностей и связей. При этом оба варианта могут быть рабочими, а выбор лучшею из них будет результатом личных предпочтений.

С целью повышения наглядности и удобства проектирования для представления сущностей, экземпляров сущностей и связей между ними используются следующие графические средства:

- диаграммы ER-экземпляров,
- диаграммы ER-типа, или ER-диаграммы.

На рис. 5.1 приведена диаграмма ER-экземпляров для сущностей ПРЕПО ДАВАТЕЛЬ и ДИСЦИПЛИНА со связью *ВЕДЕТ*.

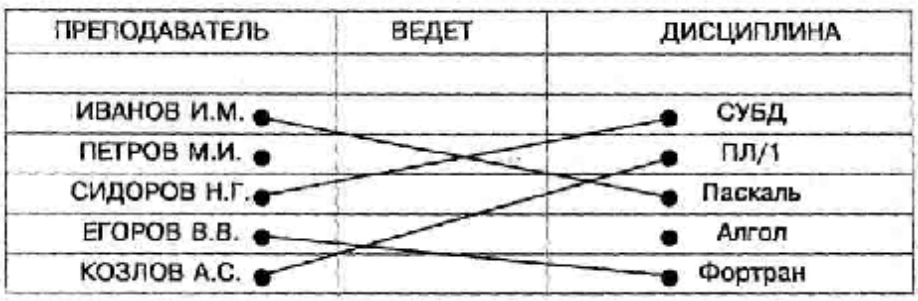


Рис. 5.1. Диаграмма ER-экземпляров

Диаграмма ER-экземпляров показывает, какую конкретно дисциплину (СУБД, ПЛ/1 и т.д.) ведет каждый из преподавателей. На рис. 5.2 представлена диаграмма ER-типа, соответствующая рассмотренной диаграмме ER-экземпляров.

Рис. 6.2. Диаграмма ER-типа

На начальном этапе проектирования БД выделяются атрибуты, составляющие ключи сущностей.

На основе анализа диаграмм ER-типа формируются отношения проекти-

руемой БД. При этом учитывается *степень связи сущностей* и *класс их принадлежности,* которые, в свою очередь, определяются на основе анализа диаграмм ER-экземпляров соответствующих сущностей.

Степень связи является характеристикой связи между сущностями, которая может быть типа: 1:1, 1:M,M:1,M:M.

Класс принадлежности (KII) сущности может быть: *обязательным* и *необязательным*.

Класс принадлежности сущности является *обязательным*,, если все экземпляры этой сущности обязательно участвуют в рассматриваемой связи, в противном случае класс принадлежности сущности является *необязательным.*

Варьируя классом принадлежности сущностей для каждого из названных типов связи, можно получить несколько вариантов диаграмм ER-типа. Рассмотрим примеры некоторых из них.

Пример 1. Связи типа 1:1 и необязательный класс принадлежности.

В приведенной на рис. 5.1 диаграмме степень связи между сущностями 1:1, класс принадлежности обеих сущностей необязательный. Действительно, из рисунка видно следующее:

- каждый преподаватель ведет не более одной дисциплины, а каждая дисциплина ведется не более чем одним преподавателем (степень связи 1:1);
- некоторые преподаватели не ведут ни одной дисциплины и имеются дисциплины, которые не ведет ни один из преподавателей (класс принадлежности обеих сущностей необязательный).

Пример 2. Связи типа 1:1 и обязательный класс принадлежности.

На рис. 5.3 приведены диаграммы, у которых степень связи между сущностями 1:1, а класс принадлежности обеих сущностей обязательный.

В этом случае каждый преподаватель ведет одну дисциплину и каждая дисциплина ведется одним преподавателем.

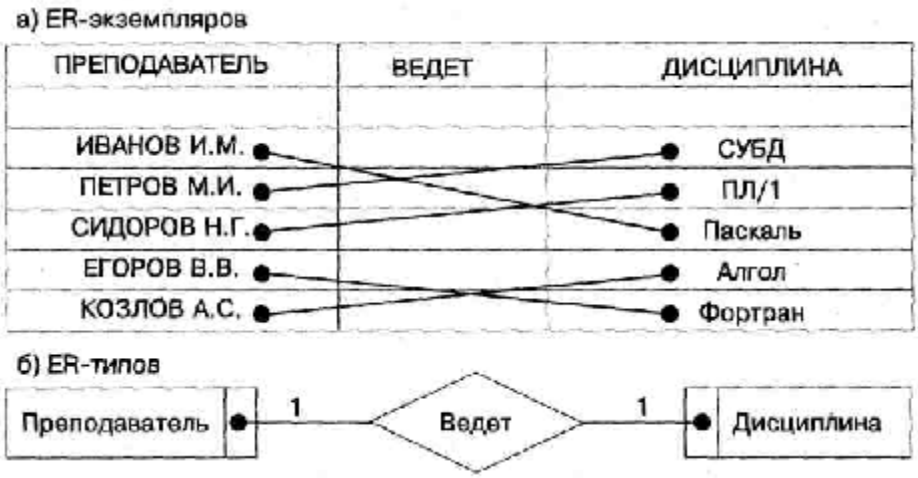


Рис. 5.3. Диаграммы для связи 1:1 и обязательным КП обеих сущностей

Возможны два промежуточных варианта с необязательным классом принадлежности одной из сущностей.

Под каждым блоком, соответствующим некоторой сущности, указывается ее ключ, выделяемый подчеркиванием. Многоточие за ключевыми атри-

бутами означает, что возможны другие атрибуты сущности, но ни один из них не может быть частью ее ключа. Эти атрибуты выявляются после формирования отношений.

На практике степень связи и класс принадлежности сущностей при проектировании БД определяется спецификой предметной области. Рассмотрим примеры вариантов со степенью связи 1:М или М:1.

Пример 3. Связи типа 1.М.

Каждый преподаватель может вести несколько дисциплин, но каждая дисциплина ведется одним преподавателем.

Пример 4. Связи типа М:1.

Каждый преподаватель может вести одну дисциплину, но каждую дисциплину могут вести несколько преподавателей.

Примеры с типом связи 1:М или М:1 могут иметь ряд вариантов, отличающихся классом принадлежности одной или обеих сущностей. Обозначим обязательный класс принадлежности символом «О», а необязательный - символом «Н», тогда варианты для связи типа 1:М условно можно представить как: О-О, О-Н, Н-О, Н-Н. Для связи типа М.i также имеются 4 аналогичных варианта.

Пример 5. Связи типа 1:М вариант Н-О.

Каждый преподаватель может вести несколько дисциплин или ни одной, но каждая дисциплина ведется одним преподавателем (рис. 5.4).

По аналогии легко составить диаграммы и для остальных вариантов.

Пример 6. Связи типа М:М.

Каждый преподаватель может вести несколько дисциплин, а каждая дисциплина может вестись несколькими преподавателями.

Как и в случае других типов связей, для связи типа М:М возможны 4 варианта, отличающиеся классом принадлежности сущностей.

Пример 7. Связи типа М:М и вариант класса принадлежности О-Н.

Допустим, что каждый преподаватель ведет не менее одной дисциплины, а дисциплина может вестись более чем одним преподавателем, есть и такие дисциплины, которые никто не ведет. Соответствующие этому случаю диаграммы приведены на рис.5.5.

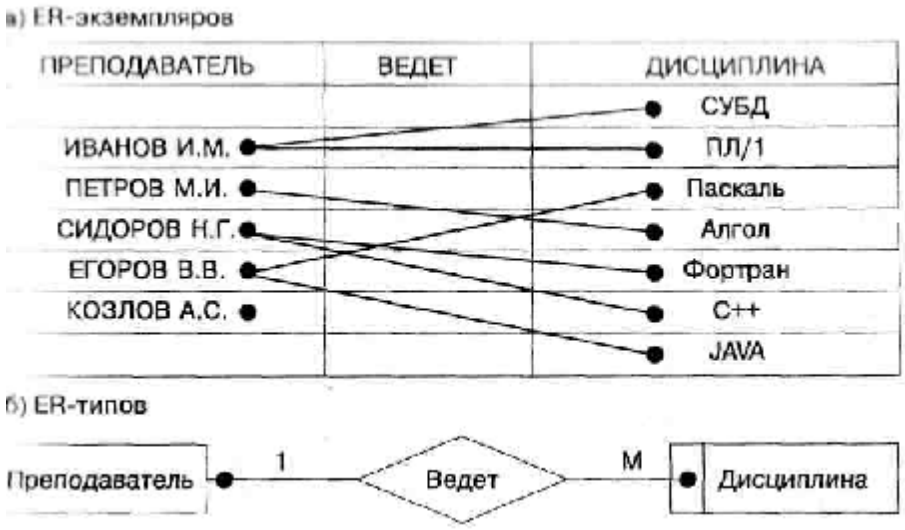


Рис. 5.4. Диаграммы для связи типа 1:M варианта H-O

Выявление сущностей и связей между ними, а также формирование на их основе диаграмм ER-типа выполняется на начальных этапах метода *сущность-связь*. Рассмотрим этапы реализации метода.

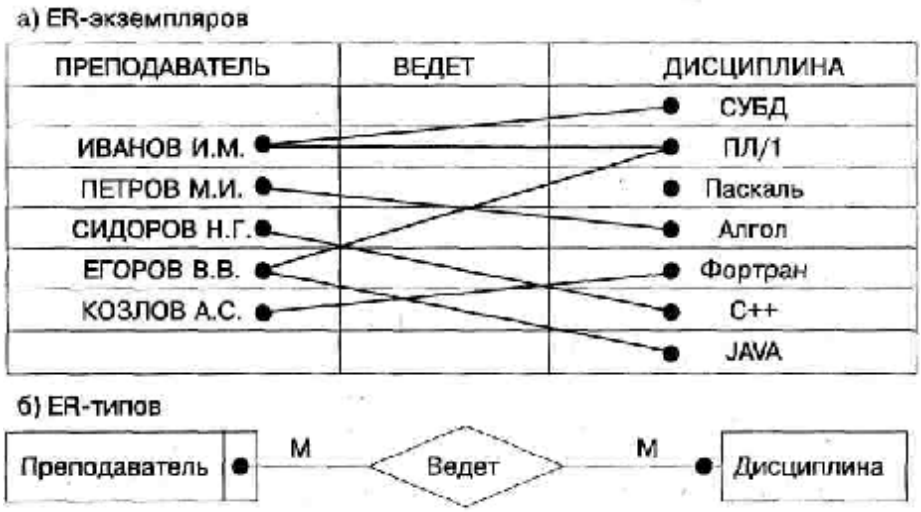


Рис. 5.5. Диаграммы для связи типа M : M и варианта O-H

5.2. Этапы проектирования

Процесс проектирования базы данных является итерационным - допускающим возврат к предыдущим этапам для пересмотра ранее принятых решений и включает следующие этапы:

1. Выделение сущностей и связей между ними.
2. Построение диаграмм ER-типа с учетом всех сущностей и их связей.
3. Формирование набора предварительных отношений с указанием предполагаемого первичного ключа для каждого отношения и использованием диаграмм ER-типа.
4. Добавление неключевых атрибутов в отношения.
5. Приведение предварительных отношений к нормальной форме Бойса Кодда, например, с помощью метода нормальных форм.
6. Пересмотр ER-диаграмм в следующих случаях:

• некоторые отношения не приводятся к нормальной форме Бойса - Кодда,

• некоторым атрибутам не находится логически обоснованных мест в предварительных отношениях.

После преобразования ER-диаграмм осуществляется повторное выполнение предыдущих этапов проектирования (возврат к этапу 1).

Одним из узловых этапов проектирования является этап формирования отношений. Рассмотрим процесс формирования предварительных отношений, составляющих первичный вариант схемы БД.

В рассмотренных выше примерах связь ВЕДЕТ всегда соединяет две сущности и поэтому является *бинарной*. Сформулированные ниже правила формирования отношений из диаграмм ER-типа распространяются именно на бинарные связи. Поэтому, когда речь идет о связях, слово «бинарные» далее опускается.

5.3. Правила формирования отношений

Правила формирования отношений основываются на учете следующего:

• степени связи между сущностями (1:1, 1:M, M:1, M:M);

• класса принадлежности экземпляров сущностей (обязательный и необязательный). |

Рассмотрим формулировки шести правил формирования отношений на основе диаграмм ER-типа.

Формирование отношений для связи 1:1

Правило 1. Если степень бинарной связи 1:1 и класс принадлежности обе их сущностей обязательный, то формируется одно отношение. Первичным ключом этого отношения может быть ключ любой из двух сущностей.

На рис. 6.6 приведены диаграмма ER-типа и отношение, сформированное по правилу 1 на ее основе.

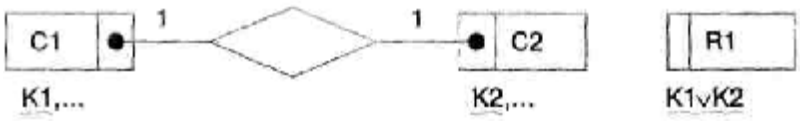


Рис. 5.6. Диаграмма и отношения для правила 1

На рисунке используются следующие обозначения:

CI, C2 -- сущности 1 и 2;

Kl, K2 - ключи первой и второй сущности соответственно;

R1 - отношение 1, сформированное на основе первой и второй сущностей;

K1vK2,... означает, что ключом сформированного отношения может быть либо К1, либо К2.

Это и другие правила будем проверять, рассматривая различные варианты связи ПРЕПОДАВАТЕЛЬ ВЕДЕТ ДИСЦИПЛИНУ. Пусть сущность ПРЕПОДАВАТЕЛЬ характеризуется атрибутами НП (идентификационный

номер преподавателя), ФИО (фамилия, имя и отчество), Стаж (стаж преподавателя). Сущность ДИСЦИПЛИНА характеризуется соответственно атрибутами КД (код дисциплины), Часы (часы, отводимые на дисциплину). Тогда схема отношения, содержащего информацию об обеих сущностях, и само отношение для случая, когда степень связи равна 1:1, а КП обязательный для всех сущностей, могут иметь вид, показанный на рис. 5.7.

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА(НП,ФИО, Стаж, КД, Часы) ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

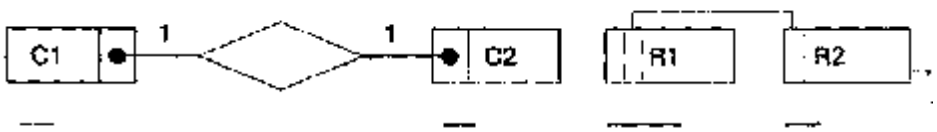
НП	ФИО	Стаж	КД	Часы
П1	Иванов И.М.	5	К1	62
П2	Петров М.И.	7	К2	74
П3	Сидоров Н.Г.	10	К3	102
П4	Егоров В.В.	5	К4	80

Рис. 5.7. Полученные по правилу 1 схема и отношение

Сформированное отношение содержит полную информацию о преподавателях, дисциплинах и о том, как они связаны между собой. Так, преподаватель Иванов ведет только дисциплину с кодом К1, а дисциплина К1 ведется только Ивановым (связь 1:1). В этом отношении отсутствуют пустые поля (КП обязательный для всех сущностей), т. к. нет преподавателей, которые бы что-то не вели, и нет дисциплин, которые никто не ведет. Таким образом, одного отношения в данном случае достаточно. В качестве перничного ключа может быть выбран ключ первого отношения НП или ключ второго отношения КД.

Правило 2. Если степень связи 1:1 и класс принадлежности одной сущности обязательный, а второй - необязательный, то под каждую из сущностей формируется по отношению с первичными ключами, являющимися ключи ми соответствующих сущностей. Далее к отношению, сущность которого имеет обязательный КП, добавляется в качестве атрибута ключ сущности с необязательным КП.

На рис. 5.8 приведены диаграмма ER-типа и отношения, сформированные по правилу 2 на ее основе.



К1,... К2,... К1.К2 К2, ...

Рис. 5.8. Диаграмма и отношения для правила 2

Чтобы убедиться в справедливости правила, рассмотрим следующий пример. На рис. 5.9 приведено исходное отношение, содержащее информацию о преподавателях и дисциплинах. Оно представляет вариант, в котором класс сущности ПРЕПОДАВАТЕЛЬ является обязательным, а сущности ДИСЦИПЛИНА - необязательным. При этом пробелы «—» (пустые поля) присутствуют во всех кортежах с информацией о дисциплинах, которые не

ведутся ни одним из преподавателей.

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

нп	ФИО	Стаж	КД	Часы
П1	Иванов И.М.	5	К1	62
П2	Петров М.И.	7	К2	74
П3	Сидоров Н.Г.	10	К3	102
—	...	—	К4	80

Рис. 5.9. Исходное отношение

Избежать этой ситуации можно, применив правило 2, в соответствии с которым, выделяются два отношения, приведенные на рис. 5.10.

ПРЕПОДАВАТЕЛЬ(НП, ФИО, Стаж, КД)

ПРЕПОДАВАТЕЛЬ

НП	ФИО	Стаж	КД
111	Иванов И.М.	5	К1
112	Петров М.И.	7	К2
113	Сидоров Н.Г.	10	К3
114	Егоров В.В.	5	К4

ДИСЦИПЛИНА(КД, Часы)

ДИСЦИПЛИНА

кд	Часы
К1	62
К2	74
К3	102
К4	80

Рис. 510. Отношения, полученные по правилу 2

В результате мы избежали пустых полей в отношениях, не потеряв данных. Добавив атрибут КД - ключ сущности ДИСЦИПЛИНА (с необязательным КП) в качестве внешнего ключа в отношение, соответствующее сущности ПРЕПОДАВАТЕЛЬ (с обязательным КП), мы связали отношения (рис. 5.11).

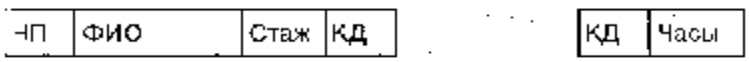
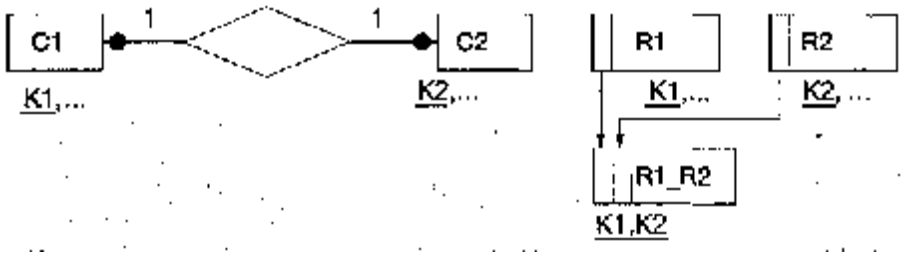


Рис.5.11. Связь отношений по внешнему ключу

Точнее говоря, мы создали условия для связывания отношений. Это свя-1ывание при работе с базой данных позволяет, например, получать одновременно данные о преподавателе и о ведущихся им дисциплинах (часах).

Правило 3. Если степень связи 1:1 и класс принадлежности обеих сущностей является необязательным, то необходимо использовать три отношения. Два отношения соответствуют связываемым сущностям, ключи которых являются первичными в этих отношениях. Третье отношение является связным между первыми двумя, поэтому его ключ объединяет ключевые атрибуты свя-

зываемых отношений.



На рис. 5. 12 приведены диаграмма ER-типа и отношения, сформированные по правилу 3 на ее основе.

На рис. 5.13 приведены примеры отношений, подтверждающие необходимость использования трех отношений при наличии необязательного КП обеих связываемых сущностей.

а) одно отношение

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

нп	ФИО	Стаж	**.кд**	Часы
П1	Иванов	5	К1	62
П2	Петров	7	—	...
ПЗ	Сидоров	10	К2	74
—	—	—	КЗ	102

б) два отношения

ПРЕПОДАВАТЕЛЬ

НП	ФИО	Стаж	кд
П1	Иванов	5	К1
П2	Петров	7	—
ПЗ	Сидоров	10	К2

ДИСЦИПЛИНА

кд	Часы	НП
К1	62	П1
К2	74	ПЗ
КЗ	102	—

в) три отношения

ПРЕПОДАВАТЕЛЬ

НП	ФИО	Стаж
П1	Иванов	5
П2	Петров	7
ПЗ	Сидо-	10

ВЕДЕТ

НП	**кд**
П1	К1
ПЗ	К2

ДИСЦИПЛИНА

кд	Часы
К1	62
К2	74
КЗ	102

Рис. 5.13. Варианты отношений для правила 3

Использование одного отношения в рассматриваемом случае приводит к наличию нежелательных пустых полей в этом отношении (рис. 5.13а). При использовании двух отношений (рис. 5.136) нам пришлось добавить ключи каждой из сущностей в отношение) соответствующее другой сущности, чтобы не потерять сведения о том, какую дисциплину ведет каждый преподаватель и наоборот. При этом также появились пустые поля.

Выход заключается и использовании трех отношений, сформированных по правилу 3 (рис. 5.13в). Объектные отношения (с атрибутами сущностей) содержат полную информацию обо всех преподавателях и дисциплинах соответственно.

Формирование отношений для связи 1:М

Если две сущности С1 и С2 связаны как 1:М, сущность С1 будем называть ни >связной (1-связной), а сущность С2 - многосвязной (М-связной). Определяющим фактором при формировании отношений, связанных этим видом связи, является класс принадлежности М-связной сущности. Так, если класс принадлежности М-связной сущности обязательный, то в результате применим правила получим два отношения, если необязательный - три отношения. Класс принадлежности односвязной сущности не влияет на результат.

Чтобы убедиться в этом, рассмотрим отношение ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА (рис. 5.14), соответствующее диаграммам, приведенным на рис 5.4, т. е. случаю, когда: связь типа 1:М, класс принадлежности М-связи и сущности обязательный, 1-связной - необязательный.

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

нп	ФИО	Стаж	**кд**	Часы
П1	Ивано-	5	К1	62
П1	Ивано-	5	К2	74
П2	Пет-	7	К4	80
ПЗ	Сидоров	10	К5	96
ПЗ	Сидоров	10	Кб	120
П4	Егоров	5	КЗ	102
П4	Егоров	5	К7	89
П5	Козлов	8	—	—

Рис. 5.14. Исходное отношение

Проблемы:

• имеются кортежи с пустыми полями (преподаватель не ведет дисцип-

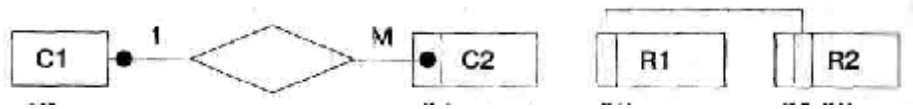
лину)

• избыточное дублирование данных (повторяется стаж преподавателя) в кортежах со сведениями о преподавателях, ведущих несколько дисциплин

Если бы класс принадлежности 1-связной сущности был обязательным (нет преподавателя без дисциплины), то исчезли бы пустые поля, но повторяющие данные в атрибутах преподавателя сохранились бы. Для устранения названных проблем отношения могут быть сформированы по следующему правилу.

Правило 4. Если степень связи между сущностями 1:M (или M:1) и класс принадлежности M-связной сущности обязательный, то достаточно формирование двух отношений (по одному на каждую из сущностей). При этом первичными ключами этих отношений являются ключи их сущностей. Кроме того, ключ 1-связной сущности добавляется как атрибут (внешний КЛЮЧ в отношение, соответствующее M-связной сущности.

На рис. 5.15 приведены диаграмма ER-типа и отношения, сформированные по правилу 4.



К2,... **К1** **К2.-** **К1.К2**

Рис.5.15. Диаграмма и отношения для правила 4

В соответствии с правилом 4 преобразуем отношение на рис. 5.14 в два отношения (рис. 5.16).

ПРЕПОДАВАТЕЛЬ

нп	ФИО	Стаж
П1	Ивано-	5
П2	Пет-	7
ПЗ	Сидоров	10
П4	Егоров В.	5
П5	Козлов	8

ДИСЦИПЛИНА

кд	Часы	НП
К1	62	П1
К2	74	П1
КЗ	102	П4
К4	80	П2
К5	96	ПЗ
Кб	120	ПЗ
К7	89	П4

Рис. 5.16. Отношения, полученные по правилу 4

На рис. 5. 16 видно, что пустые поля и дублирование информации удалось устранить. Потери сведении о том, кто из преподавателей ведет какую дисциплину, не произошло благодаря введению ключа НП сущности ПРЕПОДАВАТЕЛЬ) в качестве внешнего ключа в отношение ДИСЦИПЛИНА.

Пример. Связь между сущностями 1:М, а класс принадлежности М-связи сущности необязательный.

Пусть класс принадлежности 1-связной сущности также необязательный, хотя это и не принципиально, так как определяющим является класс принадлежности М-связной сущности. Посмотрим, к чему может привести использование одного отношения в этом случае (рис. 5.17).

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

НП	ФИО	Стаж	**кд**	Часы
П1	ИвановИ.М.	5	К1	62
П1	ИвановИ.М.	5	К2	74
П2	ПетровМ.И.	7	К4	80
—	—	—	К5	96
ПЗ	Сидоров Н.Г.	10	Кб	120
П4	Егоров В. В.	5	КЗ	102
П4	Егоров В. В.	5	К7	89
П5	Козлов А.С.	8	—	—

Рис. 5.17. Исходное отношение

С приведенным отношением связаны следующие проблемы:

1. Имеются пустые поля в кортежах, которые содержат следующее:
а) данные о преподавателях, не ведущих дисциплин;
б) данные о дисциплинах, которые не ведутся преподавателями.
2. Избыточное дублирование данных о преподавателях, ведущих более одной дисциплины.

В случае обязательного класса принадлежности 1-связной сущности исчезают проблемы 1 а). Для устранения всех проблем нужно перейти к трем отношениям в соответствии со следующим правилом.

Правило 5. Если степень связи 1:М (М:1) и класс принадлежности М-связной сущности является необязательным, то необходимо формирование трех отношений (рис. 5.18). Два отношения соответствуют связываемым сущностям, ключи которых являются первичными в этих отношениях. Третье отношение является связным между первыми двумя (его ключ объединяет ключевые атрибуты связываемых отношений).

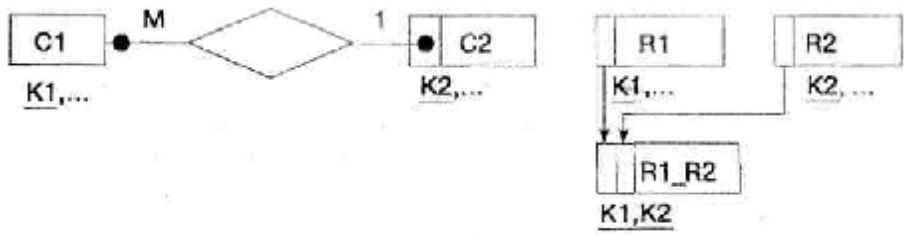


Рис. 5.18. Диаграмма и отношение для правила 5

В результате применения правила 5 к рассматриваемому отношению со держащиеся в нем данные (рис. 5.17) распределяются по трем отношениям (рис. 5.19).

ПРЕПОДАВАТЕЛЬ

нп	ФИО	Стаж
П1	ИвановИ.М.	5
П2	ПетровМ.И.	7
П3	Сидоров Н.Г.	10
П4	Егоров В.В.	5
П5	Козлов А.С.	8

ВЕДЕТ

НП	КД
П1	К1
П1	К2
П2	К4
П3	Кб
П4	К3
П4	К7

ДИСЦИПЛИНА

кд	Часы
К1	62
К2	74
К3	102
К4	80
К5	96
Кб	120
К7	89

Рис. 5.19. Отношения, полученные по правилу 5

Таким образом, указанные проблемы удалось разрешить. Ключ в связном отношении ВЕДЕТ является составным и включает в себя ключевые атрибуты обоих связываемых отношений (сущностей). В практических ситуациях связное отношение может содержать и другие характеризующие связь атрибуты.

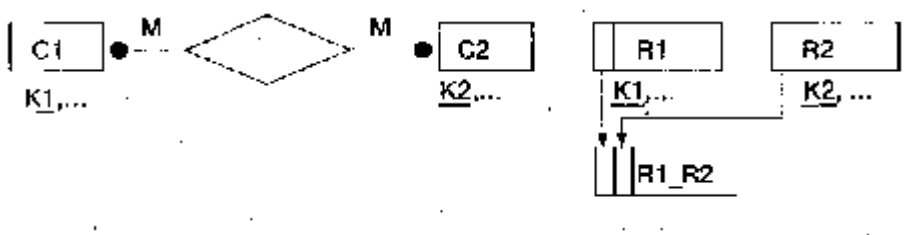
Подчеркнем, что определяющим фактором при выборе между 4-м или 5-м правилом является класс принадлежности М-связной сущности.

Формирование отношений для связи М:М

При наличии связи М:М между двумя сущностями необходимо три отношения независимо от класса принадлежности любой из сущностей. Использование одного или двух отношений н атом случае не избавляет от пустых ми или избыточно дублируемых данных.

Правило 6. Если степень связи М:М, то независимо от класса принадлежности сущностей формируются три отношения. Два отношения соответствуют связываемым сущностям и их ключи являются первичными ключами этих имений. Третье отношение является связным между первыми двумя, а его ключ объединяет ключевые атрибуты связываемых отношений.

На рис. 5.20 приведены диаграмма ER-типа и отношения, сформированное правилу 6. Нами показан вариант с классом принадлежности сущностей Н-Н, хотя, согласно правилу 6, он может быть произвольным.



К1,К2

Рис. 5.20. Диаграмма и отношения для правила 6

Применим правило 6 к примеру, приведенному на рис, 6.5. В нем степень связи равна М:М, класс принадлежности для сущности ПРЕПОДАВАТЕЛЬ обязательный, а для сущности ДИСЦИПЛИНА - необязательный. Соответствующее этому примеру исходное отношение показано на рис. 5.21.

ПРЕПОДАВАТЕЛЬ_ДИСЦИПЛИНА

нп	ФИО	Стаж	**кд**	Часы
П1	Ивано-	5	К1	62
П1	Ивано-	5	К2	74
П2	Пет-	7	К4	80
—	—	—	К3	102
пз	Сидоров	10	Кб	120
П4	Егоров В.	5	К2	74
П4	Егоров В.	5	К7	89
П5	Козлов	8	К5	96

Рис. 5.21. Исходное отношение

В результате применения правила 6 получаются три отношения (рис. 5.22)

ПРЕПОДАВАТЕЛЬ

нп	ФИО	Стаж
П1	Ивано-	5
П2	Пет-	7
П3	Сидоров	10
П4	Егоров	5
П5	Козлов	8

ВЕДЕТ

НП	КД
П1	К1
П1	К2
П2	К4
П3	Кб
П4	К3
П4	К7

ДИСЦИПЛИНА

кд	Часы
К1	62
К2	74
К3	102
К4	80
К5	96
Кб	120
К7	89

ТЕМА 6. ИНФОРМАЦИОННЫЕ СИСТЕМЫ В СЕТЯХ

6.1. Модели архитектуры клиент-сервер

При построении распределенных ИС, работающих с БД, широко используется архитектура клиент-сервер. Ее основу составляют принципы организации взаимодействия клиента и сервера при управлении БД.

Чтобы охарактеризовать основные схемы взаимодействия процессов управления БД, воспользуемся Эталонной моделью Архитектуры открытых систем OSI. Согласно этой модели, функция управления БД относится к прикладному уровню.

Остановимся на двух самых верхних уровнях: *прикладном* и *представительском*, которые в наибольшей степени являются предметом внимания со стороны разработчика и пользователя. Остальные функции будем считать связными функциями, необходимыми для реализации двух первых. При этом будем придерживаться широкого толкования термина СУБД, понимая под ним все программные системы, которые используют информацию из БД.

Как поддерживающая интерфейс с пользователем программа, СУБД, в общем случае, реализует следующие основные функции:

- управление данными, находящимися в базе;
- обработка информации с помощью прикладных программ;
- представление информации в удобном для пользователя виде.

При размещении СУБД в сети возможны различные варианты распределения функции по узлам. В зависимости от числа узлов сети, между которыми выполняется распределение функций СУБД, можно выделить *двухзвенные* модели, *трехзвенные* модели и т. д. Место разрыва функций соединяется коммуникационными функциями (средой передачи информации в сети).

Двухзвенные модели распределения функций *Двухзвенные модели* соответствуют распределению функций СУБД между *двумя узлами сети*. Компьютер (узел сети), на котором обязательно присутствует функция управления данными, назовем *компьютером-сервером*. Компьютер, близкий к пользователю и обязательно занимающийся вопросами представления информации, назовем ***компьютером-клиентом.***

Наиболее типичными вариантами (рис. 6.1) разделения функций между компьютером-сервером и компьютером-клиентом являются следующие:

- распределенное представление;
- удаленное представление;
- распределенная функция;
- удаленный доступ к данным;
- распределенная БД.

Перечисленные способы распределения функций в системах с архитектуре >й клиент-сервер иллюстрируют различные варианты: от мощного сервера, мила практически вся работа производится на нем, до

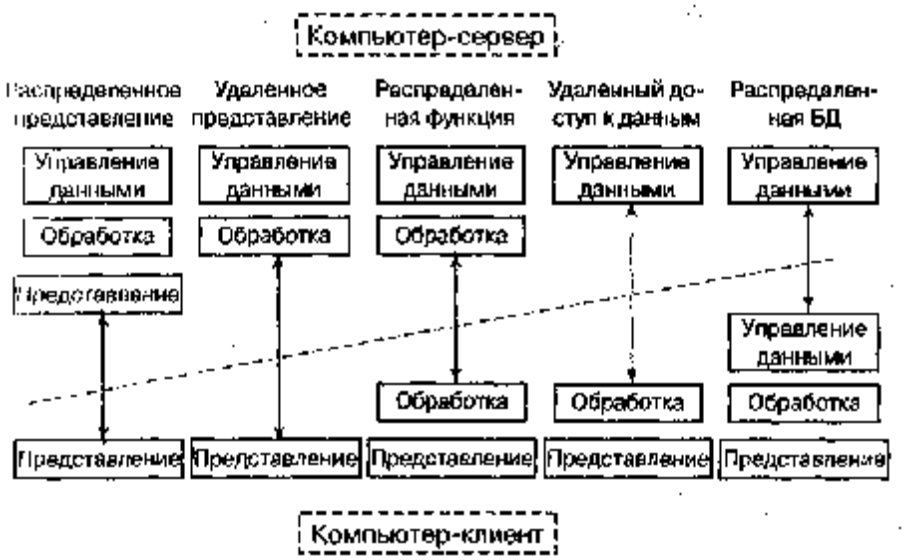


Рис. 6.1. Спектр моделей архитектуры клиент-сервер

мощного клиента, когда большая часть функций выполняется на рабочей станции, а сервер обрабатывает поступающие к нему по сети SQL-вызовы.

В моделях удаленного доступа к данным и удаленного представления производится строгое распределение функций между компьютером-клиентом и компьютером-сервером. В других моделях имеет место выполнение одном из следующих функций одновременно на двух компьютерах: управления данными (модель распределенной БД), обработки информации (модель распределенной функции), представления информации (модель распределенного представления).

Рассмотрим сначала модели *удаленного доступа к данным* и *удаленного представления (сервера БД)* как наиболее широко распространенные.

В модели ***удаленного доступа к данным*** (Remote Data Access — RDA) программы, реализующие функции представления информации и логику прикладной обработки, совмещены и выполняются на компьютере-клиенте. Обращение за сервисом управления данными происходит через среду передачи с помощью операторов языка SQL или вызовом функций специальной библиотеки API (Application Programming Interface — интерфейса прикладного программирования).

Основное *достоинство* RDA-модели состоит в большом обилии готовых СУБД, имеющих SQL-интерфейсы, и существующих инструментальных I средств, обеспечивающих быстрое создание программ клиентской части. Средства разработки чаще всего поддерживают графический интерфейс пользователя в MS Windows, стандарт интерфейса ODBC и средства автоматической генерации кода. Подавляющее большинство средств разработки использует языки четвертого поколения.

Недостатками RDA-модели являются, во-первых, довольно высокая

загрузка системы передачи данных вследствие того, что вся логика сосредоточена в приложении, а обрабатываемые данные расположены на удаленном узле, Как увидим далее, во время работы приложений обычно по сети передаются целые БД.

Во-вторых, системы, построенные на основе модели RDA, неудобны с точки зрения разработки, модификации и сопровождения. Основная причина состоит в том, что в получаемых приложениях прикладные функции и функции представления тесно взаимосвязаны. Поэтому даже при незначительном изменении функций системы требуется переделка всей прикладной ее части, усложняющая разработку и модификацию системы.

Модель ***сервера* БД** (DataBase Server — DBS) отличается от предыдущей модели тем, что функции компьютера-клиента ограничиваются функциями представления информации, в то время как прикладные функции обеспечиваются 1 приложением, находящимся на компьютере-сервере. Эта модель является более технологичной чем RDA-модель и применяется в таких СУБД, как Ingress, Sybase и Oracle. При этом приложения реализуются в виде *хранимых процедур.*

Процедуры обычно хранятся в словаре БД и разделяются несколькими клиентами. В общем случае хранимые процедуры могут выполняться в режимах компиляции и интерпретации.

Достоинствами модели DBS являются: возможность хорошего централизованного администрирования приложений на этапах разработки, сопровождения и модификации, а также эффективное использование вычислительных и коммуникационных ресурсов. Последнее достигается за счет того, что выполнение программ в режиме коллективного пользования требует существенно меньших затрат на пересылку данных в сети.

Один из *недостатков* модели DBS связан с ограничениями средств разработки хранимых процедур. Основное ограничение — сильная привязка операторов хранимых процедур к конкретной СУБД. Язык написания хранимых процедур, по сути, является процедурным расширением языка SQL, и не может соперничать по выразительным средствам и функциональным возможностям с традиционными языками третьего поколения, такими как и Pascal. Кроме того, в большинстве СУБД нет удовлетворительных средств отладки и тестирования хранимых процедур, что делает их механизм опасным инструментом — неотлаженные программы могут приводить к некорректностям БД, зависаниям серверных и клиентских программ во время работы системы и т. п.

Другим *недостатком* DBS-модели является низкая эффективность использования вычислительных ресурсов ЭВМ, поскольку не удается организовать управление входным потоком запросов к программам компьютера-сервера, а также обеспечить перемещение процедур на другие компьютеры-серверы.

В модели ***распределенного представления*** имеется мощный компьютер-сервер, а клиентская часть системы практически вырождена. Функцией клиентской части является просто отображение информации на экра-

не монитора и связь с основным компьютером через локальную сеть.

СУБД подобного рода могут иметь место в сетях, поддерживающих работу так называемых *X-терминалов*. В них основной компьютер (хост-машина) должен иметь достаточную мощность, чтобы обслуживать несколько X-терминалов. X-терминал тоже должен обладать достаточно быстрым процессором и иметь достаточный объем оперативной памяти (дисковые накопители отсутствуют). Часто X-терминалы создают на базе RISC-компьютеров (restricted [reduced] instruction set computer) — компьютеров с сокращенным набором команд. Все программное обеспечение находится на хост-машине. Программное обеспечение X-терминала, выполняющее функции управления представлением и сетевые функции, загружается по сети с сервера при включении X-терминала.

Модель распределенного представления имели СУБД ранних поколений, которые работали на малых, средних и больших ЭВМ. В роли X-терминалов выступали дисплейные станции и абонентские пункты (локальные и удаленные). В этом случае основную чаги, функций представления информации реализовывали сами СУБД, а окончательное построение изображений на терминалах пользователя выполнялось на оконечных устройствах.

По модели распределенного представления построены системы обслуживания пользователей БД в гетерогенной (неоднородной) среде. Серверная часть таких систем обычно обеспечивает некоторый унифицированный интерфейс, а клиентские части реализуют функции учета специфики оконечного оборудования или преобразования одного формата представления информации в другой.

Модель распределенного представления реализует централизованную схему управления вычислительными ресурсами. Отсюда следуют ее основные *достоинства* — простота обслуживания и управления доступом к системе и относительная дешевизна (ввиду невысокой стоимости оконечных терминалов). *Недостатками* модели являются уязвимость системы при невысокой надежности центрального узла, а также высокие требования к серверу по производительности при большом числе клиентов.

В модели ***распределенной функции*** логика обработки данных распределена по двум узлам. Такую модель могут иметь ИС, в которых общая часть прикладных функций реализована на компьютере-сервере, а специфические функции обработки информации выполняются на компьютере-клиенте. Функции общего характера могут включать в себя стандартное обеспечение целостности данных, Например, в виде хранимых процедур, а оставшиеся прикладные функции реализуют специальную прикладную обработку. Подобную модель имеют также ИС, использующие информацию из нескольких неоднородных БД.

Модель ***распределенной БД*** предполагает использование мощного компьютера-клиента, причем данные хранятся на компьютере-клиенте и на компьютере-сервере. Взаимосвязь обеих баз данных может быть двух разновидностей: а) в локальной и удаленной базах хранятся отдельные части еди-

ной БД; б) локальная и удаленная БД являются синхронизируемыми друг с другом копиями.

Достоинством модели распределенной БД является гибкость создаваемых на ее основе ИС, позволяющих компьютеру-клиенту обрабатывать локальные и удаленные БД. При наличии механизмов координации соответствия копий система в целом, кроме того, обладает высокой живучестью, так как разрыв соединения клиента и сервера не приводит к краху системы, а ее работа может быть восстановлена с возобновлением соединения. К *недостатку* модели можно отнести высокие затраты при выполнении большого числа одинаковых приложений на компьютерах-клиентах.

Трехзвенная модель распределения функций

Трехзвенная модель распределения функций представляет собой типовой вариант, при котором каждая из трех функций приложения реализуется на отдельном компьютере. Варианты распределения функций приложения на большее число компьютеров могут иметь место, но ввиду их редкого применения рассматриваться не будут. ,

Рассматриваемая нами модель имеет название *модель сервера приложений*, или *AS-моделъ* (Application Server), и показана на рис. 6.2,

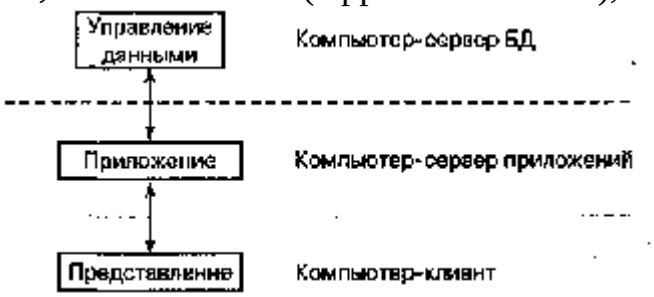


Рис. 6.2. Трехзвенная модель сервера приложений

Согласно трехзвенной AS-модели, отвечающий за организацию диалога с конечным пользователем процесс, как обычно, реализует функции представления информации и взаимодействует с компонентом приложения так же, как в модели DBS. Компонент приложения, располагаясь на отдельном компьютере, в свою очередь, связан с компонентом управления данными подобно модели RDA.

Центральным звеном AS-модели является сервер приложений. На сервере приложений реализуется несколько прикладных функций, каждая из которых оформлена как служба предоставления услуг всем требующим этого программам. Серверов приложений может быть несколько, причем каждый из них предоставляет свой вид сервиса. Любая программа, запрашивающая услугу у сервера приложений, является для него клиентом. Поступающие от клиентов к серверам запросы помещаются в очередь, из которой выбираются в соответствии с некоторой дисциплиной, например, по приоритетам.

Компонент, реализующий функции представления и являющийся клиентом для сервера приложений, в этой модели трактуется более широко, чем обычно. Он может служить для организации интерфейса с конеч-

ным пользователем, обеспечивать прием данных от устройств, например, датчиков, или быть произвольной программой.

Достоинством AS-модели является гибкость и универсальность вследствие разделения функций приложения на три независимые составляю еще. Во многих случаях эта модель оказывается более эффективной по сравнению с двухзвенными. Основной *недостаток* модели — более высокие затраты ресурсов компьютеров на обмен информацией между компонентами при сравнении по сравнению с двухзвенными моделями.

Примерами программных продуктов, реализующих среду (функционирования приложений на компьютерах-серверах приложений, являются BEA WebLogic Server (BEA Systems Corp.), Inprise Application Server (Inprise Corp.) и IBM WebSphere Application Server (IBM Corp.).

Сложные схемы взаимодействия

Возможны более *сложные схемы взаимодействия,* например, схемы, в которых элемент, являющийся сервером для некоторого клиента, в свою очередь, выступает в роли клиента по отношению к другому серверу (рис. 6.3). Пример этого мы наблюдали в AS-модели.

Рис. 6.3. Цепочка взаимодействий типа клиент-сервер

Возможно также, что в распределенной вычислительной системе при работе с БД имеются *множественные* связи (статические), когда один объект по отношению к одним является клиентом, а по отношению к другим — сервером (рис. 6.4).

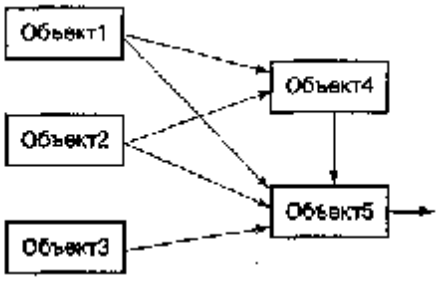


Рис. 6.4. Множественные связи взаимодействия типа клиент-сервер

При рассмотрении взаимодействия объектов в *динамике* получаются еще более сложные схемы взаимодействия. Примером такой схемы является случай, когда в процессе работы роли объектов меняются: объект, являющийся в некоторый момент времени клиентом по отношению к другому объекту, в последующем становится сервером для другого объекта.

Модель монитора транзакций

Как отмечалось, наиболее гибкой и универсальной моделью распределением функций СУБД является AS-модель. Она описывает взаимодействие трех основных элементов: клиента, сервера приложений и сервера БД, но не затрагивает вопросы организации функционирования программ-

ного обеспечения при обработке информации, в частности при выполнение транзакций. Для преодоления этого *недостатка* предложена *модель монитора транзакций.*

Мониторы обработки транзакций (Transaction Processing Monitor — TPM), или *мониторы транзакций*, представляют собой программные системы категории промежуточного слоя (Middleware), обеспечивающие эффективное управление информационно-вычислительными ресурсами в распределенной вычислительной системе. Основное их назначение — организация гибкой, открытой среды для разработки и управления мобильными приложениями, оперативно обрабатывающими распределенные транзакции. Применение мониторов транзакций наиболее эффективно в гетерогенных вычислительных системах.

Приложение TPM позволяет выполнять масштабирование системы, поддерживать функциональную полноту и целостность приложений а также повысить производительность обработки данных при невысокой стоимости накладных расходов.

Принципы организации обработки информации с помощью монитора транзакций описываются ***моделью монитора транзакций*** X/Open DTP (Distributed Transaction Processing — обработка распределенных транзакций). Эта модель (рис. 6.5) включает в себя три объекта: прикладную программу, менеджер ресурсов (Resource Manager — RM) и монитор, или менеджер транзакций (Transaction Manager — TM).

В качестве прикладной программы может выступать произвольная программа-клиент. RM выполняет функции сервера ресурса некоторого

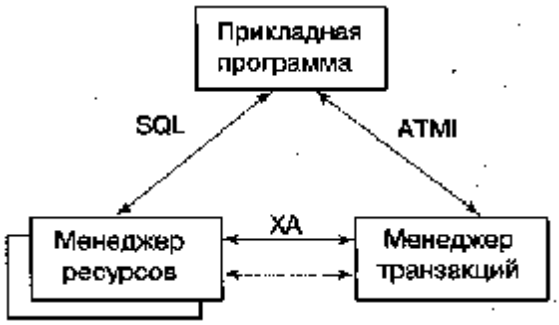


Рис. 6.5. Модель обработки транзакций X/Open DTP

вида. Прикладная программа взаимодействуете КМ с помощью набора специальных функций либо посредством операторов SQL (когда сервером является сервер БД).

Интерфейс ATMI (Application Transaction Monitor Interface — интерфейс монитора транзакций приложения) позволяет вызывать функции TPM ill некотором языке программирования, например С.

Функции менеджера ресурсов обычно выполняют серверы БД или СУБД. В задачах организации управления обработкой распределенных транзакций (транзакций, затрагивающих программные объекты вычислительной сети) TM взаимодействует с RM, который должен поддерживать протокол двухфазной фиксации транзакций и удовлетворять стандарту X/Open XA. Примерами СУБД, поддерживающих протокол двухфазной

фиксации транзакции, являются Oracle 7.0, Open INGRES и Informix-Online 5.0.,

Понятие транзакции в TPM несколько шире, чем в СУБД, где транзакции включает в себя элементарные действия над данными базы. Здесь транзакция может охватывать и другие необходимые действия: передачу сообщения, запись информации в файл, опрос датчиков и т. д. Это значит, что TPM предоставляет более мощные средства управления вычислительным процессом. Транзакции, которые поддерживают TPM, называют также *прикладными* или) *бизнес-транзакциями*.

Модель X/Open DTP не раскрывает структуру TM в деталях, а определяет состав компонентов распределенной системы обработки информации и как эти компоненты взаимодействуют друг с другом. Практические реализации этой модели, естественно, могут отличаться друг от друга. В числе примеров реализаций мониторов транзакций можно назвать ACMS, CICS, и TUXEDO^ System.

Для *разработчиков приложений* мониторы обработки транзакций TPM предоставляют удобства, связанные с возможностью декомпозиции приложений по нескольким функциональным уровням со стандартными интерфейсами, что позволяет создавать легко модифицируемые ИС со стройной архитектурой.

Прикладные программы становятся практически независимыми от конкретной реализации интерфейса с пользователем и от менеджера ресурсов. Пер вое означает, что для реализации функций представления можно выбрать любое удобное и привычное для разработчика средство (от языка С до какой-либо CASE-системы). Независимость от менеджера ресурсов подразумевает возможность легкой замены одного менеджера ресурсов на другой, лишь бы они поддерживали стандарт взаимодействия с прикладной программой (для, СУБД - язык SQL).

Если TPM поддерживает множество аппаратно-программных платформ! как, например, TUXEDO System, то разрабатываемые приложения становятся, кроме того, мобильными.

Для ***пользователей распределенных систем*** TPM позволяют улучшить показатели пропускной способности и времени отклика, снизить стоимость наработки данных в оперативном режиме на основе эффективной организации вычислительного процесса.

Улучшение показателей функционирования достигается благодаря осуществлению статической и динамической балансировки нагрузки. Управление загрузкой состоит в запуске или остановке AS-процессов (программных компонентов AS-модели) в зависимости от заранее установленных параметров и текущего состояния системы. При необходимости TPM может тиражировать копии AS-процессов на этом или других узлах сети.

Администраторы распределенных систем, имея TPM, получают возможность легкого масштабирования ИС и увеличения производительности обработки информации. Здесь, кроме вертикального и горизонтального масштабирования, можно обеспечить так называемое *матричное*

масштабирование. Суть его — введение дополнительных ресурсов в любую точку гетерогенной вычислительной среды без изменения архитектуры приложения, выполняемого в новой среде. Это означает, что без остановки серверов приложений в любое время может быть добавлен, например, компьютер или менеджер ресурсов.

Кроме того, администраторы систем получают возможность снизить общую стоимость программного обеспечения систем клиент-сервер. Снижение стоимости можно достигнуть простым уменьшением количества подключений к серверам БД.

Стоимость серверов БД (или СУБД) в сильной степени зависит от числа одновременных подключений к программе. Уменьшение количества подключений к серверу БД достигается путем мультиплексирования запросов или, того то же самое, логического преобразования потока запросов от многих источников к потоку от одного или нескольких источников. Выигрыш в стоимости и производительности при эффективной реализации самих TPM может оказаться весьма существенным.

6.3. Управление распределенными данными

С управлением данными в распределенных системах связаны следующие группы проблем: поддержка соответствия БД вносимым изменениям и обеспечение совместного доступа нескольких пользователей к общим данным.

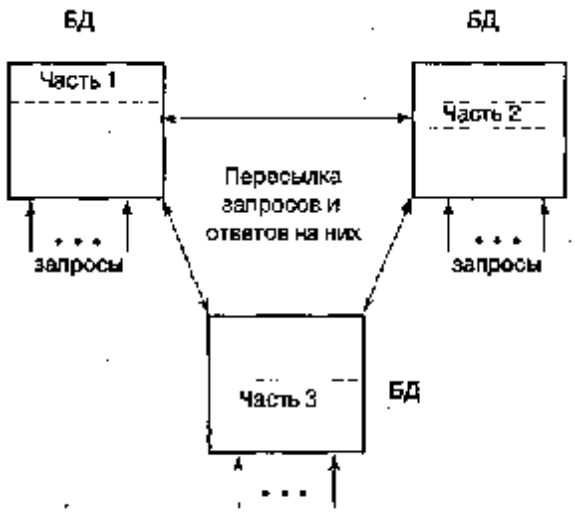
Проблема совместного доступа к распределенным данным тесно связана с тупиками. Одним из средств избегания тупиков являются протоколы фиксации транзакций.

Поддержка соответствия БД вносимым изменениям

В современных распределенных системах информация может храниться *централизованно* или *децентрализованно*. В первом случае проблемы идентичности представления информации для всех пользователей не существует, так как все последние изменения хранятся в одном месте. На практике чаще информация изменяется одновременно в нескольких узлах распределенной вычислительной системы. В этом случае возникает проблема контроля за всеми изменениями информации и предоставления ее в достоверном виде всем пользователям.

Существуют две основные технологии децентрализованного управления БД: *распределенных БД* (Distributed Database) и *тиражирования*, или *репликации, БД* (Data Replication).

Распределенная БД состоит из нескольких фрагментов, размещенных на разных узлах сети и, возможно, управляемых разными СУБД. С точки зрения программ и пользователей, обращающихся к распределенной БД, последняя воспринимается как единая локальная БД (рис. 6.6).



Информация о местоположении каждой из частей распределенной БД и другая служебная информация хранится в так называемом *глобальном словаре данных*. В общем случае этот словарь может храниться на одном из узлов или тоже быть распределенным.

Для обеспечения корректного доступа к распределенной БД в современных системах чаще всего применяется протокол (метод) *двухфазной фиксации транзакций* (two-phase commit). Суть этого метода состоит в двухэтапной синхронизации выполняемых изменений на всех задействованных узлах. На первом этапе в узлах сети производятся изменения (пока обратимые) в их БД, о чем посылаются уведомления компоненту системы, управляющему обработкой распределенных транзакций.

На втором этапе, получив от всех узлов сообщения о правильности выполнения операций (что свидетельствует об отсутствии сбоев и отказов аппаратно-программного обеспечения), управляющий компонент выдает всем узлам команду фиксации изменений. После этого транзакция считается завершенной, а ее результат необратимым.

Основным *достоинством* модели распределенной БД является то, что пользователи всех узлов (при исправных коммуникационных средствах) получают информацию с учетом всех последних изменений. Второе достоинство состоит в экономном использовании внешней памяти компьютеров, что позволяет организовывать БД больших объемов.

К *недостаткам* модели распределенной БД относится следующее: жесткие требования к производительности и надежности каналов связи, а также большие затраты коммуникационных и вычислительных ресурсов из-за их называния на все время выполнения транзакций. При интенсивных обращениях к распределенной БД, большом числе взаимодействующих узлов, низкоскоростных и ненадежных каналах связи обработка запросов по этой схеме становится практически невозможной.

Модель ***тиражирования данных,*** в отличие от технологии распределенных БД, предполагает дублирование данных (создание точных копий) в узлах сети (рис. 6.7). Данные всегда обрабатываются как обычные локальные. Поддержку идентичности копий друг другу в асинхронном режиме

обеспечивает компонент системы, называемый *репликатором* (replicator). При этом между узлами сети могут передаваться как отдельные изменения, так и группы изменений. В течение некоторого времени копии БД могут отличаться друг от друга.

К основным *достоинствам* модели тиражирования БД (в сравнении с предыдущей моделью) относятся: более высокая скорость доступа к данным, так как они всегда есть в узле; существенное уменьшение передаваемого по каналам связи потока информации, поскольку происходит передача не всех операций доступа к данным, а только изменений в БД; повышение надежности механизмов доступа к распределенным данным, поскольку нарушение связи не приводит к потере работоспособности системы (предполагается буферизация потока изменений, позволяющая корректно возобновить работу после восстановления связи).

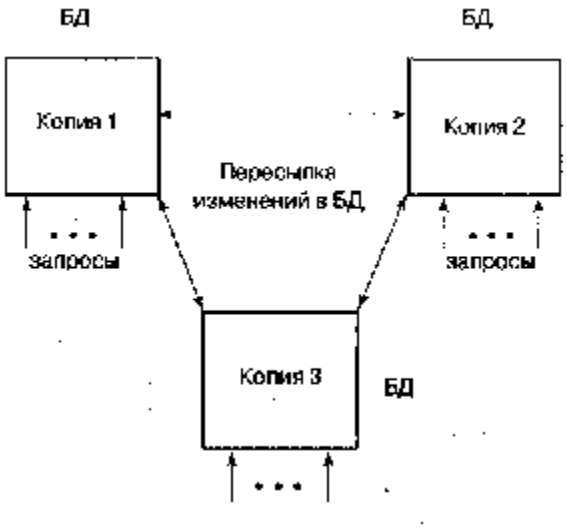


Рис. 6.7. Модель тиражирования БД

Основной *недостаток* модели тиражирования БД заключается в том, **что** на некотором интервале времени возможно «расхождение» копий БД. Если отмеченный недостаток некритичен для прикладных задач, то предпочтительно иметь схему с тиражированием БД.

Доступ к общим данным

При обслуживании обращений к общим данным средства управления БД должны обеспечивать по крайней мере два основных метода доступа: монопольный и коллективный. Основными объектами доступа в различных системах могут быть целиком БД, отдельные таблицы, записи, поля записей. В СУБД, предоставляющих возможность разработки, объектами доступа также могут выступать спецификации отчетов и экранных форм, запросы и программы.

Монопольный доступ обычно используется в двух случаях:

• во-первых, когда требуется исключить доступ к объектам со стороны других пользователей (например, при работе с конфиденциальной ин-

формацией);

• во-вторых, когда производятся *ответственные* операции с БД, не допускающие других действий, например, изменение структуры БД.

В первом случае пользователь с помощью диалоговых средств СУБД или прикладной программы устанавливает явную *блокировку*. Во втором случае пользователь тоже может установить явную блокировку, либо положиться на СУБД. Последняя обычно автоматически устанавливает неявную (без ведома пользователя или приложения) блокировку, если это необходимо.

В режиме ***коллективного доступа*** полная блокировка на используемые объекты, как правило, не устанавливается. Коллективный доступ возможен, например, при одновременном просмотре таблиц. Попытки получить монопольный доступ к объектам коллективного доступа должны быть пресечены.

Для организации коллективного доступа в СУБД применяется *механизм блокировок*. Суть блокировки состоит в том, что на время выполнения какой-либо операции в БД доступ к используемому объекту со стороны других потребителей временно запрещается или ограничивается. Например, при копировании таблицы она блокируется от изменения, хотя и разрешено просматривать ее содержимое.

Рассмотрим некоторый типичный набор блокировок. В конкретных программах схемы блокирования объектов могут отличаться от описываемой. Выделим четыре вида блокировок, перечисленные в порядке убывания строгости ограничений на возможные действия:

• полная блокировка;
• блокировка от записи;
• предохраняющая блокировка от записи;
• предохраняющая полная блокировка.

Полная блокировка. Означает полное запрещение всяких операций над основными объектами (таблицами, отчетами и экранными формами). Этот вид блокировки обычно применяется при изменении структуры таблицы.

Блокировка от записи. Накладывается в случаях, когда можно использовать таблицу, но без изменения ее структуры или содержимого. Такая блокировка применяется, например, при выполнении операции слияния данных и.) двух таблиц.

Предохраняющая блокировка от записи. Предохраняет объект от наложения на него со стороны других операций полной блокировки, либо блокировки от записи. Этот вид блокировки позволяет тому, кто раньше «захватил» объект, успешно завершить модификацию объекта. Предохраняющая блокировка от записи совместима с аналогичной блокировкой (предохраняющей блокировкой от записи), а также с предохраняющей полной блокировкой. Примером необходимости использования этой блокировки является режим совместного редактирования таблицы несколькими пользователями.

Предохраняющая полная блокировка. Предохраняет объект от наложения мл него со стороны других операций только полной блокировки. Обеспечивает максимальный уровень совместного использования объектов. Такая блокировка может использоваться, например, для обеспечения одновременного просмотра несколькими пользователями одной таблицы. В группе пользователей, работающих с одной таблицей, эта блокировка не позволит никому изменить структуру общей таблицы.

При незавершенной операции с некоторым объектом и запросе на выполнение новой операции с этим же объектом производится попытка эти операции совместить. Совмещение возможно тогда, когда совместимыми оказываются блокировки, накладываемые конкурирующими операциями, В отношении перечисленных выше четырех блокировок действуют следующие правила совмещения:

- при наличии полной блокировки над объектом нельзя производить операции, приводящие хотя бы к одному из видов блокировок (полная блокировка несовместима ни с какой другой блокировкой);
- блокировка от записи совместима с аналогичной блокировкой и предохраняющей полной блокировкой;
- предохраняющая блокировка от записи совместима с обоими видами предохраняющих блокировок;
- предохраняющая полная блокировка совместима со всеми блокировками, кроме полной.

Обычно в СУБД каждой из выполняемых с БД операций соответствует определенный вид блокировки, которую эта операция накладывает на объект. Пользователям современных СУБД, работающим в интерактивном режиме, не нужно помнить все тонкости механизма блокировки, поскольку система достаточно «разумно» осуществляет автоматическое блокирование во всех случаях, когда это требуется. При этом система сама стремится предоставить всем пользователям наиболее свободный доступ к объектам. При необходимости пользователь и программист могут воспользоваться командными или языковыми средствами явного определения блокировок. Например, в СУБД Paradox для явного блокирования отдельной записи во время редактирования таблицы используется команда Record | Lock.

Тупики

Если не управлять доступом к совместно используемым объектам, то между потребителями ресурсов могут возникать тупиковые ситуации (клинчи, «смертельные объятия» или блокировки). Следует отличать понятие блокировки в смысле контроля доступа к объектам (мы придерживаемся такого термина) от блокировки в смысле тупикового события.

Существует два основных вида тупиков: *взаимные* (deadlock) и *односторонние* (livelock).

Простейшим случаем ***взаимного тупика*** является ситуация, когда каждый из двух пользователей стремится захватить данные, уже захваченные другим пользователем (рис. 6.8а). В этой ситуации пользователь-1 ждет освобождения ресурса N, в то время как пользователь-2 ожидает ос-

вобождения от захвата ресурса M. Следовательно, никто из них не может продолжить работу.

В действительности могут возникать и более сложные ситуации, когда выполняются обращения трех и более пользователей к нескольким ресурсам. Пример одной из таких ситуаций приведен на рис. 6.8б.

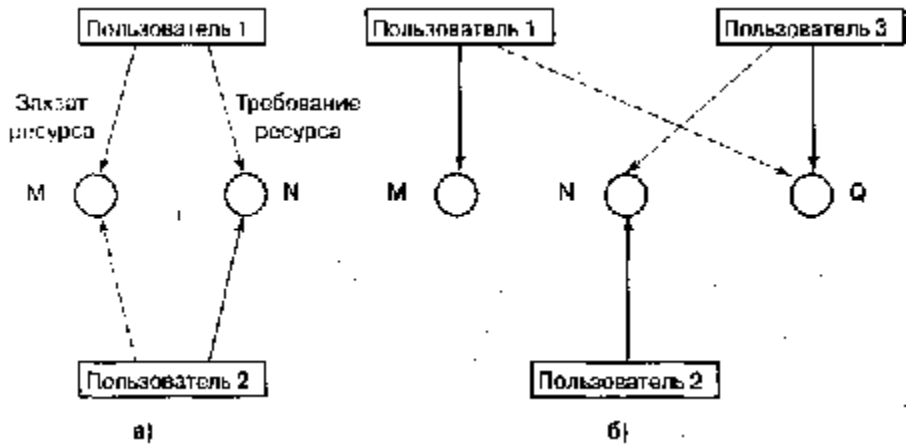


Рис. 6.8. Примеры взаимных тупиков в распределенных БД

Односторонний тупик возникает в случае требования получить монопольный доступ к некоторому ресурсу как только он станет доступным и невозможности удовлетворить это требование.

Системы управления распределенными БД, очевидно, должны иметь соответствующие средства обнаружения или предотвращения конфликтов, а также разрешения возникающих конфликтов. Одной из наиболее сложных является задача устранения конфликтов в неоднородных системах в случае, гели некоторая программа не обрабатывает или обрабатывает некорректно сигналы (уведомления) о наличии конфликтов. При этом важно не только сохранить целостность и достоверность данных в распределенных БД, но и восстановить вычислительный процесс, иногда парализующий пользователи и программы ожиданием чего-то.

Пользователи и разработчики приложений в распределенной среде должны предусматривать обработку сигналов о тупиках.

Протоколы фиксации транзакций

В настоящее время наиболее широко используются два протокола фиксации транзакций: двухфазный и трехфазный. Рассмотрим их вкратце.

Прежде всего, будем считать, что выполняется распределенная транзакция, с которой связан некоторый узел, функционирующий как координатор (диспетчер). Обычно координатором является узел сети, где транзакция была инициирована. Узлы, на которых глобальная транзакция создает агентов, назовем участниками (диспетчерами ресурсов). Координатор транзакции знает идентификаторы всех участников, а каждый участник знает идентификатор координатора, но не обязан знать идентификаторы других участников.

Двухфазный протокол фиксации транзакций выполняется в два этапа: голосование (voting phase) и принятие решения (decision phase).Этот протокол предполагает, что каждый узел имеет свой собственный журнал, с помощью которого можно зафиксировать или отменить транзакцию. Для этого журналы координатора и участников вносятся записи о приеме и посылкой команд (сообщений). Во избежание блокирования из-за бесконечного ожидания ответов от других участников протокола в протоколе используется механизм тайм-аутов.

На первом этапе координатор опрашивает всех участников командой PREPARE, готовы ли они к фиксации транзакции и переходит к ожиданию ответов.

На втором этапе если хотя бы один из участников потребует отката (команда ABORT) или не ответит на запрос в течение установленного интервала времени, координатор указывает всем участникам на необходимость выполнения отката данной транзакции (команда GLOBAL_ABORT). В случае получения подтверждений о фиксации транзакций от всех участников (команды READY_COMMIT), координатор отправляет всем участникам команду глобальной фиксации транзакции GLOBAL_COMMIT). Если участник не получает от координатора в установленное время команды GLOBAL_COMMIT или GLOBALABORT, то он просто выполняет откат транзакции.

Когда некоторый участник требует отката транзакции, то он имеет право выполнить его немедленно. По существу каждый участник имеет право выполнить откат своей субтранзакции в любое время до отправки согласия на ее фиксацию. Такой тип отката называют *односторонним откатом*.

Действия, которые выполняются в случае возникновения тайм-аутов, описываются *протоколом аварийного завершения* (termination protocol). Предпринимаемые при этом действия зависят от того, у кого возникло это событие (у координатора или участника), в каком состоянии был объект, а также какое сообщение было получено. Например, в случае появления тайм-аута на стороне координатора во время ожидания поступления от всех участников уведомлений о принятом глобальном решении по некоторой транзакции, координатор просто повторяет рассылку о принятом решении на те узлы, которые не прислали ожидаемого подтверждения.

Действия, которые выполняются отказавшими узлами после перезапуска, описываются *протоколом восстановления*. Предпринимаемые действия системы после отказа также зависят от обстоятельств, и, главным образом, от этапа обработки транзакции координатором или участником. Пусть, например, отказ произошел у участника в его начальном состоянии (до момента получения им команды PREPARE). Поскольку координатор не может принять решение о глобальной фиксации транзакции в случае, когда хотя бы один узел не отвечает, поэтому для этого участника целесообразно выполнить откат транзакции в одностороннем порядке.

Протокол двухфазной фиксации транзакций позволяет управлять

процессом обработки распределенных транзакций в достаточно широком диапазоне реальных ситуаций. Так, в случае отказа основного управляющего объекта-координатора, у участника имеется возможность обратиться за информации о принятом глобальном решении к другим участникам взаимодействия.

Если же отказал только узел координатора, а все остальные узлы «живы», то участники могут выбрать на роль координатора один из имеющихся узлов. Эти процедуры описываются *протоколами проведения выборов*. Одним из простых и эффективных протоколов проведения выборов является протокол, в котором все узлы упорядочены и имеют идентификаторы, а роль координатора выполняет узел с наименьшим идентификационным номером из числа функционирующих в произвольный момент времени. Согласно протоколу, каждый узел должен послать сообщения на узлы с большими порядковыми номерами. Тогда координатором будет тот узел, который не получит сообщение от узлов с меньшими, чем у него порядковыми номерами.

Существует несколько способов обмена сообщениями при реализации протокола двухфазной фиксации транзакций: централизованный (через узел-координатор), линейный (каждый узел взаимодействует только с узлами, имеющими ближайшие меньшие и большие номера) и распределенный (сообщения пересылаются по схеме «каждый — каждому»). Эти способы имеют свои достоинства и недостатки, которые влияют на живучесть системы, а также количество рассылаемых сообщений или время принятия решения.

Протокол двухфазной фиксации транзакций не гарантирует отсутствие блокировок в некоторых случаях. Блокировка, например, может иметь место и случае, когда на некотором узле истек тайм-аут после отправки своего сообщения на фиксацию транзакции, но так и не получена информация о принятом глобальном решении от координатора, а те узлы, с которыми данный узел может обмениваться сообщениями, также не имеют информации о решении координатора. Подобные проблемы решаются в усовершенствованном варианте — трехфазном протоколе фиксации транзакций.

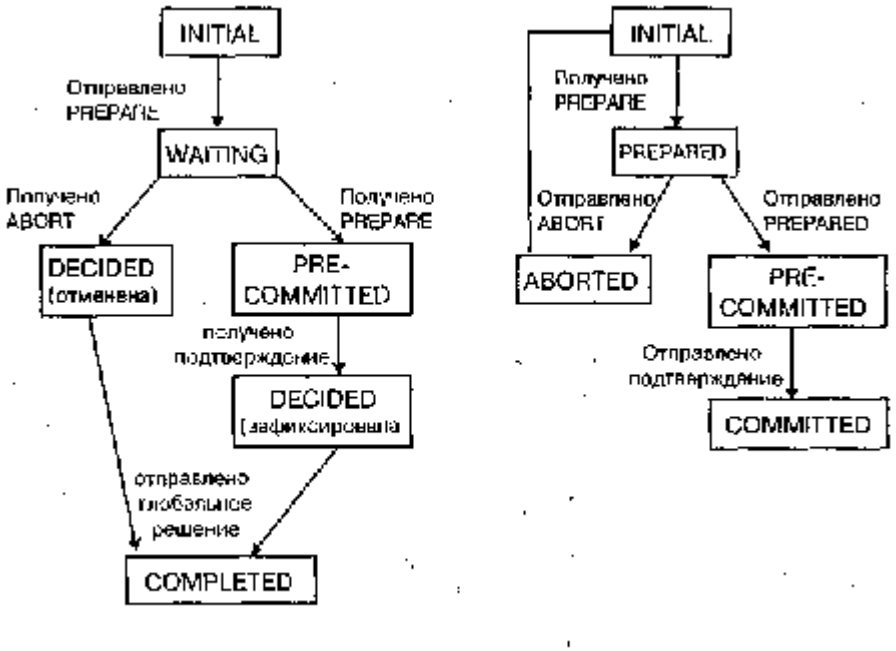
Протокол **трехфазной фиксации транзакций** не приводит к блокировкам и условиях отказа узлов, за исключением случая отказа всех узлов. Однако отказы линий связи могут привести к тому, что на разных узлах будут приняты разные решения, что является нарушением принципа неразрывности глобальной транзакции. Для использования протокола необходимо соблюдение трех условий: сеть является неразделяемой, всегда доступен по крайней мере один узел и отказать одновременно могут не более К узлов сети (такие системы называют К-устойчивыми).

Основным преимуществом протокола является устранение периода ожидания в состоянии неопределенности с момента подтверждения участникам и своего согласия о фиксации транзакции до момента получения от координатора сообщения о фиксации или отмене транзакции. Для этого в

протокол фиксации транзакций введена дополнительная фаза *предфиксации*. Таким образом протокол включает в себя три фазы: голосование, предфиксация транзакции и принятие глобального решения.

В целом взаимодействие координатора с участниками происходит следующим образом. После получения результатов голосования от всех участиков координатор рассылает сообщение о предфиксации PRE-COMMIT, извещающее о готовности всех участников зафиксировать результаты; транзакции. В ответ на него каждый участник подтверждает получение этого сообщения. После приема подтверждений от всех участников координатор рассылает команду глобальной фиксации транзакции. Обработка ситуации, когда некоторый участник потребовал отката транзакции, выполняется так же, как и в протоколе двухфазной фиксации транзакции.

Диаграммы переходов для координатора и участника приведены на рис. 6.9. На этапе предфиксации координатор и участники также переходят на некоторое время в состояние ожидания, однако все функционирующие процессы получают информацию о глобальном решении зафиксировать транзакцию еще до того, как первый процесс выполнит фиксацию результатов транзакции. Это позволяет участникам действовать независимо друг от друга в случае отказа.



а) б)

Рис. 6.9. Диаграммы переходов для протокола трехфазной фиксации транзакций: а) координатор; б) участник

6.4. Информационные системы в локальных сетях

Локальная вычислительная сеть дает пользователю прежде всего возможность более эффективной организации групповых работ и совместного использования аппаратных ресурсов: принтеров, факсов, модемов, сканеров, дисков и т. д., а также программно-информационных ресурсов, в

том числе данных. Рассмотрим основные *схемы организации работы с данными* в ЛВС.

Спектр возможных схем построения информационной системы в ЛВС существенно зависит от возможностей используемых СОС.

Основным сервисом современных ЛВС, независимо от типа управления в них (централизованные или одноранговые), является предоставление доступа одного компьютера (компьютера-клиента) к дискам, каталогам (папкам) и файлам другого компьютера (компьютера-сервера). Так, при обращении к внешнему файлу СОС сначала передает по сети файл (часть файла) в оперативную память компьютера, а по завершении работы с ним при необходимости возвращает обновленную версию. Кроме того, полезной функцией является возможность запуска на компьютере программ, хранящихся на другом компьютере.

Эти возможности примерно в равной мере предоставляются сетевыми ОС такими, как Novell NetWare 3.x (4.x), Windows NT и Windows 95/98. Более слабые сетевые пакеты и утилиты могут предоставлять доступ к информации без автоматического запуска программ. В этом случае пользователь должен сначала скопировать программы и файлы с другого компьютера на спой, после чего запустить программу.

Как и на отдельном компьютере, в ЛВС пользователь может управлять БД с помощью средств некоторой СУБД или работая с приложением. В общем случае приложение может выполняться под управлением СУБД или ее ядра, либо быть независимым и выполняться как автономная программа. В локальной сети персональных ЭВМ выделяют следующие три варианта (издания информационной системы:

- типа файл-сервер;
- типа клиент-сервер;
- основанные на технологии интранет;

Информационные системы типа ***файл-сервер*** можно строить двумя способами:

- с использованием *несетевых* СУБД, предназначенных для применения на отдельной машине;
- с использованием *сетевых* СУБД, разрабатываемых для применения в ЛВС.

Под *сетевой СУБД здесь* понимается система с произвольной моделью данных (не обязательно сетевой), ориентированная на использование в сети.

Программы *несетевой СУБД* и используемые ею данные могут храниться на компьютере-сервере (КС) и на компьютере-клиенте (КК).

Запуск и функционирование несетевой СУБД, хранящейся на КК и работающей с локальными данными, не отличается от обычного режима работы на отдельной ПЭВМ. Если используемые данные хранятся на КС, файловая система сетевой ОС «незаметно» для СУБД выполняет подгрузку нужного файла с удаленного компьютера. Заметим, что не каждая несетевая СУБД без проблем работает в среде любой сетевой ОС.

Если несетевая СУБД используется несколькими пользователями сети, то ее программы, а также БД или ее часть в целях экономии дисковой памяти эффективнее хранить на КС. Хранимую на КС БД будем называть *центральной БД* (ЦБД), а хранимую на КК БД — *локальной БД* (ЛБД). При запуске СУБД в таком варианте на каждый КК обычно пересылается полная копия основной программы СУБД и один или несколько файлов ЦБД (рис. 5.10). После завершения работы файлы ЦБД должны пересылаться с КК обратно на КС для согласования данных.

Существенным *недостатком* такого варианта применения несетевых СУБД является возможность нарушения целостности данных при одновременной работе с одной БД нескольких пользователей. Поскольку каждая копия СУБД функционирует «не зная» о работе других

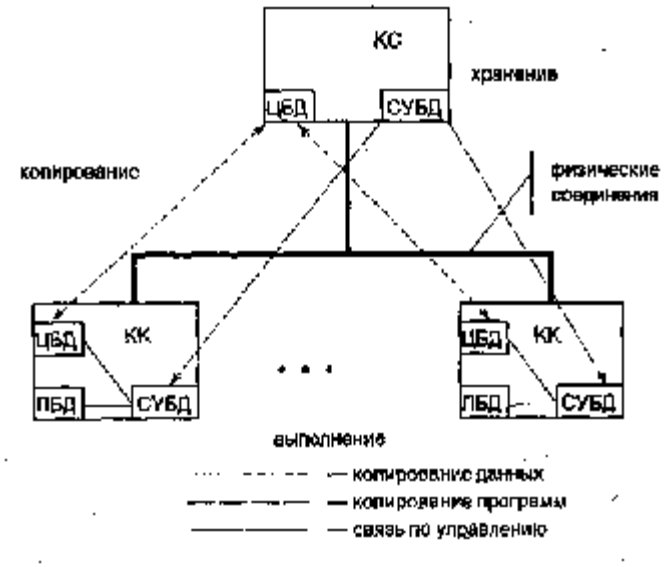


Рис.6.10. Система типа файл-сервер с несетевой СУБД

копий СУБД, то никаких мер по исключению возможных конфликтов не принимается. При этом элементарные операции чтения-записи одних и тех же файлов, как правило, контролирует сетевая ОС.

В качестве примеров несетевых СУБД можно назвать первые версии системы dBase **III** Plus, dBase IY и FoxBase.

Сетевые СУБД не имеют указанного недостатка, так как в них предусматривается «контроль соперничества» (concurrency control). Средства контроля позволяют осуществлять координацию доступа к данным, например, введение м блокировок к файлам, записям и даже отдельным полям записей (рис. 5.11).

В сетевых СУБД с коллективным использованием файлов БД попрежнему вся обработка информации производится на КК, а функции КС сводится к предоставлению большой дисковой памяти. Такой подход нельзя считать эффективным, так как для обеспечения приемлемой скорости процесса обработки информации КК должен обладать высоким быстродействием и иметь большую емкость оперативной памяти. Кроме того, пересылка копий файлов БД и команд управления блокировками по линиям

связи существенно увеличивает нагрузку на подсистему передачи данных, что сникает общую производительность сети.

Примерами сетевых СУБД являются: FoxPro 2.5 для Windows, dBase IY, Paradox 3.5 для DOS.

Информационные системы типа ***клиент-сервер*** отличаются от систем мша файл-сервер прежде всего тем, что программы СУБД функционально разделены на две части, называемые *сервером* и *клиентом.* Между клиентской и серверной частями системы возможны различные варианты распределения функций.

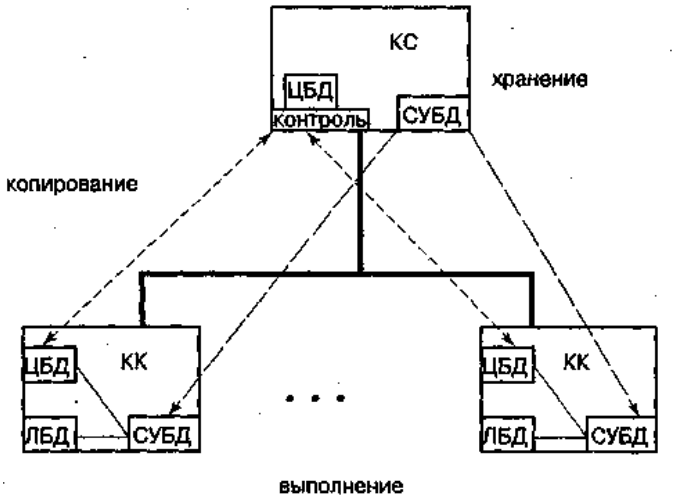


Рис. 6.11. Система типа файл-сервер с сетевой СУБД

Клиент, или *фронтальная программа,* отвечает за интерфейс с пользователем, для чего принимает и проверяет синтаксис вводимых пользователем запросов, преобразует их в команды запросов к серверной части, а при получении результатов выполняет обратное преобразование и отображение информации для пользователя.

В роли клиента выступает пользовательская (разрабатываемая для решения конкретной прикладной задачи программа) или готовая программа, имеющая интерфейс с серверной программой. В качестве готовых клиентских программ могут использоваться текстовые процессоры, табличные процессоры и даже СУБД (например, Access, FoxPro и Paradox).

Сервер является основной программой, выполняющей функции управления и защиты данных в базе. На сервере принимаются запросы к базе со стороны клиентов с проверкой их полномочий, последующим выполнением и возвратом результатов, обеспечивается параллельный доступ к данным, контролируется соблюдение ограничений целостности, реализуются функции управления восстановлением. В случаях, когда вызов функций сервера выполняется на языке SQL, а именно так часто и происходит, его называют ***SQL-сервером.***

В качестве сервера может использоваться ядро профессиональной реляционной СУБД (например, Informix 7.x и Sybase System 10) или некоторый SQL-сервер (например, Novell NetWare SQL и Microsoft SQL Server).

Структуру информационных систем типа клиент-сервер упрощенно можно представить как показано на рис. 5.12.

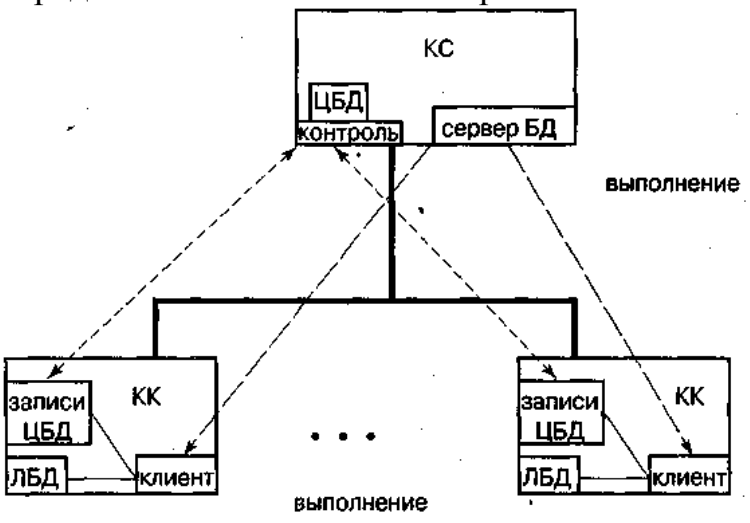


Рис. 6.12. Информационная система типа клиент-сервер

Основная часть обработки информации по формированию запросов, составлению отчетов, представлению данных в удобной для пользователя виде и т. д. выполняется на КК. Полные копии файлов БД из КС на КК и обратно как в случае систем на базе сетевых СУБД) не пересылаются, поскольку для организации полноценного взаимодействия, как правило, достаточно иметь на КК необходимые в данный момент времени записи БД. Все это существенно снижает траффик в сети, ослабляет требования по ресурсам, позволяет создавать более эффективные и надежные информационные системы.

В последнее время на компьютере-сервере, кроме собственно данных, хранят программы обработки данных и запросы. Это обеспечивает увеличение скорости обработки данных (программа обработки либо запрос находится «рядом » с данными), а также эффективность хранения и администрирования программ и запросов общего пользования в одном месте (на компьютере-сервере).

Хранимые на компьютере-сервере программы (процедуры) обработки данных называют ***хранимыми процедурами.***

Разновидностью хранимой процедуры является так называемый ***триггер.*** Триггер (триггерная процедура) автоматически вызывается при возникновении определенных событий в БД. В качестве событий могут быть следующие: операции вставки, обновления и удаления отдельных записей, колонок и полей записей и другие. Примером триггера является программа, запускающая процесс посылки сообщения по электронной почте при достижении размера БД (количества записей) предельного значения.

В БД сервера некоторых систем можно хранить и сами запросы, называемые ***хранимыми командами.*** Совокупность хранимых команд — это поименованная совокупность команд, получаемых в результате компиляции SQL-запроса. Хранимые команды выполняются значительно быстрее,

чем соответствующий SQL-запрос. Основная причина ускорения состоит в том, что при выполнении хранимых команд не требуется синтаксический разбор запросов. Дополнительное ускорение выполнения запросов может быть получено в тех случаях, когда сервер БД не просто сохраняет коды команд запросов, а производит оптимизацию сохраняемого кода.

С хранимыми процедурами и командами связано понятие ***курсора,*** отличающееся от привычного понятия курсора как указателя текущей позиции на экране монитора. В разных СУБД — это близкие, но несколько отличающиеся понятия. Наиболее широко это понятие трактуется в СУБД SQLBase. 1 Здесь курсор может означать следующее:

- идентификатор сеанса связи пользователя с СУБД;
- идентификатор хранимых команд и процедур;
- идентификатор результирующего множества;
- указатель текущей строки в результирующем множестве, обрабатываемом клиентским приложением.

Программы сервера (основные, хранимые процедуры и триггсры) могут быть выполнены как обычные программы (Windows 95/2000), сетевые (Windows NT/2k), либо как специально загружаемые модули сетевой ОС (NLM-модули в сети Novell). Программы клиента в общем случае хранятся на КС или на КК, или в обоих местах.

В настоящее время среди программных продуктов существует огромное количество универсальных (в смысле пригодности работы с различными серверами БД) средств разработки систем типа клиент-сервер, к числу которых относятся: Delphi (Borland), Power Builder (Powersoft), ERwin (LogicWorks), Visual Basic (Microsoft), CA-Visual Objects (Computer Associates), SQL Windows (Gupta) и другие. Кроме того, существуют средства разработки в рамках определенных СУБД (например, для Oracle 7 — Designer/2000). Все подобные средства, как правило, относятся к CASE-системам.

При построении информационных систем типа клиент-сервер возникает проблема доступа со стороны СУБД или приложений, разработанных в одной среде, к данным, порожденным другой СУБД. В среде Windows эта проблема решается с помощью стандартного интерфейса ODBC (Open Database Connectivity — совместимость открытых баз данных) фирмы Microsoft. Основное его назначение заключается в обеспечении унифицированного доступа к локальным и удаленным базам данных различных производителей.

Схема доступа приложений к базам данных с помощью ODBC показана на рис. 5.13. Доступ приложения к данным происходит путем вызова на языке SQL стандартных функций интерфейса ODBC. На компьютере-клиенте при этом должна функционировать операционная система MS Windows с интерфейсом ODBC. /

Взаимодействие приложения с данными производится с помощью менеджера (диспетчера) драйверов, который подключает необходимый

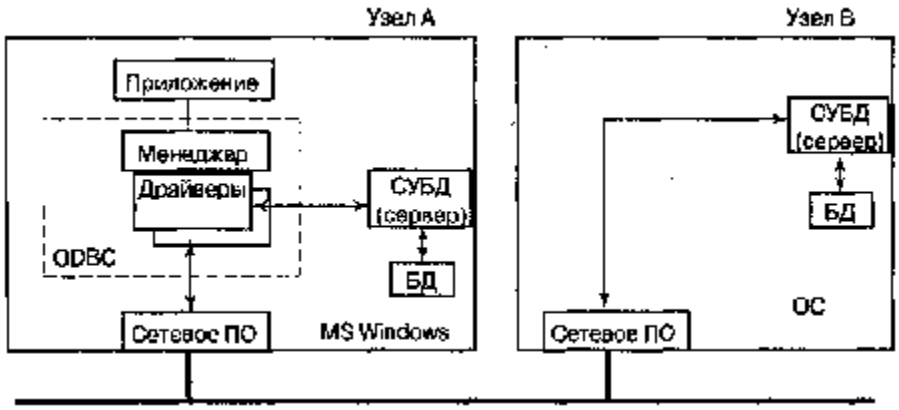


Рис. 6.13. Схема доступа к БД с помощью ODBC

драйвер в соответствии с форматом данных СУБД. Драйвер СУБД, используя сетевые средства, как правило, коммуникационные модули конкретной СУБД, перемет SQL-операторы серверу СУБД. Результаты выполнения запросов на сервере передаются обратно в приложение.

Рассмотренные схемы функционирования СУБД и внешних приложений касаются наиболее типичных вариантов их построения.

6. Методические указания к лабораторным занятиям

Темы лабораторных занятий:

Тема 1. Знакомство со средой Access (2ч)
Зачет по теме 1 (2ч)
Тема 2. Конструктор БД. Создание таблиц. (2ч)
Тема 2. Конструктор БД. Создание форм (2ч)
Тема 2. Конструктор БД. Создание отчетов (2ч)
Зачет по теме 2. (2ч)
Тема 3. Запросы. Простые запросы на выборку(2ч)
Тема 3. Запросы. Запросы на выборку из 2 и более таблиц (2ч)
Тема 3. Запросы. Запросы с группировкой (2ч)
Тема 3. Запросы. Подзапросы (2ч)
Тема 3. Запросы. Модификация таблиц БД (2ч)
Зачет по теме 3.(2ч)
Тема 4. Создание БД. Разработка структуры БД (2ч)
Тема 4. Создание БД. Разработка сложных форм (2ч)
Тема 4. Создание БД. Разработка отчетов на основе запросов (2ч)
Тема 4. Создание БД. Макросы (4ч)
Зачет по теме 4.(2ч)

Лабораторная работа 1.

Тема 1. Знакомство с Microsoft Access

Программа имеет три основных режима работы:

— режим конструктора, в котором создаются и модифицируются объекты базы данных;

— режим запуска, в котором можно выполнять некоторые операции, не открывая базу данных;

— режим выполнения, в котором отображаются окна объектов базы данных. Данный режим имеет различные названия, что зависит от того, с каким объектом работает пользователь. Так, при работе с таблицей этот режим называется режимом таблицы, при работе с формой— режимом формы и т.д.)

В окне базы данных систематизированы объекты базы данных — таблицы, запросы, формы, отчеты, макросы и модули. Изначально окно новой базы данных является пустым.

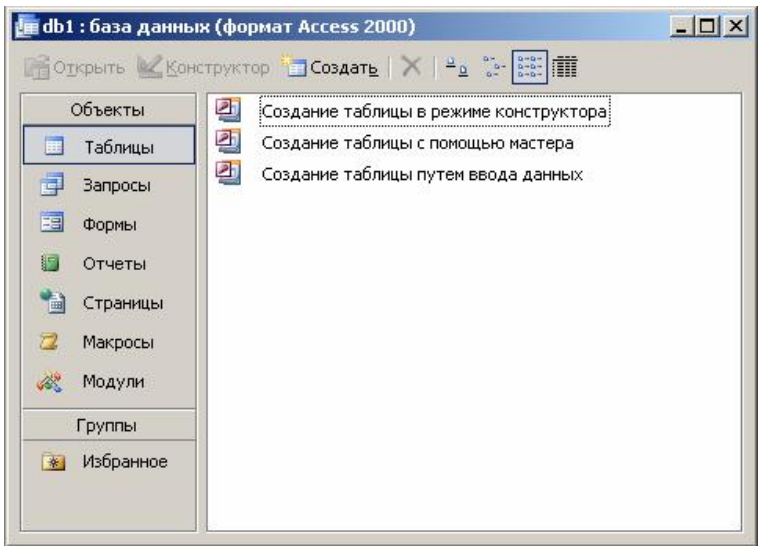


Рис. 1 Окно базы данных

Окно базы данных имеет свою строку заголовка, в которой отображается ее имя.

База данных может содержать множество объектов, отсортированных по категориям и расположенных на разных вкладках ее окна. Access позволяет просмотреть состав базы данных и без перехода с одной вкладки на другую. Полный перечень объектов открытой базы данных находится на вкладке Состав окна свойств, которое открывается в результате активизации команды Свойства базы данных из контекстного меню базы или из меню Файл.

Заполнение таблиц

Новая Access-таблица состоит из одной пустой записи. После ввода данных пустая запись смещается в конец таблицы. Именно в ней осуществляется ввод информации.

На листе данных активная запись обозначается треугольным маркером, а пустая запись — звездочкой. Для обозначения записи, в которой выполняется ввод, используется изображение карандаша. Все маркеры появляются в столбце маркировки, расположенном в левой части листа данных.

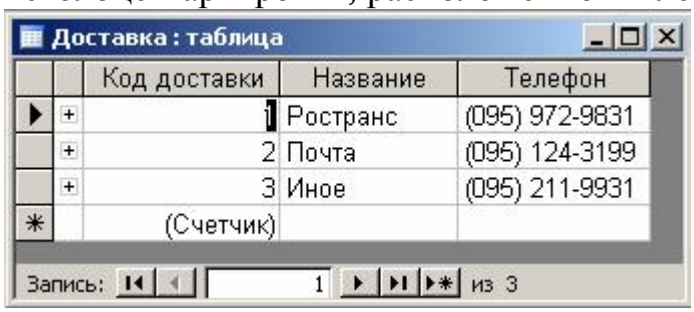


Рис. 2 Маркеры в столбце маркировки

Поле счетчика **(Счетчик)** заполняется автоматически. Access самостоятельно увеличивает значение переменной в этом поле, присваивая каждой записи новый номер.

Запись таблицы активизируется при выполнении на ней щелчка. Поскольку программа автоматически сохраняет каждую запись по завершении ее обработки, необходимости в промежуточном сохранении таблицы нет.

Access позволяет скрыть существующие записи в процессе ввода в таблицу новых данных. Для этого предназначена команда **Ввод данных** из меню **Записи.** Возможность восстановить отображение всех записей предоставляет команда **Удалить фильтр** из меню **Записи.** Меню **Записи** доступно только в режиме заполнения таблицы.

Наиболее удобным средством перемещения по таблице являются кнопки из области **Запись,** расположенной в левом нижнем углу листа данных. С их помощью можно быстро перейти к любому блоку данных. Эти кнопки соответствуют командам из подменю **Перейти** меню **Правка.** Для активизации записи с известным номером достаточно ввести этот номер в специальное **Поле номера записи,** и нажать клавишу [Enter].

Операции поиска

Диалоговое окно поиска открывается в результате активизации команды **Найти** из меню **Правка** или щелчка на кнопке с изображением бинокля.

Существенно расширить круг поиска можно, применив символы подстановки * и ?. Звездочка заменяет любое количество символов, а знак вопроса — только один символ.

По умолчанию в диалоговом окне поиска установлена опция **Только в текущем поле,** вследствие чего поиск осуществляется только в активном поле. Это позволяет быстрее получить результат. Название активного поля появляется в строке заголовка окна поиска. Если необходимо выполнить поиск во всей таблице, следует отключить опцию **Только в текущем поле.** Программа начинает выполнять поиск после нажатия кнопки **Найти.** При обнаружении искомого объекта Access 97 выделяет его, а в строке состояния появляется сообщение *Образец поиска обнаружен.* С помощью кнопки **Найти далее** можно проверить остальные поля на наличие в них объекта поиска.

Другое ограничение сферы поиска устанавливается с помощью опции **С учетом формата записей,** которую можно использовать только совместно с опцией **Только в текущем поле.**

Удаление данных

В Access для удаления данных предназначена команда **Удалить** из меню **Правка.** Удаляемую запись необходимо маркировать, иначе указанная команда не будет доступна. Выделение записей осуществляется посредством колонки маркировки.

Для удаления маркированных записей наряду с командой **Удалить** из меню **Правка** можно применять клавишу [Delete]. После попытки удалить запись программа открывает окно для подтверждения удаления, так как удаленные данные будут безвозвратно утеряны.

Изменение размеров полей таблицы

Внешний вид таблицы мало зависит от ее структуры. Его можно изменить, не изменяя структуру таблицы и не переходя в режим конструктора.

Ширина колонок устанавливается на листе данных путем перемещения разделительных линий в области заголовков полей. В режиме изменения ширины столбцов указатель мыши приобретает вид двунаправленной стрелки с вертикальной чертой посредине.

Чтобы, учитывая объем данных в поле, подобрать для него оптимальную ширину, надлежит установить указатель мыши на правой границе заголовка столбца и выполнить двойной щелчок. Можно также воспользоваться командой **Ширина столбца** из меню Формат и в окне **Ширина столбца** нажать кнопку **По ширине данных.** В поле **Ширина столбца** определяется точная ширина поля. В нем указывается количество символов, отображаемых на экране в маркированном поле.

Отображение полей

В таблицах с большим количеством полей часть информации не видна на экране. С целью решения этой проблемы можно уменьшить размер шрифта или упорядочить поля, сгруппировав самые необходимые.

Еще одно решение заключается в отмене отображения ненужных в данный момент полей. Скрытые поля легко сделать видимыми, если понадобится информация, содержащаяся в них.

Отображение одного или нескольких полей отменяется таким образом:

- Выделите поля, отображение которых вы хотите отменить.
- Выберите команду **Скрыть столбцы** в меню **Формат.**

Для восстановления отображения скрытых полей предназначена команда **Отобразить столбцы** из меню **Формат.** Диалоговое окно этой команды содержит список полей активной таблицы, в котором видимые поля отмечены. В этом окне можно как отменить, так и восстановить отображение полей — достаточно удалить или установить контрольный индикатор напротив их имен, а затем закрыть окно с помощью кнопки **Закрыть.**

Другой способ повышения удобочитаемости таблицы — фиксация полей. Зафиксированные поля всегда отображаются на экране в левой части таблицы, они не смещаются при перемещении по ней. Рекомендуется фиксировать поля, содержащие сведения, которые нужны пользователю постоянно.

Фиксация полей осуществляется следующим образом:

- Маркируйте поле в таблице, которое вы хотите зафиксировать.
- Выберите в меню **Формат** команду **Закрепить столбцы.** После активизации указанной команды маркированное поле перемещается в левую часть таблицы и даже после перехода в последние поля остается видимым.

Отменить фиксацию позволяет команда **Освободить все столбцы** меню **Формат.** После отмены фиксации автоматический возврат поля в его исходную позицию не выполняется. Пользователь должен сам переместить его с помощью мыши.

Фильтрация данных в форме

Фильтры можно использовать в тех же целях, что и запросы на выборку данных, однако фильтры уступают запросам по функциональным возможностям: при фильтрации нельзя подавить отображение отдельных полей и выполнить вычисления и, наконец, фильтр позволяет только отобрать и отсортировать нужные записи.

Существует три способа фильтрации данных в форме:

- **Обычный фильтр** — отбор записей по содержимому нескольких

полей (критерий отбора формируется с помощью логического оператора или).

— **Фильтр по выделенному фрагменту** — фильтрация путем выделения данных.

— **Расширенный фильтр** — построение расширенного фильтра.

ЗАДАНИЯ

1. Откройте учебную базу «Борей». Определите состав базы данных. Определите способ переключения между режимами работы с базой данных.
2. Занесите данные в таблицу «Клиенты»: непосредственно в таблицу и через форму «Клиенты».
3. Найти в таблице «Клиенты» всех клиентов из города Лондон 2-мя способами: через поиск и с использованием фильтра.
4. Выполните задание 3, находясь в форме «Клиенты».
5. Просмотрите структуру таблицы «Товары», определите, какие в поля и какие значения подставляются по умолчанию. Измените одно из этих значений. В режиме выполнения проверьте реализацию.
6. Измените вид таблицы «Клиенты» таким образом, чтобы были видны только следующие столбцы: «Название», «Город», «Адрес», «Телефон», именно в таком порядке. Отобразите скрытые поля.
7. Измените вид таблицы «Клиенты» таким образом, чтобы при прокрутке таблицы вправо на экране оставались видны поля «Название» и «Город».
8. С помощью мастера сводных диаграмм (меню «Вид», подменю «Сводная диаграмма») постройте диаграмму оставшихся на складе кондитерских изделий. Информация об этом хранится в таблице «Товары».
9. С помощью мастера сводных таблиц (меню «Вид», подменю «Сводная таблица») постройте таблицу о заказах, размещенных в период с 1 по 15 декабря 1996г. Таблица должна содержать информацию о стоимости доставки по каждому клиенту и по каждому сотруднику, оформившему сделку. Информация об этом хранится в таблице «Заказы».

Лабораторная работа 2.

Тема 2. Конструктор БД. Создание таблиц.

В реляционных базах данных вся информация хранится в виде таблиц. Каждая строка таблицы представляет собой запись, а столбец — поле. Запись содержит набор данных об одном объекте, а поле — однородные данные о всех объектах (например, адреса всех занесенных в таблицу фирм).

Каждая таблица состоит из записей и полей. Количество полей в записи и их тип определяются в процессе конструирования таблицы. Например, на этапе составления таблицы адресов необходимо создать поля для фамилий,

адресов, почтовых индексов, названий населенных пунктов и т.п.

Величина полей устанавливается пользователем. В соответствии с ней ограничивается количество символов, вводимых в поле при заполнении таблицы. Следите за тем, чтобы размер поля не был слишком велик, так как при этом бесполезно расходуется память (она резервируется в полной мере независимо от того, заполняется поле полностью или частично).

Существует несколько способов построения таблиц: на листе данных, в окне конструктора таблиц и с помощью мастера таблиц.

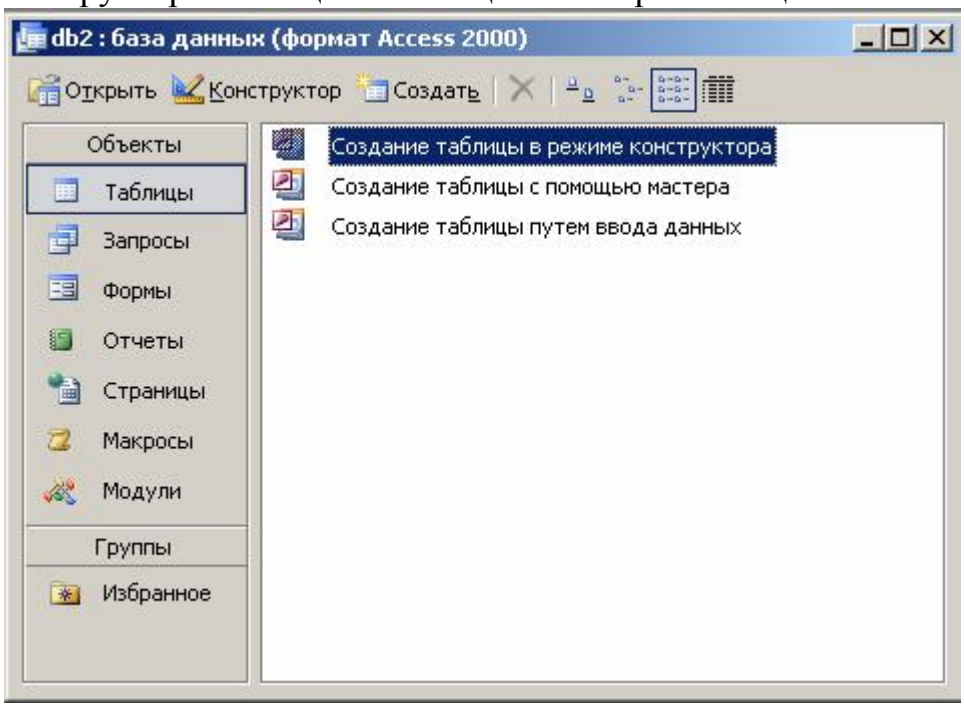


Рис.1. Варианты создания новой таблицы БД

В результате выбора способа **Конструктор** открывается окно структуры таблицы

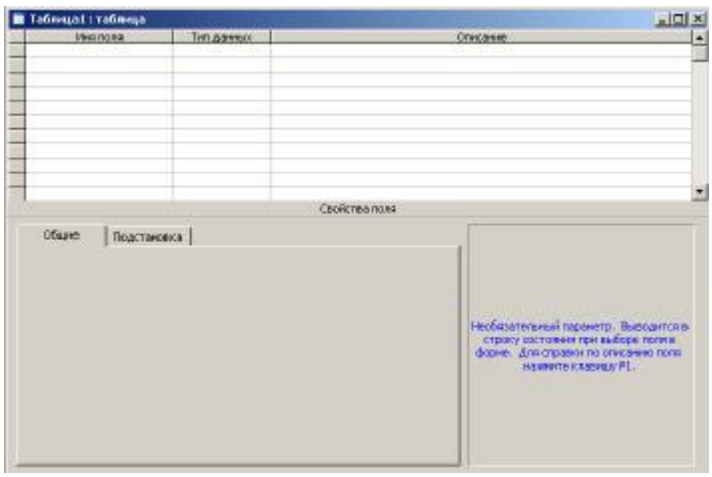


Рис. 2. Окно конструктора таблицы

Состав таблицы определяется в области проекта таблицы, который состоит из трех колонок — **Имя поля, Тип данных** и **Описание. В** первой колонке указывается имя поля, во второй — его тип, а в третьей вводится инфор-

мация о назначении поля.

Обязательным условием нормальной работы с базой данных является однозначная идентификация каждой записи. Для этого каждой записи Access автоматически назначает индивидуальный номер. Автоматическая нумерация записей выполняется в том случае, если таблица включает поле с типом данных *Счетчик*. Чтобы значение в таком поле увеличивалось на 1 для каждой новой записи, свойство *Новые значения* должно иметь значение *Последовательные*.

Поле первичного ключа помечается специальным маркером. *Ключевое поле* — это одно или несколько полей таблицы, однозначно определяющих содержимое других полей. Большинство таблиц содержит только одно поле первичного ключа, по которому выполняется индексация таблицы.

Формат поля — определяет способ отображения содержимого поля на экране и на бумаге после печати. В формате *С разделителями разрядов* Access автоматически устанавливает пробел в качестве разделителя тысяч (триад), запятую в качестве знака, отделяющего дробную часть числа от целой, и задает точность — два разряда после запятой. При выборе формата *Основной* (Общепринятый формат числа) число отображается так, как его ввел пользователь.

Свойство **Маска ввода** обеспечивает ввод данных в указанном формате. В состав Access входит мастер ввода масок, который запускается щелчком мыши на кнопке с тремя точками, появляющейся в строке **Маска ввода** после установки в ней курсора ввода.

Если значение свойства **Подпись** не задавать, то в формах и отчетах в качестве названий полей система использует имена полей таблицы. -

Свойство **Обязательное поле** позволяет указать, что данное поле должно быть обязательно заполнено.

Индексированное поле: при индексации данного поля время на поиск записи по его содержимому существенно сокращается. Вполне вероятно, что поиск в базе данных будет осуществляться по содержимому поля кода модели.

Установка поля первичного ключа

После определения всех полей таблицы следует указать по меньшей мере одно поле для использования в качестве поля первичного ключа, что не позволит вводить в таблицу повторяющиеся записи, поскольку поле первичного ключа содержит однозначный идентификатор для каждой записи. Это поле не может содержать одинаковое значение для двух различных записей.

Значения в поле таблицы могут подставляться из фиксированного списка значений или из другой таблицы. Реализовать это можно указав значение подстановки для поля. Выбор типа источника значений осуществляется в поле «Тип источника строк». В случае подстановки из фиксированного списка значений, значения указываются в поле «Источник строк» через точку с запятой. В случае подстановки из таблицы указывается имя таблицы.

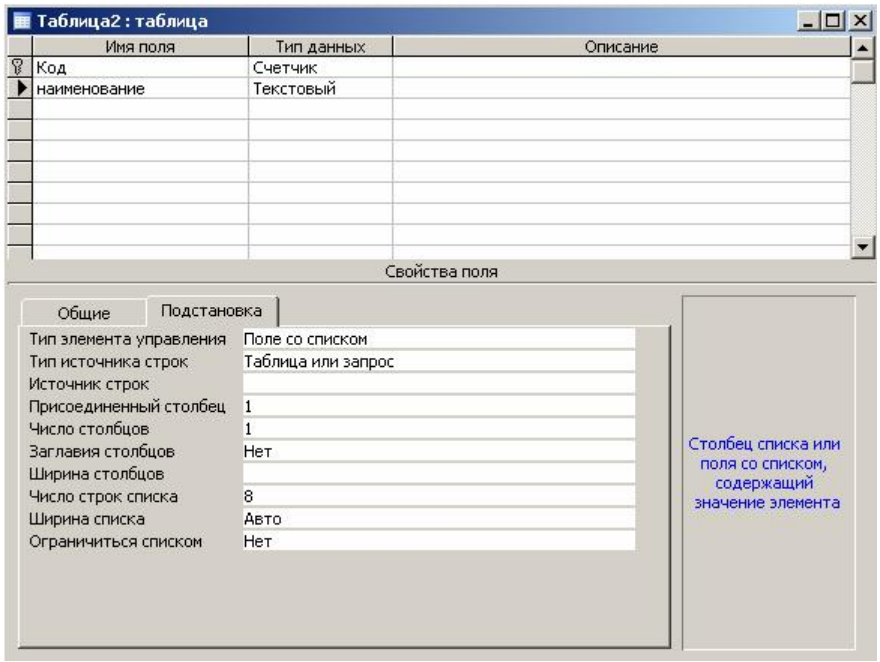


Рис.3. Использование подстановки.

ЗАДАНИЯ

1. Создайте новую базу данных.
2. Создайте таблицу «Сотрудники», имеющую следующий формат:

Имя поля	**Тип данных**	**Размер ПОЛЯ**	**Формат ПОЛЯ**	**Индексированное поле**
Фамилия	Текстовый	30		Да (Допускаются совпадения)
Имя	Текстовый	30		
Адрес	Текстовый	30		
Почтовый индекс	Числовой	Длинное целое		
Населённый пункт	Текстовый	25		
Телефон	Текстовый	15		
Телефакс	Текстовый	15		
День рождения	Дата_время		Краткий формат	

Заполните таблицу, внеся в нее 3-4 новые записи.

3. Измените таблицу «Сотрудники». Добавьте поле «пол», значения которого «Муж» или «Жен» будут выбираться из поля со списком.
4. Создайте таблицу «Филиалы фирмы», используя способ «Путем ввода данных», определив самостоятельно состав полей таблицы. В конструкторе задайте обязательные поля.
5. Измените таблицу «Сотрудники». Добавьте поле «Филиал» таким

образом, чтобы его значение выбиралось из списка филиалов из таблицы «Филиалы».

6. Измените таблицу «Сотрудники». Добавьте условие на ввод данных даты рождения: дата рождения не может быть больше текущей даты.

Лабораторная работа 3.

Тема 2. Конструктор БД. Создание форм

Форма представляет собой бланк, подлежащий заполнению, или маску, накладываемую на набор данных. Форма-бланк позволяет упростить процесс заполнения базы, что дает возможность поручить ввод информации персоналу невысокой квалификации. С помощью формы-маски можно ограничить объем информации, доступной пользователю, обращающемуся к базе.

Формы используются для достижения комфорта в работе с Access и при обработке базы данных несколькими пользователями. Форма может служить средством защиты базы от действий неквалифицированных пользователей, а также ширмой, заслоняющей от любопытных глаз конфиденциальную информацию.

В Access существует несколько способов создания форм;

— **Мастер форм** — создание формы с помощью мастера; в зависимости от назначения формы мастер предлагает на выбор стандартные шаблоны и стили оформления.

— **Конструктор** — создание формы на основе пустого бланка при помощи инструментальных средств конструктора форм.

— **Мастер диаграмм** — создание формы с диаграммой на основе выбранных полей таблицы. ,

— **Мастер сводных таблиц** — создание сводной таблицы на основе таблиц или запросов Access.

Формы создаются на основе таблиц и запросов. При каждом открытии сохраненной формы обновляются данные запроса, на основе которого создается форма. Благодаря этому содержимое формы всегда соответствует информации в таблицах и запросах.

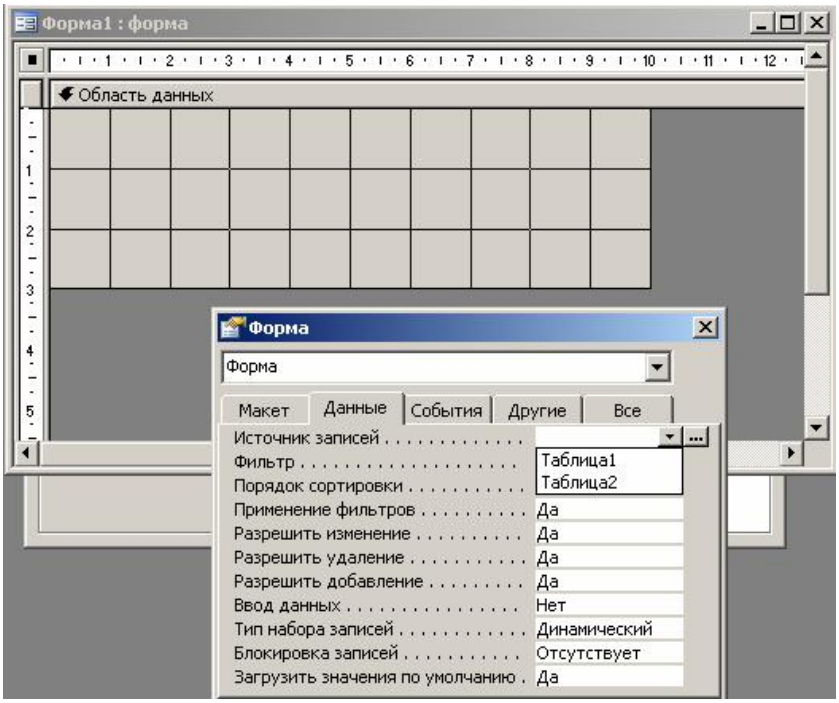
Для обработки готовых форм предназначен конструктор форм.

Существуют различные виды форм. Вы можете ознакомиться *с* ними в следующей таблице.

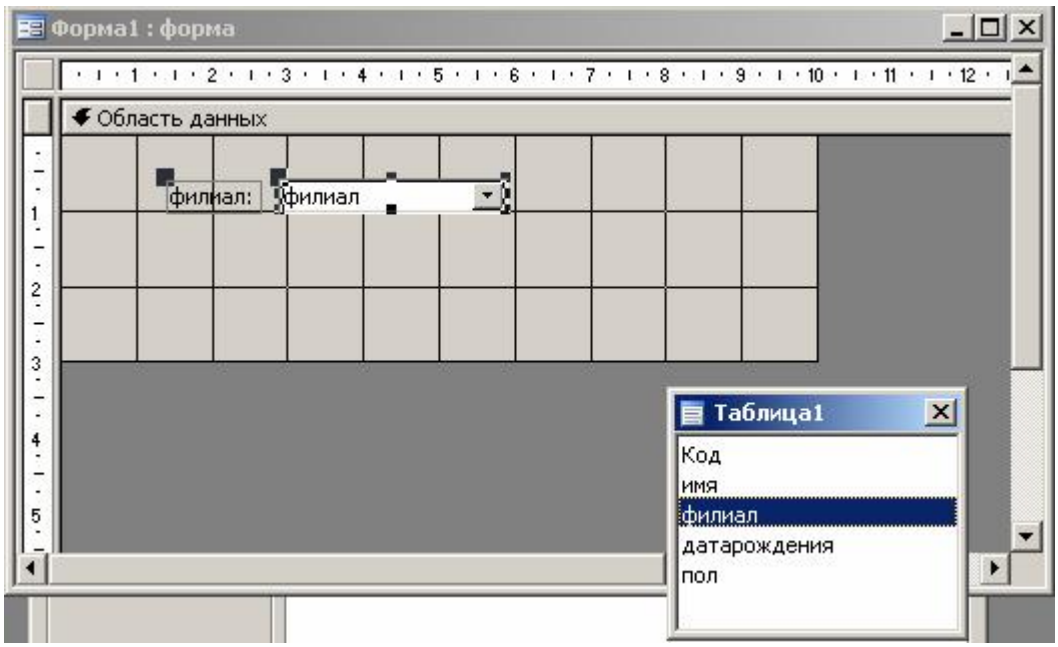
Вид формы	Особенности
Автоформа: в столбец	Для каждой записи отводится отдельная страница формы. Подходит для записей с большим числом полей
Автоформа: ленточная	Каждая запись размещается в одной строке таблицы. Позволяет наглядно представить нескольких записей

Автоформа: табличная	Имеет вид таблицы и выступает в качестве подчиненной формы в составных формах
Сводная таблица	Форма с итоговыми данными, полученными с помощью мастера сводных таблиц Microsoft Excel
Диаграмма	Диаграмма создается на основе числовых значении одного или нескольких полей таблицы

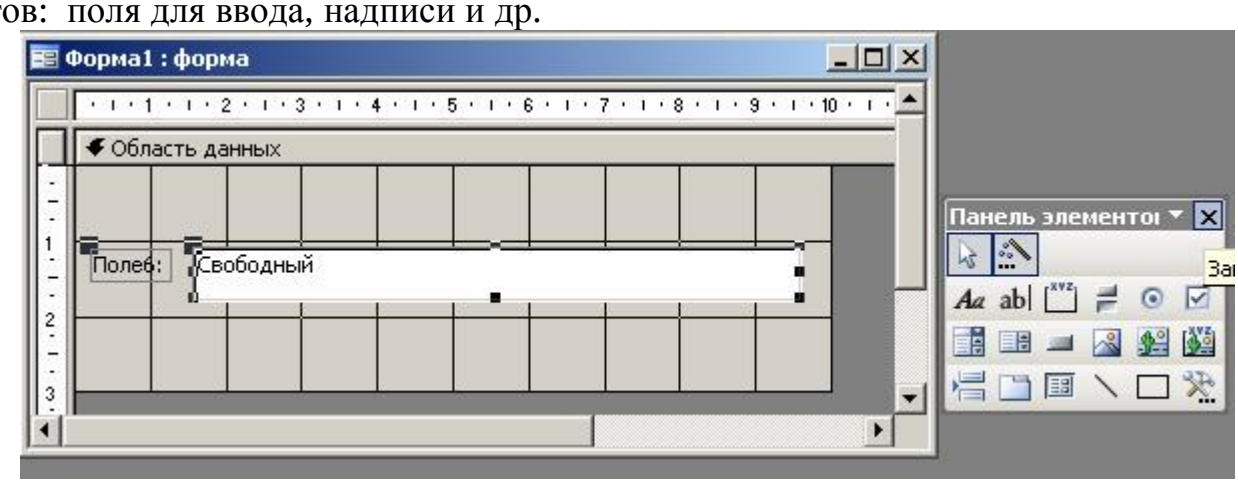
При создании формы с помощью **Конструктора** необходимо указать источник данных формы. Его можно указать в свойствах формы на вкладке «Данные».



Из открывшегося после выбора таблицы списка полей, можно перетащить на форму необходимые поля.



Также на форму можно добавлять другие элементы из таблицы элементов: поля для ввода, надписи и др.



Значения этих полей можно также задавать с помощью установки свойств поля, вкладка «Данные», например, с использованием мастера функций.

ЗАДАНИЯ

1. Создайте разные типы форм для таблицы «Сотрудники» с помощью мастера форм.
2. Создайте простую форму для таблицы «Сотрудники» с помощью конструктора форм. Добавьте на нее поле, содержащее текущую дату.
3. Задайте условия на ввод данных в поля формы.
4. Измените таблицу «Сотрудники», добавьте в нее поле «Наличие авто». Добавьте на форму флажок «Наличие авто», в котором будет отражаться содержимое поля. Посмотрите, как зафиксировался результат ввода данных в таблице.
5. Создайте таблицу «Товары», содержащую обязательные поля «Наименование» и «Цена». Состав остальных полей подберите самостоятельно.

6. Создайте форму «Товары» для таблицы «Товары» с помощью конструктора. Добавьте на форму поле «Цена со скидкой», в котором для каждого товара будет отражаться его цена со скидкой 30%. (Примечание: используйте конструктор формул)
7. Измените форму «Товары» таким образом, чтобы размер скидки можно также было задавать на форме.

Лабораторная работа 4.

Тема 2. Конструктор БД. Создание отчетов

Отчеты служат для отображения итоговых данных из таблиц и запросов в удобном для просмотра виде. В Access существуют разнообразные способы оформления отчетов.

При создании отчета с помощью мастера необходимо проделать следующие шаги:

1. Запустить мастер отчетов.
2. Выбрать таблицу (или запрос), на основе которых будет построен отчет, выбрать поля таблицы (запроса), которые войдут в отчет.
3. Выбрать уровни группировки (если необходимо). Данные будут сгруппированы по значениям выбранных полей.
4. Задать способ сортировки
5. Выбрать стиль отчета.

Мастер отчетов позволяет создавать стандартные итоговые отчеты, Составив такой отчет, вы сможете отредактировать его и изменить оформление по своему усмотрению. Создание нестандартных и более сложных отчетов лучше начинать с незаполненного шаблона.

При создании отчета с помощью конструктора открывается окно, разделенное на области: верхний и нижний колонтитулы, в которых выводится информация заголовка и подвала отчета, и область данных – многократно повторяющаяся область, содержащая непосредственно данные отчета.

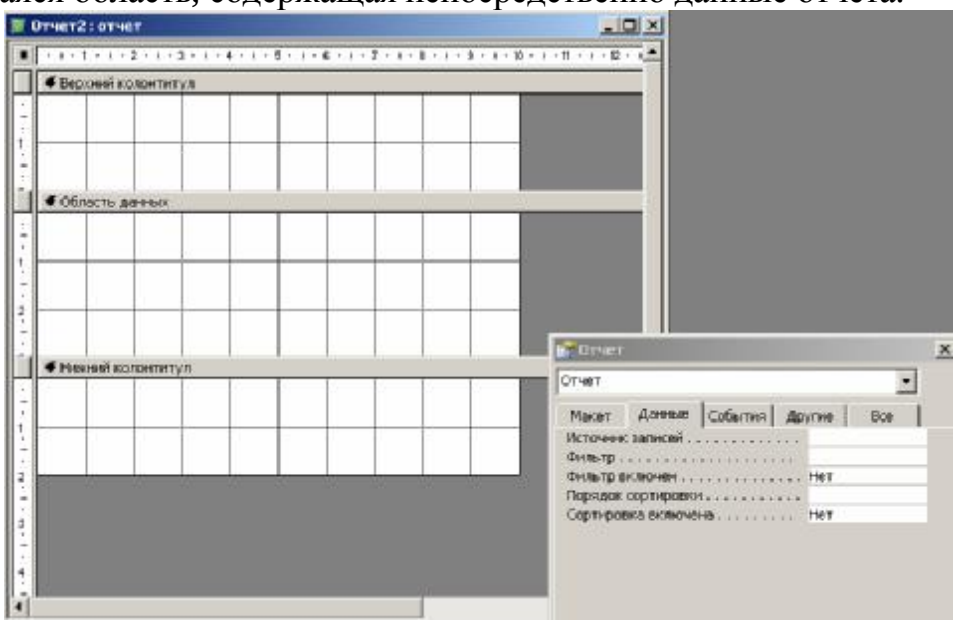


Рис.1. Макет отчета

Как и на форму, на макет отчета можно перетащить поля таблицы, указанной как источник данных. Поля, размещенные в областях верхнего и нижнего колонтитулов будут выведены единожды и должны содержать агрегированную информацию. Поля, размещенные в области данных, будут выведены столько раз, сколько строк содержит источник данных.

ЗАДАНИЯ.

1. Дополнить таблицу «Товары» полем «Тип товара», значения которого будут выбираться из фиксированного списка.
2. Создать отчет по товарам, используя мастер отчетов. Отчет должен быть выведен по типам товаров.
3. Создать аналогичный отчет с помощью конструктора. В заголовок отчета поместить шапку «Отчет по товарам на <дату>» и подвал отчета «Составитель».
4. Добавьте в отчет количество товаров каждой группы.

Лабораторная работа 5.

Тема 3. Запросы. Простые запросы на выборку(2ч)

Запросы предназначены для просмотра, изменения и анализа данных. Они используются также в качестве источника записей при создании форм и отчетов. Одним из наиболее распространенных запросов является запрос на выборку, который выполняет отбор данных из одной или нескольких таблиц в соответствии с заданными пользователем критериями.

Запросы можно создавать в режиме мастера, в режиме конструктора, а также непосредственно на языке SQL.

В окне конструктора запросов появится небольшое окно с заголовком таблицы и списком ее полей. По двойному щелчку на поле таблицы имя этого поля будет помещено в строку **Поле** бланка запроса (нижняя половина окна конструктора запросов). В бланке запроса указываются параметры запроса и данные, которые нужно отобрать, определяется способ их отображения на экране. В Access существует несколько способов сортировки данных, отобранных посредством запроса. Быстро выполнить сортировку в окне запроса позволяют команда **Сортировка** из меню **Записи,** а также кнопки **Сортировка по возрастанию** и **Сортировка по убыванию** панели инструментов **Запрос в режиме таблицы.** Для этого следует включить в запрос поля таблицы, по которым будут рассортированы записи, и определить способ сортировки. Данные можно упорядочить по алфавиту, а также по убыванию или возрастанию. При алфавитно-цифровой сортировке по возрастанию данные располагаются в таком порядке: сначала — элементы, начинающиеся со знаков пунктуации или специальных символов, затем — элементы, начинающиеся с цифр, а затем — элементы, которые начинаются с букв.

Написание запросов на языке SQL.

Для выборки информации из одной или более таблиц базы данных используется команда SELECT. Это исключительно мощная команда, способная выполнять действия, эквивалентные операторам реляционной алгеб-

ры selection, projection и join, причем в пределах одной выполняемой команды. Общий формат команды SELECT имеет вид:

```
SELECT [DISTINCT | ALL] {* | [column [AS new_name]] [, …]}
FROM table_name [alias] [, …]
[WHERE condition]
[GROUP BY column_list] [HAVING condition]
[ORDER BY column_list]
```

Здесь параметр *column* представляет собой имя столбца или выражение из нескольких имен. Параметр *table_name* является именем существующих в базе данных таблицы или представления, к котрым необходимо получить доступ. Необязательный параметр *alias* – это сокращение, устанавливаемое для имени таблицы *table_name*. Обработка элементов команды SELECT выполняется в следующей последовательности:

FROM	Определяются имена используемой таблицы или нескольких таблиц
WHERE	Выполняется фильтрация строк объекта в соответствии с заданными условиями
GROUP BY	Образуются группы строк, имеющих одно и то же значение в указанном столбце
HAVING	Фильтруются группы строк объекта в соответствии с указанным условием
SELECT	Устанавливается, какие столбцы должны присутствовать в выходных данных
ORDER BY	Определяется упорядоченность результатов выполнения оператора

Порядок предложений и фраз в команде SELECT *не может* быть изменен. Только два ключевых слова команды – SELECT и FROM – являются обязательными. Команда SELECT является замкнутой: результатом запроса к таблице является друга таблица.

Запрос может иметь параметр – переменную запроса, которую заносит пользователь перед началом выполнения запроса. Так, если занести параметр [сотр] (имя переменной-параметра всегда находится в квадратных скобках), а в тексте SQL-запроса указать WHERE (((КодСотрудника)=[сотр])), при выполнении запроса пользователю будет предложено занести данные в переменную (например, 3), а в запрос попадут только тот сотрудник, код которого=3. Чтобы добавить параметр в запрос необходимо: в режиме конструктора – указать параметр в таблице параметров в меню «запрос», подменю «параметры»; в режиме SQL – добавить строку PARAMETERS [имя] <тип> перед оператором SELECT.

ЗАДАНИЯ

1. Откройте базу данных «Борей». Создайте запрос на выборку всех клиентов, находящихся в Лондоне или Париже.
2. Измените запрос из задания 1 таким образом, чтобы выбирались все клиенты из Лондона и Парижа, должность которых – Продавец.
3. Просмотрите получившийся запрос в режиме SQL. Измените его так, чтобы выбирались все клиенты, либо проживающие в Лондоне, либо имеющие должность «Продавец».
4. **В режиме SQL** создайте запрос, содержащий информацию о заказах, размещенных в первом квартале 1996 года.
5. **В режиме SQL** создайте запрос, содержащий информацию обо всех сотрудниках, фамилия которых начинается на букву «К».
6. Измените запрос из задания 4 таким образом, чтобы выбирались заказы, сделанные конкретным сотрудником (сотрудника задайте самостоятельно).
7. В режиме конструктора создайте запрос на вывод всех товаров одного типа, тип выбирается пользователем.

Лабораторная работа 6.

Тема 3. Запросы. Запросы на выборку из 2 и более таблиц

Запросы к нескольким таблицам

Запрос для таблиц, между которыми установлена связь, создается следующим образом:

- Откройте окно конструктора запросов, нажав кнопку **Создать** в разделе **Запросы** окна базы данных.
- Выполните двойной щелчок на имени главной таблицы в диалоговом окне **Добавление таблицы,** вследствие чего она будет помещена в QBE-область.
- Повторите эту операцию для подчиненной таблицы.
- Закройте диалоговое окно Добавление **таблицы** 'посредством кнопки **Закрыть.**
- Включите в запрос необходимые поля.
- Запустите запрос.

Для определения связей между таблицами используется Схема Данных (меню «Сервис», подменю «Схема данных»).

Написание запросов на языке SQL.

Для отображения данных из 2 или более связанных таблиц необходимо задать простое условие соединения в предложении WHERE

SELECT *таблица.столбец, таблица.столбец*
FROM *таблица1, таблица2*
WHERE *таблица1.столбец1=таблица2.столбец2*

где *таблица.столбец* – таблица и столбец, из которых производится выборка данных;

таблица2.столбец2 – условие, соединяющее таблицы

таблица1.столбец1 (или задающее их взаимосвязь)

ЗАДАНИЯ

1. Создайте запрос в режиме конструктора, содержащий номер заказа, клиента, адрес клиента, город клиента для всех заказов, размещенных в августе 1996г.
2. **В режиме SQL** создайте запрос, содержащий информацию о сотрудниках и размещенных ими заказах.
3. **В режиме SQL** создайте запрос, содержащий информацию о содержании заказов клиента Wartian Herkku

Лабораторная работа 7.

Тема 3. Запросы. Запросы с группировкой

Написание запросов на языке SQL.

По умолчанию, все строки таблицы рассматриваются как одна группа. Для разбиения таблицы на меньшие группы строк используется предложение GROUP BY. Кроме того, для отбора возвращаемых групп используется предложение HAVING.

Синтаксис:

SELECT *столбец, групповая функция*
FROM *таблица*
[WHERE *условие*]
[GROUP BY *выражение_группирования*]
[HAVING *условие_группы*]
[ORDER BY *столбец*]

выражение_группирования задает столбец, на основе значения которого группируются строки,

условие_группы включает в выходной результат только те группы, для которых заданное условие истинно.

групповая функция – одна из нижеследующих:

Функция	**Описание**
AVG(DISTINCT/ALL n)	Среднее значение n
COUNT(DISTINCT/ALL/*выражение*/*)	Количество строк только с определенными результатами вычисления *выражения*. По * подсчитываются все строки, включая повторяющиеся значения и строки с неопределенным значением
MAX(DISTINCT/ALL/*выражение*)	Максимальное значение *выражения*
MIN(DISTINCT/ALL/*выражение*)	Минимальное значение *выражения*
STDDEV(DISTINCT/ALL/*n*)	Стандартное отклонение n
SUM(DISTINCT/ALL/*n*)	Сумма значений n

VARIANCE(DISTINCT/ALL/*n*)	Дисперсия n

Если задано слово DISTINCT, функция учитывает только неповторяющиеся значения, при наличии слова ALL (этот вариант принимается по умолчанию) рассматриваются все значения.

Все групповые функции, кроме COUNT, игнорируют неопределенные значения. Для подстановки значения вместо неопределенного используют NVL.

ЗАДАНИЯ.

1. В режиме SQL создайте запрос, подсчитывающий количество заказов, оформленных каждым сотрудником.
2. В режиме SQL создайте запрос, подсчитывающий количество товаров каждого типа.
3. В режиме SQL создайте запрос, стоимость каждого заказа, размещенного в январе 1996г.
4. Создайте запрос, подсчитывающий общую стоимость всех заказов для каждого клиента.

Лабораторная работа 8.

Тема 3. Запросы. Подзапросы

Подзапрос – команда SLECT, вложенная в предложение другой команды SQL. Подзапросы полезны при написании команд SELECT для выборки значений по неизвестному условному значению. Подзапросы можно использовать в разных предложениях команд SQL:

- предложение WHERE
- предложение HAVING
- предложение FROM команды SELECT или DELETE.

Синтаксис:

SELECT *список_выбора*
FROM *таблица*
WHERE *выражение оператор*
(SELECT *список выбора*
FROM *таблица*)

где *оператор* – оператор сравнения (например, >, = или IN)

Подзапрос выполняется первым, а результат его используется для полного определения условия во внешнем запросе.

Указания:

- подзапрос должен быть заключен в скобки;

- подзапрос должен находиться после оператора сравнения;
- в подзапросе нельзя использовать предложение ORDER BY. На каждую команду SELECT разрешается только одно предложение ORDER BY, и если оно используется, то должно быть последним в главной команде SELECT.

ЗАДАНИЯ

1. Подсчитайте количество товаров, цена которых равна 1000р., 2000р или 3000р.
2. Выведите все заказы, сделанные клиентами из Лондона.
3. Подсчитайте количество заказов, сделанных клиентами из Лондона.
4. Подсчитайте количество заказов, оформленных сотрудниками, подчиненными Кротову Андрею.

Лабораторная работа 9.
Тема 3. Запросы. Модификация таблиц БД

Модификация данных может выполняться с помощью предложений **DELETE** (удалить), **INSERT** (вставить) и **UPDATE** (обновить).

Предложение DELETE имеет формат:
DELETE
FROM *имя_таблицы*
[WHERE *условие*]

Например:
DELETE FROM dept WHERE deptno=13;
Если предложение WHERE опущено, удаляются все строки таблицы.

Предложение INSERT имеет один из следующих форматов:
INSERT
INTO *имя_таблицы*
[(*столбец* [, *столбец*]……)]
VALUES *значения*
Или
INSERT
INTO *имя_таблицы*
[(*столбец* [, *столбец*]……)]
подзапрос

В первом формате в таблицу вставляется строка со значениями полей, указанными в перечне фразы VALUES (значения), причем i-е значение соответствует i-му столбцу в списке столбцов; столбцы, не указанные в спи-

ске, заполняются NULL-значениями. Если в списке VALUES указаны все столбцы модифицированной таблицы, и порядок их перечисления соответствует порядку столбцов в описании таблицы, то список столбцов в фразе INTO можно опустить.

Предложение UPDATE также имеет два формата:

UPDATE *имя_таблицы*
SET *столбец=значение* [,*столбец=значение*…..]
[WHERE *условие*]
или
UPDATE *имя_таблицы*
SET *столбец*= (*подзапрос*)
[WHERE *условие*]

В первом формате *значение* – это столбец/выражение/константа, может включать столбцы лишь из обновляемой таблицы.

Второй формат описывает предложение, позволяющее производить обновление значений модифицируемой таблицы по значениям столбцов из других таблиц.

В режиме конструктора запрос на модификацию таблиц выполняется таким же образом, как и запрос на выборку, но необходимо указать тип запроса в меню «Запрос» (по умолчанию установлен тип «Выборка»)

ЗАДАНИЯ

С помощью запроса в режиме SQL выполните следующее:

1. Из таблицы заказов удалить все заказы, сделанные 1.01.1996
2. В таблице заказов заменить значение «Почта» на значение «Экспресс-доставка»
3. Добавьте новую строку в таблицу «Типы».
4. Для всех товаров, цена которых ниже 500р. прекратите поставки.
5. Всем товарам, поставляемым «АО Германия-Россия» задайте значение «ожидается» равным 10

Лабораторная работа 10.

Тема 4. Создание БД. Разработка структуры БД

Access позволяет строить реляционные базы данных, отдельные таблицы которых могут быть связаны между собой. Связь между таблицами определяет тип отношения между их полями. Как правило, связывают ключевое поле одной таблицы с соответствующим ему полем другой таблицы, которое называют полем внешнего ключа.

Связанные поля могут иметь разные имена, однако у них должны быть одинаковые типы данных и одинаковые значения свойств.

При наличии связи между таблицами связанные данные из таблиц в отчетах, запросах и формах будут выбираться автоматически.

Для связи таблиц из меню **Сервис** следует выбрать команду **Схема данных.** На экране появится окно **Схема данных,** в котором можно определить и просмотреть связи между таблицами базы данных.

Чтобы добавить таблицы или запросы в окно **Схема данных,** необходимо выбрать опцию **Добавить таблицу** в меню **Связи** или нажать кнопку **Добавить таблицу** на панели инструментов.

Переместите используемое для связи поле одной таблицы к соответствующему полю другой с помощью мыши.

На экране появится диалоговое окно **Связи,** в котором будет предложена связь между таблицами.

Теперь путем установления типа отношений между таблицами следует определить параметры связи. Активизируйте опцию **Обеспечение целостности данных,** что обеспечит проверку ссылочной целостности связи между таблицами. Такая проверка позволит избежать ряда ошибок при удалении записей из первичной таблицы и вводе информации в связанную таблицу. Например:

- добавления в связанную таблицу записей, для которых отсутствует соответствующая запись в главной таблице;
- осуществления изменений в главной таблице, которые приведут к появлению «осиротевших» записей в связанной таблице;
- удаления записей в главной таблице, на которые ссылаются записи из связанной таблицы.

После нажатия кнопки ОК в окне **Связи** созданная связь между таблицами отображается графически.

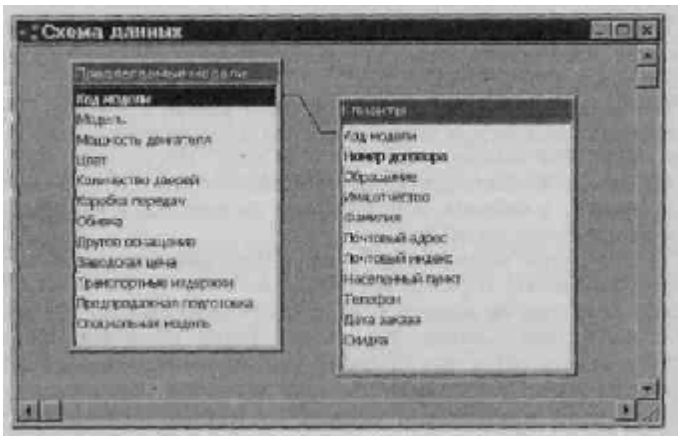


Рис. 1. Схема данных с установленной связью

Поля, через которые осуществляется связь, в главной **и** связанной таблицах могут иметь разные имена. Необходимым условием установления связи является совпадение типа данных и значений характеристик (в особенности размера).

Между двумя таблицами может быть задано только одно отношение, тип которого можно изменить.

ЗАДАНИЯ

Разработайте структуру базы данных (набор таблиц и связей между ними) для одной из следующих предметных областей:

9. Учебный центр
10. Автовокзал
11. Магазин
12. Кулинарная книга
13. Библиотека
14. Агентство недвижимости
15. Поликлиника
16. Такси

База данных должна содержать не менее 7 связанных таблиц. Для каждой таблицы необходимо задать ключевые поля, при необходимости – значения по умолчанию, значения подстановки, условия на значения.

Лабораторная работа 11.

Тема 4. Создание БД. Разработка сложных форм

Для добавления элементов управления понадобится панель инструментов, расположенная по умолчанию слева. Если при переносе полей главной формы эта панель мешает, ее, как и окно, можно переместить в любую позицию на экране. Состояние опции **Панели инструментов** в меню **Вид** определяет наличие (или отсутствие) на экране панели инструментов.

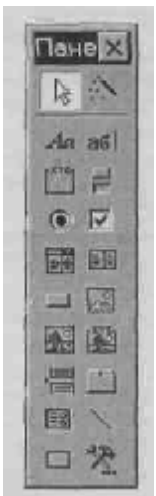


Рис. 1. Панель инструментов элементов управления

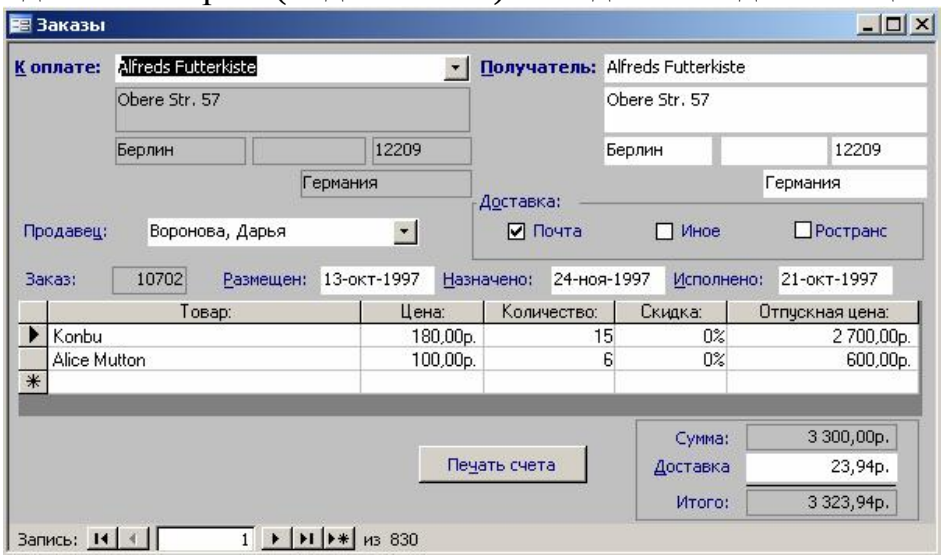
Каждая кнопка панели инструментов, за исключением кнопки **Дополнительные элементы,** представляет собой стилизованное изображение элемента управления, который можно встроить в форму. Обзор кнопок панели инструментов приведен в следующей таблице.

Пиктограмма	Название	Функция
	Выбор объектов	Выделение объектов (позволяет маркировать и перемещать поля, а также изменять их размеры, установленные по умолчанию)
	Мастера	Мастера создания элементов управления
	Надпись	Вставка в форму названия нового поля
	Поле	Отображение содержимого некоторого поля записи базы данных или вычисляемого поля
	Группа переключателей	Создание и размещение группы, в которую можно ввести контрольные переключатели и селекторные кнопки
	Выключатель	Создание выключателя, кнопки с фиксацией

Пиктограмма	Название	Функция
	Переключатель	Создание селекторного переключателя
	Флажок	Создание контрольного переключателя
	Поле со списком	Создание комбинированного списка
	Список	Создание поля списка
	Кнопка	Создание командной кнопки
	Рисунок	Встраивание статических иллюстраций (графических файлов) в форму
	Свободная рамка объекта	Создание рамки объекта, для которого нельзя установить связь
	Присоединенная рамка объекта	Создание рамки объекта, для которого будет установлена связь с файлом-источником
	Разрыв страницы	Установка принудительного конца страницы формы
	Набор вкладок	Создание формы или диалогового окна с несколькими вкладками
	Подчиненная форма/отчет	Встраивание подчиненной формы в главную форму и установка отношений между формами
	Линия	Проведение в форме разделительной линии
	Прямоугольник	Создание в форме прямоугольной рамки для группы полей
	Дополнительные элементы	Встраивание в форму элементов, не представленных на панели инструментов

Access позволяет объединить на одной форме элементы 2 связанных форм, не используя запрос. Это можно реализовать путем использования подчиненных форм. В нижеприведенном примере на одной форме

выведены поля 2 связанных форм базы Борей («Заказы» и «Заказано»), одна из которых (подчиненная) выведена в виде таблицы.



Одним из способов построения такой формы является построение с использованием мастера. В этом случае при выборе доступных полей для формы из 2 связанных таблиц Access предлагает построить форму с подчинением.

ЗАДАНИЯ

Для разработанной структуры БД постройте формы.

В состав форм должны входить формы с подчинением, формы с вычисляемыми полями.

Лабораторная работа 12.

Тема 4. Создание БД. Разработка отчетов на основе запросов (2ч)

ЗАДАНИЯ

Для разрабатываемой БД постройте отчеты на основе перекрестных запросов на выборку.

Лабораторная работа 13.

Тема 4. Создание БД. Макросы

Окно макросов открывается в результате нажатия кнопки Создать или **Конструктор** на вкладке **Макросы** окна базы данных. Оно включает четыре столбца: **Имя макроса, Условие, Макрокоманда** и **Примечание.** При создании нового макроса по умолчанию отображаются только столбцы **Макрокоманда** и **Примечание.** Показ остальных столбцов устанавливается посредством опций **Имена макросов** и **Условия** из меню **Вид.**

В столбце Имя макроса указывается имя макроса, которое надлежит задавать, если окно содержит несколько макросов. При указании имен макросов необходимо следить за тем, чтобы не было повторов. В столбце Условие осуществляется ввод условия (логического выражения) для выполнения только части макроса. В столбце Макрокоманда перечисляются подлежащие вы-

полнению действия (макрокоманды) в нужной последовательности. Столбец **Примечание,** содержащий комментарии к программе, при выполнении макроса игнорируется программой, однако заполнять его рекомендуется, поскольку в этом случае текст макроса понятнее.

Удобно разрабатывать макросы для автоматизации несложных процессов, таких как открытие и закрытие нескольких форм или отчетов, вывод на экран или печать нескольких документов, отмена/восстановление отображения панелей инструментов и т.д.

Изучая принципы создания и применения макросов, рассмотрим пример открытия нескольких объектов базы данных с помощью макроса. База данных, предназначенная для автоматизации делопроизводства, как правило, состоит из множества таблиц, форм, запросов и отчетов. Часто оператор работает с небольшим количеством одних и тех же объектов такой базы данных. В начале каждого сеанса работы с базой тратится дополнительное время на открытие необходимых объектов. Попробуем немного ускорить этот процесс: создадим макрос, открывающий необходимые документы и размещающий их на экране в определенном порядке.

- Откройте окно базы данных.
- Перейдите на вкладку Макросы и щелкните на кнопке Создать, вследствие чего откроется окно конструктора макроса.
- Активизируйте в меню Окно команду Слева направо, чтобы на экране отображались и окно базы данных, и окно макроса.
- В окне базы данных перейдите на вкладку Формы.
- Маркируйте любую форму, перетащите ее с помощью мыши в окно макроса и разместите в первой ячейке столбца Макрокоманда. В поле появится макрокоманда ОткрытьФорму. Таким образом в макрос включается операция открытия данной формы.
- В столбец **Примечание** той же строки введите такой текст: *Открытие формы.* С помощью клавиши [Enter] перейдите во вторую строку столбца Макрокоманда.
- Повторите эти действия для всех объектов, которые должны быть открыты (таблиц*).*
- Щелкните в следующей свободной ячейке столбца Макрокоманда и откройте список доступных макрокоманд. Маркируйте команду Выполнить команду.

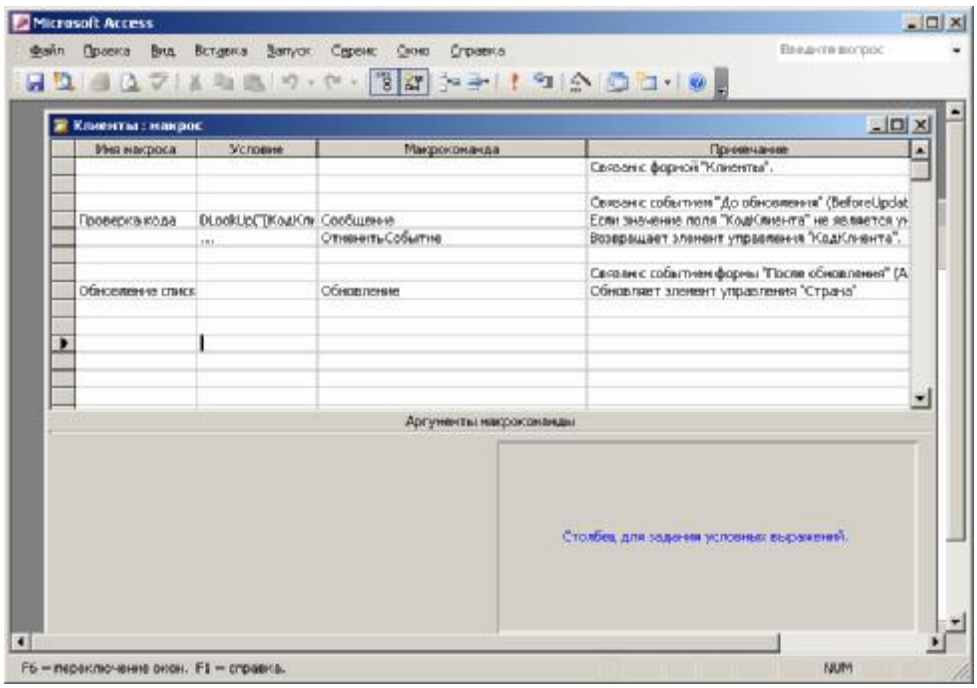


Рис. 1. *Выбор макрокоманды в списке Макрокоманда*

- В области **Аргументы макрокоманды** активизируйте поле **Команда** и выберите в списке элемент **ОкноРазделить.** Таким образом в макрокоманду будет включена операция разделения экрана в случае открытия нескольких окон.
- Вызовите команду **Сохранить** из меню **Файл** и сохраните макрос под именем *Размещение на экране*.

Запуск макроса может осуществляться следующими способами.

1. Щелчок на кнопке **Запуск** (на ней изображен восклицательный знак).
2. Вызов команды **Запуск макроса** из меню **Сервис.**
3. Маркировка имени макроса и нажатие кнопки **Запуск** на вкладке **Макросы** окна базы данных (можно также выполнить двойной щелчок на имени запускаемого макроса).

После запуска созданного нами макроса *Размещение на экране* будут загружены и размещены должным образом все нужные объекты базы данных.

Выполнение каждой макрокоманды зависит от значений ее аргументов (некоторые макрокоманды, например Развернуть, не имеют аргументов). Пользователь не должен запоминать синтаксис макрокоманды — все аргументы вводятся в специально отведенные для этого поля, расположенные в нижней части окна макроса. Если набор допустимых значений для некоторого аргумента фиксирован, Access оформляет их в виде списка. Аргументы можно вводить с помощью клавиатуры, однако лучше выбирать их из списка, чтобы не задать несуществующее значение. Аргументы рекомендуется задавать в том порядке, в каком они расположены в нижней части окна макроса.

Разработку макроса мы начали с ввода макрокоманд открытия объектов,

перетащив соответствующие объекты из окна базы данных в ячейки окна конструктора макроса. При этом Access автоматически распознает, о каком объекте идет речь, и выбирает соответствующую макрокоманду: **Открыть-Форму** для формы или ОткрытьТаблиuу для таблицы. Имя перемещенного объекта появляется в области аргументов макрокоманды в качестве значения параметра **Имя.**

В макрокоманде открытия имя объекта можно ввести в области аргументов (поля **Имя формы** и **Имя таблицы)** с клавиатуры. Важно, чтобы к моменту выполнения макроса открываемый объект уже существовал. В противном случае программа выдаст сообщение об ошибке.

Следующий аргумент макрокоманды открытия объектов, введенный в поле **Режим,** определяет режим отображения объекта на экране. Допустимые значения этого аргумента соответствуют опциям из меню **Режим.**

Значение аргумента Режим	**Применяется для**	**Описание**
Печать	Отчета	Задает вывод отчета на печать
Форма	Формы	Активизирует режим формы
Таблица	Таблицы, запросы, формы	Активизирует режим заполнения и изменения
Конструктор	Таблицы, запросы, отчета, формы	Активизирует режим конструктора
Просмотр	Таблицы, запросы, отчета, формы	Задает режим предварительного просмотра страницы

С помощью макрокоманды **ВыполнитьКоманду** можно задать выполнение большинства команд из меню Access. Имя выполняемой команды указывается в качестве аргумента в поле **Команда.**

Состав строки меню зависит от типа **и** состояния активного объекта. При этом не имеет значения, что маркировано — окно или пиктограмма. При использовании макрокоманды **ВыполнитьКоманду** обращайте внимание на то, какой объект был активизирован последним и в каком режиме он находится. От этого зависит доступность команд меню и корректность их выполнения.

Редактирование макросов

Структура таблицы в окне макросов напоминает структуру обычной

таблицы базы данных. Все знакомые вам команды редактирования текста, используемые для удаления, переноса и копирования содержимого ячеек, могут применяться и в рамках таблицы макроса.

Редактирование макроса осуществляется в режиме конструктора, для установки которого надлежит нажать кнопку Конструктор в окне базы данных. Рекомендуется вносить комментарии в столбец **Примечание.** Они могут оказаться полезными при изменении макроса автором или другим пользователем, а также позволяет получить в любой момент подробную информацию о макрокомандах без их предварительной маркировки. Аргументы **и** краткое описание макрокоманды отображаются в окне макросов только при маркировке.

Связывание макроса с кнопкой

Целесообразно создавать элементы управления (например, кнопки) для вызова макросов. Это значительно упрощает и ускоряет доступ к ним. Удобным местом для размещения таких элементов является заголовок или примечание формы.

В качестве упражнения попробуем связать с кнопкой макрос, который маркирует активную запись формы и копирует ее в буфер обмена:

- Создайте макрос *Копирование записи,* состоящий из следующих макрокоманд:

Макрокоманда	**Аргумент**
ВыполнитьКоманду Выполнить Команду	Select Record Copy

- Сохраните макрос и закройте его окно.
- Откройте форму *Каталог телефонов* и активизируйте режим конструктора.
- Вызовите команду **Слева направо** меню **Окно.**
- Маркируйте макрос *Копирование записи* в окне базы данных и перетащите его в область заголовка проекта формы.

Как только вы отпустите кнопку мыши, позиция кнопки для макроса зафиксируется. При необходимости ее размер изменить с помощью специальных манипуляторов. Перемещение кнопки осуществляется с помощью манипулятора, расположенного в левом верхнем углу маркировочной рамки.

Имя макроса автоматически появляется внутри кнопки. Если оно слишком длинное, размер кнопки следует увеличить. Чтим на кнопке была видна вся надпись, можно уменьшить размер шрифта или создать надпись, отличающуюся от имени макрокоманды. Редактирование надписи выполняется после щелчка на кнопке. Для выхода из режима редактирования достаточно щелкнуть вне кнопки.

Для изменения параметров управляющего элемента **Кнопка** надлежит выполнить на нем двойной щелчок мышью, вследствие чего откроется окно

свойств, если до этого оно отсутствовало на экране.

У каждого элемента управления свой список доступных событий. Однако действия при связывании всегда одинаковы. Например, чтобы связать некоторый макрос с перемещением мыши по форме, следует в режиме конструктора выполнить двойной щелчок в свободной области вне примечания формы и задать для события **Перемещение указателя** имя связываемого макроса.

ЗАДАНИЯ

Определите состав макросов для разрабатываемой БД и создайте их. В состав макросов должны входить макросы, выполняющие:

1. Вывод на печать;
2. Обработку открытия БД;
3. Обработку обновления таблиц;
4. Поиск записей;
5. Копирование записей.

7. Методические указания по применению современных информационных технологий для преподавания учебной дисциплины.

1. Презентации, слайды;
2. Схемы, таблицы, рисунки под медиапроектор;
3. Лазерные пленки к проектоскопу;
4. Программное обеспечение: Microsoft Access 2000

8. Методические указания профессорско-преподавательскому составу по организации межсессионного и экзаменационного контроля знаний студентов (материалы по контролю качества образования)

В процессе изучения дисциплины предусмотрены следующие виды промежуточного контроля знаний студентов:

- контрольная работа на лекции по каждой пройденной теме;

- зачет по каждому блоку лабораторных работ.

- студенты, не посещающие лекционные занятия, представляют рефераты по пропущенным темам.

К промежуточным формам контроля знаний относятся:

- блиц-опрос на лекциях по пройденному материалу;
- контрольные работы;

9. Фонд тестовых и контрольных заданий для оценки качества знаний по дисциплине

Для оценки качества знаний, полученных на лекционных занятиях, по каждой пройденной теме проводится контрольная работа.

Вопросы контрольной работы.

1. Вопросы по теме «Основы построения баз данных»

1.1 Дайте определение понятия информационной системы в широком и узком смысле.

1.2 Что представляет собой банк данных и какие компоненты входят в его состав?

1.3 Каково назначение СУБД?

1.4 Дайте определение приложения, укажите, в каких случаях оно разрабатывается,

1.5 Укажите назначение словаря данных.

1.6 Перечислите функции администратора базы данных.

1.7 Охарактеризуйте архитектуру клиент-сервер и назовите варианты ее реализации, укажите достоинства и недостатки.

1.8 Изобразите структуру информационной системы с файл-сервером.

1.9 Изобразите структуру информационной системы с сервером баз данных.

1.10 Дайте классификацию СУБД.

1.11 Назовите основные функции СУБД.

1.12 Укажите понятие транзакции. Назовите виды транзакций.

2. Вопросы по теме «Модели и типы данных»

2.1 Перечислите классические и современные модели представления данных.

2.2 Укажите достоинства и недостатки иерархической модели данных.

2.3 Как организуется физическое размещение данных в БД иерархического типа? ' .

2.4 Охарактеризуйте сетевую модель данных.

2.5 Охарактеризуйте реляционную модель данных.

2.6 В чем отличие между постреляционной и реляционной моделями данных?

2.7 Укажите достоинства и недостатки постреляционной модели.

2.8 Охарактеризуйте многомерную модель данных.

2.9 Назовите и поясните смысл операций, выполнимых над данными в случае многомерной модели.

2.10 Дайте определение и приведите примеры проявления принципов инкапсуляции, полиморфизма и наследования применительно к объектно-ориентированным базам данных.

2.11 Укажите достоинства и недостатки объектно-ориентированной модели представления данных.

2.12 Охарактеризуйте типы данных, используемые в современных СУБД.

2.13 Охарактеризуйте мультимедиа-возможности реляционных систем.

3. Вопросы по теме «Реляционная модель данных»

3.1 Приведите математическое описание понятия отношения.

3.2 Что такое домен отношения?

3.3 Дайте определение схемы отношения.

3.4 Что представляет собой первичный ключ отношения, для чего он задается?

3.5 Назовите условия, при соблюдении которых таблицу можно считать

отношением.

3.6 Что такое индекс, для чего используется индексирование?

3.7 Изобразите схему одноуровневой индексации и дайте ей характеристику.

3.8 Изобразите схему двухуровневой индексации и дайте ей характеристику.

3.9 Что такое вторичный индекс, в чем его отличие от первичного индекса?

3.10 Опишите действие механизма контроля целостности при манипулировании данными в таблицах.

3.11 Дайте общую характеристику теоретических языков запросов.

3.12 Назовите операции реляционной алгебры, предложенной Коддом, и приведите графическую интерпретацию для операций пересечения и произведения.

3.13 Охарактеризуйте общий и частные случаи операции соединения.

3.14 Назовите правила записи выражений реляционной алгебры.

3.15 Назовите и охарактеризуйте дополнительные операции реляционной алгебры, предложенные Дейтом.

3.16 Охарактеризуйте варианты реляционного исчисления.

3.17 Дайте определение понятия элемента примера и приведите пример его использования в шаблоне запроса на выборку.

3.18 Назовите предполагаемые направления совершенствования языка QBE в современных СУБД.

3.19 Охарактеризуйте язык SQL.

4. Вопросы по теме «Проектирование баз данных»

4.1 Назовите подходы к проектированию структур данных.

4.2 В чем состоит избыточное и неизбыточное дублирование данных?

4.3 Назовите и охарактеризуйте основные виды аномалий.

4.4 Как формируется исходное отношение при проектировании БД?

4.5 Приведите примеры явной и неявной избыточности.

4.6 Назовите основные виды зависимостей между атрибутами отношений.

4.7 Приведите примеры функциональной и частичной функциональной зависимостей.

4.8 Приведите примеры отношений с зависимыми атрибутами.

4.9 Охарактеризуйте нормальные формы.

4.10 Дайте определение первой нормальной формы.

4.11 Дайте определение второй нормальной формы.

4.12 Дайте определение третьей нормальной формы.

4.13 Дайте определение усиленной третьей нормальной формы.

4.14 Поясните на примере используемых в разделе таблиц требования 4НФ.

4.15 Поясните на примере используемых в разделе таблиц требования 5НФ.

4.16 Сформулируйте основное правило создания таблиц сущностей.

4.17 Назовите рекомендации по организации связи сущностей.

4.18 Дайте определение физической и логической целостности БД.

4.19 Приведите примеры ограничений значений и структурных ограничений.

4.20 Поясните понятия внешнего и первичного ключей таблиц.

5. Вопросы по теме «Метод сущность-связь»

5.1 Что представляют собой диаграммы ER-экземпляров и диаграммы ER-типа.

5.2 Что определяет степень связи между сущностями?

5.3 Каким может быть класс принадлежности?

5.4 Приведите пример диаграммы ER-экземпляров со степенью связи между сущностями 1:1 и обязательным классом принадлежности двух сущ-

ностей.

5.5 Как на диаграммах ER-типа обозначаются степень связи, обязательное и необязательное участие в связи экземпляров сущности?

5.6 Приведите пример диаграммы ER-экземпляров для связи типа 1:M

5.7 Назовите этапы проектирования базы данных.

5.8 Как осуществляется формирование отношений для связи 1:1?

5.9 Сформулируйте правило 3 формирования отношений, если степень связи 1:1 и класс принадлежности обеих сущностей является необязательным.

5.10 Сформулируйте правило формирования отношения для случая степени связи между сущностями 1:M (M:1) и обязательного класса принадлежности M-связной сущности.

5.11 Укажите правила формирования отношений для связи M:M.

5.12 Покажите, что полученные в прикладном примере из раздела отношения находятся в нормальной форме Бойса - Кодда.

Для оценки качества знаний, полученных на лабораторных занятиях, по каждой теме лабораторных занятий сдается зачет. Зачет представляет собой решение задач по теме.

Задачи:

1. Примерные задачи по теме «Среда разработки Access»

1.1 Просмотрите структуру таблицы, определите, какие в поля и какие значения подставляются по умолчанию. Измените одно из этих значений.

1.2 Измените вид таблицы таким образом, чтобы были видны только определенные столбцы.

1.3 С помощью мастера сводных диаграмм постройте диаграмму.

1.4 С помощью мастера сводных таблиц постройте таблицу.

2. Примерные задачи по теме «Конструктор БД»

2.1 Задайте в таблице поле таким образом, чтобы его значение выбиралось из другой таблицы.

2.2 Создайте таблицу, имеющую определенный формат.

2.3 Добавьте в таблицу условие на ввод данных даты рождения.

2.4 Создайте простую форму для таблицы с помощью конструктора форм

2.5 Создать отчет с помощью конструктора

3. Примерные задачи по теме «Запросы»

3.1 Вывести всех служащих, названия их отделов и зарплату. Отчет организовать так, чтобы служащие одного отдела находились последовательно друг за другом в порядке убывания зарплат.

3.2 Подсчитать количество сотрудников, получающих меньше 1500

а. Выведите всех сотрудников, заработная плата которых не лежит в промежутке от 1000 до 3000; отчет оформить по шаблону <сотрудник> получает <зарплата>

3.3 Для каждого сотрудника, оформлявшего заказы, подсчитать количество оформленных заказов.

3.4 Вывести номер служащего, его фамилию, дату поступления на работу, фамилию его менеджера и дату полугодовой аттестации для всех служащих, работающих менее года.

3.5 Подсчитать, во сколько раз отличается максимальная зарплата от минимальной

3.6 Вывести фамилии всех служащих, нанятых после 15 ноября 1996 года, и названия отделов, где они работают

3.7 Вывести все возможные кредитные рейтинги и количество клиентов в каждой категории

3.8 Вывести номера заказов, в которых заказано более 3 видов товаров

3.9 Вывести всех служащих, вторая буква фамилии которых «в», и названия отделов, где они работают

3.10Вывести данные о самом первом заказе.

3.11Вычислить остаток от деления зарплаты на комиссионные для всех служащих, получающих менее 1500

3.12 Вывод средней, максимальной и минимальной зарплаты для сотрудников, нанятых с 1992 по 1999 годы.

3.13Вывести всех служащих, предпоследняя буква фамилии которых «о», и названия отделов, где они работают

3.14Подсчитать, сколько дней прошло между заказом и выполнением самого долговыполняемого заказа.

3.15Подсчитать, сколько отделов в каждом регионе

3.16Подсчитать количество полных недель, отработанных каждым служащим отдела № 15

3.17 Для каждого заказчика, общая сумма заказа которого превышает 1000 руб., вывести наименование заказчика и заказанные им товары

3.18Вывести фамилии служащих, нанятых после 1997 года, дату поступления на работу и фамилии их менеджеров. Исключить сотрудников, принятых в субботу и воскресенье

3.19Вывести все заказы и названия входящих в них товаров, общая сумма которых превышает 1000 рублей

3.20Вывести фамилии всех работников отделов продаж.

3.21Подсчитайте, какую сумму заработал сотрудник, оформивший 623 заказ. Выведите его фамилию и округленный до целого заработок.

3.22Для каждого клиента, делавшего заказы, подсчитать количество оформленных заказов.

10. Контрольные вопросы к зачету

1. Базы данных и информационные системы.
2. Банки данных. Базы данных.
3. Система управления базами данных.
4. Словарь данных. Администратор базы данных.
5. Архитектура информационной системы.
6. Сервер БД, клиент. Файл-сервер, SQL-сервер.
7. Классификация СУБД.
8. Средства разработки программ работы с БД.

9. Способы разработки и выполнения приложений. Схема обмена данными при работе с БД

10. Модель представления данных

11. Иерархическая модель. Достоинства и недостатки модели.

12. Сетевая модель. Достоинства и недостатки модели.

13. Реляционная модель. Понятие отношения. Достоинства и недостатки модели.

14. Многомерная модель. Измерения, ячейки. Достоинства и недостатки модели.

15. Объектно-ориентированная модель.

16. Основные типы данных СУБД.

17. Определение реляционной модели. Отношение, сущность, атрибуты, домен.

18. Схема отношения. Первичный ключ, ссылочная целостность.

19. Индексирование. Индекс, методы поиска

20. Связывание таблиц. Основные виды связи таблиц 21. Контроль целостности связей

21. Теоретические языки запросов. Реляционная алгебра.

22. Языки исчислений.

23. Структурированный язык запросов SQL. Основные операторы языка.

24. Логическое проектирование, проектирование структур.

25. Избыточное дублирование данных и аномалии

26. Формирование исходного отношения.

27. Метод нормальных форм.

28. Зависимости между атрибутами. Функциональная взаимозависимость.

29. Первая нормальная форма (1НФ);

30. Вторая нормальная форма (2НФ);

31. Третья нормальная форма (3НФ); усиленная третья нормальная

форма, или нормальная форма Бойса -Кодда(БКНФ);

32. Четвертая нормальная форма (4НФ);
33. Пятая нормальная форма (5НФ).
34. Организация связи сущностей. Обеспечение целостности
35. Основные понятия метода «сущность-связь»
36. Этапы проектирования.
37. Правила формирования отношений.
38. Модели архитектуры клиент-сервер.
39. Двухзвенные модели распределения функций.
40. Трехзвенная модель распределения функций.
41. Сетевые СУБД.

11. Карта обеспеченности дисциплины кадрами профессорско-преподавательского состава.

Вид учебной нагрузки	ППС
Лекции	Тафинцева В.В., ассистент
Лабораторные занятия	Тафинцева В.В., ассистент
Экзамен	Тафинцева В.В., ассистент