

**Максимов Н.В.  
Голицына О.Л.  
Васина Е.Н.**

# **Управление данными**

Голицына О.Л., Максимов Н.В., Васина Е.Н. Управление данными. - М.: Московский международный институт эконометрики, информатики, финансов и права. - 2004. – 337 с.

В пособии подробно рассмотрены основные подходы и направления развития систем баз данных. Анализируются классические машинно-ориентированные формы представления информации и данных. Рассматриваются типовые модели физической и логической организации данных. Исследуется архитектура средств доступа к данным. Достаточно подробно представлены возможности SQL, как базового языка для работы с реляционными базами данных. Большое внимание уделено проблемам моделирования и проектирования баз данных.

© Голицына О.Л., 2004

© Максимов Н.В., 2004

© Васина Е.Н., 2004

© Московский международный институт эконометрики, информатики, финансов и права, 2004

## Содержание

Предисловие.....	7
Глава 1. Введение в базы и банки данных.....	12
1.1. Понятие базы и банка данных.....	12
1.2. Компоненты банка данных .....	15
1.2.1. Информационная база.....	16
1.2.2. Лингвистические средства .....	17
1.2.3. Программные средства .....	19
1.2.4. Технические средства .....	21
1.2.5. Организационно-административные подсистемы.....	22
1.3. Пользователи баз данных .....	22
1.4. Типология баз данных .....	23
1.4.1. Типология баз данных с точки зрения информационных процессов .....	24
1.5. Семантика баз данных .....	26
1.6. Типология моделей .....	32
Глава 2. Базовые технологии и основные этапы развития машинной обработки данных .....	37
2.1. Введение в технологии машинной обработки данных и основные определения .....	37
2.2. Схема организации файлового ввода-вывода .....	40
2.3. Эволюция концепций обработки данных .....	43
2.3.1. Простые (линейные) файлы данных (начало 60-х годов).....	44
2.3.2. Методы доступа к записям (конец 60-х годов).....	44
2.3.3. Первые системы управления базами данных (начало 70-х годов) .....	45
2.3.4. Системы управления базами данных .....	46
2.4. Схема управления данными в СУБД .....	48
2.5. Данные и управление их обработкой.....	49
2.5.1. Типы, форматы, структуры данных .....	49
2.5.2. Описание и обработка файлов .....	51
2.6. Особенности и компромиссы реализаций баз данных .....	52
Глава 3. Модели и структуры данных.....	55
3.1. Многоуровневые модели предметной области.....	55
3.2. Идентификация объектов и записей .....	59
3.3. Поиск записей.....	61
3.4. Представление предметной области и модели данных.....	64
3.5. Структуры данных .....	67
3.5.1. Линейные структуры .....	70
3.5.2. Нелинейные структуры .....	71
3.5.3. Сетевые структуры.....	76
3.6. Реляционная модель данных.....	79
3.6.1. Основные понятия реляционной модели данных.....	79
3.6.2. Основы реляционной алгебры .....	83

Глава 4. Физические модели баз данных .....	90
4.1. Организация данных на машинных носителях .....	90
4.1.1. Типы записей .....	91
4.1.2. Организация файлов - способ размещения записей .....	93
4.1.3. Способы адресации и методы доступа к записям.....	95
4.1.4. Схемы организации данных на внешних носителях .....	99
4.2. Физическое представление иерархических структур.....	102
4.2.1. Физически последовательное размещение.....	104
4.2.2. Левосписковые структуры с переполнениями.....	104
4.2.3. Использование указателей на «подобные» и «порожденные» .....	105
4.3. Физическое представление сетевых структур .....	106
4.3.1. Физически последовательное размещение.....	107
4.3.2. Использование указателей .....	108
4.3.4. Физическое представление с разделением данных и связей.....	109
4.4. Архитектура файловой организации баз данных .....	112
4.4.1. Файл-ориентированная организация данных.....	112
4.4.2. Страничная организация данных .....	113
4.5. Модели распределения данных по физическим носителям .....	114
Глава 5. Модели и этапы проектирования баз данных .....	117
5.1. Модели многоуровневой архитектуры систем баз данных .....	117
5.2. Стадии проектирования и объекты моделирования.....	122
5.3. Системный анализ предметной области.....	126
5.4. Модели и технологии инфологического проектирования реляционных БД.....	129
5.4.1. Инфологическое проектирование и семантическая модель .....	129
5.4.2. Модель «Сущность-Связь» .....	131
5.4.3. ER- диаграмма .....	137
5.4.4. Нормальные формы ER-диаграмм .....	139
5.5. Дatalogические модели .....	139
5.5.1. Получение реляционной схемы из ER-диаграммы .....	140
5.6. Физические модели .....	141
Глава 6. Проектирование реляционной базы данных .....	143
6.1. Универсальное отношение.....	143
6.2. Функциональная и многозначная зависимости .....	148
6.3. Нормальные формы .....	149
6.4. Процедура нормализации.....	151
6.5. Пример проектирования реляционной БД .....	153
6.5.1. Построение ER-диаграммы .....	153
6.5.2. Построение реляционной схемы .....	157
6.5.3. Нормализация таблиц .....	158
Глава 7. Введение в SQL .....	163
7.1. Основные понятия и компоненты .....	165
7.1.1. Инструкции и имена .....	165

7.1.2. Типы данных.....	166
7.1.3. Встроенные функции.....	167
7.1.4. Значения NULL .....	169
7.2. Ограничения целостности.....	169
7.2.1. Первичный ключ таблицы.....	169
7.2.2. Внешний ключ таблицы .....	170
7.2.3. Определение уникального столбца .....	173
7.2.4. Определение проверочных ограничений.....	173
7.2.5. Определение значения по умолчанию .....	174
7.3. Управление таблицами .....	175
7.3.1. Команда создания таблицы – CREATE TABLE .....	175
7.3.2. Изменение структуры таблицы – команда ALTER TABLE .....	181
7.3.3. Удаление таблиц – команда DROP TABLE.....	184
7.4. Управление данными .....	185
7.4.1. Извлечение данных – команда SELECT .....	185
7.4.3. Изменение данных – команда UPDATE .....	221
7.4.4. Удаление данных – команда DELETE .....	222
Глава 8. Распределенная обработка данных.....	224
8.1. Основные условия и требования к распределенной обработке данных .....	224
8.2. Архитектура распределенной обработки данных.....	226
8.2.1. Базовые архитектуры распределенной обработки.....	227
8.3. Технологии и средства доступа к удаленным БД.....	235
8.3.1. Программное обеспечение распределенных приложений.....	235
8.3.2. Доступ к базам данных в двухзвенных моделях клиент-сервер.....	237
8.4.1. Спецификация вызова удаленных процедур.....	245
8.4.2. Мониторы обработки транзакций .....	246
8.4.3. Корпоративные серверы приложений.....	248
8.4.4. Доступ к данным с помощью ADO.NET .....	252
Глава 9. Транзакции и целостность БД.....	254
9.1. Модели транзакций .....	255
9.2. Журнал транзакций .....	258
9.3. Параллельное выполнение транзакций.....	260
9.4. Сериализация транзакций .....	264
9.5. Захват и освобождение объекта.....	264
Глава 10. Управление базами данных в СУБД .....	267
10.1. Планирование БД.....	268
10.2. Управление доступом .....	270
10.2.1. Тип подключения к SQL Server .....	270
10.2.2. Пользователи базы данных .....	271
10.2.3. Роли.....	272
10.3. Управление обработкой. Представления, хранимые процедуры, триггеры .....	274

10.3.1. Представления .....	275
10.3.2. Хранимые процедуры .....	276
10.3.3. Триггеры.....	277
10.4. Управление транзакциями.....	279
10.5. Резервное копирование и восстановление.....	281
10.6. Пример администрирования базы данных в среде MicroSoft SQL Server.....	282
Глава 11. Направления развития концепций и систем обработки данных .....	292
11.1. Еще раз о проектировании и реализации систем баз данных .....	292
11.2. Объектно-ориентированные базы данных .....	295
11.3. Интеграция БД и хранилища данных.....	299
11.3.1. Основы технологии интеграции распределенных данных .....	300
11.3.2. Аналитическая обработка данных.....	302
11.4. Базы данных и Internet .....	303
11.5. Еще раз о проблемах и решениях .....	304
Глоссарий .....	306
Приложение .....	314
Литература .....	336



## Предисловие

Сегодня трудно себе представить сколько-нибудь значимую информационную систему, которая бы не имела бы в качестве основы, или важной составляющей базу данных. Концепции и технологии баз данных складывались постепенно и всегда были тесно связаны с развитием систем автоматизированной обработки информации. Создание баз данных после появления реляционного подхода превратилось из искусства в науку, но, как показала практика последних лет, все же окончательно его не исключившая. Тем не менее, сейчас это вполне сложившаяся дисциплина (хотя являющаяся скорее инженерной, чем чисто научной), основанная на достаточно формализованных подходах и включающая широкий спектр приемов и методов создания баз данных.

Как отмечается в [4], базам свойственна «перманентность» данных. Соответственно назначение систем управления базами данных – обеспечение в течение длительного времени их сохранности, а также возможностей выборки и актуализации. Данные существуют всегда, пока есть потребность в их использовании<sup>1</sup>, хотя характер использования, как и пути извлечения практической пользы могут быть самыми разными: от оперативной актуализации значений до уничтожения данных, от их использования для совершенствования сложных систем управления до формирования «чемоданов компромата».

Базы данных в стремительно, а в какой-то степени и сумбурно, развивающихся информационных технологиях – это сравнительно консервативное направление, где СУБД и сами базы представляют собой «долговременные сооружения». Элементная база ЭВМ и парадигмы программирования меняются быстрее, чем хранимые данные теряют актуальность. В таких условиях, в отличие от прикладных программистов, создатели баз данных (от разработчиков СУБД до администраторов БД) должны постоянно помнить о проблеме «наследственности» - о том, как интегрировать в создаваемую систему наследуемые данные, находящиеся под управлением устаревшей СУБД, и о том, как построить систему, чтобы вновь создаваемые данные могли быть, в свою очередь, наследованы следующим поколением систем и разработчиков.

Достаточно консервативны и концепции баз данных. Эта консервативность не только следствие свойства «долговечности», но и того факта, что базы вторичны по отношению к описываемым ими реальным процессам и объектам, достаточно стабильным и типичным. Кроме того, модели данных строились в значительной степени «по аналогии» с организационными и технологическими структурами – иерархическими, сетевыми, матричными.

---

<sup>1</sup> Правильнее было бы говорить, что данные создаются, но создаются не ради их самих, а для того, чтобы в дальнейшем они были использованы в каком-то процессе.

Широкое использование баз данных различными категориями пользователей привело, с одной стороны, к созданию интерфейсов, требующих минимум времени на освоение средств управления системой, а с другой - к построению мощных, гибких СУБД имеющих, в том числе, развитые средства защиты данных от случайного или преднамеренного разрушения. Появились и средства автоматизации разработки, позволяющие создать базу данных любому пользователю, даже не владеющему основами теории БД.

Но, как было отмечено ранее, база данных - это важная, но не основная (функционально), а обеспечивающая (информационная) составляющая некоторой, обычно, достаточно крупной *человеко-машинной* системы. И здесь интересно отметить принципиальное отличие в развитии способностей взаимодействующих субъектов (человек-машина). Разделение информации на *табличную (числовую), текстовую и графическую* отражает последовательность, в которой эти виды информации "осваивались" компьютерами. Первые языки программирования были рассчитаны исключительно на обработку числовой информации (Fortran, Algol). Первыми появляются и табличные базы данных, также преимущественно рассчитанные на обработку числовых таблиц (файлов). Затем осваиваются текстовые файлы и текстовые БД (автоматизированные информационно-поисковые системы с библиографическими и полнотекстовыми базами). Наконец, с существенным повышением быстродействия и ёмкости памяти компьютеров, на сцену выходят графические и мультимедийные базы.

Эта последовательность прямо противоположна той, в которой данные виды информации осваивает человек. Действительно, сначала он знакомится с графическими образами (птички, цветочки и бабочки на шкафчиках для одежды в детском саду), затем - учится читать и писать, а только потом осваивает таблицу умножения.

Создание практически полезной «серьезной» базы данных в равной степени зависит как от «фундаментальности» знаний разработчика в области концепций и технологий СУБД, так и от степени понимания им сегодняшних и будущих прикладных задач пользователя, не только от адекватности применения тех или иных типовых или оригинальных решений, но и от качества представления (описания) этих решений, с той или иной степенью успешности позволяющих использовать, сопровождать и развивать систему после разработчика.

Кроме того, возможности накапливать и оперативно обрабатывать большие объемы информации, характеризующие деятельность предприятий за достаточно длительные периоды и в различных аспектах, дали новый импульс к развитию аналитических систем. Такого рода *системы поддержки принятия решений* обычно используются для оценки и выбора альтернативных решений, прогнозирования, идентификации объектов и состояний и т.д. Однако, поскольку для получения необходимых



данных в этих случаях нужно использовать сложные SQL-запросы или специализированные процедуры, и при этом обрабатывать большие объемы записей, то уже это может приводить к сознательному отказу от классических нормализованных схем, т.к. чем выше степень нормализации, тем больше число операций соединения отношений и, соответственно, больше времени необходимо для получения конечного результата.

Базы данных – это уже достаточно хорошо проработанная научная дисциплина. Существует множество, в том числе и фундаментальных, работ и учебников (на материал которых авторы опирались при подготовке этого учебника, и убедительно рекомендуют их тем, кто серьезно интересуется этой проблематикой), среди которых необходимо выделить такие монографии, как «Организация баз данных в вычислительных системах» Дж. Мартина, «Введение в системы баз данных» К. Дейта, «Алгоритмы и структуры данных» Н. Вирта, «SQL» Дж. Гроффа и П. Вайнберга.

В своей работе авторы руководствовались и тем, что материал должен не только в компактной и наглядной форме представлять существо конкретной темы, но и подвести читателя к пониманию обоснованности (или условности) того или иного решения. Авторы сознательно избегали описаний языков и технологий, применяемых в конкретных системах, предполагая, что полноценное освоение материала курса связано с практикой и, соответственно, с неизбежным изучением конкретных подходов, языков и технологий, свойственных выбранной системе, и изложенных в специальных пособиях, учебниках и руководствах.

Материал курса, представленный в одиннадцати главах и приложении, условно можно отнести к следующим разделам:

- введение в машинную обработку данных и структуры данных;
- общесистемные основы и технологии проектирования баз данных;
- язык управления данными (SQL);
- основы организации и технологии доступа к данным.

В первой главе определены основные понятия, относящиеся к базам и банкам данных, приведена классификация компонент систем управления данными, определены их назначение и основные функции. Рассмотрен важнейший вопрос семантики баз данных в контексте информационных систем и определено соотношение понятий «информация» и «данные». Приведена типология моделей, охватывающая все этапы создания БД и представляющая различные подходы к представлению информации и организации данных.

Во второй главе представлены базовые технологии машинной обработки данных и рассмотрены ключевые моменты, определяющие эф-

фективность процессов управления данными. Приведены характеристические черты систем управления данными разных поколений. Примерные схемы управления данными в файловой системе ОС и СУБД дают для этих двух случаев наглядное представление о принципиальных различиях организации процессов и разделении функций между компонентами.

В главе 3 обсуждаются основы формализованного представления информации и вводится понятие многоуровневой модели. Представлены типовые подходы к идентификации объектов и дана типология запросов атрибутивного поиска описаний объектов. Определены различия между подходами, используемыми в фактографических и документальных базах данных. Приведены основные структуры организации данных, рассматриваемые как базовые конечные средства представления информации в вычислительной среде. Вводится понятие модели данных.

В главе 4 приведены типовые модели физической организации данных, акцентирующие внимание на различиях в вариантах структур и связей. Рассматриваются схемы организации данных для линейных, иерархических и сетевых структур. Обсуждаются архитектуры организации данных на уровне файловых компонент. Материал этой главы является ключевым для понимания существа внутримашинной обработки данных и, соответственно, путей построения высокоэффективных систем БД.

Глава 5 посвящена проблемам моделирования баз данных. Определяются стадии проектирования и объекты моделирования. Обсуждаются отличия подходов к моделированию предметных областей, характерных для фактографических и документальных баз данных. Подробно рассматривается содержание инфологического и даталогического этапа проектирования.

В главе 6 описывается пример проектирования реляционной базы данных, включая технологию проектирования и нормализации отношений.

Глава 7 полностью посвящена описанию SQL, который является стандартным языком для работы с реляционными базами данных. Возможности использования операторов языка рассматриваются на серии примеров, иллюстрирующих этапы создания и использования базы данных, описание проектирования которой приведено в предыдущей главе.

В главе 8 представлены модели и технологии распределенной обработки данных. Проведен сравнительный анализ базовых архитектур, отражающих характер распределения данных и процессов между компонентами распределенной системы. Рассмотрены типовые технологии и средства доступа к данным, распределенным в узлах вычислительной сети.

Глава 9 посвящена понятию «транзакция», рассматриваемого как основа технологии параллельной обработки данных. Обсуждаются модели транзакций и способы управления транзакциями.

Глава 10 знакомит с основными понятиями и процессами управления базами данных в СУБД. Здесь затрагиваются вопросы физического планирования БД, организация управления доступа пользователей к объектам БД, программирование процессов управления обработкой данных (представления, хранимые процедуры, триггеры), а также управление репликациями и резервным копированием.

Глава 11 кратко характеризует основные проблемы и направления развития систем баз данных. Рассматривается объектно-ориентированная парадигма, хранилища данных, работа с базами данных в Internet.

Приложение содержит примерное описание физических структур реальных СУБД. Сравнительный анализ этих примеров позволит прилежному читателю уяснить различия и, что нам кажется важнее, сходство решений, а также практически оценить роль моделей, которым уделялось так много внимания в большинстве глав.

Данное пособие написано в предположении, что читатели владеют математическими основами баз данных и в том числе реляционной алгеброй, а также знакомы с современными языками программирования.

Книга также может рассматриваться как введение в проблематику теории и практики информационных систем, основанных на базах данных. Для заинтересованного читателя материал книги должен стать отправной точкой, пособием для освоения специальных разделов, каковыми авторы считают «Информационные системы» и «Проектирование информационных систем», что позволит читателям заменить общие, и местами схематичные, формулировки и положения строгими выводами и исчерпывающе полными объяснениями и примерами.

## Глава 1. Введение в базы и банки данных

### 1.1. Понятие базы и банка данных

Развитие вычислительной техники и появление емких внешних запоминающих устройств прямого доступа предопределило интенсивное развитие автоматических и автоматизированных систем разного назначения и масштаба, в первую очередь заметное в области бизнес-приложений. Такие системы работают с большими объемами информации, которая обычно имеет достаточно сложную структуру, требует оперативности в обработке, часто обновляется и в то же время требует длительного хранения. Примерами таких систем являются автоматизированные системы управления предприятием, банковские системы, системы резервирования и продажи билетов и т.д.

Другими направлениям, стимулировавшим развитие, стали с одной стороны, системы управления физическими экспериментами, обеспечивающими сверхоперативную обработку в реальном масштабе времени огромных потоков данных от датчиков, а с другой - автоматизированные библиотечные информационно-поисковые системы (см., например, рис. 1.1).

Это привело к появлению новой информационной технологии интегрированного хранения и обработки данных — *концепции баз данных*, в основе которой лежит механизм предоставления обрабатывающей программе из всех хранимых данных только тех, которые ей необходимы, и в форме, требуемой именно этой программе. При этом сама форма (структура данных и форматы полей, входящих в эту структуру) описывается на логическом, т.е. «видимом» из программы, уровне. Более того, поскольку различные программы могут по-разному «видеть» (а, следовательно, и использовать) одни и те же данные, то система должна сделать «прозрачными» для программы все данные, кроме тех, которые для нее являются «своими».

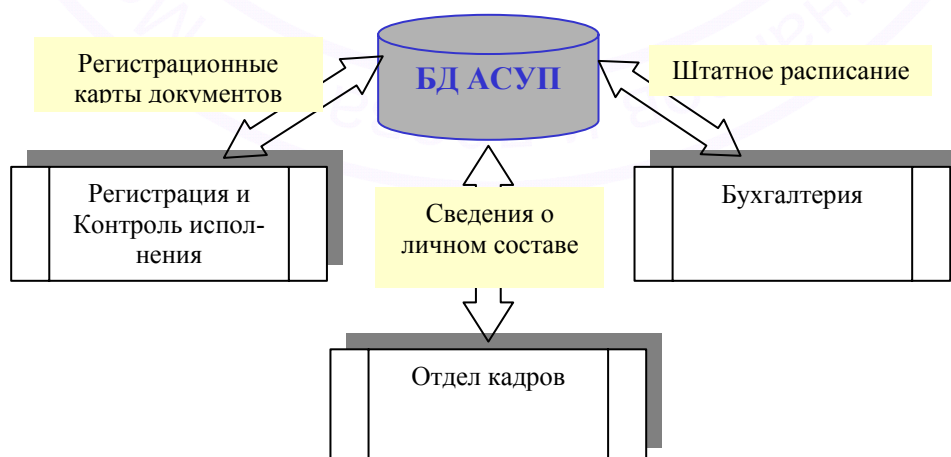


Рис. 1.1. Схема автоматизированной информационной системы



*Банк данных (БНд)* - это система специально организованных данных, программных, языковых, организационных и технических средств, предназначенных для централизованного накопления и коллективного многоцелевого использования данных<sup>2</sup>.

Под *базой данных (БД)* обычно понимается именованная совокупность данных, отображающая состояние объектов и их отношений в рассматриваемой предметной области. Характерной чертой баз данных является *постоянство*: данные *постоянно* накапливаются и используются; состав и структура данных, необходимых для решения тех или иных прикладных задач обычно *постоянны* и стабильны во времени; отдельные или даже все элементы данных могут меняться – но и это есть проявление постоянства – *постоянная* актуальность.

*Система управления базами данных (СУБД)* - это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Иногда в составе банка данных выделяют *архивы*. Основанием для этого является особый режим использования данных — только часть данных находится под оперативным управлением СУБД. Все остальные данные (собственно архивы) обычно располагаются на носителях, оперативно не управляемых СУБД. Одни и те же данные в разные моменты времени могут входить как в базы данных, так и в архивы. Банки данных могут не иметь архивов, но если они есть, то в состав банка данных может входить и система управления архивами.

Проблемы совместного использования данных и периферийных устройств компьютеров и рабочих станций быстро породили модель вычислений, основанную на концепции файлового сервера - сеть создает основу для коллективной обработки, сохраняя простоту использования персонального компьютера, позволяет совместно использовать данные и периферию.

В этом смысле главной отличительной чертой баз данных является использование централизованной системы управления данными, причем как на уровне файлов, так и на уровне элементов данных. Централизованное хранение совместно используемых данных приводит не только к сокращению затрат на создание и поддержание данных в актуальном состоянии, но и к сокращению избыточности информации, упрощению процедур поддержания непротиворечивости и целостности данных.

Эффективное управление внешней памятью является основной функцией СУБД. Эти, обычно специализированные, средства настолько важны с точки зрения эффективности, что при их отсутствии система просто не сможет выполнять некоторые задачи уже потому, что их выполнение будет занимать слишком много времени. При этом, ни одна из

---

<sup>2</sup> Следует отметить, что термин «банк данных» используется сравнительно редко, а некоторыми авторами признается даже архаичным. В современной, в основном переводной литературе – например [4], понятию банк данных соответствует понятие *системы баз данных*, хотя, по нашему мнению, «банк данных» вполне адекватное и более широкое понятие.



таких специализированных функций, как построение индексов, буферизация данных, организация доступа и оптимизация запросов, не являющиеся видимыми для пользователя и обеспечивают независимость между логическим и физическим уровнями системы: прикладной программист не должен писать программы индексирования, распределять память на диске и т.д.

Развитие теории и практики создания информационных систем, основанных на концепции баз данных, создание унифицированных методов и средств организации и поиска данных позволяют хранить и обрабатывать информацию о все более сложных объектах и их взаимосвязях, обеспечивая многоаспектные информационные потребности различных пользователей. Основные требования, предъявляемые к банкам данных, можно сформулировать следующим образом.

- **Многократное использование данных:** пользователи должны иметь возможность использовать данные различным образом;

- **Простота:** пользователи должны иметь возможность легко узнать и понять, какие данные имеются в их распоряжении;

- **Легкость использования:** пользователи должны иметь возможность осуществлять (процедурно) простой доступ к данным, при этом все сложности доступа к данным должны быть скрыты в самой системе управления базами данных;

- **Гибкость использования:** обращение к данным или их поиск должен осуществляться с помощью различных методов доступа;

- **Быстрая обработка запросов на данные:** запросы на данные, в том числе незапланированные, должны обрабатываться с помощью высокоуровневого языка запросов, а не только прикладными программами, написанными с целью обработки конкретных запросов (разработка таких программ в каждом конкретном случае связана с большими затратами времени). Пользователь должен иметь возможность кратко выразить нетривиальные запросы (в нескольких словах или несколькими нажатиями клавиш мыши). Это означает, что средство формулирования должно быть достаточно «декларативным», т. е., упор должен быть сделан на "что", а не на "как". Кроме того, средства обработки запросов не должно зависеть от приложения, т. е., оно должно работать с любой возможной базой данных;

- **Язык взаимодействия конечных пользователей с системой** должен обеспечивать конечным пользователям возможность получения данных без использования прикладных программ;

- **База данных – это основа для будущего наращивания прикладных программ:** базы данных должны обеспечивать возможность быстрой и дешевой разработки новых приложений;

- **Сохранение затрат умственного труда:** существующие программы и логические структуры данных (на создание которых обычно затрачивается много человеко-лет) не должны переделываться при внесении изменений в базу данных;

- **Наличие интерфейса прикладного программирования:** Прикладные программы должны иметь возможность просто и эффективно выполнять запросы на данные; программы должны быть изолированы от расположения файлов и способов адресации данных;

- **Распределенная обработка данных:** система должна функционировать в условиях вычислительных сетей и обеспечивать эффективный доступ пользователей к любым данным распределенной БД, размещенным в любой точке сети;

- **Адаптивность и расширяемость:** с целью увеличения производительности база данных должна быть настраиваемой, причем настройка не должна вызывать перезапись прикладных программ. Кроме того, поставляемый с СУБД набор предопределенных типов данных должен быть расширяемым – в системе должны быть средства для определения новых типов и не должно быть различий в использовании системных и определенных пользователем типов;

- **Контроль за целостностью данных:** система должна осуществлять контроль ошибок в данных и должна выполнять проверку взаимного логического соответствия данных;

- **Восстановление данных после сбоев:** Автоматическое восстановление без потери данных транзакции. В случае аппаратных или программных сбоев система должна возвращаться к некоторому согласованному состоянию данных;

- **Вспомогательные средства** должны позволять разработчику или администратору базы данных предсказать и оптимизировать производительность системы;

- **Автоматическая реорганизация и перемещение:** система должна обеспечивать возможность перемещения данных или автоматическую реорганизацию физической структуры.

## ***1.2. Компоненты банка данных***

Определение банка данных предполагает, что с функционально-организационной точки зрения банк данных является сложной человеко-машинной системой, включающей в себя все подсистемы, необходимые для надежного, эффективного и продолжительного во времени функционирования.

В структуре банка данных выделяют следующие компоненты (подсистемы):

- информационная база;
- лингвистические средства;
- программные средства;
- технические средства;
- организационно-административные подсистемы и нормативно-методическое обеспечение.

### 1.2.1. Информационная база

Данные, отражающие состояние определенной предметной области и используемые информационной системой, принято называть *информационной базой*. Информационная база состоит из двух компонент: 1) коллекции записей собственно данных и 2) описания этих данных — метаданных.

Данные отделены от описаний, но в то же время данные не могут использоваться без обращения к соответствующим описаниям.

Уже из определения базы данных и приведенных ранее основных требований следует, что данные могут использоваться (т.е., представляться) по-разному. С одной стороны, разные прикладные задачи требуют разных наборов данных, в совокупности обеспечивающих функциональную полноту информации, а с другой — они должны быть различны для различных категорий субъектов (разработчиков или пользователей). Также должны быть различными и способы описания самих данных, их природы, формы хранения, условий взаимной непротиворечивости.

В литературе по базам данных упоминаются три уровня представления данных — концептуальный, внутренний и внешний (рис. 1.2).

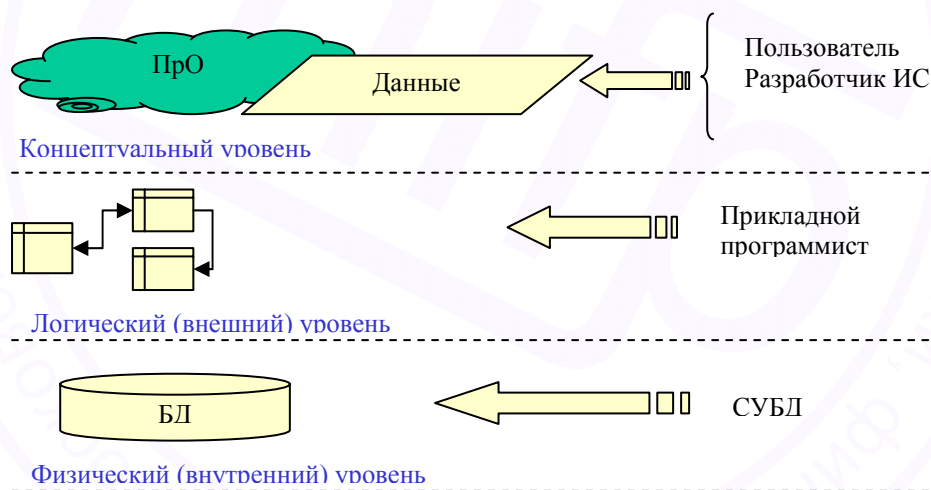


Рис. 1.2. Уровни представления данных

Эти уровни представлений введены исходя из различного рассмотрения БД. Например, прикладному программисту требуются не все данные БД, а только некоторая их часть, используемая в его программе. Внешний уровень представления обеспечивает именно эту форму обмена данными.

Внутренний уровень - глобальное представление БД, определяет необходимые условия для организации хранения данных на внешних запоминающих устройствах.

Описание БД на концептуальном уровне представляет собой обобщенный взгляд на данные с позиций предметной области (разра-

ботчика приложений, пользователя или внешней информационной системы).

Внешний уровень представления данных не затрагивает физической организации (размещения) данных во внешней памяти, поэтому его называют иногда логическим уровнем. Соответственно внутренний уровень называют физическим уровнем.

### *1.2.2. Лингвистические средства*

Многоуровневое представление БД предполагает соответствующие описания данных на каждом уровне и согласование одних и тех же данных на разных уровнях. С этой целью в состав СУБД включаются специальные языки для описания представлений внутреннего и внешнего уровней. Кроме того, СУБД должна включать в себя язык манипулирования данными (ЯМД). Желательно, также наличие тех или иных дополнительных сервисных средств, например, средств генерации отчетов.

Работа с базами данных предполагает несколько этапов: описание БД; описание частей БД, необходимых для конкретных приложений (задач, групп задач); программирование задач или описание запросов в соответствии с правилами конкретного языка и использованием языковых конструкций для обращения к БД; загрузка БД и т. д.

Для выражения обобщенного взгляда на данные применяют *язык описания данных (ЯОД)* внутреннего уровня, включаемый в состав СУБД<sup>3</sup>. Описание представляет собой модель данных и их отношений, т. е. структур, из которых образуется БД.

ЯОД позволяет определять схемы базы данных, характеристики хранимых и виртуальных данных и параметры организации их хранения в памяти, и может включать в себя средства поддержки целостности базы данных, ограничения доступа, секретности.

ЯМД обычно включает в себя средства запросов к базе данных и поддержания базы данных (добавление, удаление, обновление данных, создание и уничтожение БД, изменение определений БД, обеспечение запросов к справочнику БД).

Исторически первым типом структур данных, который был включен в языки программирования, была иерархическая структура. Некоторые ранние СУБД также предполагали использование в качестве основной модели иерархические структуры типа дерева. Основанием для такого выбора было удобство представления (моделирования) естественных иерархических структур данных, существующих, например, в организациях.

В ряде предметных областей структура данных имеет более сложный вид, в котором поддерживаются связи типа «многие к одному», и которые могут быть представлены ориентированным графом. Такие структуры называют сетевыми. Для управления БД сетевой структуры

---

<sup>3</sup> Отсюда следует, что одна и та же БД может описываться по-разному на ЯОД различных СУБД.



международной ассоциацией Кодасил была предложена обобщенная архитектура системы с ЯОД схемы (модели БД) и подсхемы (модели части БД для конкретного приложения), а также ЯМД для оперирования с данными БД в прикладных программах.

В настоящее время разработаны десятки языков, основанных на реляционном исчислении, различие которых обусловлено особенностями математических теорий, положенных в основу их построения. Среди этих языков, можно выделить языки, базирующиеся на *С*-исчислении, предложенном Коддом, и *Р*-исчислении, предложенном Пиротти.

*С*-исчисление базируется на классическом прикладном исчислении предикатов. *Р*-исчисление представляет собой разновидность прикладного многотипного исчисления предикатов. Существенное различие между этими исчислениями, а, следовательно, и языками заключается в том, что в *С*-исчислении в качестве области изменения значений предметной переменной используется множество выборок (кортежей) отношения, а в *Р*-исчислении каждому типу переменных или констант соответствует определенный домен базы данных.

Функциональные характеристики языков отражают возможности описания данных, средств представления запроса, обновления, поддержки целостности и секретности, включения в языки программирования, управления форматом ответов, средств запроса к словарю данных БД и т.д.

Качественные характеристики языков запросов могут определяться такими свойствами, как полнота, селективная мощность, простота изучения и использования, степень процедурности и модульности, унифицированность, производительность и эффективность. Рассмотрим некоторые из этих понятий.

*Селективная мощность* языков запросов характеризует возможность выбора данных по разным критериям. Данное понятие плохо поддается формализации: можно сказать, что язык с большей селективной мощностью позволяет сформулировать большинство запросов так, что ответ на них содержит меньше ненужных данных. Языки, обладающие малой селективной мощностью, в общем случае уже требуют привлечения дополнительных средств для анализа ответов на запросы (например, оценки пользователя).

*Простота изучения* является во многом субъективной оценкой и может быть в некоторой мере охарактеризована степенью его близости к естественному языку, требуемым для его освоения временем и необходимым уровнем подготовки пользователя.

*Высокий уровень процедурности*, свойственный реляционным языкам, определяется присущими реляционной модели свойствами, в частности, полным отделением логической структуры данных от структур хранения и стратегий доступа. Снижение уровня процедурности увеличивает свободу в выборе способов реализации языка, что позволяет осуществить его реализацию более оптимальным способом. Но необ-



ходимо отметить, что меньшая степень процедурности еще не означает автоматически меньшую сложность написания запросов. Некоторые сложные запросы можно более просто сформулировать в виде алгоритма поиска ответа, в то время как его формулировка в декларативном виде может оказаться достаточно трудной.

*Модульность построения языка* характеризует возможность существования нескольких уровней языка и зависит от специфических свойств математической теории, лежащей в его основе. Минимальный уровень языка, обычно легко понимаемый пользователем, бывает достаточным для формулирования большинства запросов, и лишь формулировка сложных запросов может потребовать использования всех выразительных средств языка, о существовании которых пользователи начального уровня могут и не знать. Языки, не обладающие модульностью, требуют от пользователя знания почти всего объема средств языка, что усложняет процесс их изучения.

Наиболее распространенным языком для работы с базами данных является SQL (Structured Query Language), в своих последних реализациях предоставляющий не только средства для спецификации и обработки запросов на выборку данных, но так же и функции по созданию, обновлению, управлению доступом и т.д.

По существу SQL уже соединяет в себе и язык описания данных и язык манипулирования данными. Он не является полноценным языком программирования и, в случае его использования для организации доступа к БД из прикладных программ, SQL-выражения встраиваются в конструкции базового языка.

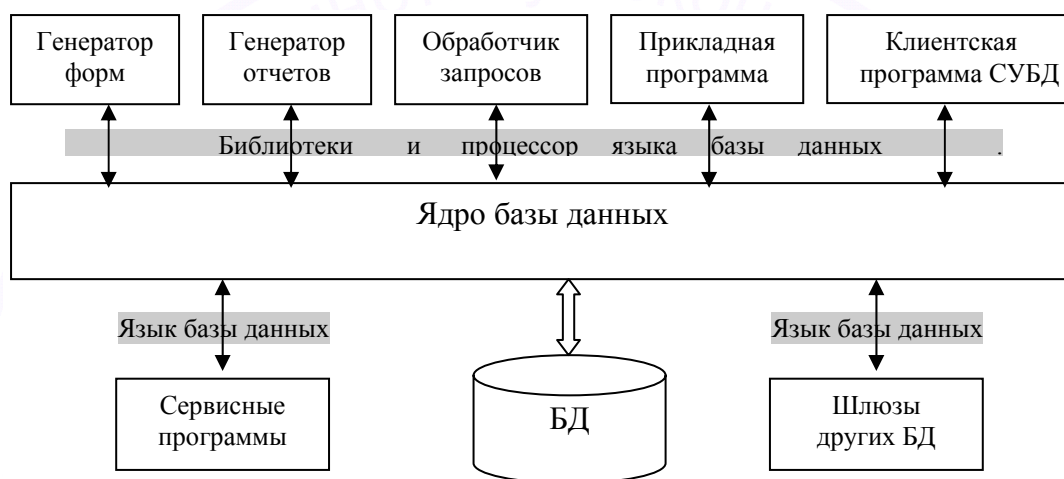
Являясь внутренним языком баз данных, SQL естественно отражает особенности конкретной СУБД. Сегодня это единственный стандартизованный язык фактографических баз данных, достаточно мощный и в то же время, простой для понимания и использования язык. Сочетание этих факторов вместе с поддержкой ведущих производителей, таких как IBM и Microsoft, привели не только к широкому его распространению, но и совершенствованию. Сегодня, благодаря независимости от конкретных СУБД и межплатформенной переносимости, SQL стал языком распределенных баз данных и языком шлюзов, позволяющим совместно использовать СУБД разного типа.

### *1.2.3. Программные средства*

Обработка данных и управление этой обработкой в вычислительной среде, а также взаимодействие с операционной системой и прикладными программами осуществляется комплексом программных средств, взаимосвязь которых иллюстрируется рис.1.3. В составе комплекса обычно выделяют следующие компоненты:

- **ядро**, обеспечивающее управление данными во внешней и оперативной памяти, а также протоколирование изменений;

- **процессор языка базы данных**, обеспечивающий обработку (трансляцию или компиляцию) и оптимизацию запросов на выборку и изменение данных;
- **подсистему (библиотеку) поддержки программных вызовов**, которая обслуживает прикладные программы управления данными, взаимодействующие с СУБД через средства пользовательского интерфейса;
- **сервисные программы** (системные и внешние утилиты), обеспечивающие настройку СУБД, восстановление после сбоев и ряд дополнительных возможностей по обслуживанию.



*Рис. 1.3. Программные средства СУБД*

Большинство СУБД работают в среде операционной системы и тесно с ней связаны. Многопользовательские приложения, обработка распределенных запросов, защита данных требуют эффективно использовать ресурсы, управление которыми обычно является функцией ОС. Использование многопроцессорных систем и мультитемных технологий обработки данных позволяет эффективно обслуживать параллельно выполняемые запросы, но требует координации использования ресурсов между ОС и СУБД. Соответственно, управление доступом и обеспечение защиты также обычно интегрируются с соответствующими средствами операционной системы.

Именно централизованное управление данными обеспечивает:

- сокращение избыточности в хранимых данных;
- совместное использование хранимых данных;
- стандартизацию представления данных, упрощающую эксплуатацию БД;
- разграничение доступа к данным;
- целостность данных, обеспечиваемую процедурами, предотвращающими включение в БД неверных данных и ее восстановление после отказов системы.

#### 1.2.4. Технические средства

Сегодня большинство банков данных создается и функционирует на основе универсальных вычислительных машин<sup>4</sup>. Однако, для больших баз данных, функционирующих в промышленном режиме, обеспечение эффективной и бесперебойной работы должно основываться на использовании адекватных аппаратных средств.

Устройства ввода-вывода и накопители внешней памяти - традиционно узкое место любой базы данных. Объем и быстродействие накопителей являются, очевидно, важными параметрами. Однако, столь же значима и отказоустойчивость. Здесь следует отметить необходимость согласованных решений при распределении ролей между аппаратными и программными компонентами управления операциями ввода-вывода. Например, наличие буферной памяти в накопителе ускоряющей ввод-вывод (аппаратное кэширование) при сбоях системы во время выполнения операции записи в БД может привести к потере данных: переданные для записи данные еще будут находиться в буфере, а т.к. СУБД уже отметит операцию записи как уже завершившуюся и откат для восстановления данных станет невозможен.

Для повышения надежности хранения часто используют специализированные дисковые подсистемы – RAID (Redundant Array of Inexpensive Disk). Один логический RAID-диск - это несколько физических дисков, объединенных в одно устройство, управляемое специализированным контроллером, что позволяет распределять основные и системные данные между несколькими носителями (дисками), в том числе дублировать данные. Таким образом, в случае повреждения одного из дисков, можно оперативно восстановить потерянные данные.

Не менее значима роль центрального процессора. Многие промышленные СУБД поддерживают многопроцессорную обработку запросов. Теоретически использование еще одного процессора позволит ускорить обработку. Однако на практике многопроцессорные системы требуют повышенного внимания при приобретении оборудования: надежно работают только сертифицированные системы, использующие соответствующие периферийные устройства.

Для распределенных и удаленно используемых баз данных также важно сетевое окружение: связанное оборудование и сетевые протоколы. Здесь важны не только показатели быстродействия, но и поддерживаемые ими возможности обеспечения безопасности.

---

<sup>4</sup> Здесь следует упомянуть достаточно интенсивно развивавшееся в 80-90гг. направление создания машин баз данных – аппаратной реализации «нечисловой» обработки, в том числе параллельной и конвейерной обработки, ассоциативных процессоров и памяти [Озкарахан]. Сегодня для реализации промышленных БД используются специализированные *серверы баз данных* – машины с повышенной отказоустойчивостью, высокопроизводительными подсистемами ввода-вывода и развитой периферией.

### 1.2.5. Организационно-административные подсистемы

Организационно-методические средства не являются технической компонентой системы, однако трудно рассчитывать на устойчивое и долговременное функционирование банка данных, если будут отсутствовать необходимые методические и инструктивные материалы, регламентирующие работу пользователей, различных по своему статусу и уровню подготовленности.

### 1.3. Пользователи баз данных

В информационных системах, создаваемых на основе СУБД, способности организации данных и методы доступа к ним перестали играть решающую роль, поскольку оказались скрытыми внутри СУБД. Массовый, так называемый *конечный пользователь*, как правило, имеет дело только с внешним интерфейсом, поддерживаемым СУБД.

Эти преимущества, как уже понятно, не могут быть реализованы путем механического объединения данных в БД. Предполагается, что в системе существует (как неотъемлемая составная часть) специальное должностное лицо (группа лиц) — *администратор базы данных (АБД)*, который несет ответственность за проектирование и общее управление базой данных. АБД определяет информационное содержание БД. С этой целью он идентифицирует объекты БД и моделирует базу, используя язык описания данных. Получаемая модель служит в дальнейшем справочным документом для администраторов приложений и пользователей. Администратор решает также все вопросы, связанные с размещением БД в памяти, выбором стратегии и ограничений доступа к данным. В функции АБД входят также организация загрузки, ведения и восстановления БД и многие другие действия, которые не могут быть полностью формализованы и автоматизированы.

*Администратор приложений* (или, если таковой специально не выделяется - администратор БД) определяет для приложений подмодели данных. Тем самым разные приложения обеспечиваются собственным «взглядом» но не на всю БД, а только на требуемую для конкретного приложения («видимую») ее часть. Вся остальная часть БД для данного приложения будет «прозрачна».

*Прикладные программисты* имеют, как правило, в своем распоряжении один или несколько языков программирования, с помощью которых генерируются прикладные программы.



#### 1.4. Типология баз данных

Классификация баз и банков данных может быть произведена по разным признакам (и относящихся к разным компонентам и сторонам функционирования БНД), среди которых выделяют, например, в [5] следующие.

**По форме представляемой информации** можно выделить фактографические, документальные, мультимедийные, в той или иной степени соответствующие цифровой, символьной и другим (не цифровой и не символьной) формам представления информации в вычислительной среде. К последним можно отнести картографические, видео, аудио, графические и другие БД.

**По типу хранимой (не мультимедийной) информации** можно выделить фактографические, документальные, лексикографические БД. Лексикографические базы – это классификаторы, кодификаторы, словари основ слов, тезаурусы, рубрикаторы и т.д., которые обычно используются в качестве справочных совместно с документальными или фактографическими БД. Документальные базы подразделяются по уровню представления информации – полнотекстовые (так называемые «первичные» документы) и библиографическо-реферативные («вторичные» документы, отражающие на адресном и содержательном уровне первичный документ).

**По типу используемой модели данных** выделяют три классических класса БД: иерархические, сетевые, реляционные. Развитие технологий обработки данных привело к появлению постреляционных, объектноориентированных, многомерных БД, которые в той или иной степени соответствуют трем упомянутым классическим моделям.

По *топологии хранения* данных различают локальные и распределенные БД.

**По типологии доступа и характеру использования** хранимой информации БД могут быть разделены на специализированные и интегрированные<sup>5</sup>.

**По функциональному назначению** (характеру решаемых с помощью БД задач и, соответственно, характеру использования данных) можно выделить операционные и справочно-информационные. К последним можно отнести ретроспективные БД (электронные каталоги библиотек, БД статистической информации и т.д.), которые используются для информационной поддержки основной деятельности, и не предполагают внесение изменений в уже существующие записи, например, по результатам этой деятельности. Операционные БД предназначены для управления различными технологическими процессами. В этом слу-

---

<sup>5</sup> В последнем случае правильнее говорить об интегрированных информационных системах, объединяющих в общей среде разнородные данные, хранимые возможно в разнотипных базах, но используемых для решения одной прикладной задачи.



чае данные не только извлекаются из БД, но и изменяются (в том числе добавляются) в том числе в результате этого использования.

**По сфере возможного применения** можно различать универсальные и специализированные (или проблемно-ориентированные) системы.

**По степени доступности** можно выделить общедоступные и БД с ограниченным доступом пользователей. В последнем случае говорят об управляемом доступе, индивидуально определяющем не только набор доступных данных, но и характер операций которые доступны пользователю.

Следует отметить, что представленная классификация не является полной и исчерпывающей. Она в большей степени отражает исторически сложившееся состояние дел в сфере деятельности, связанной с разработкой и применением баз данных.

#### *1.4.1. Типология баз данных с точки зрения информационных процессов*

С другой стороны, БД могут соотноситься с различными уровнями *информационных процессов*: уровень информационных технологий (ИТ), уровень системы (ИС), уровень информационных ресурсов (ИР).

На уровне информационных технологий БД определяется как взаимосвязанная совокупность файлов ОС, содержащих данные о предметной области решаемой задачи. При этом основное внимание уделяется *физической структуре БД*.

На уровне информационных систем БД рассматривается как компонента, представляющая собой информационную модель предметной области. Здесь наиболее важной является проблема *логической структуры БД*.

При рассмотрении БД на уровне информационных ресурсов БД трактуется как элемент мировых ИР. Основной характеристикой здесь является *содержание БД*, хотя и структуры данных также немаловажны.

Основное внимание в данном пособии будет уделяться рассмотрению БД на уровне технологии и систем, уровень ИР будет вкратце рассмотрен только в настоящей главе.

**Программные средства баз данных.** *Оболочки информационных систем* (системы программирования ИС) представляют собой гибкие программные комплексы, настраиваемые на задачи пользователя. Наиболее распространенными классами данных программных средств являются *системы управления базами данных (СУБД)* и *оболочки автоматизированных информационно-поисковых систем (АИПС)*.

**Информационно-поисковые системы.** В узком смысле под АИПС принято понимать открытый (обычно) или замкнутый (реже) программный продукт, предназначенный для реализации практически большинства функций (процессов) - *ввод, обработка, хранение, поиск, представление данных* (организованных в записи или документы, нахо-

дящиеся в БД). В этом смысле часто отождествляют АИПС с АИС, и это трудно оспаривать.

Среди АИПС в узком смысле принято выделять:

- *фактографические системы* (отличающиеся фиксированной структурой данных или записей), для разработки которых как правило, используются СУБД, поддерживающие *табличные (реляционные) БД*;

- *документальные системы* (отличающиеся неопределенной или переменной структурой данных или документов) и для разработки которых часто (но не обязательно) применяют оболочки АИПС.

В более широком смысле под АИПС подразумеваются также *программные оболочки, ориентированные на разработку продуктов типа АИПС* (в узком смысле). Это связано с тем фактом, что первые системы типа СУБД и оболочек АИПС были предложены в 60-е - 70-е годы фирмой IBM (и сотрудничавшими с ней организациями) и включали в себя:

- IMS/360 (Information Management System) - по-видимому, первая реальная СУБД, поддерживавшая т.н. *иерархическую модель данных* (понятие появилось позже, в связи с необходимостью систематизации СУБД), нашедшая достаточно широкое применение (в частности, для информационного обеспечения проекта Apollo, завершившегося, как известно, высадкой граждан США на Луну в 1969 г.);

- DPS/360 (Document Processing System) - первый промышленный пакет прикладных программ (ППП), предназначенный для реализации документальных АИПС. В дальнейшем путем развития принципов DPS, фирмой в 1972 г. был выпущен пакет STAIRS (STorage And Information Retrieval System), предназначенный для диалогового обслуживания множества (удаленных) пользователей;

- IRMS (Information Retrieval and Management System), TEXT-RAC и другие аналогичные пакеты.

Как это следует из наименований продуктов, разработчики понимали под АИПС именно ППП-оболочки.

### **Системы управления базами данных и программирования АИС**

Среди различных программных средств данного класса следует различать три типа:

- СУБД в “чистом виде” (IMS, SETOP и пр.);
- СУБД с элементами систем программирования АИС (ADABAS/NATURAL, ORACLE);
- системы программирования АИС с элементами СУБД (FoxBase / FoxPro, Clipper).

Первый тип фактически относится к начальному этапу развития систем второго (реже - третьего) типов. В этом случае СУБД состоит только из системы интерпретации вызовов (обращений) из пользовательской программы (call-interface) на выборку (корректировку, занесение) информации из (в) БД, причем программа написана на одном из универсальных языков программирования (ЯП), таких как, Кобол, Фортран, Паскаль и пр., получивших название *включающие языки СУБД*. Данная система в последующих СУБД (второй тип) получила наименование *ядра*. Соглашения о форматах и структурах такого взаимодействия обычно пытаются оформить в виде некоторого формального *языка (языка ядра)*. В частности, вдохновленная успехами в разработке и распространении универсального ЯП PL/1 (Programming Language #1), фирма IBM разработала описание форматов интерфейса пользовательских программ с БД IMS в форме языка DL/1 (Data Language #1), который, однако, значительного успеха не имел.

Второй тип представляет собой расширение первого в направлении создания универсальной системы разработчика АИС, включающей также специализированные языковые средства. В этом случае СУБД представляет собой совокупность специализированных программных средств, вспомогательных файлов и управляющих таблиц (иногда находящихся в составе БД, реже это файлы ОС), которая обеспечивает доступ пользователей к БД при соблюдении следующих существенных критериев: целостность и непротиворечивость данных, описывающих различные аспекты объектов реального мира, защита информации от несанкционированного доступа на чтение/обновление содержимого БД, установление и поддержание связей между зависимыми данными, удобство использования данных.

Третий тип представляют собой (разработанные обычно для ПК) системы, содержащие как элементы непроцедурного типа (язык запросов), так и процедурного (язык программирования) во входном языке, предназначенном для управления данными и обработки информации. Элементы СУБД здесь также заключаются в наличии простейшего словаря данных, возможности создания модели предметной области в форме совокупности таблиц, связанных между собой простейшим образом, а также в наличии средств генерации отчетов и управления доступом пользователей.

### ***1.5. Семантика баз данных<sup>6</sup>***

Как уже отмечалось, база данных не может рассматриваться в отрыве от назначения и особенностей ее использования для решения практических задач, причем обязательно в составе более крупных информа-

---

<sup>6</sup> Материал этого и следующего параграфов является не только введением в проблематику проектирования и эксплуатации баз данных, но и, может быть, несколько опережающим обобщением того, что будет представлено в дальнейших главах.

ционных или технологических автоматизированных систем. Задачи таких систем – это не только планирование и управление предприятием, но и интеграция разработки и сопровождения основных и технологических объектов и процессов, диагностика, мониторинг, моделирование. Соответственно, задачи и назначение БД, как системы хранящей информацию обо всех этих составляющих – обеспечить информационную поддержку этих процессов.

База данных – это отражение реальной предметной области, «действующая» информационная модель<sup>7</sup>, которая, обеспечивая субъект информацией для принятия решения, позволяет, в том числе, и управлять объектами и процессами в отражаемой предметной области (ПрО). Такая функциональная направленность (и, естественно, предполагающая достижение эффективности в первую очередь за счет использования именно БД) обуславливает и обратную зависимость: объекты, процессы и события ПрО выделяются таким образом, чтобы было возможно их представление в виде системы взаимосвязанных данных и процессов, удобных для их последующей (человеко-машинной!) обработки.

В каком-то смысле базу данных можно сравнить с сообщением о состоянии предметной области, воспринимаемым некоторым субъектом, задачей которого и является преобразование объектов этой ПрО, причем в своей деятельности субъект руководствуется информацией извлекаемой именно из этого «сообщения». Схема этого соотношения, приведенная на рис. 1.4, иллюстрирует еще и то, что система, преобразующая объект, принципиально является комплексной (состоящей, по крайней мере, из двух компонент, работающих с объектами разной природы: субъект преобразования взаимодействует преимущественно с материальными объектами, а БД – с информационными).

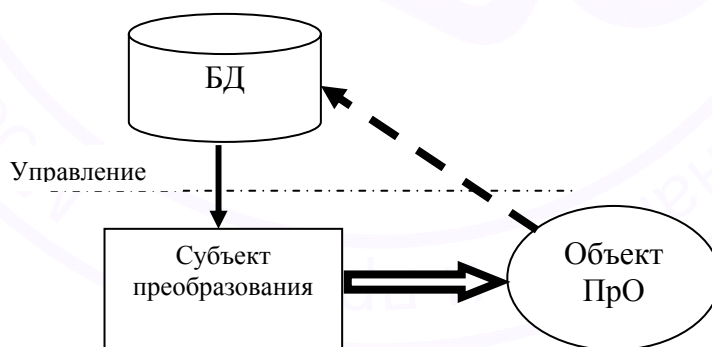


Рис. 1.4. Информационная модель преобразования

В общем случае, поскольку для многокомпонентных систем с многоуровневым представлением семантики, эффективность обработки достигается через специализированность представления объектов или процессов (а для вычислительных систем - как среды хранения информации

<sup>7</sup> Модель – лишь в том смысле, что она – представление, описание на уровне данных только некоторых аспектов, и только некоторой части реального мира, и поэтому не может быть тождественна реальным объектам. Но в тоже время БД и сама является частью реального мира.



- с *единственно возможной двоичной* формой представления) и, в первую очередь, путем сведения представления множества (локально) обрабатываемых объектов к однородности природы и формы их представления, то для реализации эффективного межуровневого взаимодействия (на каждом из которых объекты представлены в виде, наиболее адекватном функциональным средствам этого уровня) любая величина должна быть преобразована в соответствии с «контекстом» этого уровня для получения такого ее представления, которое будет «значимо» для воспринимающего уровня, т.е. может быть обработано средствами этого уровня.

Здесь «контекст» - это декларативное или, иногда, процедурное определение способа использования элементарных составляющих величины для получения значения. Например, порядок использования байтов при преобразовании вещественного числа, представленного в двоичной форме, в символьный формат.

Соотношение понятий «величина», «контекст» и «значение» приведено на рис. 1.5. Здесь значение, получаемое на первом уровне, на следующем рассматривается в свою очередь как величина, которая будет интерпретироваться в соответствии с контекстом своего уровня<sup>8</sup>.

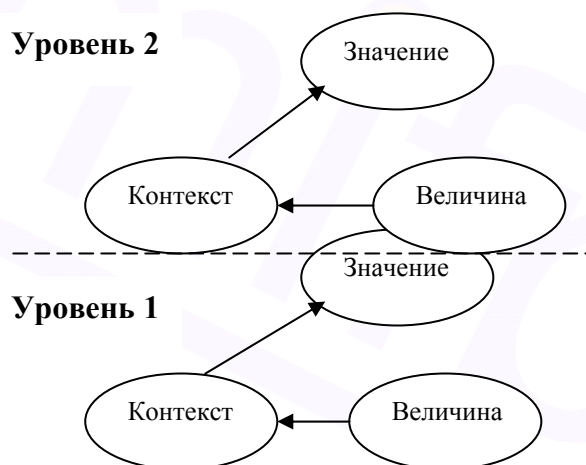


Рис. 1.5. Соотношение понятий «величина», «контекст» и «значение»

Таким образом, можно сказать, что значение в общем случае определяется парой <контекст, величина>. Причем, поскольку *контекст* и *величина* имеют разную природу, они должны быть представлены в вычислительной среде самостоятельными, скорее всего, разнотипными объектами.

Такое, хотя и упрощенное, представление БД как средства информационных коммуникаций, позволяет тем не менее увидеть взаимосвязь вида информации (способа реализации смысла) с формой ее представления и особенностью ее использования.

<sup>8</sup> Соотношение понятий «величина» и «значение» аналогично соотношению понятий «данные» и «информация». Информация - это значимые для приемника данные, например изменяющие его внутреннее состояние.



В этом смысле (с точки зрения способа представления и, соответственно, восприятия) в отдельный класс можно выделить *фактографическую информацию*: такое представление реально существующих событий и явлений, когда они могут быть описаны как *факты*, задаваемые парой *<имя, значение>*, где *имя* – знак, уникально определяющий (идентифицирующий) факт в заданной предметной области, и обычно не нуждающийся в явном определении или доопределении его существа; а *значение* – характеристика, задающая одно из множества возможных состояний.

Т.е., здесь факт (его значение) задается величиной, например, числовой для физически измеримых параметров, в том числе и логическими величинами «истина» / «ложь» для указания свершилось событие или нет<sup>9</sup>.

Можно сказать, что особенностью фактографической информации является практическая очевидность (минимальная неопределенность, не требующая использования сложных или нечетких процедур) идентификации и интерпретации «факта», как его имени, так и состояния. Т.е., контекст в этом случае в достаточной степени определяется однозначно понимаемым объявлением о назначении базы данных и таким именовании полей данных, когда в качестве имени используется общепринятое, не зависящее от прикладных задач, *имя свойства* (и таким образом определяются характеристические признаки). Такая ситуация предопределяет для пользователя возможность адекватного восприятия содержания: способ интерпретации данных в этом случае практически не может быть неоднозначным, причем для пользователя *определение способа* происходит *неявно* (не требует от него явных действий для определения и использования контекста). Это, с одной стороны, позволяет свести представление предметной области к точной теоретико-множественной модели, а с другой – обуславливает возможность непосредственного использования данных в задачах обработки (на уровне прикладных программ) для генерации новой информации без участия субъекта (человека), внешнего по отношению к машинной среде, обеспечивающего определение и использование контекста. Например, OLAP-технологии баз данных, позволяющие строить на основе множества данных, количественно характеризующих состояние объектов предметной области и представленных обычно регулярными таблицами, новые значения, отражающими это состояние на ином *качественном* уровне, например интегральные показатели, диаграммы, графики и т.д.

Однако большинство задач, решаемых человеком, не могут быть сведены к «фактографическому» представлению и описываются (и, соответственно, представляются в машинной среде) средствами естествен-

---

<sup>9</sup> И, следует отметить, что такая форма в наибольшей степени соответствует машинным формам представления информации.

ного или специализированного языков, оперирующих *лингвистическими переменными*, значение которых может зависеть не только от контекста предметной области, но также и от контекста ближайшего окружения – значения соседних переменных. Причем, появление нового смысла (факта) не обязательно приводит к появлению новой переменной: новый факт представляется с помощью уже существующих переменных. Например, словесные определения философских или географических понятий.

В отличие от ранее рассмотренного фактографического представления, для вербальной формы представления факта (выражениями языка с использованием лингвистических переменных) характерно то, что для задания *имени, значения и контекста* может использоваться единый способ и средства – лингвистические переменные одного и того же языка. Например, описание весовых свойств может быть представлено несколькими, но имеющими один смысл, вариантами предложений: «Чугунная заготовка весом 29 килограмм» или «Чугунная заготовка имеет свойство  $m = 29$ , где  $m$  – вес в килограммах».

Автоматическое приведение такого рода представлений к очевидной наилучшей для этого случая табличной форме, потребовало бы применения трудно реализуемых процедур морфологического и семантического анализа. Но, с другой стороны, выделение смысла (и генерация новой информации) обычно производится человеком, сознание которого (как среда преобразования) ориентировано именно на обработку лингвистических переменных.

Рассматривая процесс автоматизированной генерации новой информации (рис. 1.6), где в качестве источника исходных данных используются БД, нужно сказать, что отбор и обработка должны быть выделены в отдельные процессы, т.к. с точки зрения общей (суммарной) эффективности один из них (обычно поиск) должен быть опосредованным – оценка полезности найденной информации производится обычно человеком, т.к. сознание человека – внешняя по отношению к машине среда, работает со слабоструктурированной информацией эффективнее машин.

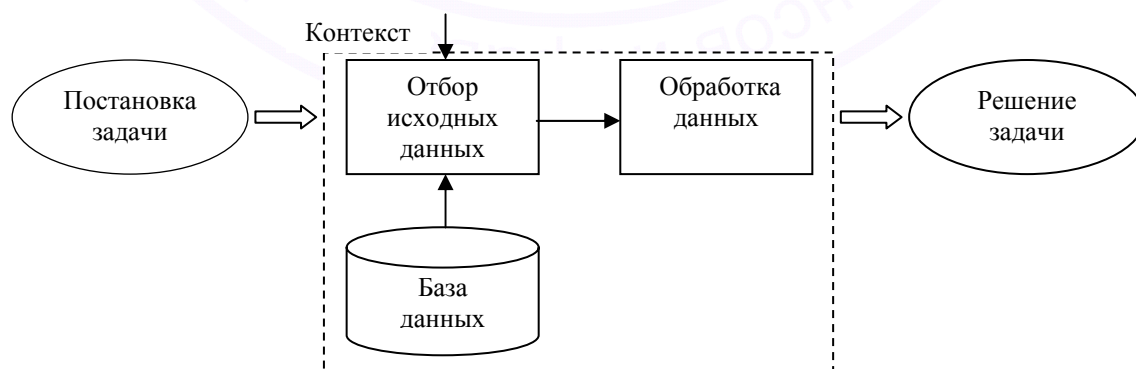


Рис. 1.6. Схема процесса автоматизированного решения задач

Случаи, когда информация представляется в форме не адекватной архитектуре Фон-Неймановских машин, могут быть обусловлены разными факторами. Рассмотрим следующие случаи.

1. Хорошо структурированная информация представляется в графическом или специальном формате. Например, структурные химические формулы, конструкторская документация и т.д. В этом случае для автоматической обработки требуются узко специализированные средства, что приводит к общей не унифицированности представления семантических элементов (например, графических примитивов) на уровне данных.

2. Информация точная по содержанию, но вариантно представляемая по форме. Например, описание в текстовом виде численно задаваемых параметров изделия. Лингвистические переменные в этом случае имеют точное значение, однако построение универсальной процедуры автоматического выделения факта из текста трудоемко и потому нецелесообразно.

3. Слабоструктурированная информация, обычно представляемая в текстовой форме. Например учебная или научная публикация, где новые понятия строятся на основании ранее определенных. В этом случае значения лингвистических переменных могут принимать новые, ранее не определенные значения, которые определяются контекстом - ближним (словосочетания) или общим (темой сообщения).

Возвращаясь к процедуре поиска, как важнейшей составляющей использования баз данных, еще раз отметим, что критерий отбора должен содержать не только величину (например, слово), но и контекст.

В реальных системах поиск документальной информации<sup>10</sup>, представленной в текстовой форме, производится по вторичным документам – специально создаваемым поисковым образам точно идентифицирующим сам документ как единицу хранения, и приблизительно, в краткой форме путем *перечисления* основных понятий, отражающий смысловое содержание. Такой подход позволяет построить процедуры поиска на основе теоретико-множественной модели с точной логикой отбора по критерию наличия заданного сочетания терминов запроса в списке терминов поискового образа. Однако контекст использования терминов должен быть доопределен отдельно – либо во время поиска, например указанием тематической области, либо после отбора из базы – во время ознакомления человека с содержанием найденного.

Определение контекста предметной области в целом осуществляется с помощью тезаурусов терминологических систем, фиксирующих с помощью родо-видовых и других отношений роль и семантику дескрип-

---

<sup>10</sup> Это соответствует третьему из вышеперечисленных случаев. Два первых мы не рассматриваем, т.к. в этих случаях используются специализированные системы.

торов – выделенных терминов, которые используются для формирования поисковых образов документов.

Для доопределения смысла термина в составе поискового образа документа в первых поколениях автоматизированных информационных систем применялись специальные указатели роли, однако их использование было трудоемко и требовало специальной подготовки пользователя, поэтому в современных системах не применяется.

Другой важный фактор, влияющий на эффективность работы человека с информацией это форма хранения и представления – структура и оформление документа. Это особенно заметно при работе с объемными полнотекстовыми документами, причем иногда это определяется на уровне машинного формата (например, DOC, PDF, HTML и т.д.), от выбора которого зависит возможность дальнейшей обработки.

В том случае, когда для хранения информации используются базы данных, структура документов может быть определена двумя путями<sup>11</sup>:

- 1) так же как и для фактографических БД, заданием схемы – последовательности именованных типизированных полей данных;
- 2) контекстным определением – использованием специализированных языков разметки (например, HTML или XML), задающим индивидуальные особенности представления материала каждого документа.

Использование встраиваемых определений структуры позволяет ввести «самоопределяемые» форматы представления документов. Это обеспечивает практически неограниченную гибкость при организации хранения коллекций разнородных документов, однако создает проблемы семантические проблемы согласованного использования материала (из-за возможности различной интерпретации определений), что в свою очередь требует создания доступного всем пользователям репозитория метайнформации – описаний природы и способов представления информации.

## **1.6. Типология моделей**

Основные отличия любых методов представления информации заключаются в том, каким способом фиксируется семантика предметной области. Но, следует особо отметить, что для всех уровней и для любого метода представления предметной области (но для нас важно, что *в контексте создания и использования машинных баз данных*) в основе отображения (т.е., собственно формирования представления) лежит *кодирование* понятий и отношений между понятиями. Многоуровневая система

---

<sup>11</sup> Для реляционной СУБД MS SQL Server 2000 реализован импорт/экспорт документов, представленных в XML-формате, в том числе с использованием схем сопоставления, определяющих соотношение элементов XDR-схем таблицам, а атрибутов – столбцам.



моделей представления информации иллюстрируется рис. 1.7. Рассмотрим далее основные из них.

Ключевым этапом при разработке любой информационной системы является проведение системного анализа: формализация предметной области и представление системы как совокупности компонент. Системный анализ позволяет, с одной стороны лучше понять «что надо делать» и «кому надо делать» (аналитику, разработчику, руководителю, пользователю), а с другой - отслеживать во времени изменения рассматриваемой модели и обновлять проект.

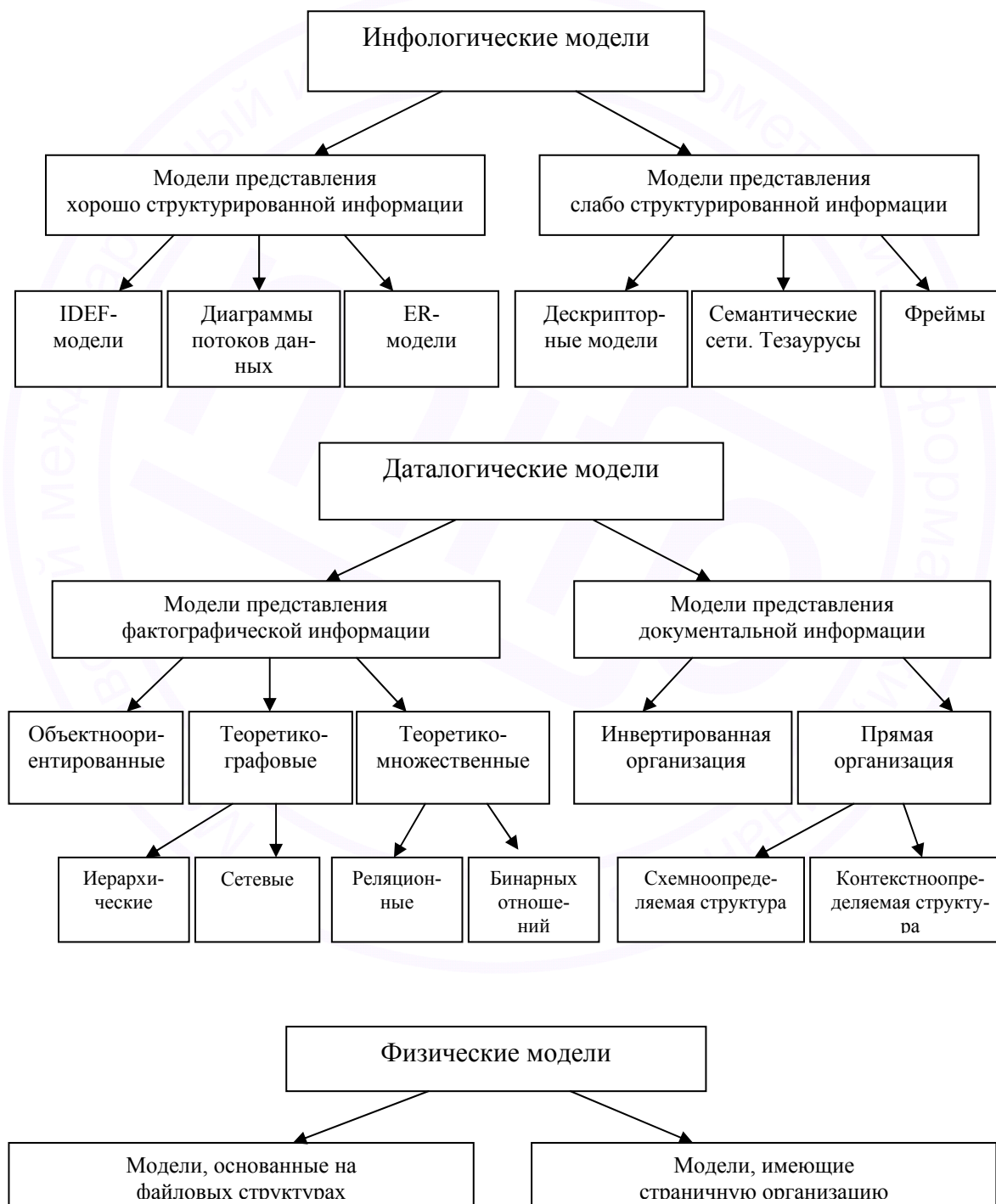


Рис. 1.7. Система моделей представления информации

Декомпозиция, как основа системного анализа, может быть функциональной (построение иерархий функций) или объектной.

Однако в большинстве систем, если говорить, например, о базах данных, типы данных являются более статичным элементом, чем способы их обработки. Поэтому получили интенсивное развитие такие методы системного анализа, как диаграммы потоков данных (Data Flow Diagram). Развитие реляционных баз данных в свою очередь стимулировало развитие методик построения моделей данных, и в частности, ER-диаграмм (Entity Relationship Diagram). Однако и функциональная декомпозиция и диаграммы потоков данных дают только некоторый срез исследуемой предметной области, но не позволяют получить представление системы в целом.

Различаются и методы отображения, используемые на этапе построения даталогических моделей, отражающих способ идентификации элементов и связей, но, что особенно важно, в контексте их будущего представления в одномерном пространстве памяти вычислительной машины. Модели подразделяются на фактографические - ориентированные на представление хорошо структурированной информации, и документальные - представляющие наиболее распространенный способ отражения слабоструктурированной информации. Если в первом случае говорят о реляционной, иерархической или сетевой моделях данных, то во втором — о семантических сетях и документальных моделях.

При проектировании информационных систем свойства объектов (их характеристики) называются атрибутами. Именно значения атрибутов позволяют выделить в предметной области как различные объекты (типы объектов), так и среди объектов одного типа — их различные экземпляры. Представление атрибутов удобнее всего моделируется теоретико-множественными отношениями. Отношение наглядно представляется как таблица, где каждая строка — кортеж отношения, а каждый столбец (домен) представляет множество значений атрибута. Список имен атрибутов отношения образует схему отношения, а совокупность схем отношений, используемых для представления БД, в свою очередь образует схему базы данных.

Представление схем БД в виде схем отношений упрощает процедуру проектирования БД. Этим объясняется создание систем, в которых проектирование БД ведется в терминах реляционной модели данных, а работа с БД поддерживается СУБД одного из описанных в данном пособии типов.

Основное отличие методов представления информации заключается в том, каким способом фиксируется семантика предметной области. Первые, фактографические БД, задают четкую 'схему соответствия, в рамках которой и отображается предметная область. Подобное построение по сути своей является довольно статичным, требует априорного знания типов отношений. В нем достаточно сложно вводить информацию о новых типах отношений между объектами, но, с другой стороны,

зафиксированная схема базы данных позволяет довольно эффективно организовать поиск информации.

Во втором случае предметная среда отображается (по крайней мере, на уровне модели) в виде однородной сети, любые изменения которой, как по вводу новых классов объектов, так и новых типов отношений, не связаны с какими-либо структурными преобразованиями сети. В силу большого количества типов отношений манипулирование подобной "элементарной" информацией достаточно затруднено, поэтому для данного случая характерно введение большого количества более общих понятий (и соответствующих им отношений), что упрощает работу с сетью.

Модель данных должна, так или иначе, дать основу для описания данных и манипулирования данными, а также дать средства анализа и синтеза структур данных. Любая модель, построенная более или менее аккуратно с точки зрения математики, сама создает объекты для исследования и начинает жить как бы параллельно с практикой.

Реляционная модель данных в качестве основы отображения непосредственно использует понятие отношения. Она ближе всего находится к так называемой концептуальной модели предметной среды и часто лежит в основе последней.

В отличие от теоретико-графовых моделей в реляционной модели связи между отношениями реализуются неявным образом, для чего используются *ключи отношений*. Например, отношения иерархического типа реализуется механизмом первичных / внешних ключей, когда в подчиненном отношении должен присутствовать набор атрибутов, связывающих это отношение с основным. Такой набор атрибутов в основном отношении будет называться первичным ключом, а в подчиненном – вторичным.

Прогресс в области разработки языков программирования, связанный, в первую очередь с типизацией данных и появлением объектно-ориентированных языков, позволил подойти к анализу сложных систем с точки зрения иерархических представлений - классам объектов со свойствами инкапсуляции, наследования и полиморфизма, схемы которых отображают не только данные и их взаимосвязи, но и методы обработки данных.

В этом смысле объектно-ориентированный подход является гибридным методом и позволяет получить более естественную формализацию системы в целом. В итоге это позволяет снизить существующий барьер между аналитиками и разработчиками (проектировщиками и программистами), повысить надежность системы и упростить сопровождение, в частности, интеграцию с другими системами. Модель будет структурно объектно-ориентированной, если она поддерживает сложные объекты; модель будет поведенчески объектно-ориентированной, если она обеспечивает процедурную расширяемость; для того чтобы модель

была полностью объектно-ориентированной, она должна обладать обоими свойствами.

Разделение на фактографические и документальные в этой группе моделей является достаточно условным. Документ, как последовательность полей может быть представлен, в том числе, и реляционной моделью. И в этом случае выбор специализированного решения чаще всего обуславливается требованием общей эффективности.

Представленная здесь типология моделей не претендует на полноту, и она не является классификацией в точном смысле этого слова. Она скорее иллюстрирует эклектичность преобладающих в разное время взглядов, методов и решений, используемых при проектировании и реализации баз данных.

### *Контрольные вопросы*

1. Дайте определение понятиям «База данных» и «банк данных».
2. Каковы предпосылки создания баз и банков данных.
3. Перечислите преимущества и недостатки использования БД.
4. Определите соотношение понятий «информация» и «данные».
5. Перечислите и определите назначение основных компонентов БД.
6. Определите основные функции и назначение СУБД
7. Назовите отличительные особенности БД.
8. Перечислите основные требования, предъявляемые к БД.
9. Какие технические средства используются для создания баз данных.
10. Перечислите основные признаки классификации БД.
11. Определите понятие и назначение лингвистических средств БД.
12. Перечислите основные категории пользователей БД.
13. Перечислите основные функции администратора БД.
14. Укажите взаимосвязь этапов создания БД и используемых моделей предметной области.
15. В чем различие между структурированной и слабоструктурированной информацией.
16. Приведите классификационную схему моделей БД.



## Глава 2. Базовые технологии и основные этапы развития машинной обработки данных

### 2.1. Введение в технологии машинной обработки данных и основные определения

Реальные базы данных промышленного масштаба содержат миллионы записей, данные которых описывают состояния и взаимосвязи многих и многих объектов реального мира. Требования, предъявляемые пользователями к автоматизированным или автоматическим системам обрабатывающим эти данные, обуславливают и требования к параметрам подсистем внешней памяти, в первую очередь предполагают высокую оперативность доступа.

Важной особенностью здесь является то, что архитектура систем и технологий управления данными непосредственно связана с двумя следующими значительными, хотя и противоположными обстоятельствами:

- непредсказуемой вариантностью представления данных в прикладной программе, зависящей от разнообразных особенностей пользовательских задач;
- жесткостью технических решений устройств внешней памяти, выражающейся в функциональной простоте<sup>12</sup> операций и ограниченности форм представления данных.

Высокая эффективность решений в области обработки данных достигается введением промежуточных слоев специализированных технических и программных средств. Характер проблем и архитектурно-технологические решения такого рода достаточно полно иллюстрируются приведенной на рис.2.1 примерной схемой реализации операций ввода-вывода - взаимодействия прикладной программы с компонентами операционной системы и устройствами внешней памяти. Здесь *специализация* компонентов выражается в том, что по существу каждый из них реализует различные способы работы с потоком данных (и, в частности, его фрагментацию на блоки), что и обеспечивает с одной стороны необходимый уровень декомпозиции и идентификации логических/физических записей, а с другой – независимость физического и логического уровней представления данных.

Здесь термины *логический* и *физический* отражают различия аспектов представления данных. *Логическое представление* указывает на то,

---

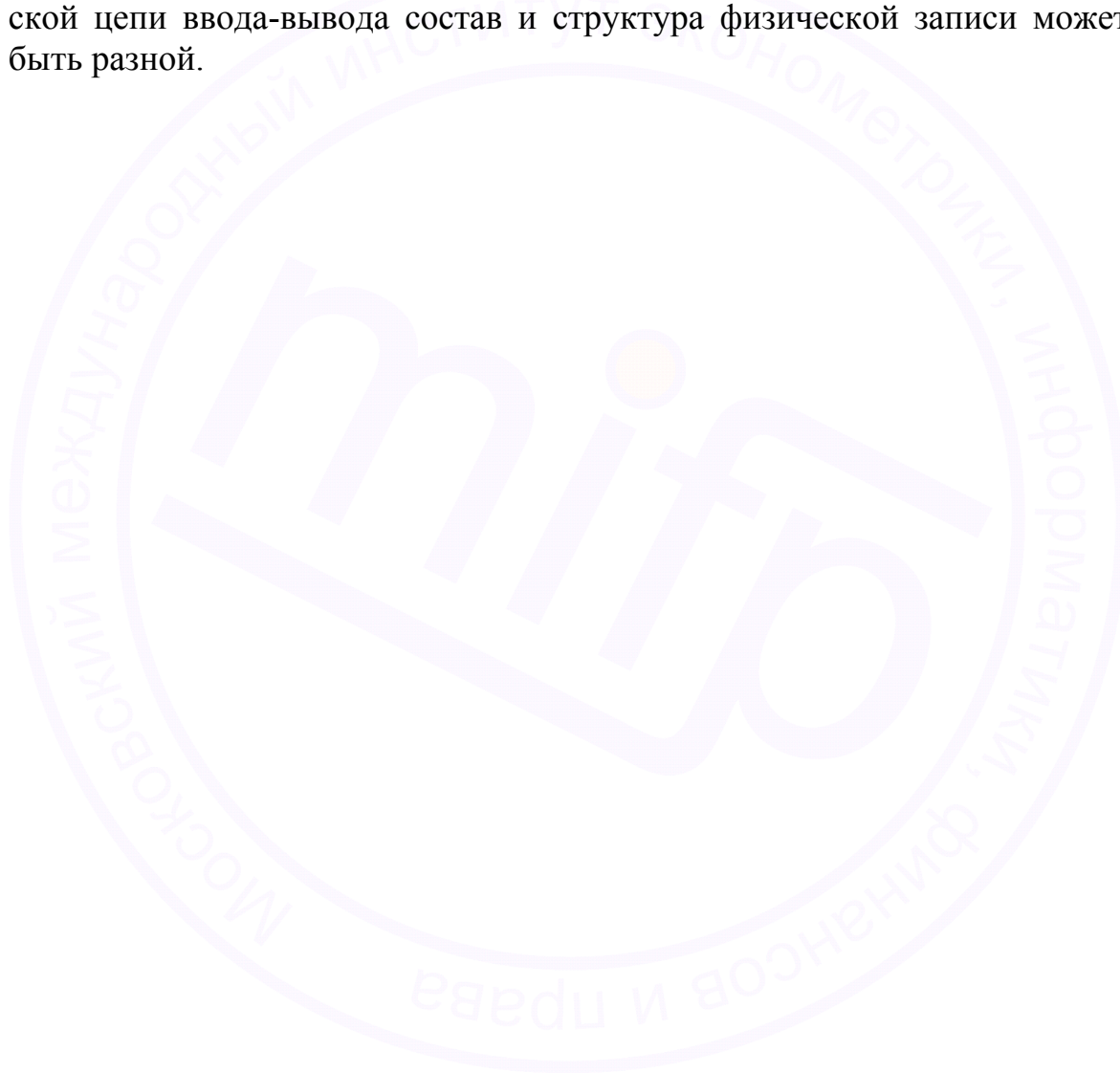
<sup>12</sup> Требование операционной простоты определяется производственными и экономическими причинами: устройство должно быть надежным в использовании и дешевым в изготовлении (т.е., содержать минимум механических компонент и сложной логики).

Функциональная ограниченность управления данными, кроме того, диктуется еще и требованием *унифицированности*: устройство должно одинаково эффективно и стандартным способом использоваться в составе различных вычислительных и операционных систем, причем даже если со временем отдельные компоненты систем будут принципиально меняться.

как данные используются в прикладной программе, т.е. – отражают логику обработки. *Физическое представление* – это то, как данные хранятся на *физическом носителе*.

Будем считать *логической записью* идентифицируемую (именованную) *совокупность элементов или агрегатов данных*, воспринимаемую прикладной программой как единое целое при обмене информацией с внешней памятью (по крайней мере, для операций ввода-вывода).

*Физической записью* будем считать совокупность данных, которая может быть считана или записана как единое целое одной *командой ввода-вывода*. Важно, что для компонент различного уровня в технологической цепи ввода-вывода состав и структура физической записи может быть разной.



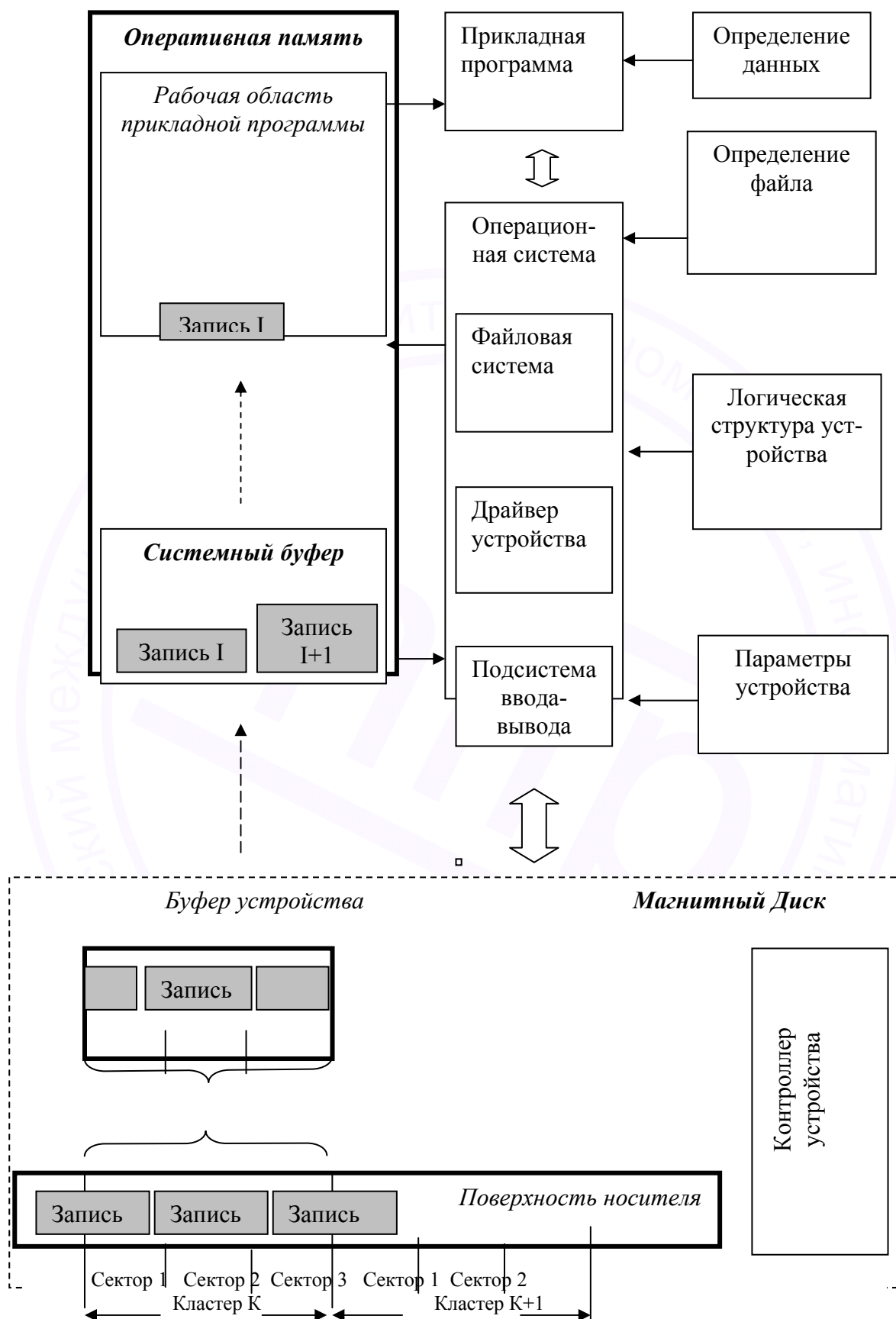


Рис. 2.1. Примерная схема организации ввода-вывода

Структура данных и их взаимосвязь в случаях логического и физического представления могут не совпадать. Например, а) одна физическая запись может включать несколько логических; б) порядок следования элементов данных в физической записи может быть изменен для оптимизации использования пространства памяти. То есть, если логическая структура может варьироваться в широком диапазоне и даже представляться, например, вариантными записями, то физическая - практически всегда представлена жесткой структурой, причем в значительной степени определяемой типом носителя.

## ***2.2. Схема организации файлового ввода-вывода***

Рассмотрим представленные на рис. 2.1 основные способы адресации и последовательность операций<sup>13</sup> выборки данных, обеспечивающих чтение прикладной программой с тома внешней памяти (например, магнитного диска ПЭВМ) некоторой произвольной (I-ой) записи. Отметим еще раз, что «специализация» компонент, участвующих в операциях ввода-вывода, выражается прежде всего в используемом способе адресации.

Прикладная программа использует одномерную (или сводимую к одномерной) сквозную адресацию данных на уровне логических записей: запись определяется номером, например, соответствующему порядку их размещения.

Система управления физическим вводом-выводом (в рассматриваемом примере - BIOS ПЭВМ) использует трехмерную систему координат: адрес записи составляется из номера дорожки, номера головки чтения-записи (номер поверхности) и номера сектора. Т.е., операционная система использует одномерную сквозную систему координат: сектора нумеруются от края диска к центру последовательно, причем сначала в рамках одного сегмента цилиндра (кластера), далее сектора следующего сегмента дорожки, после чего происходит переход к следующей дорожке.

Этот способ адресации и, соответственно, порядок использования пространства отчасти отражает специфику аппаратных решений, ориентированных на временную оптимизацию операций ввода-вывода: большее количество данных будет считано при одном обращении к диску за счет одновременного обращения через головки чтения-записи к данным, размещенным на параллельных дорожках в одном секторе одного цилиндра. Фиксированное количество битов, равное размеру сектора, определенное при разметке, умноженному на число головок, будет прямо (без дополнительной обработки, например, проверки логических усло-

---

<sup>13</sup> В целях общности в этом примере не рассматриваются подготовительные операции, такие как открытие файла и выделение памяти для рабочих и системных буферов, хотя они также достаточно ресурсоемки.



вий конца файла или записи) предано в буфер оперативной памяти устройства или операционной системы.

Таким образом, если система адресации в прикладной программе является относительной и отражает логику взаимосвязи записей (например, порядок создания файла), то для подсистем ввода-вывода она является абсолютной и определяется *физическим форматом носителя*: размером сектора, количеством секторов на дорожке, количеством поверхностей и дорожек и т.д. При этом независимость от особенностей физического размещения и механизма адресации обеспечивается на уровне *логической структуры носителя*.

Например, логически последовательная выборка записей файла обеспечивается таблицей размещения файлов, определяющей используемое файлом пространство как цепочку кластеров, физически находящихся в любой доступной части диска. Доступ к файлу производится по идентификатору (составному имени) через систему каталогов, связывающих идентификатор файла с началом цепочки указателей на кластеры данных в таблице размещения файлов. Кроме того, логическая структура содержит (в составе загрузочной записи) информацию, идентифицирующую пространство в целом, а также данные, *определяющие физическую структуру* (физический формат носителя, рассмотренный ранее).

В общем случае операция чтения физической записи включает следующие действия:

1. Определение адреса записи в координатах устройства (например, для файлов с записями фиксированной длины - пересчетом номера нужной записи в относительный адрес сектора и далее определение абсолютного номера сектора на диске);

2. Перемещение головки чтения в соответствующую координату: позиционирование к дорожке и сектору на дорожке, складывающееся из двух действий – собственно радиального перемещения головки на расстояние от текущего положения до нужной дорожки и ожидания подхода указанного сектора вращающегося диска к позиции, где находится головка. Следует также отметить, что высокая плотность записи данных означает, что промежуток между секторами<sup>14</sup> и дорожками сравнительно мал (сопоставим с погрешностями механизма перемещения и тепловым расширением), и поэтому правильность позиционирования определяется

---

<sup>14</sup> Если контроллер не успевает завершить обработку передачи и подготовиться к передаче данных, размещаемых на физически следующем секторе, то придется ожидать завершения полного оборота диска. С целью исключения таких потерь диск форматируется так, что логически последовательные секторы разделены одним или несколькими физическими секторами (коэффициент чередования) так, что контроллер будет готов выполнить операцию со следующим логическим сектором не ожидая дополнительного оборота.

по служебным данным заголовка<sup>15</sup> сектора, считываемым до начала передачи прикладных данных;

3. Пересылка данных, расположенных в области кластера, в буфер, который физически может быть как частью устройства, так и областью оперативной памяти;

4. Завершение операции (проверка корректности чтения, например по контрольной сумме) и возврат управления ОС для обработки считанных данных;

5. Выделение системой данных, относящихся к затребованным записям. Причем во многих случаях в системный буфер считываются не только данные логической записи, нужные прикладной программе, но и соседние. Это позволяет сократить суммарные затраты времени при чтении нескольких записей, исключив наиболее долгую операцию позиционирования. Указание на такое *блокирование* может выдаваться явно прикладной программой при открытии файла, или операционной системой, использующей собственные механизмы кэширования для оптимизации<sup>16</sup> ввода-вывода;

6. Передача в рабочую область прикладной программы данных запрошенной ею логической записи или указателя на соответствующую область памяти в системном буфере.

В этой последовательности наиболее медленными операциями являются механическое позиционирование головок и чтение данных с поверхности носителя (выполняемые на порядки медленнее, чем операции пересылки). Поэтому выигрыш во времени может быть получен только в случае выполнения ряда запросов на доступ к данным, причем экономия может достигаться следующими путями:

1. Суммарным сокращением перемещения головок за счет организации такой последовательности обращения к записям (или такого порядка их физического размещения), когда перемещение от текущего положения к следующему будет минимальным;

2. Формированием логических записей таким образом, чтобы их формат (длина данных) соответствовала физическому формату хранения. В случае кратности длин, т.е. если длина логической записи будет кратной длине кластера или в кластере будет размещаться целое число записей, будет исключена передача данных, не запрошенных текущей операцией.

---

<sup>15</sup> Такой подход форматирования (разметки) пространства внешней памяти используется и в случае таких устройств «истинно» последовательного доступа, как магнитные ленты, для обеспечения ускоренного «прямого» доступа к сектору по его номеру – прямому адресу (еще с тех времен, когда не были созданы дисковые накопители, например, ЭВМ 2-го поколения Минск-22). При этом, поскольку данные секторов, предшествующих нужному, передавать не надо, позиционирование будет выполняться с максимальной скоростью перемещения ленты.

<sup>16</sup> Автоматическое использование системы кэширования и упреждающего чтения (не учитывающее особенности порядка обращения к данным, обусловленного алгоритмом обработки) может привести к обратному результату, например в случае обращения к логическим записям в произвольной последовательности (случайной) не соответствующей физическому следованию записей.

Непосредственное применение приведенных методов повышения эффективности, тем не менее, достаточно ограничено по целому ряду причин. По мере добавления новых типов данных или при появлении новых приложений структура записей должна будет меняться. Требования к обработке изменяются случайным образом. Если возникает необходимость модификации выбранных структур данных, то приходится соответственно переписывать и отлаживать прикладные программы. Чем больше количество прикладных программ имеется в наличии, тем более дорогой становится эта процедура. Кроме того, логическая структура записей стала бы зависимой от параметров физической структуры носителя, и планирование эффективной физической организации для конкретной структуры данных потребовало бы уже знаний системного аналитика.

Практическое решение состоит во введении *контролируемой* функциональной и информационной избыточности, обеспечивающей сокращение времени доступа за счет: 1) специализации компонент упрощение процедур преобразований, и 2) за счет построения вспомогательных структур (в той или иной степени дублирующих основную информацию). Основой этого подхода является принцип выделения и представления описательных составляющих в виде самостоятельных операционных объектов, хранимых отдельно от определяемых ими данных<sup>17</sup>.

### **2.3. Эволюция концепций обработки данных**

Характер возможных представлений данных и архитектурные решения, отражающие степень специализации компонент управления, хорошо иллюстрируется представленной в [14] эволюцией концепций обработки данных.

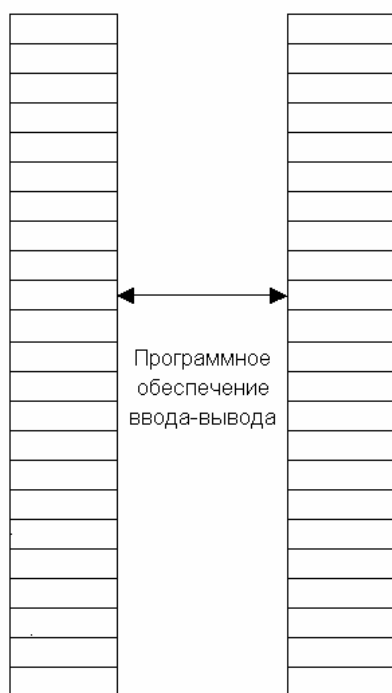
С появлением в конце 60-х годов понятия *база данных* взаимосвязь файлов (логических) и наборов данных (физических файлов) рассматривается в контексте *неизбыточности и независимости данных, их защиты и возможностью доступа в реальном времени*.

---

<sup>17</sup> Сюда относятся не только определения данных в виде описаний типов, схем баз данных, разделов деклараций, но и упомянутые ранее описания физической и логической структуры носителя, параметры подсистемы ввода-вывода и т.д.

### 2.3.1. Простые (линейные) файлы данных (начало 60-х годов)

Файл логических записей      Файл физических записей



Для линейных «простых» файлов организация хранения и доступа характеризуется следующими особенностями.

Записи в файлах размещаются и обрабатываются последовательно. Физическая структура хранения данных точно такая же, как логическая.

Программное обеспечение ввода-вывода выполняет только операции физического чтения-записи. При обновлении отдельной записи файл всегда перезаписывается на другой носитель, а предыдущие поколения данных сразу не уничтожаются.

Прикладной программист определяет физическое расположение данных и включает формирование физической структуры в прикладные программы. Если структура данных или запоминающее устройство изменяется, прикладную программу необходимо переписать.

Наборы данных обычно создаются и оптимизируются для одного приложения. Одни и те же данные редко используются для нескольких приложений.

### 2.3.2. Методы доступа к записям (конец 60-х годов)

Этот этап характеризуется изменением природы файлов и устройств. Появляются дисковые устройства с прямым доступом и возможностью обновления «по месту изменений», а программное обеспечение позволяет без перекомпиляции программы изменять расположение на-



бора данных, но без изменения структуры записей и типа организации набора.



Организация хранения и доступа в этом случае характеризуется следующими особенностями.

Логическая и физическая структуры файла различаются между собой, но взаимосвязь между ними достаточно простая. Запоминающее устройство можно менять без изменения прикладной программы.

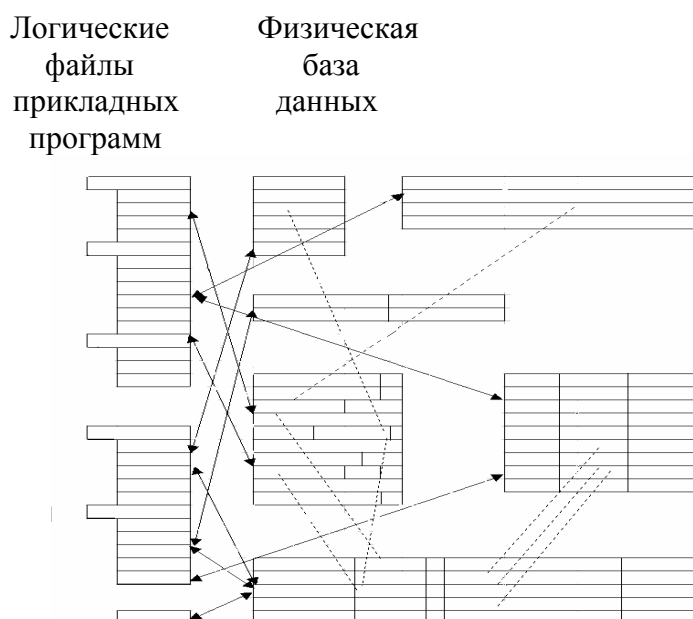
Файл создается в прикладной программе как набор данных с последовательным, индексно-последовательным или с прямым доступом (по физическому адресу). Возможен последовательный или произвольный доступ к записям (но не к полям). Поиск по многим ключам, как правило, не используется. Если используются иерархические файлы, то взаимосвязь «исходный – порожденный» программируется в прикладной программе.

Типовое программное обеспечение системы обработки данных представляет собой методы доступа, но не «управление данными». Данные в основном разрабатываются и оптимизируются для одного приложения.

Средства обеспечения защиты данных недостаточно надежны.

### 2.3.3. Первые системы управления базами данных (начало 70-х годов)

Для этого этапа характерно изменение представления о назначении и возможностях систем управления данными. По мере развития средств обработки данных становилось ясно, что прикладные программы желательно сделать независимыми не только от изменений в аппаратных средствах хранения, но также и от добавления к хранимым данным новых полей и новых взаимосвязей. Система должна быть способна обрабатывать новые типы запросов пользователей.



Организация хранения и доступа в случае систем управления данными характеризуется следующими особенностями.

Различные логические файлы могут быть получены из одних и тех же физических данных. Доступ к одним и тем же данным может осуществляться различными приложениями по различным путям, отвечающим требованиям этих приложений.

Данные адресуются на уровне полей и групп. Можно использовать поиск по многим ключам.

Физическая структура данных независима от прикладных программ. Ее можно изменять с целью повышения эффективности базы данных, не модифицируя при этом прикладные программы. Использование сложных форм организации данных не требует усложнения прикладных программ.

Элементы данных являются общими для различных приложений. Отсутствие избыточности способствует целостности данных.

#### 2.3.4. Системы управления базами данных

Требования к системе основываются на том, что структура базы данных является менее *статичной*, чем файловая структура. Элементы хранимых данных и способы их представления непрерывно изменяются.

Из одних и тех же данных могут быть получены различные логические файлы, а доступ к одним и тем же данным со стороны различных приложений может осуществляться различными путями, отвечающими требованиям этих приложений. Это часто приводит к созданию сложных структур данных. Независимо от того, каким образом данные организованы на самом деле, прикладной программист должен представлять себе файл в виде сравнительно простой структуры, которая спланирована в соответствии с требованиями его приложения.

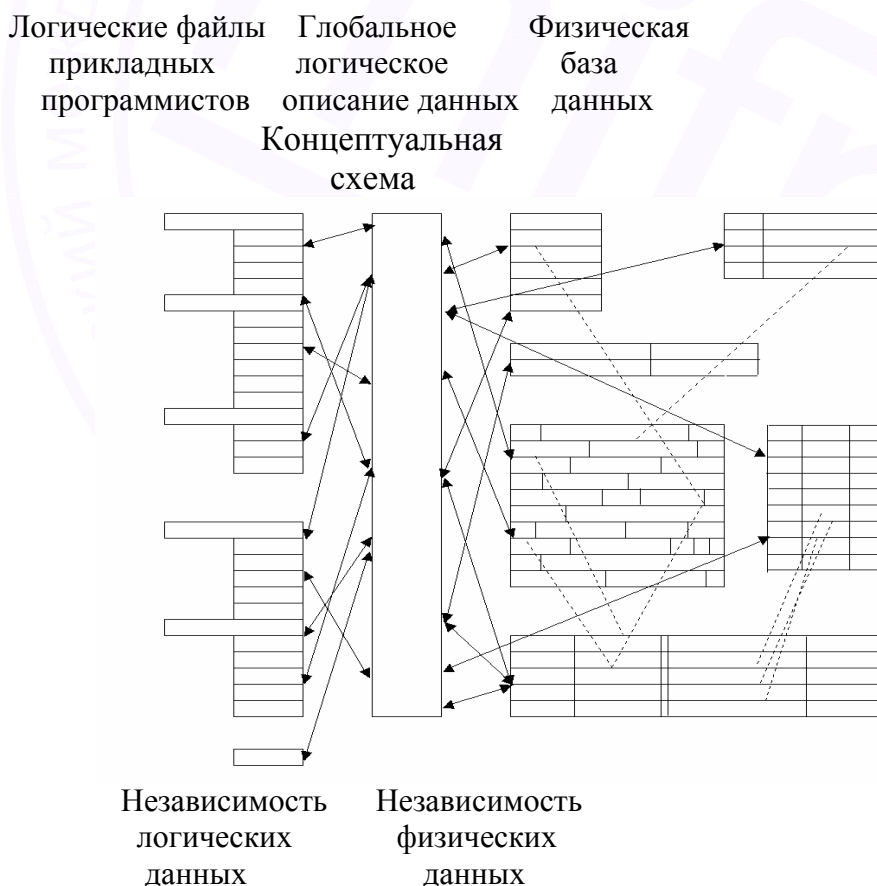
Программное обеспечение баз данных должно располагать средствами отображения файлов структуры прикладного уровня в такую физическую структуру данных, которая эффективно запоминается на реальном носителе, и наоборот.

Для этого вводятся два уровня независимости данных.

*Логическая независимость данных* означает, что общая логическая структура данных может быть изменена без изменения прикладных программ (изменение, конечно, не должно заключаться в удалении из базы данных таких элементов, которые используются прикладными программами).

*Физическая независимость данных* означает, что физическое расположение и организация данных могут изменяться, не вызывая при этом изменения ни общей логической структуры данных, ни прикладных программ.

Система обеспечивает привязку данных — связывание физического представления данных с программой, которая эти данные использует, путем преобразования обращения прикладной программы к логической записи или к элементам логической записи в машинные обращения к физической записи и ее элементам.



Физическая и логическая независимость данных обеспечивается программными средствами. Допускается существование глобального логического представления данных. Предусматривается использование

*языка описания данных* для администратора базы данных, *языка команд* для прикладного программиста и *языка запросов* для пользователя.

Для систем управления базами данных также характерны следующие особенности:

1) Так как базы данных конструируются для выдачи ответов на не запланированные заранее запросы, то используются дополнительные функционально-ориентированные структуры, например, инвертированные файлы, позволяющие осуществлять быстрый поиск в базе данных по некоторым не основным ключам.

2) Вводятся средства администрирования, которые позволяют управлять системой (в том числе управление защитой, секретностью, целостностью и безопасностью данных); проектировать структуры, оптимальные для пользователей, обеспечивать импорт-экспорт и перемещение данных.

#### **2.4. Схема управления данными в СУБД**

Рассмотрим примерную последовательность операций, обеспечивающих чтение прикладной программой из базы данных, представленную на рис. 2.2.

➤ (1) Прикладная программа (клиентское приложение) формирует и выдает системе управления базами данных запрос на чтение необходимых данных, содержащихся в базе.

➤ (2-3) СУБД отыскивает описание затребованных данных в структуре описания данных прикладного уровня (внешняя модель).

➤ (4-5) СУБД по глобальному описанию БД (концептуальная схема) определяет необходимые данные на логическом уровне.

➤ (6-7) СУБД по описанию физической структуры БД (физическая модель) определяет физическую запись (или совокупность записей), которую необходимо считать для выборки данных, затребованных прикладной программой.

➤ (8-9) СУБД через подсистему управления потоками данных выдает операционной системе запрос на чтение хранимой записи.

➤ (10-11) Подсистема управления вводом-выводом операционной системы осуществляет физическое чтение записи в системный буфер ОС.

➤ (13) СУБД выделяет необходимую логическую запись, осуществляет форматные преобразования, обусловленные различиями описаний на глобальном и прикладном уровнях, и передает для функциональной обработки приложением данные в рабочий буфер, выделяемый прикладной программой или самой СУБД.



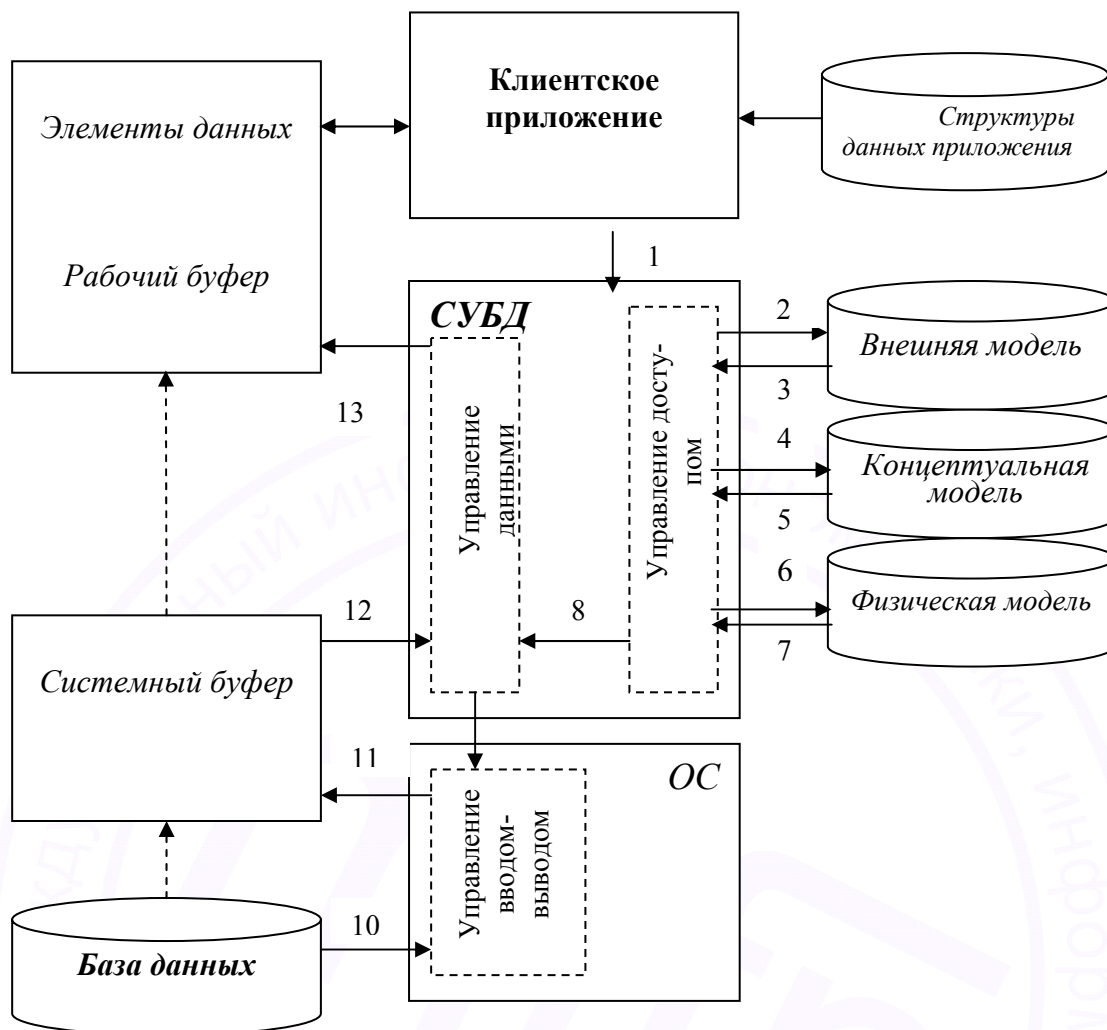


Рис. 2.2. Схема обработки запроса на выборку данных из БД

## 2.5. Данные и управление их обработкой

### 2.5.1. Типы, форматы, структуры данных

Структура информационных единиц, обрабатываемых на ЭВМ, определяется следующими понятиями:

- *тип данных*, или совокупность соглашений о программно-аппаратурной форме представления и обработки а также ввода, контроля и вывода элементарных данных;
- *структуры данных* - способы композиции простых данных в агрегаты и операции над ними;
- *форматы файлов* - представление информации на уровне взаимодействия операционной системы с прикладными программами;

**Типы данных.** Ранние языки программирования - Фортран, Алгол были ориентированы исключительно на вычисления и не содержали развитых систем типов и структур данных. Типы числовых данных Алгола: INTEGER (целое число), REAL (действительное) - различаются

диапазонами изменения, внутренними представлениями и применяемыми командами процессора ЭВМ (соответственно арифметика с фиксированной и плавающей точкой). Нечисловые данные представлены типом BOOLEAN - логические, имеющие диапазон значений {TRUE, FALSE}.

Появившиеся позже ЯП COBOL, PL/1, Pascal уже предусматривают новые типы данных:

- символьные (цифры, буквы, знаки препинания и пр.);
- числовые символьные для вывода,
- числовые двоичные для вычислений,
- числовые десятичные (цифры 0 - 9) для вывода и вычислений.

**Структуры данных.** В ЯП Алгол были определены два типа структур: *элементарные данные* и *массивы* (векторы, матрицы, тензоры, состоящие из арифметических или логических переменных). Основным нововведением, появившимся первоначально в Коболе, (затем PL/1, Паскаль и пр.) являются *агрегаты данных* (структуры, записи), представляющие собой именованные комплексы переменных разного типа, описывающих некоторый объект или образующих некоторый достаточно сложный документ.

Термин *запись* подразумевает наличие множества аналогичных по структуре агрегатов, образующих *файл* (картотеку), содержащих данные по совокупности однородных объектов, элементы данных образуют поля, среди которых выделяются элементарные и групповые (агрегатные).

Появление СУБД и АИПС приводит к появлению новых разновидностей структур:

- множественные поля данных;
- периодические групповые поля;
- текстовые объекты (документы), имеющие иерархическую структуру (документ, сегмент, предложение, слово).

**Форматы файлов.** В зависимости от типа и назначения файлов и возможностей ОС (методов доступа) файл может передаваться в прикладную программу как целое, или блоками (физическими записями) либо логическими записями (строками, словами, символами).

Например, в системе OS/360 основную роль играли два типа файлов:

- символьные (исходные программы или данные);
- двоичные (программы в машинных кодах).

В современных системах активно используется значительно большее разнообразие файлов, например, *текстовые файлы* - обобщенное название для простых и размеченных текстов, ASCII-файлов и других наборов данных символьной информации, которые интерпретируются и обрабатываются текстовыми редакторами, процессорами, анализаторами.

### 2.5.2. Описание и обработка файлов

По мере развития средств вычислительной техники и расширения спектра задач, связанных с обработкой на ЭВМ, произошел постепенный переход основных усилий разработчиков и пользователей от алгоритмической к информационной стороне вопроса.

По-видимому достаточно подробное описание структур данных и установление их связи с файлами было впервые сделано в ЯП Cobol (Common Business Oriented Language). Эта проблема была решена следующим образом - *файл* (набор данных на внешнем носителе) рассматривается как совокупность записей, одинаковой структуры, каждая из которых представляет собой набор (агрегат) разнородных данных (в более поздних ЯП - PL/1, Pascal, C за подобными объектами так и закрепилось название *структура* - *structure*).

**Проблема локализации описания данных.** Приемы распознавания программой элементов данных или записей относятся к такому типу взаимодействия программ и данных, когда описание данных размещено в программе, а файл данных организован в соответствии с этим описанием (Рис. 2.3.а). Однако этот способ может привести к нарушению функционирования или разрушению данных, если из-за ошибок программиста или оператора к программе будет подсоединен "неправильный файл".

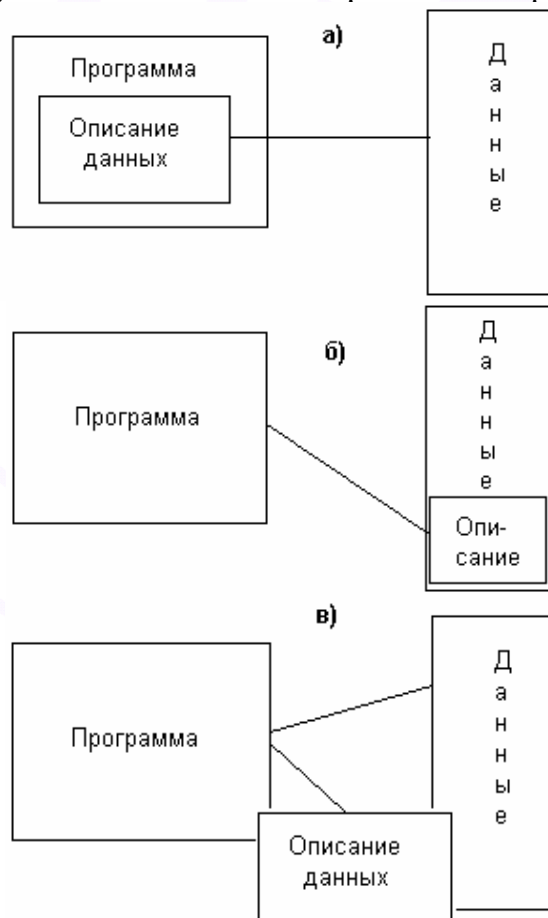


Рис. 2.3. Варианты размещения данных и их описания:

а) - в прикладной программе; б) - в файле данных; в) - отдельным набором данных (словарь данных)

Для установления независимости программ от данных в некоторых системах описание данных размещают совместно с файлом данных (Рис. 2.3б). По такому принципу организован весьма распространенный формат файла данных (dbf - формат), происходящий от систем dBase - Clipper - Foxbase - FoxPro, а затем принятый и рядом других систем. В этом случае в начале файла создается заголовок, содержащий описание полей записи файла (имя, тип, длина данного, код информации и пр.), и таким образом, описание данных файла в программе не нужно.

Недостатком такого подхода является, например, необходимость использования программистами тех же имен данных, что содержатся в описании файла.

Следующим шагом явилось полное отделение описаний от данных и программ и сосредоточение их в специальных файлах (таблицах) - словарях данных (Рис. 2.3в), которые относятся к базам данных и системам управления базами данных.

## ***2.6. Особенности и компромиссы реализаций баз данных***

В заключение приведем основные отличительные особенности обработки данных, характерные для файловых систем и систем управления базами данных.

Файлы обладают следующими свойствами:

- файл, как правило, представляет собой совокупность записей одного типа, доступ к которым определяется типом *организации* файла и осуществляется только средствами операционной системы;
- файл описывают и используют в прикладной программе, работающей с данными.

Базы данных имеют следующие особенности:

- база данных представляет собой совокупность данных разного типа, причем часто по одним данным получают другие;
- база данных существует независимо от конкретной прикладной программы – база создается с целью интеграции данных, объединяющей данные многих приложений (но определенного назначения). База данных предназначена для *совместного, многофункционального* использования *многими* пользователями *один раз введенных* данных.

Надо отметить, что с точки зрения управления данными СУБД оперируют данными на содержательном уровне, хотя физические структуры, используемые для этих целей, могут и совпадать с аналогичными структурами, создаваемые ОС.

Коренное же отличие СУБД от файловых систем ОС состоит в том, что СУБД устанавливает связь между *содержанием и адресом*, а ОС - между *именем и адресом* данных.

В то же время эта граница постоянно подвергается "атакам" с обеих сторон. Например, ОС-360 с "индексным доступом к данным", IN-



PICK, включающая язык поиска записей файлов по содержанию, UNIX, включающая команды сортировки, коррекции или объединения содержимого текстовых файлов, наподобие того, как это осуществляется с таблицами данных в СУБД. Тем не менее, следует признать это скорее исключением, чем правилом и в компетенцию ОС надо относить только связь "имя-адрес", оставляя другие зависимости на ответственность прикладных программ и оболочек СУБД и АИПС (автоматизированные информационно-поисковые системы).

В общем случае можно сказать, что основные задачи обработки данных, решаемые на основе концепций баз данных, сводятся к следующим вопросам:

- 1) Каким образом сложные нелинейные структуры данных представить в виде линейных – наиболее соответствующих принципу последовательного представления (хранения) в машинной памяти.
- 2) Каким образом организовать данные, чтобы была возможность эффективного внесения, удаления и редактирования данных.
- 3) Как организовать данные, чтобы использование пространства памяти (плотность данных) было достаточно рациональным, а скорость доступа к записям данных высокой.
- 4) Каким образом организовать данные, чтобы поиск был эффективным и позволял отыскивать записи по нескольким ключам.

При этом, с точки зрения прагматики, создание базы данных - это по существу попытка найти компромисс сразу по нескольким направлениям и сочетаниям нескольких взаимообратных факторов (с точки зрения их влияния на показатель общей эффективности системы), в том числе, следующих:

- 1) Эффективность – простота;
- 2) Скорость выборки – стоимость (сложность) аппаратных средств;
- 3) Скорость выборки – сложность процедур доступа;
- 4) Плотность данных – время доступа и сложность процедур;
- 5) Независимость данных – производительность;
- 6) Гибкость средств поиска – избыточность данных или
- 7) Гибкость поиска – скорость поиска;
- 8) Сложность процедур доступа – простота обслуживания.

### *Контрольные вопросы*

1. Перечислите основные задачи обработки данных, решаемые на основе концепций баз данных.
2. Проведите сравнительный анализ понятий «физического» и «логического» представлений.
3. Определите соотношение физической и логической записи.

4. Приведите примерную схему организации файлового ввода-вывода.
5. Проведите сравнительный анализ процессов обработки данных средствами файловой системы и СУБД.
6. Перечислите основные этапы эволюции систем обработки данных.
7. Определите отличия в концепциях обработки данных 1-го и 2-го этапов.
8. Определите отличия в концепциях обработки данных 3-го и 4-го этапов.
9. Проведите сравнительный анализ объектов и функций, используемых в разных концепциях обработки данных.
10. Перечислите систему основных показателей эффективности обработки данных.
11. Приведите схему управления данными в СУБД.

### Глава 3. Модели и структуры данных

Рассматриваемые в контексте понятия «информационная система» элементы реального мира, информацию о которых мы сохраняем и обрабатываем, будем называть *объектами*. Объект может быть материальным (например, служащий, изделие или населенный пункт) и нематериальным (например, имя, понятие, абстрактная идея). Будем называть *набором объектов* совокупность объектов, однородных с некоторой точки зрения (например, объектов *нашего* внимания, пусть даже и разнородных по своей внутренней природе).

Объект имеет различные свойства (например, цвет, вес, имя), которые важны для нас в то время, когда мы обращаемся к объекту (например, выбираем среди множества других) с какой-либо целью его использования. Причем свойства могут быть заданы как отдельными однозначно интерпретируемыми количественными показателями, так и словесными нечеткими описаниями, допускающими разную трактовку, иногда зависящую от точки зрения и наличных знаний воспринимающего субъекта.

Однако во всех случаях человек, работая с информацией, имеет дело с *абстракцией*, представляющей интересующий его фрагмент реального мира - той совокупностью *характеристических свойств (атрибутов)*, которые важны для решения его прикладной задачи. Абстрагирование – это способ *упрощения* совокупности фактов, относящихся к реальному объекту (по своей сути бесконечно сложному и разнообразному при изучении его человеком). При этом некоторые свойства объекта игнорируются, поскольку считается, что для решения данной прикладной задачи (или совокупности задач) они не являются определяющими и не влияют на конечный результат действий при решении.

Цель такого абстрагирования - построение конструктивного операбельного описания (рабочей модели), удобного в обработке, как для человека, так и для машины, позволяющего организовать эффективную обработку больших объемов информации, причем высокопроизводительной должна быть работа не только вычислительной системы, но и взаимодействующего с ней человека.

#### **3.1. Многоуровневые модели предметной области**

Обычно отдельная база данных содержит (отражает) информацию о некоторой *предметной области* – наборе объектов, представляющих интерес для актуальных или предполагаемых пользователей. То есть, реальный мир отображается совокупностью конкретных и абстрактных понятий, между которыми существуют (и соответственно, фиксируются) определенные связи. Выбор для описания предметной области (ПрО) существенных понятий и связей является предпосылкой того, что пользователь будет иметь *практически все необходимые ему в рамках задачи*

знания об объектах предметной области. Но, следует отметить, что пользователь, который хочет работать с базой данных, должен владеть основными понятиями, представляющими предметную область.

И в этом смысле абстрагирование позволяет построить такое описание (модель предметной области), которое другой человек сможет не только воспринять, но и безошибочно использовать для работы с описаниями экземпляров объектов, хранимых в базе данных.

Модель предметной области соотносится с реальными объектами и связями так же, как схема маршрутов городского пассажирского транспорта с фактической траекторией движения автобуса. Схема адекватно отражает действительность на уровне основных понятий – маршрутов и остановок: выбрав по схеме маршрут, пассажир достигнет цели (прибудет на нужную остановку) независимо от того, в каком транспортном ряду будет двигаться автобус.

Наиболее простой способ представления предметных областей в БД реализуется поэтапно: 1) фиксацией *логической точки зрения* на данные (т.е. данные рассматриваются независимо от особенностей их хранения и поиска в конкретной вычислительной среде); 2) определением физического представления данных с учетом выбранных структур хранения данных и архитектуры ЭВМ.

Абстрагированное описание предметной области с фиксированной (логической) точки зрения будем называть *концептуальной схемой*. Соответственно, систематизация понятий и связей предметной области называется *логическим* или *концептуальным проектированием*. Модель (представление логической точки зрения), используемая при абстрагировании - совокупность функциональных характеристик объектов и особенностей представления информации (например, в числовой или текстовой форме), будем называть *моделью данных*.

Отображение концептуальной схемы на физический уровень будем называть *внутренней схемой*.

Соотношение этих понятий приведено на рис. 3.1.

Отражение взгляда (точки зрения) отдельного пользователя на концептуальную схему (как вариант восприятия предметной области) будем называть *внешней схемой*. Внешняя схема использует те же абстрактные категории, что и концептуальная, а на практике соответствует логической организации данных в прикладной программе.



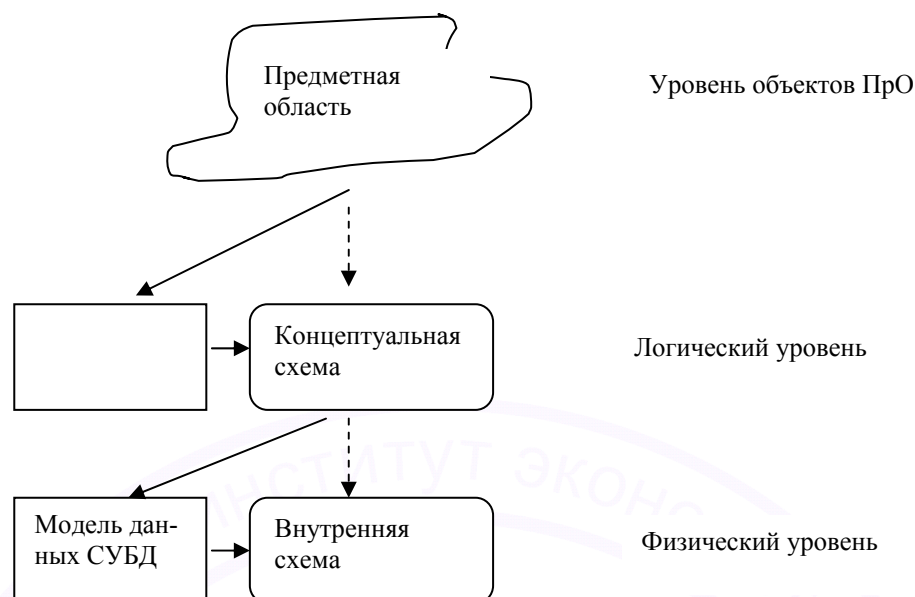


Рис. 3.1. Соотношение понятий концептуальной и внутренней схем

Теоретически вопрос о многообразии уровней абстракции был решен еще в 60 – 70-х годах. Основой для его решения является концепция многоуровневой архитектуры системы базы данных. Например, в отчете CODASYL [24] предусматривался архитектурный уровень подсхемы, который позволял для каждого конкретного приложения строить свое собственное «видение» используемого подмножества базы данных путем определения его «персональной» подсхемы базы данных.

В более общем виде этот вопрос решен в архитектурной модели ANSI/X3/SPARC [22]. Здесь на внешнем уровне может поддерживаться совсем иная модель данных (или даже несколько моделей), чем на концептуальном уровне. Поддержка разнообразных возможностей абстрагирования в такой системе достигается благодаря средствам определения и поддержки межуровневого отображения моделей данных.

Помимо этого, для решения указанной проблемы может использоваться внутримодельная структура, например, механизмы *представлений* (view). В объектных системах для этих целей может использоваться отношение наследования.

В общем случае концепция трехуровневого представления не требует более трех уровней, однако с практической точки зрения иногда удобно включать схемы дополнительных уровней. На рис. 3.2 приведены некоторые варианты решений.

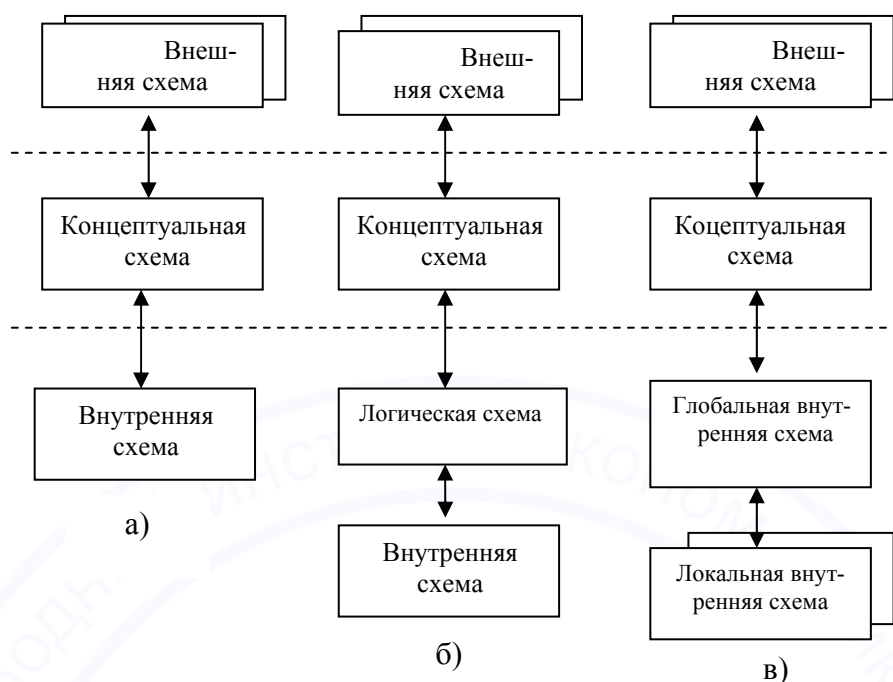


Рис. 3.2. Примеры трехуровневого представления

На рис. 3.2б выделена логическая схема, учитывающая особенности СУБД.

Пример, приведенный на рис. 3.2в, характерен для варианта распределенной базы данных, объединяющей информацию, представленную разными внутренними схемами.

Рассмотренная трехуровневая архитектура обеспечивает выполнение основных требований, предъявляемых к системам баз данных:

- адекватность отображения предметной области;
- возможность взаимодействия с БД разных пользователей при решении разных прикладных задач;
- обеспечение независимости программ и данных;
- надежность функционирования БД и защиту от несанкционированного доступа.

С точки зрения пользователей различных категорий трехуровневая архитектура имеет следующие достоинства:

- системный аналитик, создающий модель предметной области, не обязательно должен быть специалистом в области программирования и вычислительной техники;
- администратор баз данных, обеспечивающий отражение концептуальной схемы во внутреннюю, не должен беспокоиться о корректности представления предметной области;
- конечные пользователи, используя внешнюю схему, могут не вдаваться полностью в предметную область, обращаясь только к необходимым составляющим. При этом исключается возможность несанк-

ционированного обращения к данным вне объявленных внешней схемой, так как формирование ее находится в сфере деятельности администратора базы данных;

- системный аналитик, как и конечный пользователь не вмешивается во внутреннее представление данных.

Это отражает распространенную практику специализации и разделения ответственности. Главное же заключается в том, что работу по проектированию и эксплуатации баз данных можно разделить на три достаточно самостоятельных этапа. Хотя надо отметить, что на практике создание концептуальной схемы не всегда предшествует построению внешней. Иногда трудно с самого начала полностью определить предметную область, но, с другой стороны, уже известны требования пользователей (именно поэтому создание базы уже имеет смысл). И, кроме того, адекватность модели предметной области, в конце концов, должна подтверждаться практикой пользовательских представлений.

### 3.2. Идентификация объектов и записей

В задачах обработки информации, и в первую очередь в алгоритмизации и программировании, атрибуты *именуют* (обозначают) и приписывают им *значения*.

При обработке информации мы, так или иначе, имеем дело с совокупностью объектов, *информацию о свойствах* каждого из которых надо сохранять (записывать) как *данные*, чтобы при решении задач их можно было найти и выполнить необходимые преобразования.

Таким образом, любое состояние объекта характеризуется совокупностью актуализированных атрибутов<sup>18</sup> (имеющих некоторое значение в этот момент времени), которые фиксируются на некотором материальном носителе в виде *записи* – совокупности (*группы*) формализованных *элементов данных* (значений атрибутов, представленных в том или ином формате). Кроме того, в контексте задач хранения и поиска можно говорить, что значение атрибута *идентифицирует* объект: использование значения в качестве поискового признака позволяет реализовать простой критерий отбора по условию сравнения<sup>19</sup>.

Также как и в реальном мире, отдельный объект всегда уникален (уже хотя бы потому, что мы *именно его* выделяем среди других). Соответственно, запись, содержащая данные о нем, также должна быть

<sup>18</sup> В общем случае объект может описываться совокупностью записей, относящихся к его составным частям или отражающих динамику изменения состояния.

<sup>19</sup> Следует отметить некоторые семантические проблемы идентификации через значение атрибута. Значение атрибута идентифицирует запись о **состоянии** объекта, и в случае изменения значения, например – табельного номера служащего, будет невозможно ответить на вопрос: идет ли речь о том же служащем, или о новом.

узнаваема однозначно (по крайней мере, в рамках предметной области), т.е. – иметь уникальный идентификатор, причем никакой другой объект не должен иметь такой же идентификатор. Поскольку идентификатор – суть значение элемента данных, в некоторых случаях для обеспечения уникальности требуется использовать более одного элемента. Например, для однозначной идентификации записей о дисциплинах учебного плана необходимо использовать элементы СЕМЕСТР и НАИМЕНОВАНИЕ ДИСЦИПЛИНЫ, так как одна дисциплина может быть прочитана в разных семестрах.

Предложенная выше схема представляет атрибутивный способ идентификации содержания объекта (рис.3.3). Она является достаточно естественной для данных, имеющих фактографическую природу, и описывающих обычно материальные объекты. Информацию, представляемую такого рода данными, называют *хорошо структурированной*. Здесь важно отметить, что структурированность относится не только к форме представления данных (формат, способ хранения), но и к способу интерпретации значения пользователем: значение параметра не только представлено в предопределенной форме, но и обычно сопровождается указанием размерности величины, что позволяет пользователю понимать ее смысл без дополнительных комментариев. Таким образом, фактографические данные предполагают возможность их *непосредственной* интерпретации.

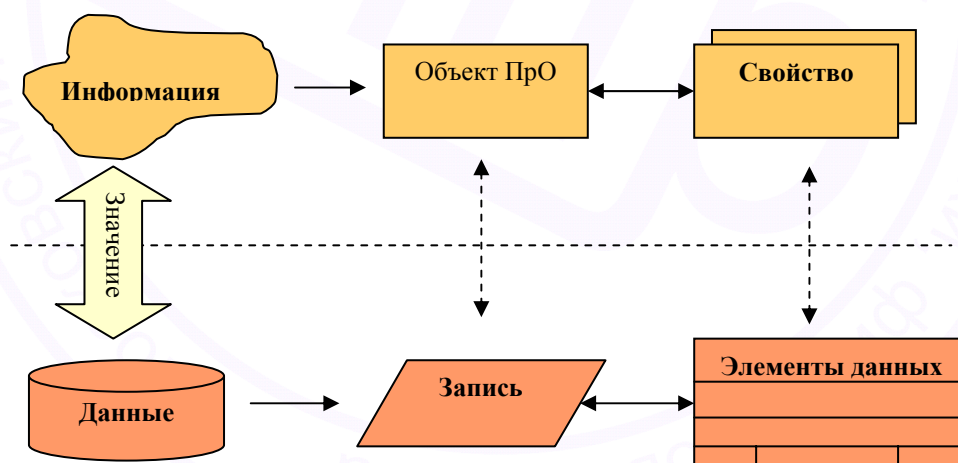


Рис. 3.3. Атрибутивный способ идентификации

Однако атрибутивный способ практически не подходит для идентификации *слабо структурированной информации*, связанной с объектами, имеющими обычно *идеальную* (умозрительную) природу – категориями, понятиями, знаковыми системами. Такие объекты зачастую определяются логически и опосредованно – через другие объекты. Для описания таких объектов используются естественные или искусственные языки (например, язык алгебры). Соответственно, для понимания смысла пользователю необходимо использовать соответствующие пра-



вила языка, и, более того, часто необходимо уже располагать некоторой информацией, позволяющей идентифицировать и связать получаемую информацию с наличным знанием. Т.е., процесс интерпретации такого рода данных имеет *опосредованный* характер и требует использования дополнительной информации, причем такой, которая не обязательно присутствует в формализованном виде в базе данных.

Такое разделение нашло отражение в традиционном разделении баз данных на *фактографические* и *документальные*.

### 3.3. Поиск записей

Программисту или пользователю необходимо иметь возможность обращаться к отдельным, нужным ему записям (описаниям объектов) или отдельным элементам данных. В зависимости от уровня программного обеспечения прикладной программист может использовать следующие способы:

- задать машинный адрес данных и в соответствии с физическим форматом записи прочитать значение. Это случай, когда программист должен быть «навигатором»;
- сообщить системе имя записи или элемента данных, которые он хочет получить, и, возможно, организацию набора данных. В этом случае система сама произведет выборку (по предыдущей схеме), но для этого она должна будет использовать вспомогательную информацию о структуре данных и организации набора. Такая информация по существу будет избыточной по отношению к объекту, однако общение с базой данных не будет требовать от пользователя знаний программиста и позволит переложить заботы о размещении данных на систему.

В качестве ключа, обеспечивающего доступ к записи, можно использовать идентификатор – отдельный элемент данных. Ключ, который идентифицирует запись единственным образом, называется *первичным (главным)*.

В том случае, когда ключ идентифицирует некоторую группу записей, имеющих определенное общее свойство, ключ называется *вторичным (альтернативным)*. Набор данных может иметь несколько вторичных ключей, необходимость введения которых определяется практической необходимостью – оптимизацией процессов нахождения записей по соответствующему ключу.

Иногда в качестве идентификатора используют составной *сцепленный ключ* – несколько элементов данных, которые в совокупности, например, обеспечат уникальность идентификации каждой записи набора данных.

При этом ключ может храниться в составе записи или отдельно. Например, ключ для записей, имеющих неуникальные значения атрибутов, для устранения избыточности может храниться отдельно. На рис.

3.4 приведены два таких способа хранения ключей и атрибутов для набора простейшей структуры.

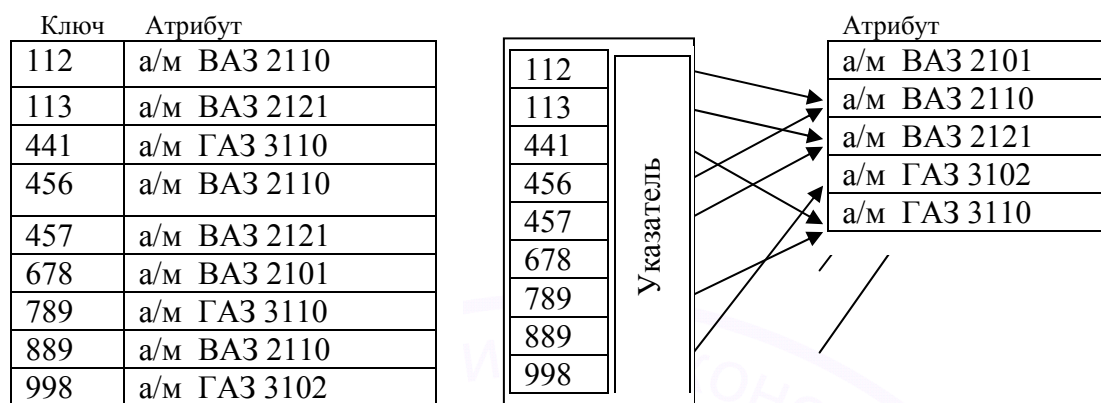


Рис. 3.4 Способы хранения ключа и атрибута

Введенное понятие ключа является логическим и его не следует путать с физической реализацией ключа – *индексом*, обеспечивающим доступ к записям, соответствующим отдельным значениям ключа.

Один из способов использования вторичного ключа в качестве входа - организация инвертированного списка, каждый вход которого содержит значение ключа вместе со списком идентификаторов соответствующих записей. Данные в индексе располагаются в возрастающем или убывающем порядке, поэтому алгоритм нахождения нужного значения довольно прост и эффективен, а после нахождения значения запись локализуется по указателю физического расположения. Недостатком индекса является то, что он занимает дополнительное пространство и его надо обновлять каждый раз, когда удаляется, обновляется или добавляется запись. На рис. 3.5 приведен инвертированный список для предыдущего примера.

а/м ВАЗ 2101	678
а/м ВАЗ 2110	112, 456, 889
а/м ВАЗ 2121	113, 457
а/м ГАЗ 3102	998
а/м ГАЗ 3110	441, 789

Рис. 3.5. Инвертированный список для ключа «Марка автомобиля»

В общем случае инвертированный список может быть построен для любого ключа, в том числе составного.

В контексте задач поиска можно сказать, что существуют два основных способа организации данных. Первый соответствует примеру, приведенному на рис. 3.3, и представляет прямую организацию массива. Второй способ является инверсией первого, он соответствует рис. 3.4. Прямая организация массива удобна для поиска по условию «Каковы

свойства указанного объекта?», а инвертированная – для поиска по условию «Какие объекты обладают указанным свойством?».

В [14] приводится следующая типология простых (атомарных) запросов:

1.  $A(E) = ?$  Каково значение атрибута  $A$  для объекта  $E$ ?
2.  $A(?) = V$  Какие объекты имеют значение атрибута равное  $V$ ?
3.  $?(E) = V$  Какие атрибуты объекта  $E$  имеют значение равное  $V$ ?
4.  $?(E) = ?$  Какие значения атрибутов имеет объект  $E$ ?
5.  $A(?) = ?$  Какие значения имеет атрибут  $A$  в наборе?
6.  $?(?) = V$  Какие атрибуты объектов набора имеют значение равное  $V$ ?

Здесь в запросах типов 2, 3, 6 вместо оператора равенства может быть использован другой оператор сравнения (*больше, меньше, не равно* или другие).

Запросы типа 1 выполняются поиском по «прямому» массиву: доступ к записи производится по первичному ключу. Запросы типа 2 выполняются поиском по инвертированному списку: доступ к записи(ям) производится по указателю, выбираемому из списка по значению вторичного ключа. Ответом в этих случаях будет *значение* атрибута или идентификатора. Запросы типа 3 имеют ответом *имя* атрибута.

Запросы типа 2, 5, 6 относятся к нескольким атрибутам, и в этом случае могут быть построены несколько индексов, облегчающих поиск по этим ключам.

*Составные условия поиска* могут использовать несколько простых условий, обычно связанных логическими (булевыми) операторами.

Следует отметить, что в контексте обработки запросов 2-го типа «Какие объекты имеют заданное значение атрибута?» можно выделить три следующих типа архитектур доступа:

1. *Системы с вторичными индексами.* В этих системах последовательность расположения записей соответствует последовательности значений первичного ключа. Как правило, используется один первичный индекс и несколько вторичных.

2. *Системы частично инвертированных файлов.* В этих системах записи могут располагаться в произвольной последовательности. В отличие от систем первого типа первичный индекс отсутствует. Вторичные индексы применяются для прямой адресации записей, что существенно облегчает включение в файл новых записей, так как допускается их размещение в любом свободном участке файла.

3. *Системы полностью инвертированных файлов.* В этих системах предусмотрено наличие файлов, содержащих значения отдельных элементов данных, входящих в состав записей – допускается раздельное хранение элементов данных записи. Значения элементов данных, со-

ставляющих конкретную запись или кортеж, в общем случае могут размещаться в памяти произвольно. Для ускорения процесса поиска в системе используют два набора индексов: *индекс экземпляров* (значений ключей) и *индекс данных* (инвертированный список). С помощью индекса экземпляров можно найти в файле элементы данных, имеющих заданное значение. С помощью индекса данных можно найти записи, связанные с заданными значениями элементов. Такая организация характерна для организации данных *документальных информационных систем*.

### **3.4. Представление предметной области и модели данных**

Если бы назначением базы данных было только хранение и поиск данных в массивах записей, то структура системы и самой базы была бы простой. Причина сложности в том, что практически любой объект характеризуется не только параметрами-величинами, но и взаимосвязями частей или состояний. Есть различия и в характере взаимосвязей между объектами предметной области: одни объекты могут использоваться только как характеристики остальных объектов, другие – независимы и имеют самостоятельное значение (рис. 3.6).

Кроме того, сам по себе отдельный элемент данных (его значение) ничего не представляет. Он приобретает смысл только тогда, когда связан с атрибутом (природой значения, что позволит интерпретировать значение) и другими элементами данных.

Поэтому физическому размещению данных (и, соответственно, определению структуры физической записи) должно предшествовать описание логической структуры предметной области – построение *модели* соответствующего фрагмента реального мира, выделяющей только те объекты, которые будут интересны будущим пользователям, и представленные только теми параметрами, которые будут значимы при решении прикладных задач. Такая модель будет иметь очень мало физического сходства с реальностью, но будет полезна как *представление* пользователя о реальном мире. Причем это представление будет задаваться (описываться) *удобными для пользователя* средствами в не адекватной человеку жесткой вычислительной среде с двоичной логикой и числовым представлением информации.



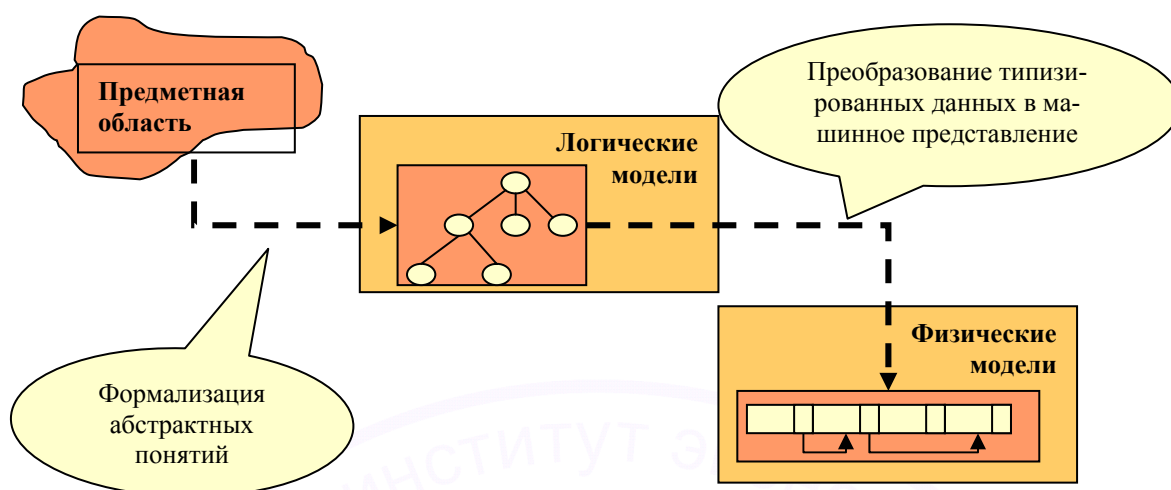


Рис. 3.6. Этапы преобразования представлений Про

Таким образом, прежде чем описывать физическую реализацию объектов и связей между ними, необходимо определить:

1. способ, с помощью которого внешние пользователи представляют (описывают) объекты и связи;
2. форму и методы внутримашинного представления элементов данных и взаимосвязей;
3. средства, обеспечивающие взаимно однозначные преобразования внешнего и внутримашинного представлений.

Такой подход является компромиссом, свойственным языкам программирования: за счет *предварительно определяемого множества абстракций*, общих для большинства задач обработки данных, обеспечивается возможность построения *надежных* программ обработки. Пользователь, используя *ограниченное множество формальных, но достаточно знакомых понятий*, выделяя сущности и связи, описывает объекты и связи предметной области; программист (или система автоматизации проектирования БД), используя такие *типовые абстрактные понятия* (как например числа, множества, последовательности, агрегаты), определяет соответствующие информационные структуры. Система управления данными, используя *двоичные формы представления типизированных данных*, обеспечивает эффективные процедуры хранения и обработки данных.

Именно введение *промежуточного* уровня абстракции позволяет иметь раздельное описание логического и физического представления, освобождает конечного пользователя от необходимости беспокоиться о деталях внутримашинного представления и обработки, поскольку он может быть уверен, что программистом выбрана наиболее эффективная форма для данной ситуации. Однако эффективность здесь имеет определенные пределы. Чем ближе система абстракций к особенностям вычислительной среды, тем выше эффективность выполнения программы, но

вынужденная «специализация» абстракций увеличивает вероятность того, что они станут неподходящими для некоторых других применений.

Модель данных должна, так или иначе, дать основу для описания данных и манипулирования данными, а также дать средства анализа и синтеза структур данных.

Необходимо отметить, что предметные среды с точки зрения описания целесообразно условно разделить на два полярных случая:

1. Предметная среда характеризуется сравнительно небольшим количеством типов отношений, но каждое отношение само есть большое множество. Эти отношения сравнительно устойчивы, а изменений в пределах каждого множества существенно меньше мощности самого отношения. Например, отношение «вхождения» элементов изделий, содержащееся в конструкторских спецификациях, для среднего предприятия содержит сотни тысяч записей. В этом случае, задав схемы отношений и ориентировочные значения их мощностей, можно достаточно полно представить структуру и масштаб предметной среды.

2. Для предметной среды характерно большое число типов отношений между объектами, но каждое отношение есть множество сравнительно малой мощности. При этом мощность потока изменений для отношений сравнима с мощностью самих отношений.

Первый случай характерен для отображения процессов на уровне автоматизированных систем управления предприятиями. Современные системы управления базами данных наиболее эффективны именно в подобном случае, при отображении статических в указанном смысле предметных сред. Обычно при этом речь идет о целых классах объектов, например, о деталях данного типа и обычно не отображается состояние каждой конкретной детали.

Второй случай характерен для описания производственного *технологического процесса*, с учетом временных и пространственных факторов нахождения конкретных объектов.

Если в первом случае говорят о реляционной, иерархической или сетевой моделях данных, то во втором — о семантических сетях и фреймах.

Основное отличие этих методов заключается в том, что первые задают четкую схему (так называемую схему базы данных), в рамках которой и отображается предметная область. Подобное построение по сути своей является довольно статичным, требует априорного знания типов отношений, в которых может находиться объект, однако зафиксированная схема базы данных позволяет довольно эффективно организовать поиск необходимой информации. Во втором случае предметная среда отображается (по крайней мере, на уровне модели) в виде однородной сети, любые изменения которой, как по вводу новых классов объектов, так и новых типов отношений, не связаны с какими-либо структурными преобразованиями сети. В силу большого количества типов отношений

манипулирование подобной "элементарной" информацией достаточно затруднено, поэтому для данного случая характерно введение большого количества общих понятий (и соответствующих им отношений), что упрощает работу с таким представлением.

В контексте машинного представления модель данных может быть использована следующим образом:

- как средство спецификации типов данных и их организации, разрешенных в конкретной БД;
- как основа разработки общей методологии построения баз данных;
- как основа минимизации влияния эволюции баз данных на уже существующие прикладные программы и работу конечных пользователей;
- как основа разработки семейства языков запросов и языков манипулирования данными;
- как основа архитектуры СУБД;
- как основа изучения динамических свойств различных организаций данных.

Таким образом, модель данных – это базовый инструментарий, обеспечивающий на формальном абстрактном уровне конкретные способы представления объектов и связей.

*Модель базы данных* охватывает более широкий спектр понятий. Основное назначение модели базы данных состоит в том, чтобы:

- определить ясную границу между логическим и физическим аспектами управления базой данных (*независимость данных*);
- обеспечить конечным пользователям и программистам, создающим БД, возможность и средства общего понимания смысла данных (*коммуникабельность*);
- определить языковые понятия высокого уровня, обеспечивающие возможность выполнения однотипных операций над большими совокупностями записей (в общем случае разнотипных данных) как единую операцию (*обработка множеств*).

### **3.5. Структуры данных**

При любом методе отображения предметной области в машинных базах данных в основе отображения лежит фиксация (кодирование) понятий и отношений между понятиями. Абстрактное понятие *структуры* ближе всего находится к так называемой концептуальной модели предметной среды и часто лежит в основе последней.

Понятие структуры используется на всех уровнях представления предметной области и реализуется как:

- *структура информации* - схематичная форма представления сложных композиционных объектов и связей реальной ПрО, выделяемых как актуально необходимые для решения прикладных задач;
- *структура данных* - атрибутивная форма представления свойств и связей ПрО, ориентированная на выражение описания данных средствами формальных языков (т.е. учитывающая возможности и ограничения конкретных средств с целью сведения описаний к стандартным типам и регулярным связям);
- *структура записей* – целесообразная (учитывающая особенности физической среды) реализация способов хранения данных и организации доступа к ним как на уровне отдельных записей, так и их элементов (с целью определения основных и вспомогательных функциональных массивов, а также совокупности унифицированных процедур манипулирования данными).

Структура является общепринятым и удобным инструментом, одинаково эффективно используемым как на уровне сознания человека при работе с абстрактными понятиями, так и на уровне логики машинных алгоритмов. Структура позволяет простыми способами свести многомерность содержательного описания к линейной последовательности записей. Именно это позволяет формализовать на общей понятийной основе взаимосвязь представлений информации в разных средах: обеспечить контролируемое сведение бесконечного разнообразия объектов и видов взаимосвязей реального мира к жестко детерминированному описанию – совокупности двоичных данных и машинно-ориентированных алгоритмов их обработки.

Выделение трех видов структур, относящихся к представлению объектов ПрО, имеет, в некотором смысле, принципиальный характер.

*Структура информации* – это неотъемлемое свойство информации (сведений, сигналов, воспринимаемых субъектом) о некоторой совокупности объектов предметной области, в контексте практической задачи (решаемой субъектом), в общем случае без учета того, будут ли для ее решения использованы средства программирования и вычислительные машины. Структурирование информации осуществляется системным аналитиком и сводится к выделению операционных объектов и определению их характеристических свойств и взаимосвязей.

*Структура данных* – это определение информационных массивов (состава и взаимосвязей данных на логическом уровне, соответствующих характеру информации и видам соответствующих преобразований). При определении структур данных необходимо не только установить состав массива, но и определить оптимальную их взаимосвязь (и, соответственно, определить критерии и методы оценки эффективности), например, выделение групп или агрегатов, имеющих иерархическую идентификацию. Эффективность в этом случае связывается с процессом построения программы («решателя» прикладной задачи) и, в каком-то



смысле – с эффективностью работы программиста. Например, при функциональной обработке массива необходимо обращаться к отдельным элементам, в то время как в операциях присваивания или при записи массива в файл поэлементное обращение приведет к увеличению размера текста программы, а в ряде случаев – к увеличению времени выполнения.

*Структура записей* – это определение структуры физической памяти: выделение, освобождение и защита областей физического носителя, способы адресации и пересылки. Эффективность в этом случае связывается с процессами обмена между устройствами оперативной и внешней памяти, искусственно вводимой для обеспечения функциональной эффективности отдельных операций (например, поиска по ключам) избыточностью данных,

Рассмотрим разновидности и типологию «компьютерных»<sup>20</sup> логических структур данных с точки зрения особенности их организации. Структура здесь в первую очередь определяет алгоритм выборки отдельных элементов данных, но в то же время необходимо отметить, что она отражает и особенности «технологии» организации и обработки информации, свойственные человеку в его повседневной деятельности.

Физически понятию *структура* соответствует *запись данных*. Запись – это упорядоченная в соответствии с характером взаимосвязей совокупность *полей* (элементов) данных, размещаемых в памяти в соответствии с их *типом*<sup>21</sup>. Поле представляет собой минимальную *адресуемую* (идентифицируемую) часть памяти – единицу данных, на которую можно сослаться при обращении к данным.

Т.о., структура данных – это способ отображения значений в памяти: размер области и порядок ее выделения (который и определит характер процедуры адресации/выборки). Зачастую именно успешность структурирования данных определяет сложность процедур их обработки [2].

Классификация структур данных должна проводиться с двух точек зрения.

1) По характеру взаимосвязи элементов структуры (с точки зрения порядка их размещения/выборки) виды структур можно разделить на линейные и нелинейные.

---

<sup>20</sup> Здесь не рассматриваются простейшие типы, к которым относятся *стандартные типы* – целые и вещественные числа, логические переменные, символы. Их состав и структура определяется в основном набором *встроенных* базовых типов данных и операций, свойственных конкретному типу ЭВМ.

<sup>21</sup> Память, отводимая для хранения значения элемента данных (*поле данных*), должна выбираться в соответствии с диапазоном значений, которые может иметь этот элемент. Поскольку для выполнения операции *присвоения значения* элементу данных (установление соответствующих битов в «0» или «1») необходимо сначала выделить память, для чего используются две схемы – статическая и динамическая. Для первой характерно выделение памяти до того, как реально появляются значения (обычно на этапе трансляции программы); для второй – в тот момент, когда программа во время исполнения получает конкретное значение. Кроме того, характер данных (тип данных) определяет способ представления и, соответственно, некоторое множество стандартных операций (*примитивов*).

2) По характеру информации, представляемой структурой (с точки зрения однородности и «элементарности» типов данных, отражающих понятийную структуру ПрО) – на однородные структуры, где все элементы находятся на одном понятийном уровне и имеют один тип данных, и неоднородные (композиционные), где элементы относятся к нескольким понятийным уровням или имеют разную природу.

### 3.5.1. Линейные структуры

К линейным структурам относятся массивы и последовательности, таблицы.

Порядок следования (и, соответственно, выборки) элементов таких структур имеет линейный характер и соответствует порядку расположения элементов в памяти: один за другим без каких-либо промежутков. Адрес элемента соответствует его положению и определяется индексом – порядковым номером элемента в последовательности размещения. К элементу имеется прямой доступ, если известен его индекс.

Особенностью линейной структуры является то, что при последовательной организации (размещении) она допускает возможность прямого доступа к произвольному элементу, поскольку условие однородности (однотипности) предполагает, что все элементы занимают расположенные строго последовательно области одинакового размера, что и позволяет достаточно просто вычислять значение физического адреса элемента по значению его индекса.

*Массив* представляет собой совокупность однотипных элементов, причем число элементов массива известно до его размещения, что позволяет строить гибкие многомерные системы адресации.

*Последовательность*<sup>22</sup>, так же, как и массив, представляет собой совокупность однотипных элементов. Однако число элементов до размещения неизвестно. И, хотя каждая конкретная последовательность имеет конечную длину, до начала обработки (и, соответственно, размещения) необходимо считать длину последовательности бесконечной. Принципиальность такого предположения выражается в том, что необходимо предусматривать специальную процедуру использования памяти (выделения/освобождения) и, возможно, алгоритм обработки последовательности по частям. Важность рассмотрения такого типа данных обусловлена тем, что именно он превалирует в операциях ввода/вывода с устройствами внешней памяти. Именно последовательный доступ по-

---

<sup>22</sup> Не рассматриваемые здесь *очереди* и *стеки* отличаются тем, что для них устанавливаются особые схемы добавления (размещения в памяти) новых элементов и их выборки. Для очереди порядок размещения/выборки определяется правилом «первым размещен – первым выбран», для стека – «первым размещен – последним выбран». Кроме того, выборка элемента влечет его удаление из последовательности.

звolyет организовать «поточковые» операции: однородность позволяет рассматривать пересылаемые данные как непрерывный поток. Поток не может быть прерван по контекстно определяемому условию, например, при пересылке текста - по значению кода «перевод строки», и это не заставляет программу анализировать значение каждого очередного элемента. И, кроме того, последовательный доступ – это простота управления памятью и устройством ввода-вывода.

*Таблица* – это последовательности, обычно представляемые строками - совокупностями разнотипных элементов. Или, иначе, таблица - это множество записей, каждая из которых представляет набор поименованных полей.

Однако с точки зрения размещения элементов таблица может быть представлена как одномерный массив (или, в случае БД - последовательность) с однородными композиционными элементами, каждый из которых представляет собой совокупность разнотипных элементов. Именно это позволяет свести ввод/вывод таких типов структур к последовательным элементарным операциям.

Кроме того, разнотипность элементов позволяет ввести отличную от перечислительной схему идентификации записей, определив одно из полей как ключ записи. Обычно ключ содержит значение, используемое в процедурах упорядочения и поиска записей.

### *3.5.2. Нелинейные структуры*

В качестве примеров нелинейных структур рассмотрим списки, деревья и сети.

Порядок следования (и, соответственно, выборки) элементов таких структур может не соответствовать порядку расположения элементов в памяти. Списки представляют собой пример линейного упорядочения, деревья – двумерного, сети – произвольного. Соответственно различаются методы и средства, обеспечивающие последовательность выборки элементов данных. Обычно для обеспечения возможности прямого доступа к произвольному элементу необходимо использовать вспомогательные структуры типа инвертированных списков.

*Список* также, как и массив, представляет собой совокупность однотипных элементов. Однако порядок выборки элементов может отличаться от порядка следования в памяти, определенного при размещении. Наиболее очевидный способ установления однонаправленного порядка выборки элементов – это сопоставить каждому элементу списка ссылку, указывающую на следующий элемент. Соответственно, для организации двунаправленного списка, допускающего также выборку в обратном порядке, каждый элемент должен иметь ссылку на предыдущий. Такая организация уже не допускает возможности прямого доступа, например, по номеру элемента.

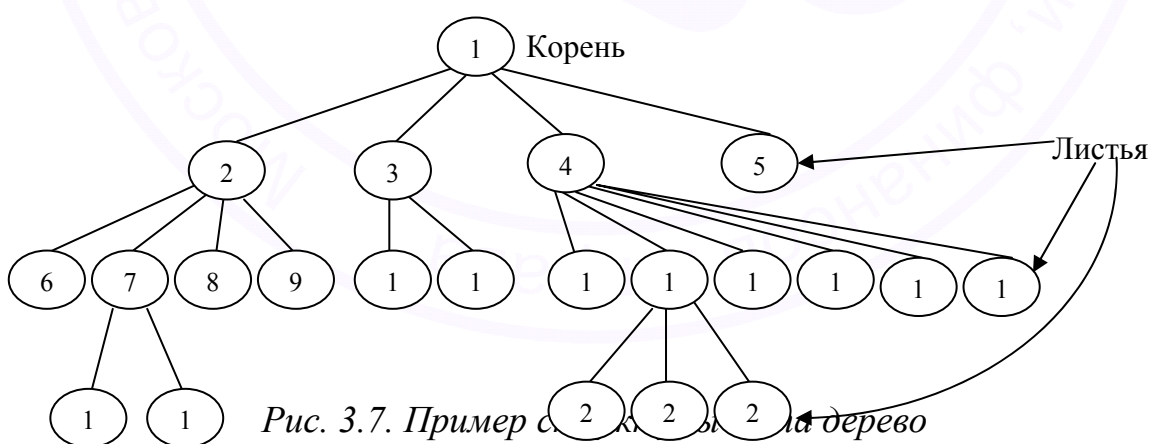
Кроме того, число элементов списка, как и в случае последовательностей, может быть неизвестно до размещения, и до начала обработки (и, соответственно, размещения) необходимо считать длину списка бесконечной, что ведет к необходимости предусматривать специальную процедуру выделения/освобождения памяти.

Таким образом, с точки зрения физической реализации элемент списка должен быть *составным*, включающим собственно информативные данные, несущий смысловое значение, и дополнительные данные (*ссылки*), определяющие порядок доступа к информативным элементам.

Понятие списка достаточно универсально. В общем случае ссылки могут указывать ответвления к другим спискам – *подпискам*. В зависимости от способа построения списка и предполагаемых путей доступа к элементам различают следующие виды ссылок: перекрестные, боковые, иерархические, множественные, что позволяет изменять «естественный» последовательный порядок прохода по элементам списка.

## Деревья

Дерево (Рис. 3.7) представляет собой иерархию элементов, называемых *узлами*. На самом верхнем уровне иерархии имеется только один узел — *корень*. Каждый узел, кроме корня, связан с одним узлом на более высоком уровне, называемым *исходным узлом* для данного узла. Каждый элемент имеет только один исходный. Каждый элемент может быть связан с одним или несколькими элементами на более низком уровне, которые называются *порожденными*. Элементы, расположенные в конце ветви, т. е. не имеющие порожденных, называются *листьями*.



Существует несколько способов представления структуры дерева. Например, дерево может быть определено как иерархия узлов с попарными связями, в которой:

1. Самый верхний уровень иерархии имеет один узел, называемый *корнем*.



2. Все узлы, кроме корня, связываются с одним и только одним узлом на более высоком уровне по отношению к ним самим.

Такое определение в части организации связей совпадает со списком, и, в частности, список представляет вырожденный случай дерева, в котором каждая вершина имеет не более одного поддерева.

Еще раз отметим, что деревья здесь рассматриваются как средство и для логического, и для физического представления данных. В логическом описании данных они используются для определения связей между элементами структуры, а при определении физической организации данных — для определения набора указателей, реализующих связи между ними.

Использование ссылок для организации доступа к отдельным элементам структуры не позволяет сократить процедуру поиска, в основу которой положен последовательный перебор. Процедура поиска будет эффективнее, если будет предварительно установлен некоторый порядок перехода к следующему элементу дерева. Такой порядок в ряде случаев определяется в отношении *метода обхода* и *регулярности* итераций, определяемой длиной пути и кратностью деления вершины.

Выделяют [2] три метода обхода: *сверху вниз, слева направо, снизу вверх*.

Регулярность обхода дерева может быть связана с *упорядоченными деревьями*, к которым относятся *сбалансированные* (Рис. 3.8) и *двоичные деревья* (Рис. 3.9).

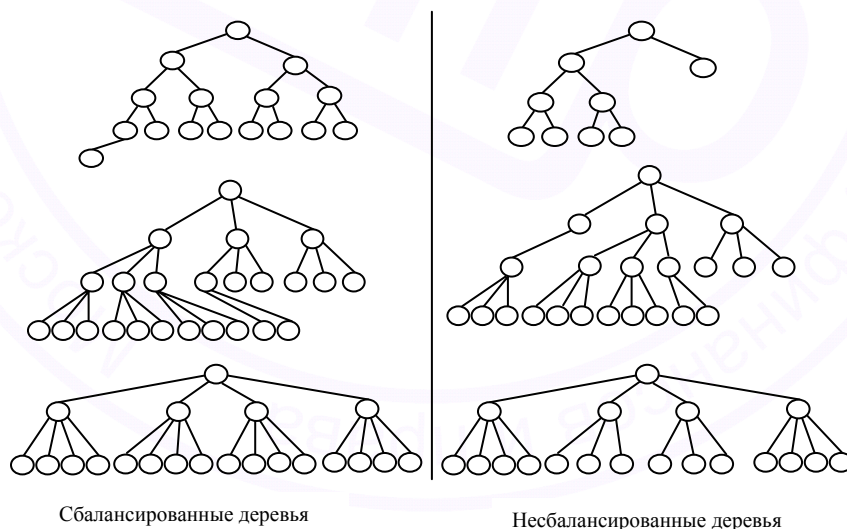
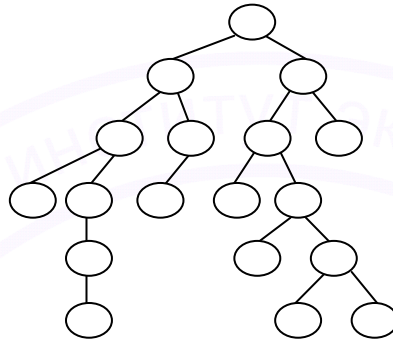


Рис. 3.8. Примеры сбалансированных и несбалансированных деревьев

*Сбалансированное дерево* в каждом узле имеет одинаковое число ветвей, причем процесс включения новых ветвей в узлы дерева идет сверху вниз, а на каждом уровне дерева — слева направо. Для дерева с фиксированным числом ветвей физическая организация данных будет более простой. Однако большая часть логических организаций данных

не может быть задана в виде сбалансированной древовидной структуры, и для их представления требуется переменное число ветвей в каждом узле. В то же время индексы могут быть построены в виде сбалансированных древовидных структур.

*Двоичные деревья* - это особая категория сбалансированных древовидных структур, в которой допускается не более двух ветвей для одного узла. На рис. 3.9 показано несбалансированное двоичное дерево.



*Рис. 3.9. Пример несбалансированного двоичного дерева*

Любые связи в дереве можно представить в виде двоичных древовидных структур. Рис. 3.10 иллюстрирует представление дерева в виде двоичного дерева.

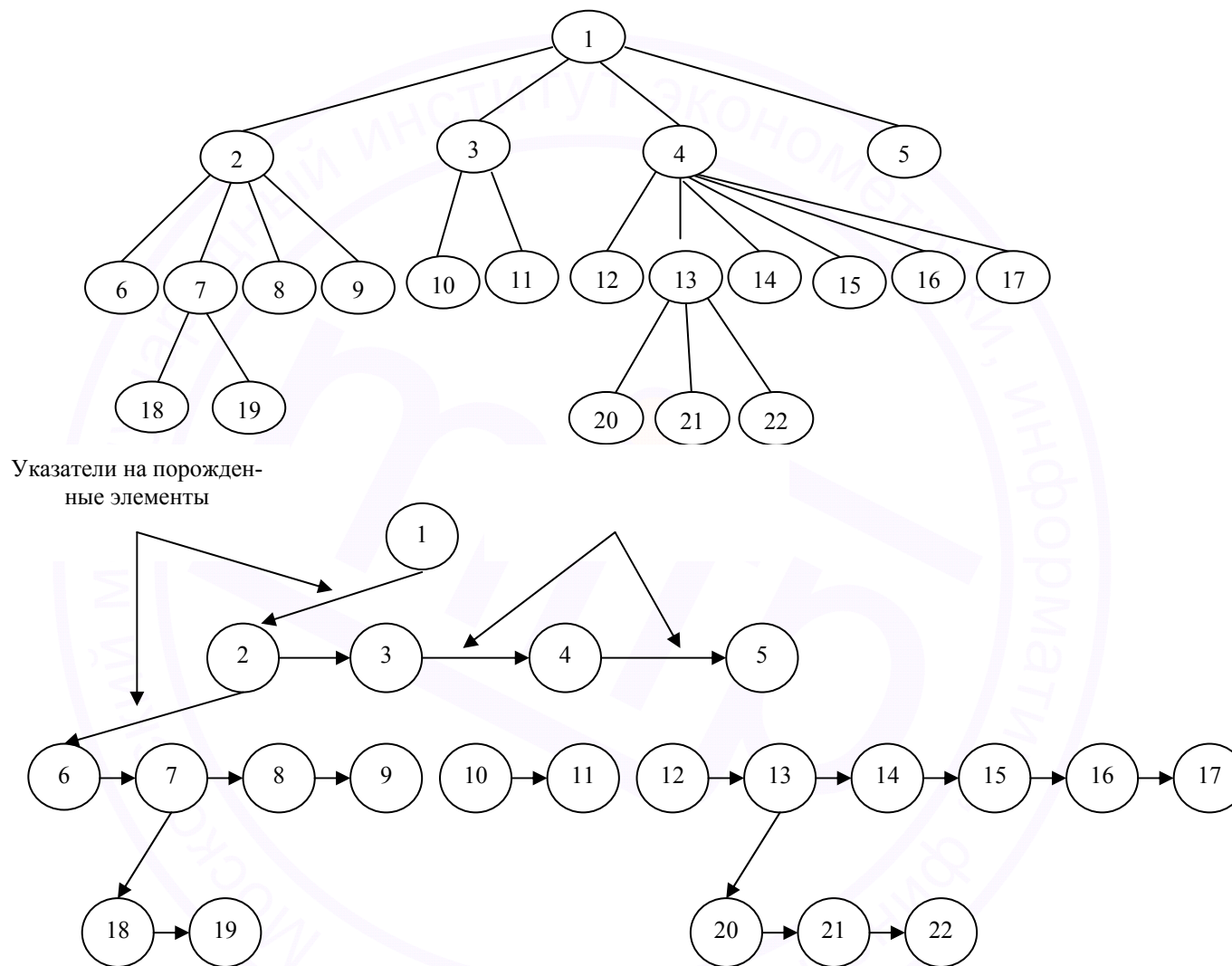


Рис. 3.10. Пример представления дерева в виде двоичного

При таком представлении каждый элемент может иметь указатели как на порожденные, так и на подобные элементы.

Различные виды двоичных деревьев, для которых характерно наличие жесткой схемы управления их ростом, достаточно эффективно используются для построения больших поисковых индексов, размещаемых обычно на устройствах внешней памяти. Кроме того, для таких деревьев можно организовать специальное «страничное» хранение поддеревьев, что сократит число физических обращений к устройству.

Заметим, что деревья поисковых индексов являются *однородными* структурами: каждый узел представлен элементами одного типа. Однако большинство баз должно поддерживать организацию данных, имеющих различную природу. В этом случае при работе с неоднородными структурами разной глубины, гарантировать регулярность, обеспечивающую эффективность процедур доступа, становится затруднительно.

### 3.5.3. Сетевые структуры

Иерархические структуры характерны для многих областей, однако во многих случаях отдельная запись требует более одного представления или связана с несколькими другими. В результате получаются обычно более сложные структуры по сравнению с древовидными. Например, генеалогическое дерево может быть представлено в виде древовидной структуры, только если для каждого элемента (личности) будет показан только один исходный элемент (родитель). Если бы были показаны оба родителя, то это была бы более сложная структура.

В сетевой структуре любой элемент может быть связан с любым другим элементом. Примеры сетевых структур приведены на рис. 3.11.

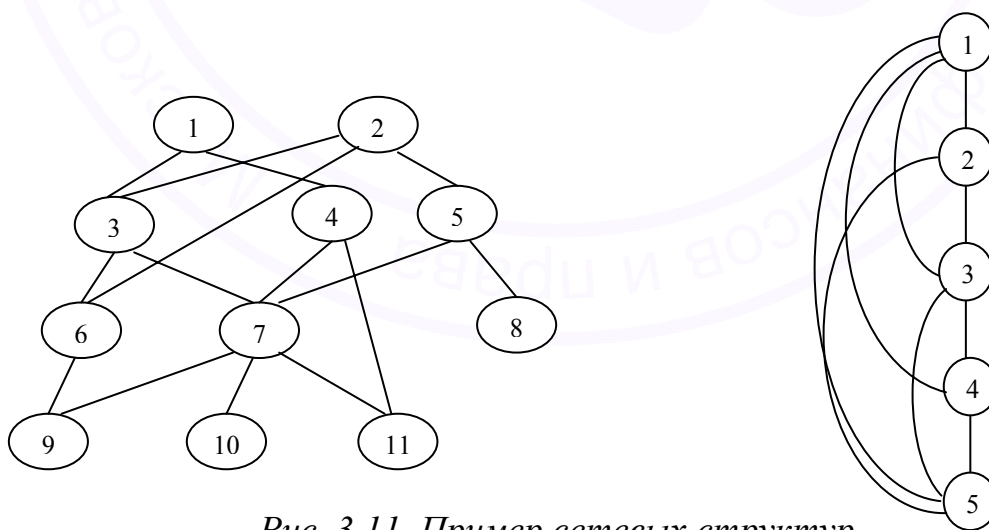


Рис. 3.11. Пример сетевых структур

Так же как и в случае древовидных структур, сетевую структуру можно описать с помощью исходных и порожденных элементов. Удобно представлять ее так, чтобы порожденные элементы располагались ниже



исходных. При рассмотрении некоторых сетевых структур естественно говорить об уровнях, так же как и в случае древовидных структур.

Во многих сетевых структурах, задающих связи между элементами, представление отношений между исходными и порожденными элементами аналогично представлению отношений в случае дерева: отношение исходный-порожденный является сложным (указывается сдвоенными стрелками), а отношение порожденный-исходный — простым (указывается одинарными стрелками).

На рис. 3.12 показана неоднородная сетевая структура с пятью типами элементов. Ни одна из их соединяющих линий не имеет сдвоенных стрелок в обоих направлениях. Каждое отношение может рассматриваться как отношение «исходный-порожденный». Запись ЗАКАЗ-НА-ЗАКУПКУ является порожденной по отношению к записи ИЗДЕЛИЕ и исходной по отношению к записи ПАРТИЯ-ТОВАРА.

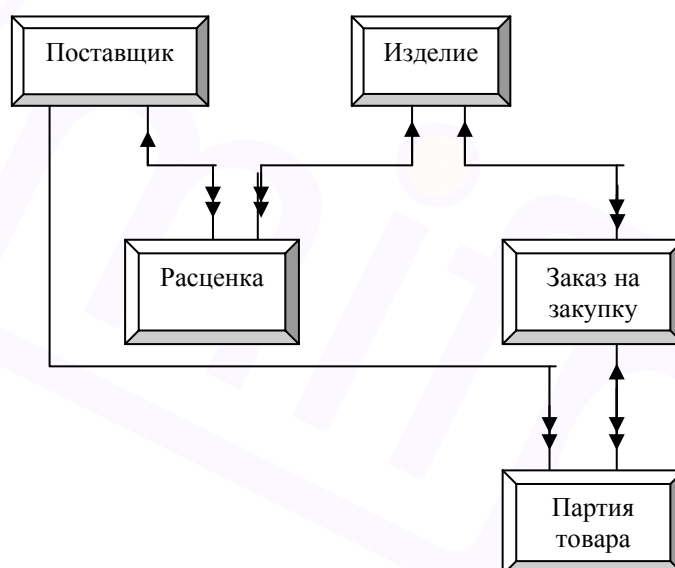


Рис. 3.12. Пример простой сетевой структуры

Желательно отличать структуры, в которых представление отношений «порожденный-исходный» является *простым* или не используется, от структур, в которых представление отношений между какими-то двумя типами данных является *сложным* в обоих направлениях. Для структур второго типа на одной из линий схемы будут сдвоенные стрелки, указывающие в разные стороны. Этот тип схемы назовем *сложной* сетевой структурой, а схему, в которой ни на одной из линий нет сдвоенных стрелок в обоих направлениях, — *простой* сетевой структурой. На рис. 3.9 показана простая сетевая структура. Она станет *сложной*, если ввести отношение ЗАКАЗ-НА-ЗАКУПКУ — ИЗДЕЛИЕ, когда один заказ может быть сделан сразу на несколько изделий. Для образования сложной сетевой структуры достаточно двух типов элементов. Например, ПОСТАВЩИК может иметь несколько порожденных записей, потому что может поставляться более одного вида изделий. С другой сто-

роны, элемент ИЗДЕЛИЕ может иметь более одного исходного элемента, поскольку это изделие может поставляться различными поставщиками.

В некоторых случаях один элемент данных может быть связан с целой совокупностью других элементов данных. Например, одно изделие может поставляться несколькими поставщиками, каждый из которых установил свою цену на это изделие. Элемент данных ЦЕНА не может быть ассоциирован только с элементом ИЗДЕЛИЕ или только с элементом ПОСТАВЩИК, а должен быть связан одновременно с двумя. Информация такого рода, т. е. данные, ассоциированные с совокупностью элементов, называют иногда *данными пересечения*.

Некоторые структуры содержат циклы. *Циклом* считается ситуация, в которой предшественник узла является в то же время его последователем. Отношения «исходный-порожденный» образуют при этом замкнутый контур. Например, завод выпускает различную продукцию. Некоторые изделия производятся на других заводах-субподрядчиках. С одним контрактом может быть связано производство нескольких изделий. Представление этих отношений и образует цикл.

Иногда элементы связаны с другими элементами того же типа. Такая ситуация называется *петлей*. На рис. 3.13 приведены две достаточно распространенные ситуации, где могут использоваться петли. В массиве служащих специфицированы связи, существующие между некоторыми служащими. В базу данных списка материалов введено дополнительное усложнение: некоторые узлы сами состоят из узлов.

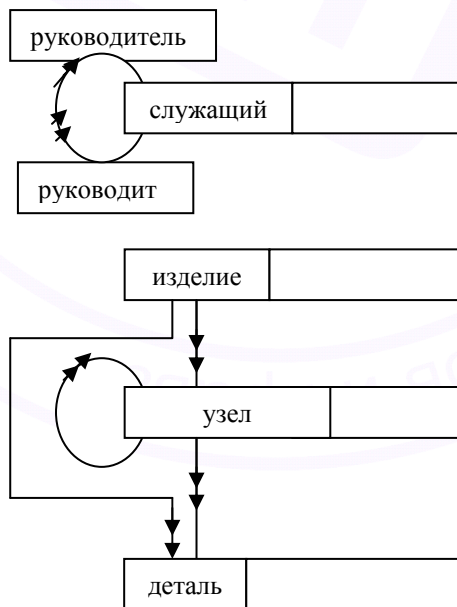


Рис. 3.13. Пример сетевой структуры с петлей

Разделение сетевых структур на простые и сложные необходимо потому, что сложные структуры требуют более сложных методов *физического* представления. Это не всегда является недостатком, поскольку

сложную сетевую структуру можно (а в большинстве случаев и следует) преобразовать к простому виду.

### 3.6. Реляционная модель данных

#### 3.6.1. Основные понятия реляционной модели данных

Реляционная<sup>23</sup> модель является удобной и наиболее привычной формой представления данных в виде таблицы. В отличие от иерархической и сетевой модели, такой способ представления 1) понятен пользователю-непрограммисту; 2) позволяет легко изменять схему – присоединять новые элементы данных и записи без изменения соответствующих подсхем; 3) обеспечивает необходимую гибкость при обработке непредвиденных запросов. К тому же любая сетевая или иерархическая схема может быть представлена двумерными отношениями.

Одним из основных преимуществ реляционной модели является ее однородность. Все данные рассматриваются как хранимые в таблицах, в которых каждая строка имеет один и тот же формат. Каждая строка в таблице представляет некоторый объект реального мира или соотношение между объектами. Пользователь модели сам должен для себя решить вопрос, обладают ли соответствующие сущности реального мира однородностью. Этим самым решается проблема пригодности модели для предполагаемого применения.

Основными понятиями, с помощью которых определяется реляционная модель, являются следующие: *домен, отношение, кортеж, кардинальность, атрибуты, степень, первичный ключ*. Соотношение этих понятий иллюстрируется рис. 3.14. Эти понятия представляют специальную терминологию, введенную авторами теоретических основ, однако они имеют и более привычные аналоги (но не во всем эквиваленты!), соответствие которых приведено в следующей таблице 3.1.

Таблица 3.1

Домен	Совокупность допустимых значений
Кортеж	Таблица
Кардинальность	Количество строк в таблице
Атрибут	Поле, столбец таблицы
Степень отношения	Количество полей (столбцов)
Первичный ключ	Уникальный идентификатор

*Домен* – это совокупность значений, из которой берутся значения соответствующих атрибутов определенного отношения. С точки зрения

<sup>23</sup> В математических дисциплинах понятию «таблица» соответствует понятие «отношение» (relation). Отсюда и произошло название модели – реляционная. Т.е., применительно к базам данных понятия «реляционная БД» и «табличная БД» по существу являются синонимами.

программирования домен – это тип данных, определяемый системой (стандартный) или пользователем.

*Первичный ключ* – это столбец или некоторое подмножество столбцов, которые уникально, т.е. единственным образом определяют строки. Первичный ключ, который включает более одного столбца, называется множественным, или комбинированным, или составным. Правило целостности объектов утверждает, что первичный ключ не может быть полностью или частично пустым, т.е. иметь значение null.

Остальные ключи, которые можно также использовать в качестве первичных, называются потенциальными или *альтернативными* ключами.

*Внешний ключ* – это столбец или подмножество одной таблицы, который может служить в качестве первичного ключа для другой таблицы. *Внешний ключ* таблицы является ссылкой на первичный ключ другой таблицы. Правило ссылочной целостности гласит, что внешний ключ может быть либо пустым, либо соответствовать значению первичного ключа, на который он ссылается. Внешние ключи являются неотъемлемой частью реляционной модели, поскольку реализуют связи между таблицами базы данных.

Внешний ключ, как и первичный ключ, тоже может представлять собой комбинацию столбцов. На практике внешний ключ всегда будет составным (состоящим из нескольких столбцов), если он ссылается на составной первичный ключ в другой таблице. Очевидно, что количество столбцов и их типы данных в первичном и внешнем ключах совпадают.

Если таблица связана с несколькими другими таблицами, она может иметь несколько внешних ключей.

Модель предъявляет к таблицам следующие требования:

1. данные в ячейках таблицы должны быть структурно неделимыми<sup>24</sup>;
2. данные в одном столбце должны быть одного типа;
3. каждый столбец должен быть уникальным (недопустимо дублирование столбцов);
4. столбцы размещаются в произвольном порядке;
5. строки размещаются в таблице также в произвольном порядке;
6. столбцы имеют уникальные наименования.

---

<sup>24</sup> Недопустимо, чтобы в ячейке таблицы содержались более одной порции информации, что иногда используется путем создания композиционных данных (информационное кодирование). Примером служит идентификационный номер автомобиля. Если записать его в одну ячейку, то будет нарушен принцип неделимости информации, поскольку в ячейке окажутся разделяемые данные, имеющие разную информационную сущность, такие, как наименование модели, номер кузова, двигателя, сведения о предприятии-изготовителе и т.д.



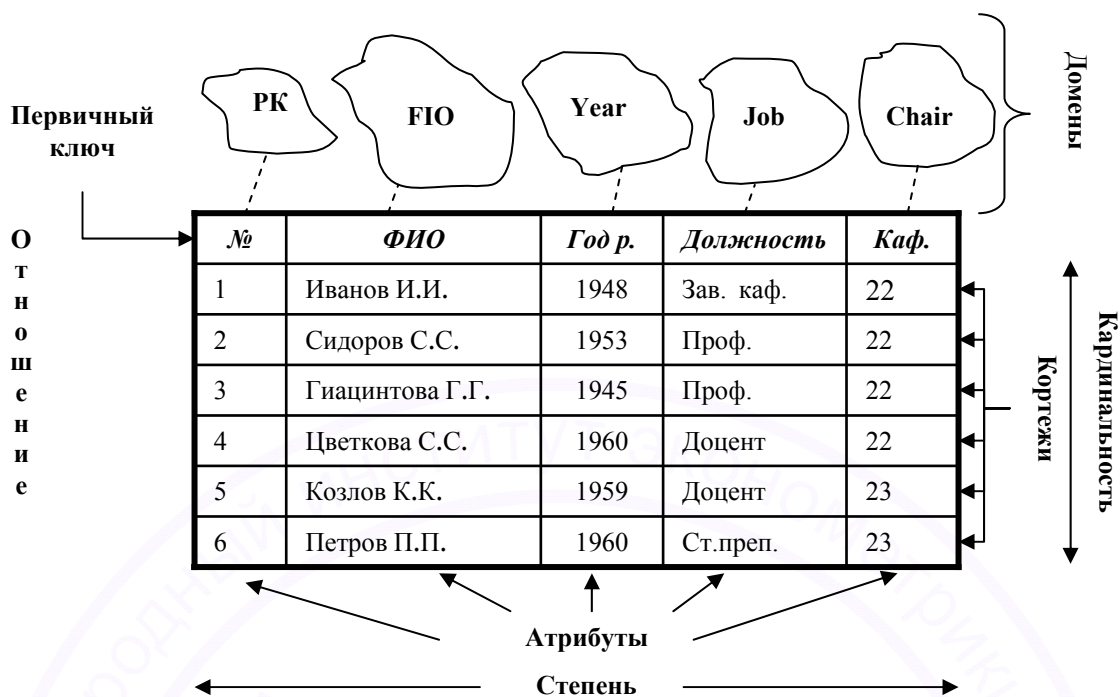


Рис 3.14. Основные понятия реляционной модели

В целом концепция реляционной модели определяется следующими двенадцатью правилами Кодда [приводится по 3]:

1. *Правило информации.* Вся информация в базе данных должна быть предоставлена исключительно на логическом уровне и только одним способом - в виде значений, содержащихся в таблицах.

2. *Правило гарантированного доступа.* Логический доступ ко всем и каждому элементу данных (атомарному значению) в реляционной базе данных должен обеспечиваться путём использования комбинации имени таблицы, первичного ключа и имени столбца.

3. *Правило поддержки недействительных значений.* В реляционной базе данных должна быть реализована поддержка недействительных значений, которые отличаются от строки символов нулевой длины, строки пробельных символов, от нуля или любого другого числа и используются для представления отсутствующих данных независимо от типа этих данных.

4. *Правило динамического каталога, основанного на реляционной модели.* Описание базы данных на логическом уровне должно быть представлено в том же виде, что и основные данные, чтобы пользователи, обладающие соответствующими правами, могли работать с ним с помощью того же реляционного языка, который они применяют для работы с основными данными.

5. *Правило исчерпывающего подязыка данных.* Реляционная система может поддерживать различные языки и режимы взаимодействия с пользователем (например, режим вопросов и ответов). Однако должен

существовать по крайней мере один язык, операторы которого можно представить в виде строк символов в соответствии с некоторым четко определенным синтаксисом и который в полной мере поддерживает определение данных; определение представлений; обработку данных (интерактивную и программную); условия целостности; идентификация прав доступа; границы транзакций (начало, завершение и отмена).

6. *Правило обновления представлений.* Все представления, которые теоретически можно обновить, должны быть доступны для обновления.

7. *Правило добавления, обновления и удаления.* Возможность работать с отношением как с одним операндом должна существовать не только при чтении данных, но и при добавлении, обновлении и удалении данных.

8. *Правило независимости физических данных.* Прикладные программы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при любых изменениях способов хранения данных или методов доступа к ним.

9. *Правило независимости логических данных.* Прикладные программы и утилиты для работы с данными должны на логическом уровне оставаться нетронутыми при внесении в базовые таблицы любых изменений, которые теоретически позволяют сохранить нетронутыми содержащиеся в этих таблицах данные.

10. *Правило независимости условий целостности.* Должна существовать возможность определять условия целостности, специфические для конкретной реляционной базы данных, на подязыке реляционной базы данных и хранить их в каталоге, а не в прикладной программе.

11. *Правило независимости распространения.* Реляционная СУБД не должна зависеть от потребностей конкретного клиента.

12. *Правило единственности.* Если в реляционной системе есть низкоуровневый язык (обрабатывающий одну запись за один раз), то должна отсутствовать возможность использования его для того, чтобы обойти правила и условия целостности, выраженные на реляционном языке высокого уровня (обрабатывающем несколько записей за один раз).

Правило 2 указывает на роль первичных ключей при поиске информации в базе данных. Имя таблицы позволяет найти требуемую таблицу, имя столбца позволяет найти требуемый столбец, а первичный ключ позволяет найти строку, содержащую искомый элемент данных.

Правило 3 требует, чтобы отсутствующие данные можно было представить с помощью недействительных значений (NULL).

Правило 4 гласит, что реляционная база данных должна сама себя описывать. Другими словами, база данных должна содержать набор *системных таблиц*, описывающих структуру самой базы данных.

Правило 5 требует, чтобы СУБД использовала язык реляционной

базы данных, например SQL. Такой язык должен поддерживать все основные функции СУБД — создание базы данных, чтение и ввод данных, реализацию защиты базы данных и т.д.

Правило 6 касается *представлений*, которые являются виртуальными таблицами, позволяющими показывать различным пользователям различные фрагменты структуры базы данных. Это одно из правил, которые сложнее всего реализовать на практике.

Правило 7 акцентирует внимание на том, что базы данных по своей природе ориентированы на множества. Оно требует, чтобы операции добавления, удаления и обновления можно было выполнять над множествами строк. Это правило предназначено для того, чтобы запретить реализации, в которых поддерживаются только операции над одной строкой.

Правила 8 и 9 означают отделение пользователя и прикладной программы от низкоуровневой реализации базы данных. Они утверждают, что конкретные способы реализации хранения или доступа, используемые в СУБД, и даже изменения структуры таблиц базы данных не должны влиять на возможность пользователя работать с данными.

Правило 10 гласит, что язык базы данных должен поддерживать ограничительные условия, налагаемые на вводимые данные и действия, которые могут быть выполнены над данными.

Правило 11 гласит, что язык базы данных должен обеспечивать возможность работы с распределенными данными, расположенными на других компьютерных системах.

Правило 12 предотвращает использование других возможностей для работы с базой данных, помимо языка базы данных, поскольку это может нарушить ее целостность.

### 3.6.2. Основы реляционной алгебры

С точки зрения внешнего представления (абстрагирования на логическом уровне) объектов реального мира *модель данных* — это основные понятия и способы, используемые при анализе и описании предметной области.

Среди многих попыток представить обработку данных на формальном абстрактном уровне реляционная модель, предложенная Э.Ф. Коддом, стала по существу первой работоспособной *моделью данных*, поскольку помимо средств описания объектов имела эффективный инструментарий преобразований этих описаний - операции реляционной алгебры.

Реляционная алгебра в том виде, в котором она была определена Э.Ф. Коддом, состоит из двух групп по четыре оператора.

1. Традиционные операции над множествами (но модифицированные с учетом того, что их операндами являются отношения, а не

произвольные множества): объединение, пересечение, разность и декартово произведение.

2. Специальные реляционные операции: выборка, проекция, соединение, деление.

Рассмотрим подробнее операции реляционной алгебры.

*Объединение* возвращает отношение, содержащее все кортежи, которые принадлежат либо одному из двух заданных отношений, либо им обоим.

## Объединение

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст.преп.	24
Лютикова Л.Л.	1977	Ассистент	24

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст.преп.	24
Лютикова Л.Л.	1977	Ассистент	24

*Пересечение* возвращает отношение, содержащее все кортежи, которые принадлежат одновременно двум заданным отношениям.

## Пересечение

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст.преп.	24
Лютикова Л.Л.	1977	Ассистент	24

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23

*Разность* возвращает отношение, содержащее все кортежи, которые принадлежат первому из двух заданных отношений и не принадлежат второму.



## Разность

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст. преп.	24
Лютикова Л.Л.	1977	Ассистент	24

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22

**Произведение** - возвращает отношение, содержащее все возможные кортежи, которые являются сочетанием двух кортежей, принадлежащих соответственно двум заданным отношениям.

## Произведение

<i>Job</i>
Зав. каф.
Проф.
Ст. преп.
Доцент
Ассистент

<i>Chair</i>
22
23

<i>Job</i>	<i>Chair</i>
Зав. каф.	22
Зав. каф.	23
Проф.	22
Проф.	23
Ст. преп.	22
Ст. преп.	23
Доцент	22
Доцент	23
Ассистент	22
Ассистент	23

A	X
B	Y
C	

 $\times$ 

X	
Y	

 $=$ 

A	X
A	Y
B	X
B	Y
C	X
C	Y

**Выборка** – возвращает отношение, содержащие все кортежи из заданного отношения, которые удовлетворяют указанным условиям.

## Выборка

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст.преп.	24
Лютикова Л.Л.	1977	Ассистент	24

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22

Chair = 22

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Петров П.П.	1960	Ст.преп.	24

(Chair = 24) AND  
(Year < 1970)

*Проекция* возвращает отношение, содержащее все кортежи (под-кортежи) заданного отношения, которые остались в этом отношении после исключения из него некоторых атрибутов.

## Проекция

<i>FIO</i>	<i>Year</i>	<i>Job</i>	<i>Chair</i>
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	23
Козлов К.К.	1959	Доцент	23
Петров П.П.	1960	Ст.преп.	24
Лютикова Л.Л.	1977	Ассистент	24

<i>FIO</i>	<i>Job</i>
Иванов И.И.	Зав. каф.
Сидоров С.С.	Проф.
Гиацинтова Г.Г.	Проф.
Цветкова С.С.	Доцент
Козлов К.К.	Доцент
Петров П.П.	Ст.преп.
Лютикова Л.Л.	Ассистент

*Соединение* возвращает отношение, содержащее все возможные кортежи, которые представляют собой комбинацию атрибутов двух кортежей, принадлежащих двум заданным, при условии, что в этих двух комбинированных кортежах присутствуют одинаковые значения в одном или нескольких общих для исходных отношений атрибутах (причем эти общие значения в результирующем кортеже появляются один раз, а не дважды).

## Соединение

<i>FIO</i>	<i>Job</i>	<i>Chair</i>
Иванов И.И.	Зав. каф.	22
Сидоров С.С.	Проф.	22
Гиацинтова Г.Г.	Проф.	22
Цветкова С.С.	Доцент	23
Козлов К.К.	Доцент	23
Петров П.П.	Ст.преп.	24
Лютикова Л.Л.	Ассистент	24

A1	B1	B1	C1	A1	B1	C1
A2	B1	B2	C1	A2	B1	C1
A3	B3	B3	C3	A3	B2	C2

<i>FIO</i>	<i>Job</i>	<i>Chair</i>	<i>Pay</i>
Иванов И.И.	Зав. каф.	22	3000
Сидоров С.С.	Проф.	22	2500
Гиацинтова Г.Г.	Проф.	22	2500
Цветкова С.С.	Доцент	23	2000
Козлов К.К.	Доцент	23	2000
Петров П.П.	Ст.преп.	24	1500
Лютикова Л.Л.	Ассистент	24	1200

<i>Job</i>	<i>Pay</i>
Зав. каф.	3000
Проф.	2500
Доцент	2000
Ст.преп.	1500
Ассистент	1200

**Деление** для заданных двух унарных отношений и одного бинарного возвращает отношение, содержащее все кортежи из первого унарного отношения, которые содержатся также в бинарном отношении и соответствуют всем кортежам во втором унарном отношении.

## Деление

Делимое



<i>Job</i>
Зав. каф.
Проф.
Доцент
Ст.преп.
Ассистент

Посредник



<i>Job</i>	<i>Chair</i>
Зав. каф.	22
Проф.	22
Доцент	22
Зав. каф.	23
Доцент	23
Ст.преп.	24
Ассистент	24

Делитель



<i>Chair</i>
22

<i>Chair</i>
22
23

<i>Job</i>
Зав. каф.
Проф.
Доцент

<i>Job</i>
Зав. каф.
Доцент

Результат выполнения любой операции над отношением также является отношением, поэтому результат одной операции может использоваться в качестве исходных данных для другой. Другими словами, можно записывать вложенные реляционные выражения, т.е. выражения, в которых операторы сами представлены реляционными выражениями, причем произвольной сложности. Эта особенность называется свойством *реляционной замкнутости*.

Важно, как отмечается в [4], что отношение имеет две части - заголовок и тело. Нестрого говоря, заголовок - это атрибуты, а тело - это кортежи. Заголовок для базового отношения, т.е. значение базовой переменной-отношения, очевидно, вполне конкретен и известен системе, поскольку он задается как часть определения соответствующей базовой

переменной-отношения. Т.к. результат обязательно должен иметь вполне определенный тип отношения, поэтому, если рассматривать свойство реляционной замкнутости более строго, каждая реляционная операция должна быть определена таким образом, чтобы выдавать результат с надлежащим типом отношения (в частности, с соответствующим набором имен атрибутов или заголовком).

Реляционная алгебра имеет набор правил вывода типов (отношений), позволяющих вывести тип (отношение) на выходе произвольной реляционной операции, зная типы (отношения) на входе этой операции. Задав такие правила для всех операций, можно гарантировать, что для реляционного выражения любой сложности будет вычисляться результат, имеющий вполне определенный тип (отношение) и, в частности, известный набор имен атрибутов.

Рассмотренные восемь операторов Кодда не являются минимальным набором, так как не все из них примитивны, т.е. часть из них можно определить через другие операторы. Действительно, операции соединения, пересечения и деления можно определить через остальные пять. Эти пять операций (выборка, проекция, произведение, объединение и разность) можно рассматривать как примитивные в том смысле, что ни одна из них не выражается через другие. Они образуют минимальный набор, но, тем не менее, необязательно единственно возможный. Кроме того, остальные три операции (в особенности операция соединения) на практике используются настолько часто, что, несмотря на то, что они не являются примитивными, имеет смысл обеспечить их непосредственную поддержку.

Предшествующее рассмотрение алгебры представлено в контексте только операций выборки данных. Однако, как отмечается в классических введениях к реляционной алгебре, ее основная цель - обеспечить запись реляционных выражений позволяющих определять:

- области выборки, т.е. тех данных, которые должны быть доставлены в результате выполнения операции выборки;
- области обновления, т.е. данных, которые должны быть вставлены, изменены или удалены в результате выполнения операции обновления;
- правила поддержки целостности данных, т.е. некоторых особых требований, которым должна удовлетворять база данных;
- производные переменные-отношения, т.е. те данные, которые должны быть включены в представления базы данных;
- требования устойчивости, т.е. данные, которые должны быть включены в контролируемую область для некоторых операций управления параллельным доступом к информации;
- ограничения защиты, т.е. данные, для которых осуществляется тот или иной тип контроля доступа.



В целом, выражения реляционной алгебры служат для символического высокоуровневого представления намерений пользователя (например, в отношении некоторого определенного запроса). И именно потому, что подобные выражения являются символическими и высокоуровневыми, ими можно манипулировать в соответствии с различными высокоуровневыми правилами преобразования, в том числе и для оптимизации процедур выполнения запросов на данные.

### *Контрольные вопросы*

1. Дайте определение понятия предметной области
2. Что является результатом абстрагированного описания предметной области?
3. Приведите варианты модели трехуровневого представления ПрО.
4. Дайте определение атрибутивного способа идентификации объектов и записей.
5. Приведите типологию простых запросов.
6. Определите понятия первичного и вторичного ключа записи.
7. Определите основные требования к модели данных.
8. Дайте сравнительную характеристику понятий *модель данных* и *модель базы данных*.
9. Перечислите операции реляционной алгебры.
10. Дайте определение понятия реляционной замкнутости.
11. Дайте определение реляционных операций соединения, пересечения и деления через пять других операций.
12. Докажите ассоциативность и коммутативность реляционной операции объединения
13. Являются ли реляционные операции умножения и деления взаимнообратными?
14. Дайте сравнительную характеристику понятий *структура данных*, *структура записи*, *структура информации*.
15. Дайте определение древовидной структуры.
16. Определите характерные свойства и отличия линейных и нелинейных структур.
17. Проведите сравнительный анализ основных типов нелинейных структур.
18. Перечислите основные свойства реляционной структуры данных.

## **Глава 4. Физические модели баз данных**

Физическая модель базы данных определяет способ размещения данных на носителях (устройствах внешней памяти), а также способ и средства организации эффективного доступа к ним. Поскольку СУБД функционирует в составе и под управлением операционной системы и база данных размещается обычно на устройствах общего доступа (разделяемый ресурс), используемого самой ОС и другими прикладными программами, то организация хранения данных и доступа к ним в значительной степени зависит от принципов и методов управления данными операционной системы. И, естественно, СУБД в той или иной степени использует не только файловую систему и подсистему ввода-вывода ОС, но и специализированные методы доступа, основанные на тех или иных принципах организации данных.

Основное внимание в этой главе будет уделено системам управления данными, построенным на основе однородных файлов, а также рассмотрению основ построения систем управления, использующих «одно-файловые» страничные модели организации данных.

### ***4.1. Организация данных на машинных носителях***

С общепринятой точки зрения к вопросам организации данных относятся:

- выбор типа записи – единицы обмена в операциях ввода-вывода;
- выбор способа размещения записей в файле и, возможно метода оптимизации размещения;
- выбор способа адресации и метода доступа к записям.

Целесообразность выделения именно таких аспектов организации была предельно очевидна на начальной стадии развития таких запись-ориентированных систем и устройств внешней памяти, как магнитные ленты и диски. Но, следует отметить, что широкое использование современных поток-ориентированных систем ввода-вывода не уменьшило принципиальное, да и практическое, значение давно известных методов и решений, построенных на запись-ориентированных принципах. Основные понятия и подходы к физической организации и обработке данных, кратко обсуждаемые ниже, иллюстрируются рис. на 4.1.

## Файл

## Типы записей

Поток-ориентированный	Запись						EOF
Записей фиксированной длины	Запись	EOF	Запись	EOF	Запись	EOF	...
С блокировкой записей фиксир. длины	Запись	Запись	EOF	...	Запись	Запись	EOF
Записей переменной длины	L	Запись	EOF	L	Запись	EOF	...
С блокировкой записей переменной длины	L	Запись	L	Запись	EOF	...	L
Записей неопределенной длины	Запись	EOF	Запись	EOF	...	Запись	EOF

Рис. 4.1. Способы организации файлов данных

### 4.1.1. Типы записей

*Логическая запись*, с которой работает прикладная программа – это совокупность элементов или агрегатов данных, воспринимаемая и, обычно, физически отдельно размещаемая в рабочей области памяти прикладной программой как единое целое. Последовательность записей в логике обработки образует *файл*.

*Физическая запись*, с которой работает файловая система - это совокупность данных, которые размещаются в файле обычно на внешнем носителе, и могут быть считаны или записаны как единое целое одной командой ввода-вывода. Здесь файл<sup>25</sup> – это последовательность физических записей, размещаемых в линейном пространстве носителя но, в общем случае, не обязательно в линейном порядке.

Организация данных в случаях логического и физического представления может не совпадать, в частности, одна физическая запись может включать несколько логических (*блокирование записей*). При этом алгоритмы выделения логических записей из физической в значительной степени зависят от *типа записи*, рассматриваемого как характер организации последовательности байтов.

На логическом уровне выделяют следующие типы:

- *записи фиксированной длины*, для размещения каждой из которых выделяется всегда память фиксированной длины, объявляемой заранее. В этом случае данные, образующие запись, имеют устойчивую природу и представляются жесткими структурами, например ряд числовых полей или символьная последовательность заданной длины;

<sup>25</sup> В некоторых операционных системах, например IBM, файл на внешних носителях называют *набором данных* в отличие от логического файла.

- *записи переменной длины*, когда каждый экземпляр записи может иметь длину отличную от длины другой записи в том же наборе. В этом случае запись содержит либо элементы данных переменной длины (например, текстовую строку), либо переменное число элементов фиксированной длины.

Организация физической записи для достаточно часто встречающегося случая блокирования логических записей фиксированной или переменной длины представлена на рис. 4.2.

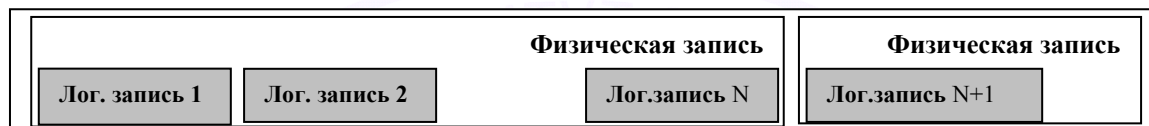


Рис. 4.2. Физическая организация логических записей

При этом структура представления логической записи *переменной длины*<sup>26</sup> отличается тем, что байтам содержания - собственно данным, образующим логическую запись, предшествуют байты значения длины содержания этой логической записи.

Существует и другая физическая структура представления записей, имеющих переменную длину – *запись неопределенной длины*, когда данные, образующим логическую запись, завершаются разделителем «конец записи»<sup>27</sup>.

Порядок доступа к записи может быть только последовательным, поскольку для определения начала следующей записи надо считать значение длины текущей.

Для файлов записей фиксированной длины доступ будет проще, так как адрес начала любой записи может быть вычислен умножением относительного номера нужной записи на длину записи.

Физические записи на носителе следуют непосредственно друг за другом. При этом выделение отдельной записи может производиться двумя способами, определяемыми технологиями записи данных на носитель.

Первый способ, применяемый в запись-ориентированных устройствах внешней памяти мэйнфреймов, основан на том, что каждая запись отделяется от соседней физическим промежутком, неиспользуемым для записи, и воспринимаемым устройством чтения как сигнал «конец записи».

Другой способ – это размещение байтов следующей записи непосредственно за последним байтом предыдущей записи без каких либо

<sup>26</sup> В современных файловых системах практически не используется.

<sup>27</sup> В поток-ориентированных файловых системах этому соответствует организация текстовых файлов, где запись – это последовательность символов, образующих строку, которая завершается специальными кодами «CR» «LF».



разделителей. Для этого способа характерна меньшая зависимость от особенностей устройства: оптимизация процессов ввода-вывода, в том числе блокирование<sup>28</sup> записей, переносится в прикладную программу.

#### *4.1.2. Организация файлов - способ размещения записей*

Записи файла обычно располагаются на носителе последовательно в том порядке, как они создаются в прикладной программе. Но иногда физическая последовательность размещения записей может отличаться от их логической последовательности.

Последовательность размещения физических записей естественно может быть только одна (если содержание логической записи сознательно не дублируется в другой форме) и она должна быть выбрана с учетом эффективности использования данных в различных приложениях.

Выбор последовательности связывается с одним из следующих обстоятельств:

1. Ускорением выполнения наиболее частых операций путем размещения записей в той последовательности, которая требуется при последующей обработке.
2. Ускорением или упрощением средств адресации файла (например, средств прямой адресации или хэширования).
3. Уменьшением размера используемого индекса и сокращением таким образом времени поиска в нем.
4. Сокращением среднего времени доступа за счет размещения в наиболее доступных местах записей, к которым происходит наиболее частое обращение.
5. Облегчением операций включения, обновления и удаления записей в интенсивно изменяемых файлах.

---

<sup>28</sup> Блокирование записей переменной и неопределенной длины в этом случае будет практически невозможно.

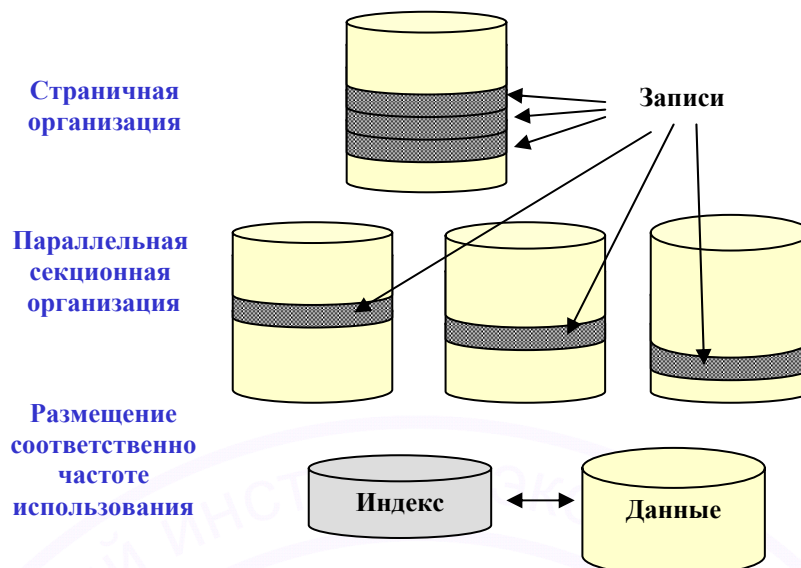


Рис. 4.3. Способы организации файлов

Можно выделить две «чистые» стратегии определения места (адреса) для размещения записей: *последовательное* (sequential) и *произвольное* (random) размещение. В этом смысле алгоритм размещения определяет *тип организации* файла.

В первом случае каждая следующая запись будет располагаться физически следом за предыдущей. Во втором – по месту, адрес которого будет определяться в зависимости от некоторых факторов, в том числе, упомянутых выше.

Хотя записи на устройствах с прямым доступом могут записываться и читаться в любой последовательности, для каждой структуры данных существует некоторая определенная последовательность, в которой записи можно читать намного быстрее, чем при других способах размещения.

Рассмотрим следующие, наиболее распространенные методы организации файлов, позволяющих оптимизировать доступ к записям.

**Страничная организация.** Данные можно перемещать между внешней и оперативной памятью страницами фиксированной длины. Размер страницы определяется системой, а не длиной записи. Там, где применяется страничная организация памяти, данные логически независимы от размера страницы, но они должны быть физически сгруппированы СУБД так, чтобы эффективно заполнять страницы.

**Параллельная секционная организация.** Если имеется несколько механизмов доступа, которые могут работать одновременно, то для минимизации времени ожидания данные могут быть расположены на запоминающих устройствах так, чтобы одновременно было задействовано как можно большее число механизмов доступа.

При параллельной секционной организации существуют два вида ожиданий. Запросы должны ожидать позиционирования механизма доступа (операция установки и задержки на вращение), а затем - ждать выполнения операции чтения-записи. Время, в течение которого запись читается, значительно меньше времени, в течение которого позиционируется механизм доступа. Следовательно, полное время доступа к записи при параллельной организации будет меньше.

В современных СУБД наиболее часто используется страничная организация данных, поскольку гораздо проще иметь весь файл целиком на одном пакете дисков, чем на нескольких, однако принципы секционной организации вновь нашли применение в системах планирования баз данных, а так же на уровне аппаратных решений RAID-массивов.

**Размещение соответственно частоте использования.** Если в системах используется несколько типов запоминающих устройств или в системе предусмотрены специальные методы доступа, то наиболее часто используемые данные можно хранить на более быстрых устройствах или в файлах с «быстрым» методом доступа.

Аналогичный принцип используется при «кэшировании», когда наиболее часто используемые записи помещаются в промежуточную память с быстрым доступом, обеспечивающимся в основном программными средствами за счет упорядочения размещения и введения избыточности.

#### *4.1.3. Способы адресации и методы доступа к записям*

Записи логического файла идентифицируются с помощью уникальной последовательности символов или некоторого числа — *ключа*. Таким ключом обычно является значение поля, расположенное в каждой записи в одной и той же позиции. Иногда бывает необходимо объединить несколько полей, чтобы обеспечить уникальность ключа, который в этом случае называется *сцепленным ключом*.

В некоторых файлах записи имеют несколько ключей. Запись ЗАКУПКА может иметь различные НОМЕР ПОСТАВЩИКА и НОМЕР ПОКУПАТЕЛЯ, каждый из которых является ключом.

Во многих приложениях требуется идентифицировать записи по ключам, которые не являются уникальными. Однако при этом все равно должен существовать *один* уникальный ключ, тот, который используется для размещения записи в файле и выборки ее из файла. Такой ключ называется *первичным ключом* или *идентификатором*.

Основную проблему при адресации файла можно сформулировать следующим образом: *как по первичному ключу определить местоположение записи с данным ключом?* и *как надо организовать набор записей, чтобы поиск потребовал как можно меньше затрат?*

При разработке схем адресации файлов и определяемого ими размещения записей в файлах большое значение имеет вопрос о том, как включаются в файл новые записи и удаляются старые.

Существует несколько различных способов адресации и поиска записей, например, на основе упорядочения, различных индексов, преобразования «ключ-адрес». Приведем обзор следующих способов, количественная оценка эффективности которых представлена в [Мартин].

**Последовательное сканирование файла.** Наиболее простым способом локализации записи является сканирование файла с проверкой ключа каждой записи. Этот способ, однако, требует слишком много времени и может применяться, когда каждая запись все равно должна быть прочитана.

**Блочный поиск.** Если записи упорядочены по ключу, то при сканировании файла не требуется чтение каждой записи. ЭВМ могла бы, например, просматривать каждую сотую запись в последовательности возрастания ключей. При нахождении записи с ключом большим, чем искомое значение, просматриваются последние 99 записей, которые были пропущены.

Этот способ называется *блочным поиском*. Записи группируются в блоки и каждый блок проверяется по одному разу до тех пор, пока не будет найден нужный блок. Иногда данный способ называют *поиском с пропусками*.

**Двоичный поиск.** При двоичном поиске в файле записей, упорядоченных по ключу, анализируется запись, находящаяся в середине поисковой области файла (изначально всего файла), а ее ключ сравнивается с поисковым ключом. Затем поисковая область делится пополам, и процесс повторяется для соответствующей половины области, пока не будет обнаружено искомое значение или длина области не станет равной 1. Число сравнений в этом случае будет меньше, чем для случая блочного поиска.

Двоичный поиск эффективен для поиска в файлах организованных в виде двоичного дерева с указателями, когда поиск происходит в направлении, задаваемом указателями. Кроме того, добавление в файл новых записей не приводит к сдвигу других записей, что требует много времени и является достаточно сложной процедурой.

Т.е., двоичный поиск более пригоден для поиска в индексе файла, чем в самом файле.

**Индексно-последовательные файлы.** Если файл упорядочен по ключам, то для адресации может использоваться таблица, называемая *индексом*, связывающая ключ хранимой записи с ее относительным или абсолютным адресом во внешней памяти.



Индекс можно определить как таблицу, с которой связана процедура, воспринимающая на входе информацию о некоторых значениях атрибутов и выдающая на выходе информацию, способствующую быстрой локализации записи или записей, которые имеют заданные значения атрибутов.

Если записи файла упорядочены по ключу, индекс обычно содержит не ссылки на каждую запись, а ссылки на блоки записей, внутри которых можно выполнять поиск или сканирование. Хранение ссылок на блоки записей, а не на отдельные записи в значительной степени уменьшает размер индекса. Причем даже в этом случае индекс часто оказывается слишком большим для поиска и поэтому используется индекс индекса.

**Индексно-произвольные файлы.** Произвольный (не упорядоченный по ключу) файл можно индексировать точно так же, как и последовательный файл. Однако при этом индекс должен содержать по одному элементу для каждой записи файла, а не для блока записей. Более того, в нем должны содержаться *полные* абсолютные (или относительные) адреса, в то время как в индексе последовательного файла адреса могут содержаться в усеченном виде, так как старшие знаки последовательных адресов будут совпадать.

Произвольные файлы в основном используются для обеспечения возможности адресации записей файла с несколькими ключами. Если файл упорядочен по одному ключу, то он не упорядочен по другому ключу. Для каждого типа ключей может существовать свой индекс: для упорядоченных ключей индекс будет иметь по одному элементу на блок записей, для других ключей индексы будут более длинными, так как должны будут содержать по одному элементу для каждой записи. Ключ, который чаще всего используется при адресации файла, обычно служит для его упорядочения.

В индексно-произвольных файлах часто используются символические, а не абсолютные адреса, так как при добавлении новых или удалении старых записей изменяется местоположение записей. Если в записях имеется несколько ключей, то индекс вторичного ключа может содержать в качестве выхода первичный ключ записи. При определении же местоположения записи по ее первичному ключу можно использовать какой-нибудь другой способ адресации. По этому методу поиск осуществляется медленнее, чем поиск, при котором физический адрес записи определяется по индексу. В файлах, в которых положение записей часто изменяется, символическая адресация может оказаться предпочтительнее.

**Адресация с помощью ключей, преобразуемых в адрес.** Известно много методов преобразования ключа непосредственно в значение адреса в файле. В тех случаях, когда возможно преобразование зна-

чения ключа непосредственно в значение адреса в файле, такой способ адресации обеспечивает самый быстрый доступ; при этом нет необходимости организовывать поиск внутри файла или выполнять операции с индексами.

В некоторых приложениях адрес может быть вычислен на основе значений некоторых элементов данных записи.

К недостаткам данного способа относится малое заполнение файла: в файле остаются свободные участки, поскольку ключи преобразуются не в непрерывное множество адресов.

Другим недостатком схем прямой адресации является их малая гибкость. Машинные адреса записей могут измениться при обновлении файла. Для устранения этого недостатка прямую адресацию обычно выполняют в два этапа. Сначала ключ преобразуется в *порядковый номер*, который затем преобразуется в машинный адрес.

**Хэширование.** Простым и полезным способом вычисления адреса является хэширование (*перемешивание*). В данном методе ключ преобразуется в квазислучайное число, которое используется для определения местоположения записи.

Более экономичным является указание на область, в которой размещается группа записей. Эта область называется *участком записей* (slot, bucket).

При первоначальной загрузке файла адрес, по которому должна быть размещена запись, определяется следующим образом:

1. Ключ записи преобразуется в квазислучайное число, находящееся в диапазоне от 1 до числа участков, используемых для размещения записей.

2. Число преобразуется в адрес участка и, если на участке есть свободное место, то логическая запись размещается на нем.

3. Если участок заполнен, запись должна быть размещена на *участке переполнения* - следующий по порядку участок либо участок в отдельной области переполнения.

При чтении записей из файла их поиск выполняется аналогично, причем может оказаться, что для поиска записи потребуется чтение нескольких участков переполнения.

Из-за вероятностной природы алгоритма в этом способе не удастся достичь 100% плотности заполнения памяти, однако для большинства файлов может быть достигнута плотность 80 или 90%; при этом память для индексов не требуется. Большинство записей можно найти за одно обращение, но для некоторых потребуется второе обращение (при переполнении). Для очень небольшой части записей потребуется третье или четвертое обращение к файлу.

Кроме того, в этом случае менее эффективно используется память, чем в индексных методах; записи не упорядочены для последовательной обработки.

**Комбинации способов адресации.** При адресации записей некоторых файлов используются комбинации перечисленных выше способов. Например, с помощью индекса может определяться ограниченная поисковая область файла, затем эта область просматривается последовательно либо в ней выполняется двоичный поиск. С помощью алгоритма прямой адресации может определяться нужный раздел индекса, и, таким образом, исчезает необходимость поиска во всем индексе.

#### *4.1.4. Схемы организации данных на внешних носителях*

Схема адресации записей в файле является определяющей для способов размещения записей в файлах, т.е. условий и процедур включения в файл новых записей, обновления и удаления старых.

Записи располагаются на внешнем запоминающем устройстве в конкретной физической последовательности. Обработка данных усложняется, если последовательность записей файла не соответствует последовательности их обработки: возникает необходимость сортировки записей, что требует значительных временных затрат. Как отмечалось ранее, другой способ - это организации доступа к записям в нужной последовательности, отличной от порядка физического размещения - использование различных систем адресации.

Для иллюстрации взаимосвязи схем адресации и организации наборов данных рассмотрим процедуру ведения массива индексно-последовательной организации

При индексно-последовательной организации записи физически размещаются последовательно (или произвольно для индексно-произвольной организации) в соответствии с возрастанием их ключей, которые чаще всего используются для адресации этих записей.

Для работы с индексно-последовательным файлом можно использовать два режима обработки: 1) *последовательную обработку*, при которой записи обрабатываются в последовательности их размещения на внешнем запоминающем устройстве, и 2) *произвольную обработку*, при которой записи обрабатываются в произвольной последовательности, не связанной с физической организацией записей на внешнем устройстве.

На рис. 4.4 приведена схема индексно-последовательного файла, в который добавлены 3 новые записи.

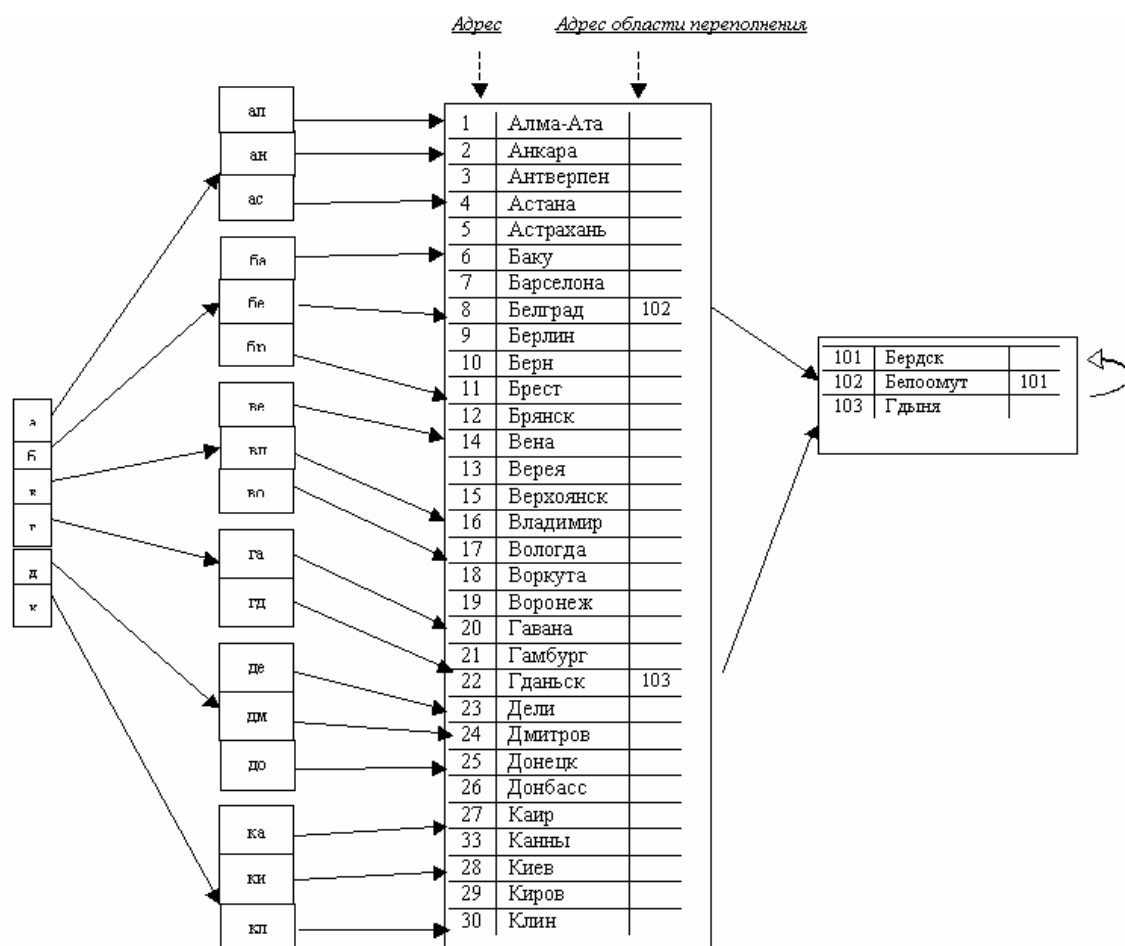


Рис. 4.4. Схема индексно-последовательного файла после добавления записей

Новые записи просто включаются в конец файла, и при этом не требуются указатели на область переполнения и выполнение специальных программ поддержки включения записей. Однако в данном случае возникает необходимость перегруппировки элементов индекса.

Существует три способа адресации записей в области переполнения. В первом методе применяются указатели, расположенные в индексах и указывающие на записи, содержащиеся в области переполнения. Кроме этого, в самой области переполнения используются указатели, связывающие записи в цепочки в порядке возрастания ключа.

Второй способ адресации отличается от первого лишь тем, что указатели на область переполнения создаются не для цепочек записей, а для каждой записи переполнения. В этом случае в индексе резервируется место для нескольких указателей. При этом существенно усложняется ведение индексов, а также увеличивается объем памяти для индексов (в связи с увеличением числа указателей).

Третий способ также позволяет избежать поиска записей по цепочкам благодаря созданию специального индекса независимой области переполнения. В этом случае для поиска записи, находящейся в области переполнения, необходимо прочитать индекс независимой области пе-



реполнения и считывать соответствующую запись. Данный способ требует больших затрат времени по сравнению с первым способом в том случае, когда записи переполнения не связаны в цепочки; но при многократном включении групп записей опасность возникновения очень больших цепочек здесь отсутствует.

### **Методы включения записей, основанные на резервировании**

Метод, основанный на резервировании участков пространства (например фиксированной части каждого блока) в файле для ожидаемого включения новых элементов данных, называется *методом распределенной свободной памяти*. Хотя, применяя данный метод, можно избежать записей переполнения, целесообразно периодически выполнять процедуры восстановления заполненных резервных позиций.

Наличие некоторого объема свободной памяти в каждом управляемом интервале приводит к тому, что большая часть вновь поступающих записей уместается в пределах соответствующих интервалов. Тем не менее, неизбежны случаи нехватки распределенной свободной памяти в интервалах для включения новых записей. В таких случаях осуществляется «*расщепление*» интервала. Предположим, что необходимо включить запись с некоторым значением ключевого поля. В соответствии со значением ключевого поля определяется интервал, в который следует включить запись. Но, так как интервал уже полностью заполнен, поэтому осуществляется его расщепление: половина его записей пересылается в свободный интервал, входящий в состав той же управляемой области.

Резюмируя, перечислим способы включения в файл новых записей:

1. При включении новых записей файл перезаписывается с размещением записей в соответствующих местах.
2. Записи размещаются в области переполнения, которая находится либо в той же области (файле), что и основная область, либо - в отдельной независимой области (файле). При этом для обеспечения доступа могут использоваться цепочки, указатели из индекса к каждой записи переполнения, отдельные индексы для каждого блока области переполнения.
3. Записи размещаются в распределенной свободной памяти, которая резервируется при создании базы на уровне физических или логических областей в пространстве файла. При переполнении первоначально зарезервированной области производится ее расщепление.

#### 4.2. Физическое представление иерархических структур

Рассмотрим физическое представление древовидных структур на примере обновления дерева с использованием следующих методов:

1. Физически последовательное размещение.
2. Указатели.
3. Цепи и кольца.

На рис. 4.5 и рис. 4.6 представлен пример иерархической структуры до и после обновления.

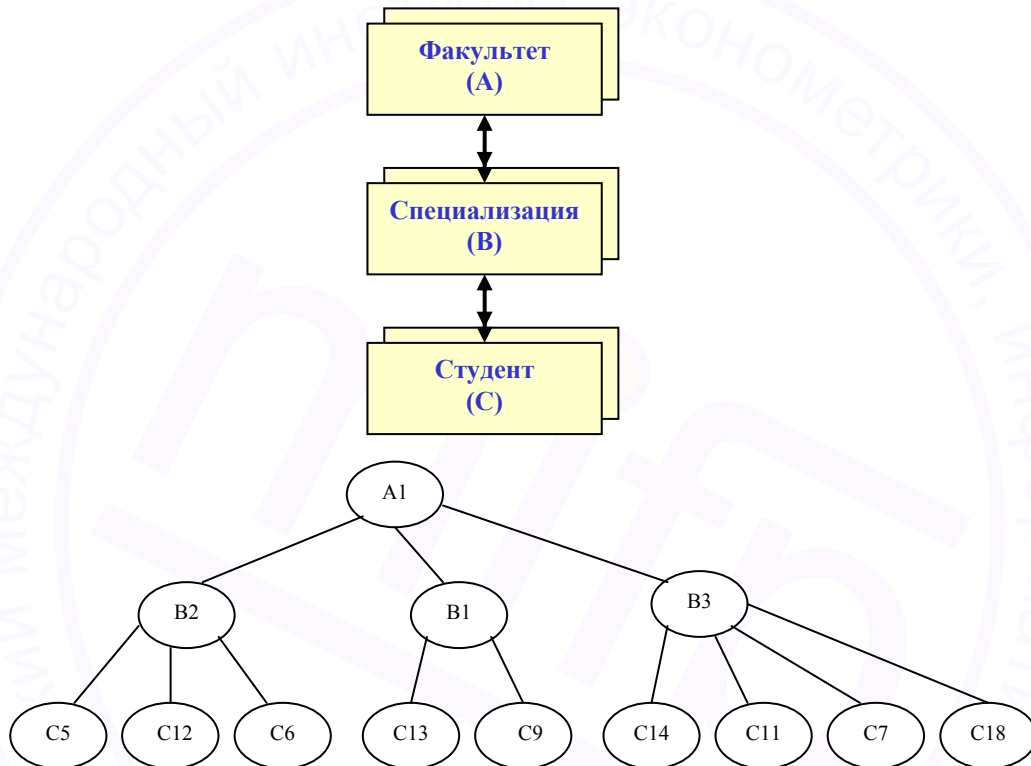
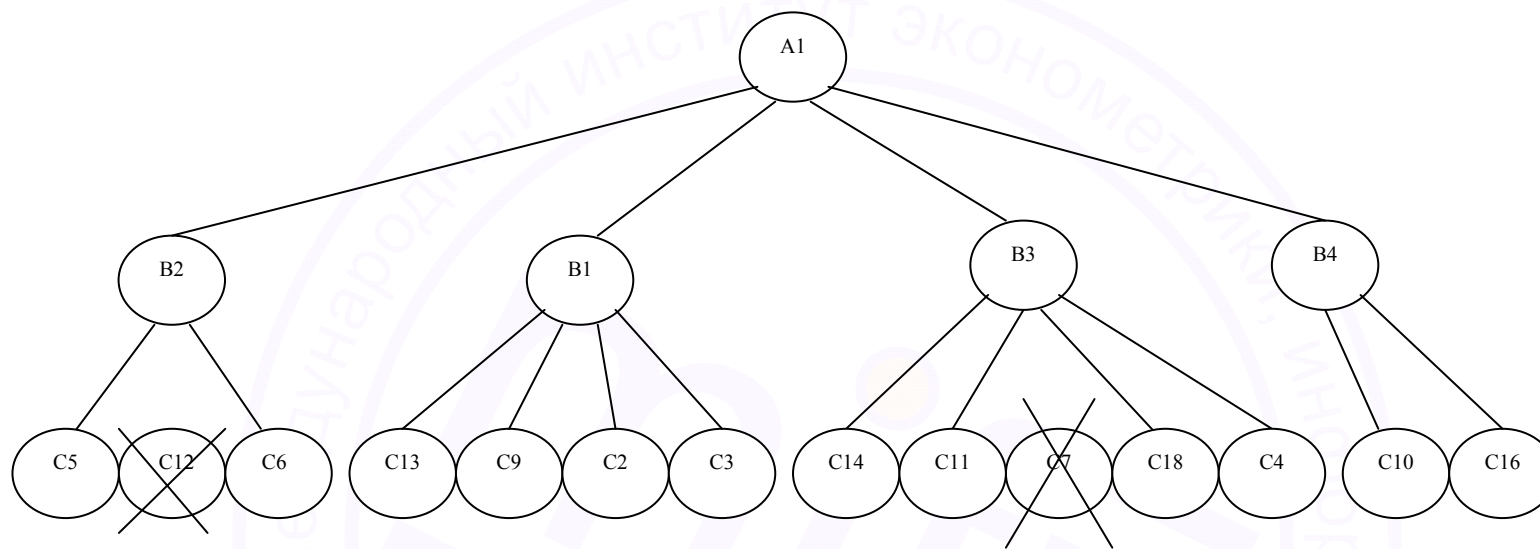


Рис. 4.5. Пример древовидной структуры



*Рис. 4.6. Пример древовидной структуры после обновления*

Записи, относящиеся к разным уровням дерева, обычно рассматриваются как главные и детальные. Поэтому при реализации такого файла для любой пары уровней дерева есть возможность выбора вариантов включения сегментов нижнего уровня в сегменты верхнего уровня. Хотя, исходя из стремления к однородности массивов, обычно все сегменты нижнего уровня размещаются отдельно от сегментов верхнего уровня.

#### 4.2.1. Физически последовательное размещение

На рис. 4.7 и рис. 4.8 представлен пример реализации иерархической структуры до и после обновления путем физически последовательного размещения данных на носителе.

A1	B2	C5	C12	C6	B1	C13	C9	B3	C14	C11	C7	C18	
----	----	----	-----	----	----	-----	----	----	-----	-----	----	-----	--

Рис. 4.7. Пример реализации древовидной структуры до обновления

A1	B2	C5	C6	B1	C13	C9	C2	C3	B3	C14	C11	C18	C4	B4	C10	C16
----	----	----	----	----	-----	----	----	----	----	-----	-----	-----	----	----	-----	-----

Рис. 4.8. Пример реализации древовидной структуры после обновления

Последовательность элементов на рис. 4.7 иногда называется *левосписковой структурой* (последовательность типа «сверху вниз - слева направо»).

Последовательность строится следующим образом: выбираются узлы, начиная от вершины дерева и вниз по самой левой ветви дерева; когда выбран узел самого нижнего уровня этой ветви, выбираются подобные узлы слева направо; процесс повторяется, причем уже выбранные узлы пропускаются.

При размещении каждой записи последовательности в памяти может указываться, к какому уровню дерева она относится. Это выполняется путем введения в каждую запись специального кода (например, тип записи может быть определен по типу ключа). Возможно также использование некоторой формы разграничения последовательности записей, например представление записи в таком виде:

A1(B2(C5C12C6)B1(C12C9)B3(C14C11C7C18))

Последовательные левосписковые структуры не позволяют осуществлять быстрый выбор элементов нижних уровней дерева, так как при этом требуется просмотр всего списка.

#### 4.2.2. Левосписковые структуры с переполнениями

Включение и удаление элементов могут быть выполнены с помощью метода переполнения или метода распределенной свободной памяти.



ти, рассмотренные ранее на примере метода ведения файлов с индексно-последовательной организацией данных.

На рис. 4.9 и рис. 4.10 представлен пример реализации иерархической структуры до и после обновления путем использования области переполнения.

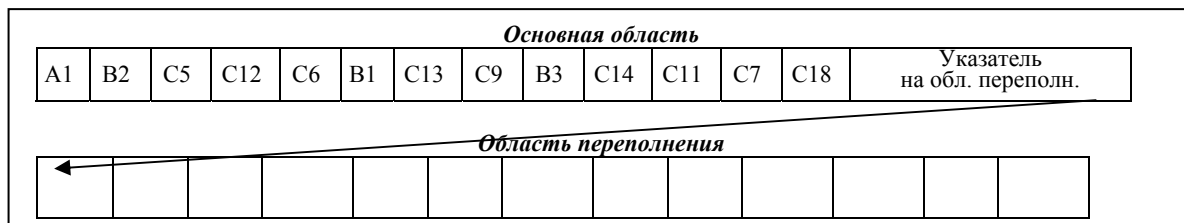


Рис. 4.9. Пример реализации древовидной структуры методом переполнения до обновления



Рис. 4.10. Пример реализации древовидной структуры методом переполнения после обновления

В этом случае для определения местонахождения записей А, В или С можно использовать индексы.

#### 4.2.3. Использование указателей на «подобные» и «порожденные»

Для обеспечения эффективных процедур выборки записей могут использоваться межзаписные ссылки следующих типов:

- указатели на порожденные записи;
- указатели на подобные записи;
- указатели на исходные записи.

При построении древовидных структур, в которых используется какой либо один тип указателя, всегда исходят из альтернативы между сложностью реализации списка указателей переменной длины на порожденные записи и увеличением времени поиска, связанным с использованием цепочки указателей на подобные записи.

Практически эффективные компромиссы могут быть достигнуты путем использования в каждой записи двух указателей каких-либо двух типов (рис. 4.11), а также использованием кольцевых структур (например, рис. 4.12).

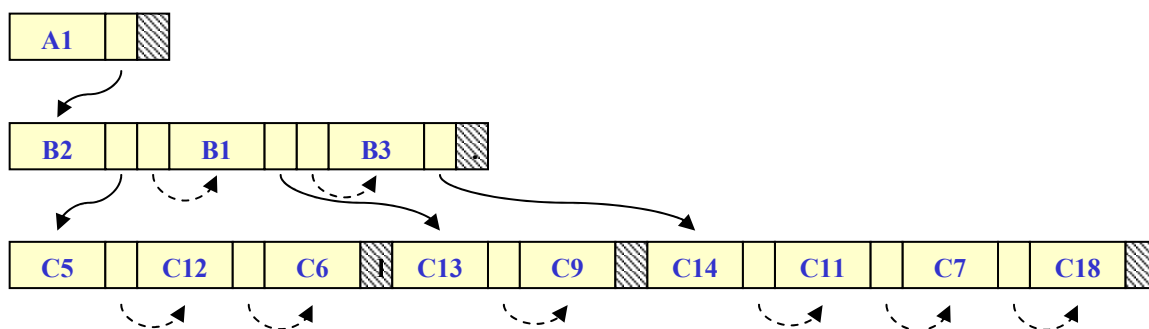


Рис. 4.11. Пример реализации древовидной структуры (рис. 4.5.) с использованием ссылок «порожденный-подобный» (штрихованные области означают конец списка)

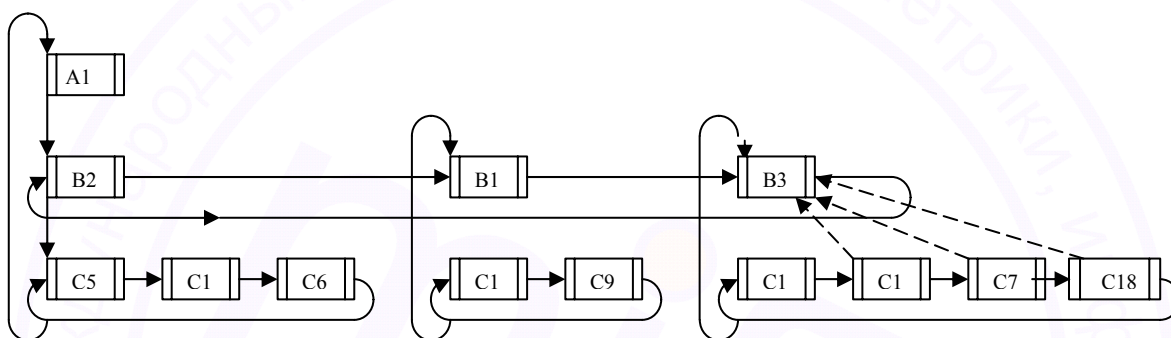


Рис. 4.12. Пример реализации древовидной структуры с использованием кольцевых ссылок

На рис. 4.12 ссылки образуют кольца двух типов: подобных записей и кольца «исходный—порожденный». В записях самого нижнего уровня показаны указатели на исходные записи. Для единообразия здесь каждая запись имеет два указателя. Однако кольца большей частью создаются двусторонними. В этом случае число указателей в каждой записи увеличится до четырех.

### 4.3. Физическое представление сетевых структур

Так же как и в случае древовидных структур, рассмотренных в предыдущей главе, связи в сетевых структурах можно представить, используя физически последовательное размещение, указатели, кольца. Рассмотрим простую сетевую структуру, представленную на рис. 4.13.

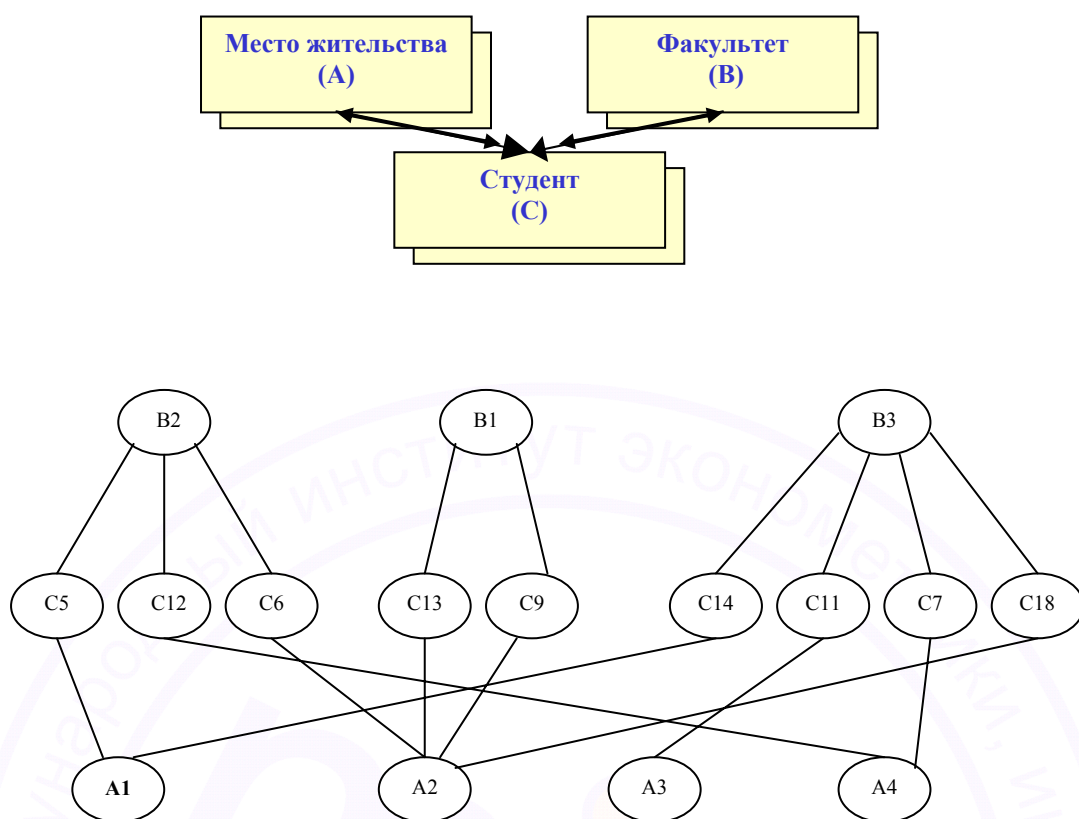


Рис. 4.13. Пример простой сетевой структуры

#### 4.3.1. Физически последовательное размещение

Если древовидные структуры можно представить без избыточности с помощью физически последовательного размещения, то для сетевых структур это обычно невозможно. Однако в некоторых случаях может оказаться удобным представить один набор связей типа «исходный-порожденный» путем физически последовательного размещения, а для остальных связей использовать другой метод. Например, можно использовать физически последовательное размещение для представления связей А и С (рис. 4.14).

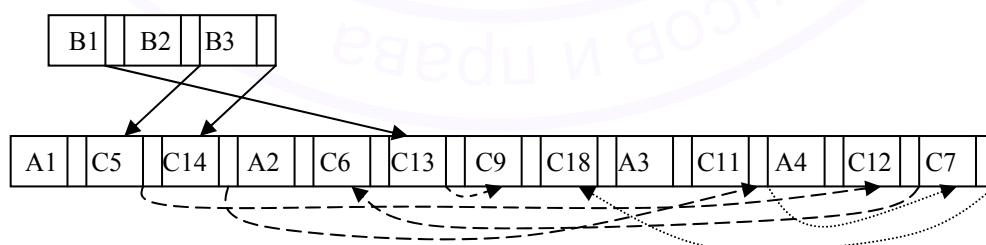


Рис. 4.14. Пример реализации сетевой структуры с последовательным размещением

В этом примере связи между В и С реализуются с помощью множественных указателей на порожденные записи, указателей на исходные записи и указателей на порожденные и подобные записи. Для множественных указателей на порожденные записи требуются списки указателей переменной длины; для указателей на порожденные и подобные записи необходимы длинные цепочки.

Обычно для представления сетевых структур физически последовательное размещение не применяется.

#### 4.3.2. Использование указателей

Если для реализации сетевых структур используются указатели, то они должны представлять все связи, причем какие-то записи должны называться исходными (например, верхние), а какие-то — порожденными (нижние записи).

На практике может использоваться много различных вариантов конфигураций указателей. На рис. 4.15 показаны кольцевые структуры, в которых имеются указатели на исходные, порожденные и подобные записи.

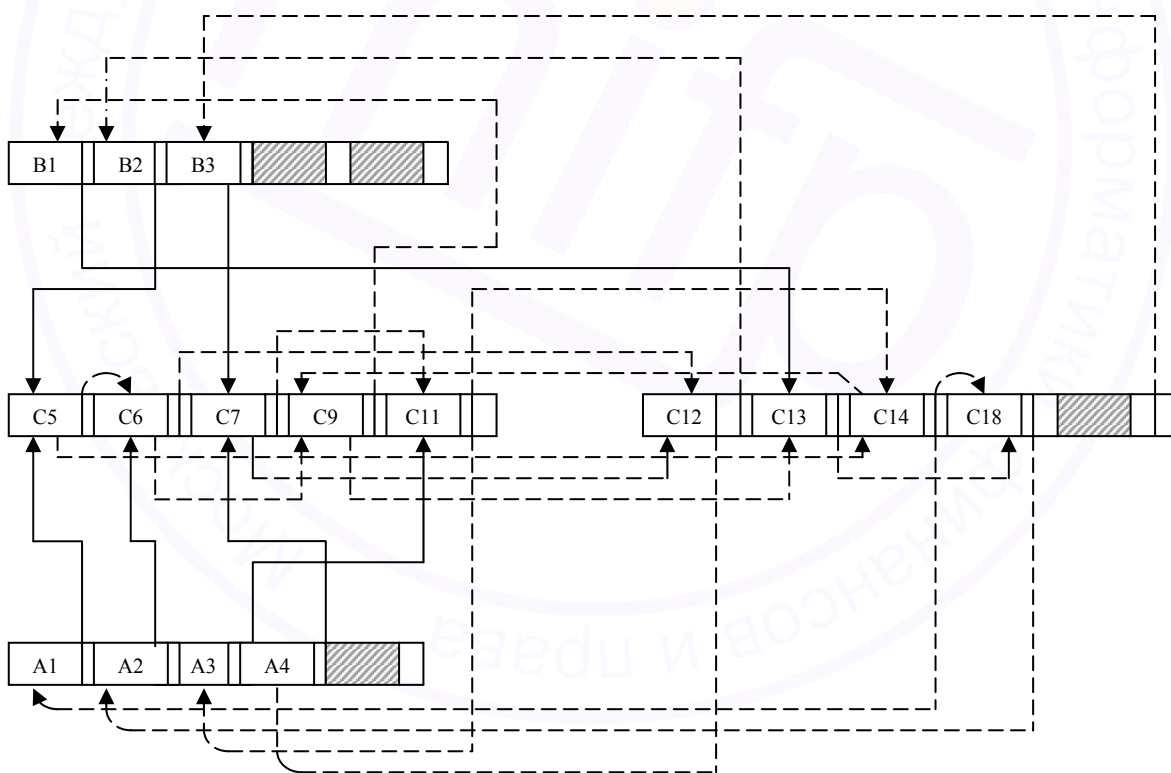


Рис. 4.15. Пример реализации сетевой структуры с использованием указателей

Однако если какая-нибудь связь между записями относится к типу «многие ко многим», то названные три метода физического представления сетевых структур оказываются непригодными. Более того, если в



простых сетевых структурах на предыдущих рисунках для хранения *указателей на исходные записи* требовался один или два указателя в каждой записи, то здесь необходимы списки указателей переменной длины.

Основной проблемой, возникающей при организации встроенных списков указателей переменной длины, является сложность их ведения. При обновлении файла должна существовать возможность сокращения и удлинения списков, что обычно приводит к необходимости периодической реорганизации. Реорганизация является сложной задачей, поскольку при перемещении записей должны быть изменены многие указатели.

Эта проблема частично решается, если использовать *символические указатели*, которые не изменяются при перемещении записей. Однако их применение отражается на механизме адресации и при поиске записей в файле: система затрачивает на поиск записей больше времени, чем при использовании прямых указателей.

#### 4.3.4. Физическое представление с разделением данных и связей

Рассматриваемые ранее структуры в основном ориентированы на то, чтобы связи между данными хранились вместе с самими данными. Такое объединение реализовалось, например, агрегированием данных (построением сложных понятийных структур и данных) или введением ссылочного аппарата, фиксирующего семантические связи, непосредственно в записи данных.

Табличная форма представления информации является наиболее распространенной и понятной. Кроме того, такие семантически более сложные формы, как деревья и сети, путем введения некоторой избыточности могут быть сведены к табличным. При этом связи между данными также будут представлены в форме двумерных таблиц.

Такой *реляционный* подход, в основе которого лежит принцип разделения данных и связей, обеспечивает с одной стороны независимость данных, а с другой – более простые способы реализации хранения и обновления.

На рис. 4.16 а,б приведен пример разделения линейных записей исходной таблицы «Штатное расписание факультета» (рис. 4.16) на связи и собственно данные.

Фамилия И.О.	Год Рожд.	Должность	Кафедра №
Иванов И.И.	1948	Зав. каф.	22
Сидоров С.С.	1953	Проф.	22
Гиацинтова Г.Г.	1945	Проф.	22
Цветкова С.С.	1960	Доцент	22
Козлов К.К.	1959	Доцент	22
Петров П.П.	1960	Ст.преп.	22
Лютикова Л.Л.	1977	Ассистент	22
Рыбин Р.Р.	1950	Зав. каф.	23
Китов К.К.	1944	Проф.	23
Раков В.В.	1958	Доцент	23
Соловьева С.С.	1958	Доцент	23
Воробьева В.В.	1959	Ст.преп.	23
Орлова О.О.	1966	Ассистент	23
Осетров С.С.	1976	Ассистент	23

Рис. 4.16. Пример набора записей табличного типа

Данные			
1	Воробьева В.В.	1	1944
2	Гиацинтова Г.Г.	2	1945
3	Иванов И.И.	3	1948
4	Китов К.К.	4	1950
5	Козлов К.К.	5	1953
6	Лютикова Л.Л.	6	1958
7	Орлова О.О.	7	1959
8	Осетров С.С.	8	1960
9	Петров П.П.	9	1966
10	Раков В.В.	10	1976
11	Рыбин Р.Р.	11	1977
12	Сидоров С.С.	1	Ассистент
13	Соловьева С.С.	2	Доцент
14	Цветкова С.С.	3	Зав. каф.
		4	Проф.
		5	Ст. преп.
		1	22
		2	23

Рис. 4.16а. Пример разделения на связи и данные набора записей табличного типа (данные)

Связи

Ф.И.О	Год Рожд.	Должн	Каф №
3	3	3.	1
12	5	4	1
2	2	4	1
14	8	2	1
5	7	2	1
9	8	2.	1
6	11	1	1
11	4	3	2
4	1	4	2
10	6	2	2
13	6	2	2
1	7	5	2
7	9	1	2
8	10	1	2

Должн	Ф.И.О.
1	6
1	7
1	8
2	14
2	5
2	10
2	13
3	3
3	11
4	12
4	2
4	4
5	9
5	1

Год рожд	Ф.И.О.
1	4
2	2
3	3
4	11
5	12
6	10
6	13
7	5
7	1
8	14
8	9
9	7
10	8
11	6

Кафедра	Ф.И.О.
1	3
1	12
1	2
1	14
1	5
1	9
1	6
2	11
2	4
2	10
2	13
2	1
2	7
2	8

Рис. 4.166. Пример разделения на связи и данные набора записей табличного типа (связи)

В разделенном варианте получены три таблицы бинарных отношений для трех вторичных ключей и одна таблица отношений в не инвертированной форме, но упорядоченная по первичному ключу. Каждое значение элемента данных представлено в одном экземпляре и имеет идентификатор (порядковый номер - ключ). Связи элементов данных также выделены в таблицы отдельно.

Такое представление обладает следующими важными свойствами:

- каждый элемент таблицы – это *один* элемент данных;
- таблица не содержит *одинаковых* строк, т.е. содержащих попарно равных значений элементов данных;
- столбцы таблицы однородны (т.к. элементы данных каждого столбца имеют общую природу) и могут быть однозначно идентифицированы *именованием*.

Для более сложных случаев, например, древовидных структур, для устранения зависимости от путей вводятся дополнительные ключевые элементы данных.

Следует отметить, что дублирование некоторых элементов в таблицах является *логическим* и не обязательно повлечет дублирование на физическом уровне, так как можно воспользоваться указателями.

Однородность реляционных баз данных, построенных на основе бинарных отношений, обеспечивает:

- унифицированность средств работы с базой: необходимы только средства для работы с бинарными таблицами;
- простоту расширения состава логической записи.

В тоже время для получения ответа по комплексному запросу необходимо обращаться к нескольким таблицам.

#### ***4.4. Архитектура файловой организации баз данных***

Файловая структура и система управления файлами являются прерогативой операционной среды, поэтому по отношению к базам данных, ориентированным на работу с элементами данных и высокую интенсивность обмена, эффективность операций ввода-вывода будет не оптимальна: стандартный язык СУБД намного богаче, чем набор операций файловой системы.

Прямое использование файловой системы для организации хранения и доступа к данным оказывается менее эффективным (отличие составляет, по крайней мере, 10%), поскольку при выполнении каждого обращения к диску со стороны СУБД в работу включается дополнительный слой системного ПО. Хранение данных в файловой системе приводит также к определенной потере емкости памяти. Файловая система, например Unix, потребляет примерно 10% от форматированной емкости дисков для метаданных о файлах и файловой системе. Более того, файловая система резервирует некоторое пространство, чтобы обеспечить быстрый поиск свободного пространства в случае расширения файлов.

Это послужило причиной того, что СУБД берут на себя непосредственное управление внешней памятью, минимально используя файловую систему ОС.

##### ***4.4.1. Файл-ориентированная организация данных***

Этот подход отражает точку зрения «идейно чистого» программирования, выражающуюся в стремлении к построению модульных процедур, ориентированных на обработку регулярных однородных данных<sup>29</sup>: «сколько типов структур записей - столько и файлов».

Таким образом, БД физически состоит из нескольких файлов: основного, индексного, файла метаданных, файлов указателей и т.д. Такого типа организация файловой структуры БД представлена в приложении примерами организации данных dBase-подобных и документальных баз данных.

---

<sup>29</sup> Именно такой подход обеспечил возможность реализации надежных достаточно эффективных СУБД, функционирующих по современным меркам в крайне скромных рамках наличных вычислительных ресурсов.



#### 4.4.2. Страничная организация данных

Другой подход отражает стремление разработчиков сосредоточить в СУБД управление данными на всех уровнях – от логической обработки до управления пространством носителя. Создание сложных специализированных процедур эффективно работающих со сложными нерегулярными структурами данных в сочетании с огромными ресурсами вычислительной мощности и оперативной памяти позволили реализовать даже однофайловую<sup>30</sup> физическую структуру СУБД.

Приведем примерную логическую схему «страничной» организации хранения данных.

При распределении дискового пространства рассматриваются следующая схема структуризации пространства в зависимости от типов данных:

*Экстент* — это непрерывная область дисковой памяти, включающая несколько страниц фиксированной длины. Новый экстент создается после заполнения предыдущего и связывается с ним ссылкой, которая располагается на последней странице экстента, либо в специальной карте размещения. Учет свободных страниц ведется внутри экстента.

Каждый экстент используется для хранения одного из нескольких типов страниц: страницы данных, страницы индексов, страницы blob-объектов<sup>31</sup> (неструктурированных данных, например большие текстовые или двоичные данные). Т.е., данные на одной странице являются однородными: страница, например, может хранить только данные или только индексы.

Основной логической единицей операций обмена (ввода-вывода) является *страница данных*, хранящая данные в виде *строк* или других специализированных структур.

Все страницы данных имеют одинаковую структуру, включающую:

- *Заголовок страницы*, содержащий номер страницы, номера предыдущей и следующей страниц, сведения о свободном пространстве на странице;
- *Содержание* – строки данных (последовательность кодов), каждая из которых имеет уникальный идентификатор в рамках всей базы данных, который состоит из номера страницы и номера строки на странице;
- *Дескрипторы строк*, задающие смещение строки на странице и длину строки, что позволяет при переупорядочении строк на страницах не производить физического перемещения строк, т.к. все манипуляции производятся с дескрипторами.

---

<sup>30</sup> Например, MS ACCESS.

<sup>31</sup> Для СУБД важно знать, что этот объект надо хранить целиком и что размеры этих объектов от записи к записи могут меняться, а в общем случае размер неограничен.

Для организации быстрого доступа создаются *страницы индексов*, которые организованы обычно в виде В-деревьев.

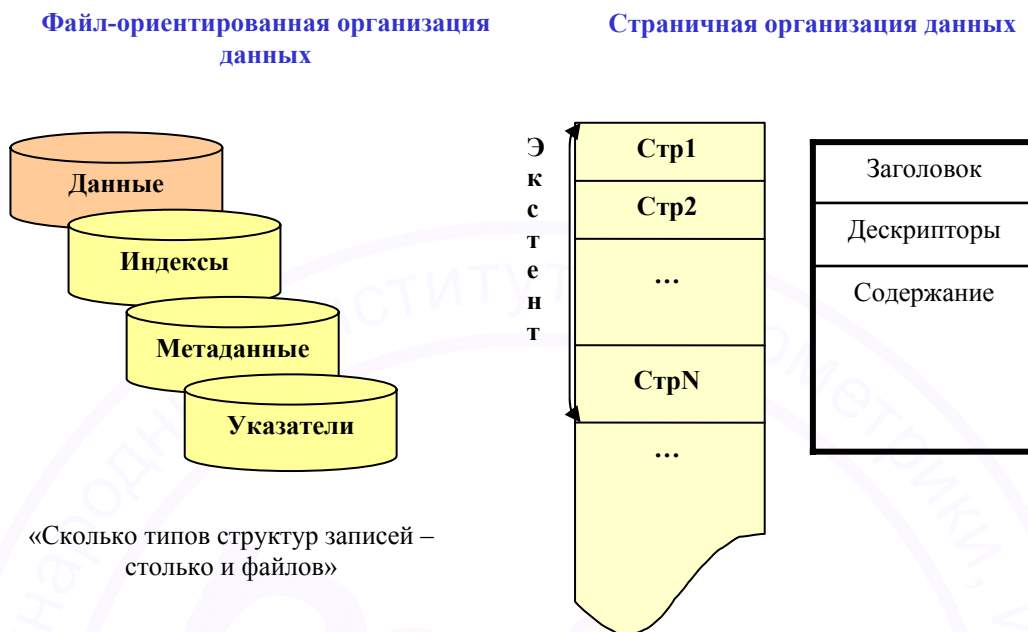


Рис. 4.17. Файл-ориентированная и страничная организация данных

#### 4.5. Модели распределения данных по физическим носителям

Важным фактором, влияющим на производительность подсистемы ввода-вывода, является распределение данных по дискам. Даже минимальная по объему высокопроизводительная система должна иметь по крайней мере четыре диска: один для операционной системы и области подкачки (swap), один для данных, один для журнала и один для индексов.

Размещение всех данных БД на одном и том же диске почти всегда приводит к неудовлетворительной производительности. В частности, может оказаться, что процесс формирования журнала, который должен записываться синхронно, в действительности будет выполняться в режиме произвольного, а не последовательного доступа к диску. Уже только эта операция будет существенно задерживать каждую транзакцию обновления базы данных.

Кроме того, выполнение запросов, выбирающих записи из таблицы данных путем последовательного сканирования индекса, будет сильно увеличивать время ожидания ввода-вывода. Обычно сканирование индекса выполняется последовательно, но в данном случае головка диска должна перемещаться для поиска каждой записи данных между выборками индексов. Наконец, следует отметить, что объединение разных функций на одних и тех же физических ресурсах приводит к резкому увеличению времени подвода головок на диске.

Примером, иллюстрирующим подход с точки зрения практических компромиссов выбора решения, являются RAID-массивы. На рис. 4.18 приведены два варианта: RAID-0, обеспечивающий максимальную производительность при «стандартной» надежности, и RAID-1, обеспечивающий «двойную» надежность при «стандартной» производительности.

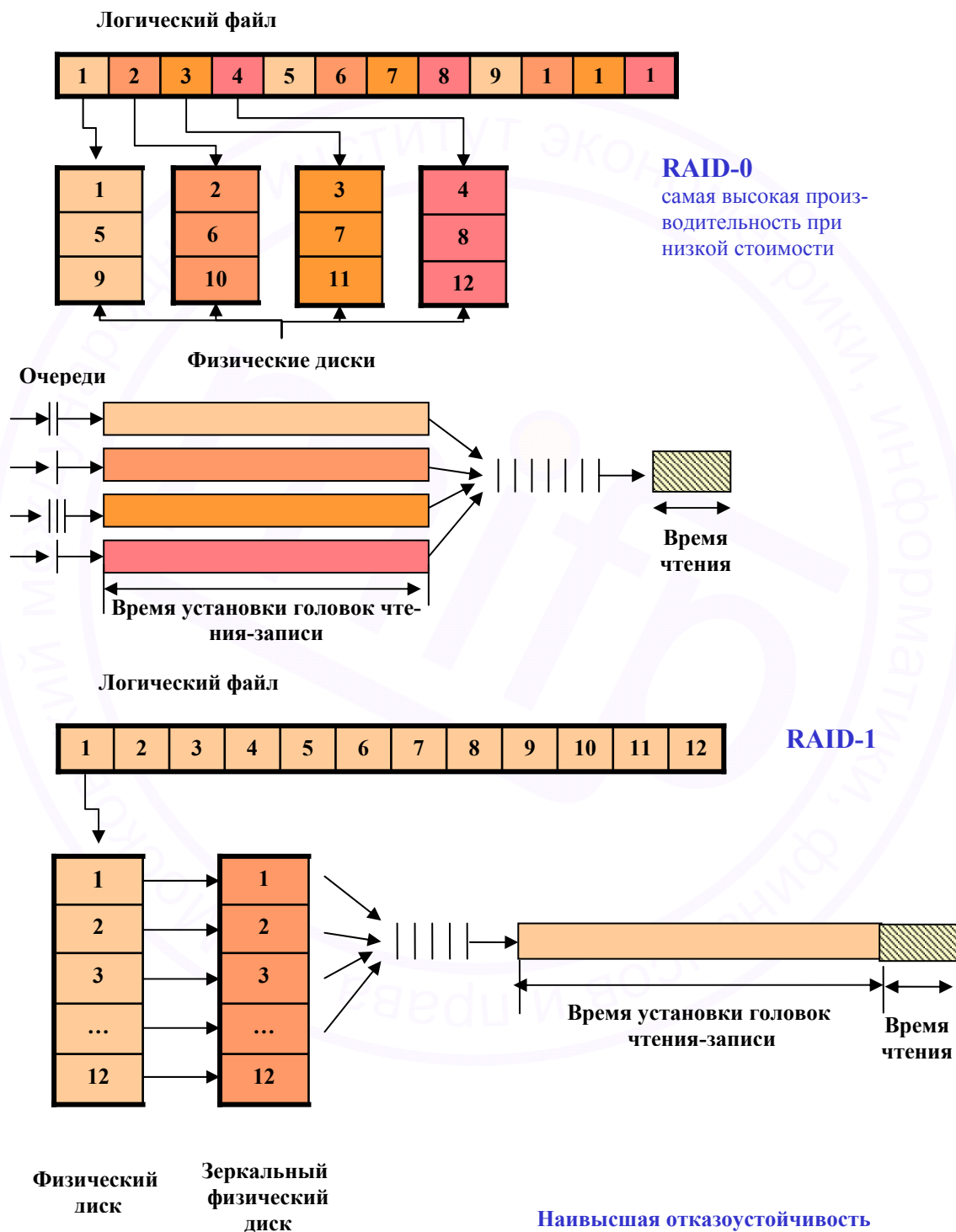


Рис. 4.18. Распределение данных в RAID-массивах

Системы управления базами данных применяют по крайней мере два механизма для распределения данных по дисковым накопителям. Для эффективного распределения доступа к данным многие СУБД имеют возможность осуществлять сцепление нескольких дисковых накопителей или файлов. Если по запросам производится произвольный доступ к данным, например, если пользователи независимо запрашивают разные записи, то возможности сцепления дисков в СУБД полностью обеспечивают распределение нагрузки по доступу к множеству дисков (при достаточно равномерном заполнении пространства базы). Если обращения по своей природе последовательны, в частности если один или несколько пользователей должны просматривать каждую строку таблицы, то больше подходит механизм расщепления дисков. Главное отличие между сцеплением и расщеплением заключается в размещении смежных данных. Когда диски сцепляются друг с другом, последовательное сканирование представляет собой тяжелую нагрузку для каждого из дисков, но эта нагрузка носит последовательный характер (только один диск участвует в обслуживании запроса). Расщепление дисков осуществляет деление данных на меньшие порции, размещаемые на разные диски, позволяя тем самым всем дискам участвовать в обслуживании даже сравнительно небольшого запроса. В результате использование расщепления существенно уменьшает загрузку дисков при выполнении последовательного доступа. Основными кандидатами для расщепления являются обычно архивные и журнальные файлы, поскольку к ним всегда осуществляется последовательный доступ, что может ограничить общую производительность системы.

#### *Контрольные вопросы*

1. Перечислите типы физических записей. Приведите примеры, показывающие соотношение физических и логических записей.
2. Перечислите факторы, влияющие на выбор метода размещения данных (организацию файла).
3. Перечислите методы организации файлов, позволяющих оптимизировать доступ к записям.
4. Приведите пример организации данных в виде индексно-последовательного файла.
5. Какие типы указателей можно использовать для реализации иерархической структуры.
6. Какие типы указателей можно использовать для реализации сетевой структуры с последовательным размещением.
7. Придумайте схему реализации сетевой структуры для случая часто изменяемых данных.
8. Какие методы организации данных, применяемые для реализации иерархических структур, неэффективны для реализации сетевых?
9. Приведите примерную структурную схему «страничной» организации хранения данных.
10. Приведите примерную схему размещения данных с использованием механизма расщепления.

## Глава 5. Модели и этапы проектирования баз данных

### 5.1. Модели многоуровневой архитектуры систем баз данных

В области проектирования и разработки систем баз данных используются различные средства моделирования, причем даже в рамках одной конкретной системы необходим целый комплекс моделей разного назначения.

Опубликованный в 1975 году отчет ANSI/X3/SPARC зафиксировал не только широкое признание концепций многоуровневой архитектуры систем баз данных, но и необходимость явного выделения специального *концептуального* уровня представления базы данных, единого для всех ее приложений и независимого от них. Кроме этого уровня предусматривались еще два уровня: внутренний уровень, который должен обеспечивать поддержку представления хранимой базы данных, и внешний, поддерживающий представления базы данных “с точки зрения” приложений. На каждом архитектурном уровне предполось использование той или иной модели данных. Кроме того, на внешнем (прикладном, пользовательском) уровне таких моделей может быть несколько. Модели, а также схемы, специфицируемые на их основе, называются, соответственно, внешней, концептуальной и внутренней.

Как очевидно конечной целью проектирования является построение конкретной базы данных, в той или иной степени воплощающей представление проектировщика о предметной области и задачах, решаемых пользователями с использованием созданной базы. Рассматривая базу данных как конкретную **реализацию** модели, мы по существу устанавливаем порядок процесса, отделяя этап определения принципов (то, какой база *должна* быть) от этапа воплощения этих принципов при реализации базы данных в конкретной среде СУБД, ОС и языках программирования. И, как показывает практика, между реализациями баз данных и принципами их построения всегда есть расхождения. Различия являются следствием разных причин, но чаще всего - это явный или неявный отказ от некоторых принципиальных ограничений, налагаемых, например, моделью данных или базовыми (встроенными) алгоритмами обработки, в пользу частного решения, которое, по мнению проектировщика, будет более эффективно, например, для понимания или обработки данных.

Важность отделения проектирования на абстрактном уровне от физической реализации состоит в том что, объявляя принципы, мы *конструктивно* ограничиваем область применения. Во-первых, размерность и сложность задачи *должна быть* сокращена до такого уровня, чтобы реализация стала возможной в данных конкретных условиях – ресурсах среды, профессионализме проектировщика, подготовленности пользователя и т.д. Во-вторых, поскольку база данных по определению предна-



значена для *многофункционального* использования *различными* пользователями, и в тоже время - для обслуживания запросов, *не предвиденных* при проектировании, такое явное объявление принципов позволит не вводить в заблуждение пользователя и не создавать приложения для решения задач, которые в силу своего принципиального отличия от тех, которые рассматривались при проектировании, обусловят неэффективную обработку данных<sup>32</sup>.

Проектирование базы данных - это упорядоченный формализованный процесс создания *системы взаимосвязанных описаний*<sup>33</sup>, т.е. таких моделей предметной области, которые связывают (фиксируют) хранимые в базе данные с объектами предметной области, описываемыми этими данными. Прикладное назначение таких описаний состоит в том, чтобы пользователь, практически не имеющий представления об организации данных в БД (физическом размещении в памяти данных и механизмах их поиска), обращая запрос к базе, имел бы практическую возможность получить адекватную информацию о состоянии объекта предметной области.

Проектирование начинается с анализа предметной области и выявления функциональных и других требований к проектируемой системе. Подробнее этот процесс будет рассмотрен ниже, а здесь отметим, что проектирование обычно выполняется человеком (группой людей) – системным аналитиком (а на практике чаще администратором базы данных), которым может быть как специально выделенный сотрудник, так и будущий пользователь базы данных, достаточно хорошо знакомый с машинной обработкой данных.

Объединяя отдельные представления о содержимом базы данных, полученные в результате опроса пользователей, и свои представления о данных, которые могут потребоваться для решения практических задач, системный аналитик сначала создает обобщенное неформальное описание создаваемой базы данных. Это описание, выполненное с использованием естественного языка, математических выражений, таблиц, графов и других средств, понятных всем людям, работающим над проектированием базы данных, называют *инфологической моделью*.

Такая человеко-ориентированная модель практически полностью независима от физических параметров среды хранения данных, которой может быть как память человека, так и ЭВМ. Поэтому инфологическая

---

<sup>32</sup> Применяемые формальные языки представления предметной области не позволяют описывать *все* отношения, которые проектировщик считает важными. С другой стороны, многие проекты (и, в частности, рассматриваемые *примеры*) воспринимаются как достаточно простые, а проектные решения кажутся очевидными. Кроме того, опытный программист всегда может предложить некоторый эмпирический и, возможно, действительно эффективный способ для целевого представления и обработки нужной информации. Однако это означает отказ от единого формализма, что при увеличении количества данных и связей значительно усложняет проблемы управления базой и в частности – понимание пользователем организации и методов доступа.

<sup>33</sup> Такие описания реализуются, например, в виде *схем*.

модель не изменяется до тех пор, пока какие-то изменения в реальном мире (той его части, которая отнесена к предметной области) не потребуют изменения в модели соответствующего фрагмента описания, чтобы эта модель продолжала адекватно отражать предметную область.

Остальные модели являются машинно-ориентированными. С их помощью СУБД дает возможность программам и пользователям осуществлять доступ к хранимым данным лишь по их именам, не заботясь о физическом расположении этих данных.

Так как доступ к данным осуществляется с помощью конкретной СУБД, то модели должны быть представлены на языке описания данных этой СУБД. Такое описание, создаваемое по инфологической модели данных, называют *даталогической моделью* данных.

Для размещения и поиска данных на внешних запоминающих устройствах СУБД использует *физическую модель* данных.

Представленная трехуровневая архитектура (инфологический, даталогический и физический уровни) позволяет обеспечить независимость хранимых данных от использующих их программ. Хранимые данные могут быть переписаны на другие носители или может быть реорганизована их физическая структура, в том числе дополнена полями для новых приложений, но это повлечет лишь изменение физической и, возможно, даталогической модели данных. Главное, такие изменения физической и даталогической моделей не будут замечены существующими пользователями системы (окажутся "прозрачными" для них) так же, как не будут замечены и вновь подключаемые пользователи. Кроме того, независимость данных обеспечивает возможность создания новых приложений для решения новых задач без разрушения существующих.

В процессе развития теории систем баз данных термин "модель данных" имел разное содержание. Для более глубокого понимания существа отдельных понятий рассмотрим некоторые особенности использования этого понятия в контексте эволюции баз данных, представленные в [11].

**О понятии «модель данных».** Первоначально понятие модели данных употреблялось как синоним структуры данных в конкретной базе данных. Структурная трактовка полностью согласовывалась с математическим определением понятия модели как *множества с заданными на нем отношениями*. Но, следует отметить, что объектом моделирования в данном случае являются не данные вообще, а конкретная база данных. Разработки новых архитектурных подходов, основанных на идеях многоуровневой архитектуры СУБД, показали, что уже недостаточно рассматривать отображение представлений конкретной базы данных. Требовалось решение на метауровне, позволяющее оперировать множествами всевозможных допустимых представлений баз данных в рамках заданной СУБД или, что эквивалентно, инструментальными средствами, используемыми для их спецификации. В этой связи возникла потребность в термине, который обозначал бы инструмент, а не результат мо-

делирования, и соответствовал бы, таким образом, множеству всевозможных баз данных некоторого класса. Т.е. инструмент моделирования баз данных должен включать не только средства структурирования данных, но и средства манипулирования данными. Поэтому модель данных в инструментальном смысле стала пониматься как алгебраическая система – множество всевозможных допустимых типов данных, а также определенных на них отношений и операций. Позднее в это понятие стали включать еще и ограничения целостности, которые могут налагаться на данные. В результате проблема отображения данных в многоуровневых СУБД и системах распределенных баз данных стала рассматриваться как проблема отображения моделей данных.

Важно подчеркнуть, что для разработчиков и пользователей СУБД точным определением реализованной в ней модели данных фактически являются языковые средства определения данных и манипулирования данными. Поэтому отождествлять такой язык со схемой базы данных (результатом моделирования) – конкретной спецификацией в этом языке – неправомерно.

Начиная с середины 70-х годов, под влиянием предложенной в тот период концепции *абстрактных типов* само понятие типа данных в языках программирования стало трансформироваться таким образом, что в него стали вкладывать не только структурные свойства, но и элементы поведения (изменения данных). В дальнейшем это послужило основой для формирования концепции объекта, на которой базируются современные объектные модели.

В связи с этим был предложен новый подход, при котором модель данных рассматривается как система типов. Такой подход обеспечивал естественные возможности интеграции баз данных и языков программирования, способствовал формированию направления, связанного с созданием так называемых систем программирования баз данных. Трактовке модели данных как системы типов соответствуют не только уже существующие широко используемые модели, но также объектные модели, завоевывающие все большее влияние.

**О развитии и конкурировании моделей.** По аналогии с ситуацией 70-х годов, когда велись споры о преимуществах сетевой, иерархической и реляционной модели данных (как известно, завершившиеся “ничейным” исходом открытой дискуссии Ч.Бахмана и Э.Кодда на Конференции ACM SIGMOD в 1975 году), в настоящее время сформировалась новая тройка конкурентов – реляционная, объектная и многомерная модели.

Хотя, строго говоря, здесь речь идет лишь о двух моделях. Действительно, многомерные модели, коммерческие реализации которых появились в начале 90-х годов для поддержки технологий OLAP, не основаны на каких-либо радикально новых идеях. Они представляют собой некоторое расширение активно исследовавшейся в 70-80-х годах модели универсальных отношений новыми операционными возможностями,



обеспечивающими, в частности, необходимые для OLAP функции агрегирования данных. Таким образом, многомерные модели представляют собой особую разновидность реляционной модели.

Многомерные модели – это “полноправные” модели данных. Также, как и другие модели, они используются для описания базы данных, определяя тем самым соответствующее ее природе представление данных, и предоставляют средства манипулирования данными. И в этом смысле они ничем не отличаются по своей функциональности от прочих моделей данных. В сегодняшних массовых реляционных технологиях многомерные модели ассоциируются с внешним уровнем архитектуры систем баз данных. Именно этим определяется их направленность на конечного пользователя. Нужно заметить, что обеспечение комфортных условий для работы конечного пользователя было также основной целью разработки модели универсального отношения.

Однако главная проблема в области массовых технологий заключается не столько в том, чтобы найти достойного преемника реляционной модели, сколько в том, что огромный балласт унаследованных систем (сегодня – уже реляционных) не дает возможности для резких технологических сдвигов и радикального обновления существующих технологий. Стремление избежать огромных кратковременных капиталовложений в случае перехода к принципиально новым технологиям приводит к необходимости лишь эволюционного пути развития. Отсюда рождение таких гибридов, как объектно-реляционные базы данных “нового поколения”.

**Документальные системы и интеграция моделей.** Приведенные выше положения разрабатывались и действительно широко используются для баз данных хорошо структурированной информации. Однако уже сегодня одной из важнейших проблем становится обеспечение интеграции неоднородных информационных ресурсов, и в частности слабоструктурированных данных. Необходимость ее решения связывается со стремлением к полноценной интеграции систем баз данных в среду Web-технологий. При этом уже недостаточно простого обеспечения доступа к базе данных традиционным способом “из-под” HTML-форм. Нужна интеграция на модельном уровне. И не просто синтаксическая интеграция. В этом случае проблема семантической интероперабельности информационных ресурсов сводится к задаче разработки средств и технологий, предусматривающих явную спецификацию метаданных для ресурсов слабоструктурированных данных на основе традиционных технологий моделирования из области баз данных. (Напомним, что в области систем баз данных аналогичная ситуация была преодолена благодаря предложениям CODASYL о необходимости явной спецификации схемы базы данных, независимой от приложений).

Именно на достижение этой цели направлены интенсивные разработки WWW-консорциумом языка XML и его инфраструктуры (фактически, новой модели данных для этой среды), объектной модели доку-

ментов и других средств, которые, как можно ожидать, в близкое время станут основой технологий управления информационными ресурсами. Это направление связано с другой глобальной проблемой - организацией распределенных неоднородных информационных систем на основе построения репозитория<sup>34</sup> метаданных, обеспечивающих возможность семантического отождествления ресурсов и, таким образом, возможность их целенаправленного повторного использования.

## ***5.2. Стадии проектирования и объекты моделирования***

Не исключено, что у читателя создалось впечатление, будто мы уже владеем современной методологией или, по крайней мере, близки к этому, что, к сожалению, не так, и, может быть, мы никогда ничего подобного не добьемся. Всегда несложно охарактеризовать методологию на концептуальном уровне, весьма трудно применить ее на практике. Камень преткновения – сложность проникновения в существо предметной области (например, сложности понимания механизма деятельности организации) и адаптации ее к новым, возможно лучшим, условиям функционирования.

Аналогичные проблемы характерны и для СУБД в целом. Система баз данных должна стать органическим элементом системы управления организацией - вот залог ее успешного применения. Однако процесс ее внедрения связан с определенными изменениями в самой организации и в деятельности ее сотрудников, и мы всегда будем сталкиваться с естественной инертностью людей, когда речь идет о восприятии изменений....

Весьма важно, чтобы средства СУБД были адекватны потребностям пользователей. Поскольку разным пользователям могут понадобиться разные модели данных, языки данных и схемы, желательно, чтобы СУБД поддерживала множество средств, а пользователь мог выбирать из них наиболее подходящие. ...

Можно, конечно, поставить под сомнение ценность таких исследований. Действительно, каким бы плохим ни был язык программирования, его, в конце концов, все-таки можно выучить. Точно также и средства СУБД можно освоить за определенный период времени. Но проблема состоит не в освоении средств, а в эффективности их использования!

Следует отметить, что положения цитаты из [20], текст которой выделен выше курсивом, по-прежнему актуальны, хотя книга издана более 20 лет назад. Действительно, средства проектирования непрерывно развиваются, но и задачи, решение которых пользователь предполагает автоматизировать с помощью систем баз данных, существенно усложнились.

---

<sup>34</sup> Этому понятию в классических работах по проектированию баз данных соответствует понятие *словарь данных*.



С точки зрения объектов моделирования необходимо различать модели предметной области и модели базы данных. Эти модели взаимосвязаны, поскольку представляют собой образы одного и того же оригинала – некоторого множества предметов реального мира, информацию о которых мы предполагаем хранить и обрабатывать с помощью проектируемой БД.

Характер взаимосвязей (и, соответственно, отличий) проявляется и в процессе проектирования системы баз данных. Модель предметной области скорее ассоциируется с неформальным<sup>35</sup> уровнем семантического моделирования, а модель базы данных – с формализованным уровнем системы (и в частности, СУБД). В идеале целью семантического моделирования является формирование систематического основания для хорошо формализованного процесса проектирования базы данных. Здесь необходимо вспомнить следующие требования, предъявляемые к базам данных, и, в частности, к способам описания данных:

- 1) описания должны быть понятны пользователю, не проектировавшему базу;
- 2) однажды принятые способы представления данных должны допускать присоединение новых элементов данных без изменения существующих схем данных и прикладных программ;
- 3) СУБД должны позволять эффективно обрабатывать произвольные запросы к базе данных.

Эти требования отражают, с одной стороны, точку зрения пользователя, для которой характерны требования высокой степени общности и широты представления (или, по крайней мере, не громоздкость детальных описаний), позволяющих ему получить достаточно сведений без затраты значительных временных или интеллектуальных ресурсов. С другой стороны – точку зрения администратора, выполняющего проектирование и оптимизацию системы баз данных, что предполагает высокую степень детализации и формализации, обеспечивающих обоснованность технических решений, а также возможность автоматизации проектирования.

Забегая несколько вперед, рассмотрим взаимосвязь двух известных методов семантического моделирования инфологического уровня – ER-диаграммы (Entity-Relation) и метод нормализации, воспринимаемых зачастую как альтернативные. На самом деле нормализация с помощью хорошо формализованных методов обеспечивает декомпозицию исходных отношений (переменных) большой размерности к возможно большему набору отношений, но меньшей размерности. Эти методы *не зави-*

---

<sup>35</sup> Правильнее было бы говорить о *неформализованности*, связанной с невозможностью обоснованного однозначного выбора (из реально существующих) как множества объектов, так и средств, используемых для моделирования (описания выделенных объектов).

сят от особенностей предметной области (семантики обрабатываемых данных), но вследствие этого и не позволяют определить исходное отношение. Для этого удобнее использовать методики, подобные ER-диаграммам - для них свойственны подходы технологии нисходящего проектирования, и которые дают представление «в целом», но именно поэтому (из-за сравнительной простоты) не позволяют провести полноценное проектирование базы. То есть, можно сказать, что метод нормализации и ER-диаграммы по существу являются взаимодополняющими.

Кроме того, разнообразие моделей связано также и с различием используемых парадигм моделирования, по существу определяющих способ представления взаимосвязи объектов на уровне *структур данных*. С этой точки зрения, различаются реляционные, сетевые, иерархические, объектные, объектно-реляционные, документальные и другие виды моделей. Соответственно различаются и описываемые их средствами схемы баз данных.

Рассмотрим основные стадии процесса<sup>36</sup> моделирования, представленные на рис. 5.1.

---

<sup>36</sup> Здесь не рассматриваются работы, относящиеся к другим этапам жизненного цикла баз данных, такие как, определение процедур заполнения и сопровождения БД, разработка конкретных программ обработки данных, развитие и улучшение структуры БД и т.д.

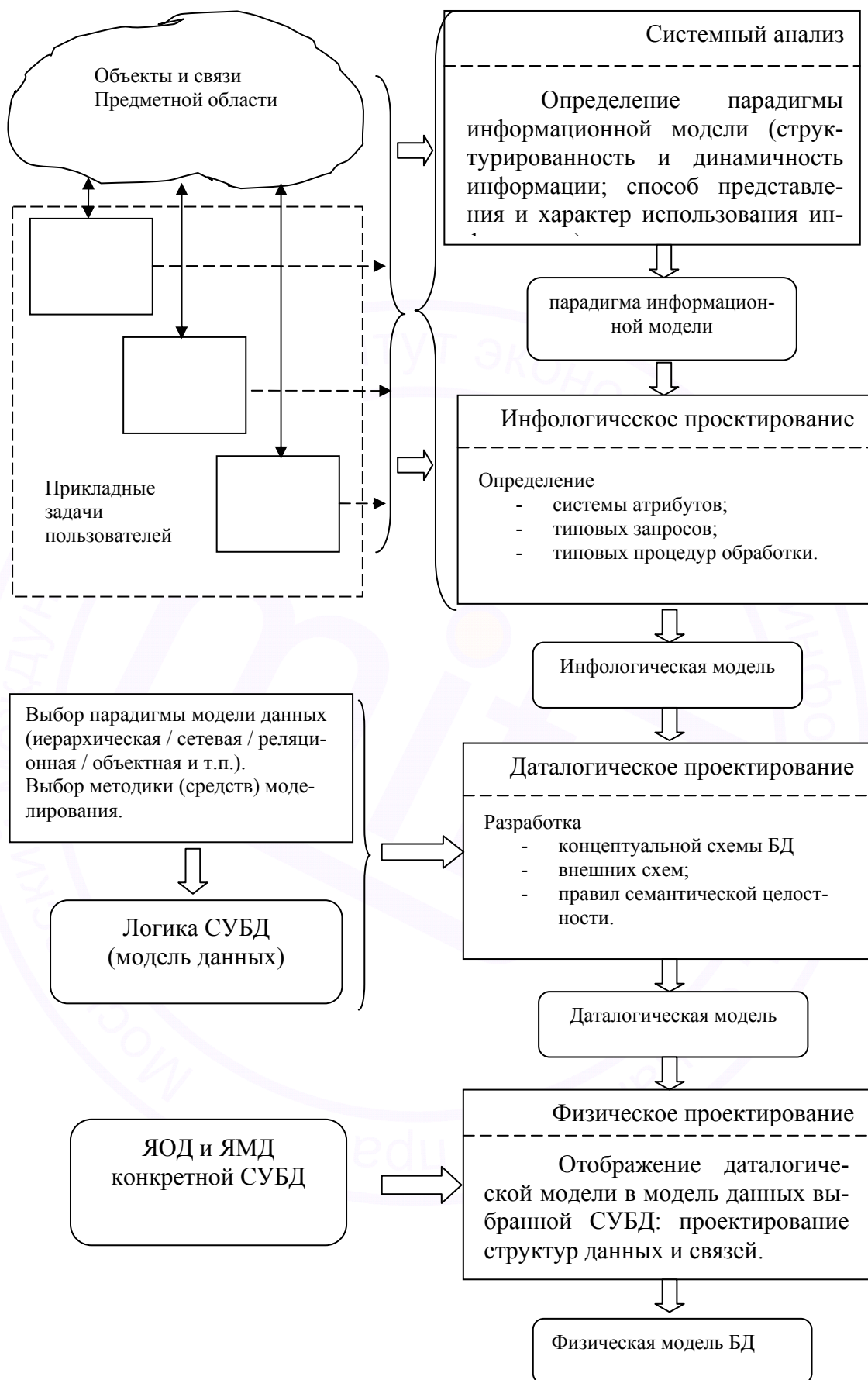


Рис. 5.1. Стадии и объекты процесса проектирования

### 5.3. Системный анализ предметной области

Как отмечалось ранее, базы данных сами по себе представляют относительную ценность. Базы данных это всегда важнейшая, но только *одна из* составляющих некоторой информационной системы (ИС). И надо отметить, что любая информационная система, предназначенная, например, для оперативного управления предприятием или архивного хранения и поиска документов – это не только программы, данные и коммуникации, но также и люди (заказчики, пользователи, аналитики, разработчики), организационные структуры, а также цели, стимулы работы предприятия или отдельных людей. И все эти компоненты должны быть понятным как проектировщику, так и пользователю, а, кроме того, непротиворечивым образом соединены в одну систему.

Главная идея процесса такого согласования состоит в том, что его надо начинать с анализа самых главных характеристик предметной области, рассматривая самые главные содержательные аспекты. И проводить его не "мысленно" и не "на словах", а на явно изложенных описаниях (моделях) объектов предметной области, позволяющих видеть все существенные взаимосвязи. Но следует отметить, что попытки использования привычных нотаций формальных моделей (структурных, объектных или каких либо других) на этом этапе приводят к более низкому (более детальному и в тоже время ограниченному) уровню представления предметной области, чем это необходимо для общего понимания.

В общем случае существуют два подхода к определению состава и структуры предметной области (рис. 5.2).

*Функциональный* подход предполагает, что проектирование начинается с анализа задач и, соответственно, функций, обеспечивающих реализацию информационных потребностей.

При *объектном* (предметном) подходе информационные потребности пользователей (задачи) жестко не фиксируются, а основное внимание сосредотачивается на выделении существенных объектов – предметов и связей, информация о которых может быть использована в прикладных задачах пользователя.

Условность такого деления достаточно очевидна, поэтому на практике используются компромиссные варианты, предполагающие по мере развития системы расширение как состава объектов, так и спектра прикладных задач.

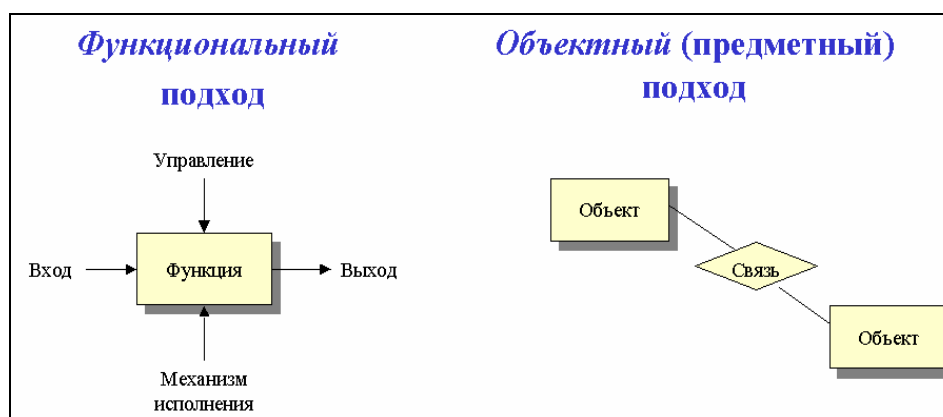


Рис. 5.2. Основные подходы к системному анализу предметной области

В [29] была предложена простая, но концептуально мощная схема, показывающая различные уровни представления архитектуры ИС, различные виды ее "обеспечения", а также их основные взаимосвязи. На рис. 5.3 показана таблица, представленная в [7], аналогичная схеме Захмана. Три столбца отражают три раздела обеспечения системы: информационный (ДАННЫЕ), функциональный (ФУНКЦИИ) и коммуникационный (СЕТЬ).

Строки таблицы отражают шесть уровней представления системы: реальная среда приложений, концептуальная модель, логическая модель, физическая модель, детальная реализация процедур, представления пользователя.

Полезность такой схемы состоит в том, что она помогает рассматривать задачи проекта в полном объеме, упорядочивать состав и структуру требований к системе, определять и фиксировать причинно-следственные связи.

	ДАННЫЕ	ФУНКЦИИ	СЕТЬ
Потребности и внешняя среда	1. .... 2. ....	1. .... 2. ....	
Бизнес-модель предприятия			
Представление аналитиков -- логическая модель			
Техническая архитектура	INDEX	SCREEN WIZARD	
Детальная реализация (субпрограмм)	CREATE TABLE	BEGIN BLOCK BEGIN .. END	C:>PING
Взгляд пользователя		Меню Ввод Печать	Wait, please

Рис. 5.3. Модель информационной системы Захмана



Позднее появилось развитие такой "плоской" модели. В [28] рассматривалась схема, представленная на рис. 5.4, включающая три новых столбца, в которых отражаются еще три раздела: побудительные причины действий системы, события и графики выполнения действий, а также "действующие лица" - люди и организационные структуры.

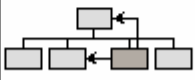
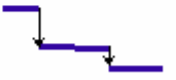
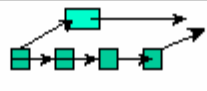


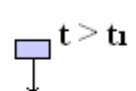
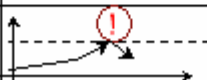
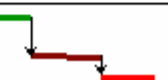
	Побудительные причины действий («ЮТИБД»)	Люди: участники бизнеса	Операционное время	Д А Н Н Ы Е	Ф У Н К Ц И И	С Е Т Ь
Главные потребности, цели и среда бизнеса	“Конкуренты, Новые товары...”	Партнеры, филиалы, клиенты	Главные события в ведении бизнеса			
Бизнес-модель предприятия	Бизнес-план (прибыль - 10%, риск - 2%)					
Представление анализиров - логическая модель	Бизнес-правила					
Техническая архитектура	Условия действия					
Детальная реализация (субподряд)	TRIGGER ALARM	read string into password;	on event t > t1..			
Выход "практика исползования"		Умения/ Ответственность				

Рис. 5.4. Развитие модели информационной системы Захмана

В результате появилось шесть разделов, которые содержат "ответы на вопросы": почему выполняются действия, когда выполняются и кто их выполняет, а также что делает система, как делает и где. При этом уровни представления (строки таблицы) остались те же.

Такое расширение позволило рассматривать потребности в контексте информационных технологий, соединять предметы и действия с человеческим фактором и операционной динамикой процессов.

Цель системного анализа предметной области как этапа проектирования – выделить предметную область как **систему объектов и их взаимосвязей**, определив при этом функционально-информационные требования к их последующему представлению в виде **системы взаимосвязанных данных**.

Главным результатом этапа системного анализа является определение **парадигмы информационной (инфологической) модели**: требования к средствам представления системы определяются на основании анализа уровня структурированности информации и характера восприятия ее семантики пользователем (точная/приблизительная, четкая/неопределенная).

Например, выбор *атрибутивной формы представления* объектов предметной области приведет, соответственно, к выбору парадигмы *фактографических баз данных*, а *вербальной* - к необходимости выбора *документальных БД*.

В дальнейшем изложении процесс и средства проектирования мы будем рассматривать только для случая фактографических баз данных, использующих реляционную модель.

#### **5.4. Модели и технологии инфологического проектирования реляционных БД**

Как отмечалось ранее, представляется вполне очевидным начинать создание базы данных с определения самих данных, выполняя проектирование базы данных в терминах отношений на основе механизма нормализации.

Однако, забегаая вперед, отметим, что такой подход часто представляет собой очень сложный и неудобный процесс для самого проектировщика. При этом проявляется ограниченность реляционной модели данных в следующих аспектах:

- реляционная модель не предоставляет достаточных средств для фиксации смысла данных, т.е. семантика предметной области не фиксируется непосредственно в отношениях;
- для многих приложений трудно моделировать предметную область на основе плоских таблиц;
- хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не имеет средств представления (отражения семантики) этих зависимостей;
- несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области ("сущностей") и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо аппарата для различения сущностей и связей в базе данных.

На практике семантическое моделирование обычно производится на первой стадии проектирования. Полученный результат - концептуальная схема базы данных (в терминах семантической модели) затем вручную или автоматически преобразуется к реляционной схеме.

##### **5.4.1. Инфологическое проектирование и семантическая модель**

Начальной стадией проектирования системы баз данных является построение *семантической модели* предметной области, которая базируется на анализе свойств и природы объектов предметной области и информационных потребностей будущих пользователей разрабатываемой

системы. Эту стадию принято называть *концептуальным* проектированием системы, а ее результат – *концептуальной моделью* предметной области (объектом моделирования здесь является предметная область будущей системы!). Этой стадии соответствуют также ранее упомянутые термины “*инфологическое проектирование*” и “*инфологическая модель*”.

Такие модели обобщенно представляют информационные потребности пользователей создаваемой системы в части использования хранимых данных<sup>37</sup> и, по существу, являются средством коммуникации как разработчиков, так и пользователей на разных стадиях жизненного цикла базы данных.

Назначение инфологических моделей определяет и некоторые специфические требования к средствам их представления. Помимо упомянутой независимости от среды (оборудования) и требования адекватности отражения предметной области отметим следующие:

- формализованность, обеспечивающую возможность автоматизированной обработки и, в том числе, например, автоматический контроль непротиворечивости;
- дружелюбность, обеспечивающую возможность использования наглядных графических средств отображения и обработки их пользователем.

К инфологическим моделям относятся различные компоненты, по-разному и разными средствами отражающие предметную область. Помимо наиболее известного описания объектов и связей между ними (модель «сущность-связь») к инфологическому уровню описания предметной области можно отнести следующие компоненты:

- систему атрибутов и средств описания предметной области. Например, логические (алгоритмические) связи между показателями или лингвистические свойства языка (синонимию, синтаксис и т.д.), используемого для вербального представления объектов;
- ограничения целостности, определяющие допустимость значения отдельных полей и взаимосвязей как на уровне семантики содержимого БД, так и ее физической структуры (отдельных файлов данных и взаимосвязей между ними);
- описание информационных потребностей пользователей, например, в виде типовых запросов, отражающих процедурные особенности обращения к данным.

---

<sup>37</sup> Еще раз отметим, что они, в отличие от моделей данных, используемых в качестве инструмента моделирования конкретных баз данных, не обязательно поддерживаются механизмами используемой СУБД, хотя это и может иметь место на практике.

#### 5.4.2. Модель «Сущность-Связь»

Одной из наиболее популярных средств формализованного представления предметной области систем, ориентированных на обработку фактографической информации, является модель “сущность-связь” [21], которая положена в основу значительного количества коммерческих CASE-продуктов, поддерживающих полный цикл разработки систем баз данных или отдельные его стадии. При этом многие из них не только поддерживают стадию концептуального проектирования предметной области разрабатываемой системы, но и позволяют осуществить на основе построенной их средствами модели стадию логического проектирования путем автоматической генерации концептуальной схемы базы данных для выбранной СУБД, например, схемы базы данных для какого-либо SQL-сервера или объектной СУБД.

Моделирование предметной области в этом случае базируется на использовании графических диаграмм, включающих сравнительно небольшое число компонентов и, самое важное – **технологии построения** таких диаграмм<sup>38</sup>.

Семантическую основу ER-модели составляют следующие предположения:

- та часть реального мира (совокупность взаимосвязанных объектов), сведения о которых должны быть помещены в базу данных, может быть *представлена* как совокупность **сущностей**;
- каждая сущность обладает характеристическими свойствами (атрибутами), отличающими ее от других сущностей и позволяющими ее *идентифицировать*;
- сущности можно классифицировать по типам сущностей: каждый экземпляр сущности (представляющий некоторый объект) может быть отнесен классу - **типу сущностей**, каждый экземпляр которого обладает общими для них свойствами и отличающим их от сущностей других классов;
- систематизация представления, основанная на классах, в общем случае предполагает иерархическую зависимость типов: сущность типа *A* является **подтипом** сущности *B*, если каждый экземпляр типа *A* является экземпляром сущности типа *B*;
- взаимосвязи объектов могут быть представлены как **связи** – сущности<sup>39</sup>, которые служат для фиксирования (представления) взаимозависимости двух или нескольких сущностей.

---

<sup>38</sup> Существует много версий ER-диаграмм, которые по-разному представляют связи, сущности и атрибуты, и которые имеют различные ограничения и условия применимости. Приведенный здесь пример не является полным и не претендует на точное соответствие какой-либо версии построения ER-диаграмм или CASE-продукту.

<sup>39</sup> Такое определение связи, как сущности особого рода, отражает существо реляционного подхода, для которого характерно единообразное представление сущностей всех типов, включая связи, посредством переменных-отношений.



Здесь следует еще раз подчеркнуть информационную природу понятия *сущность* и его соотношение с материальными или воображаемыми объектами предметной области. Любой объект предметной области обладает свойствами, часть из которых выделяется как характеристические - значимые с точки зрения прикладной задачи. При этом, например, в процессе анализа и систематизации предметной области, обычно выделяются *классы* – совокупности объектов, обладающих одинаковым набором свойств, задаваемых в виде *наборов атрибутов* (значения атрибутов для объектов одного класса естественно могут различаться). Соответственно, на уровне представления предметной области (т.е. - ее инфологической модели) объекту, рассматриваемому как понятие (объект в сознании человека), соответствует понятие *сущность*; объекту, как части материального мира (и существующему независимо от сознания человека), соответствует понятие *экземпляр сущности*; классу объектов соответствует понятие *тип сущности*.

В дальнейшем, поскольку в инфологической модели рассматриваются не отдельные экземпляры объектов, а классы, мы не будем различать соответствующие понятия этих двух уровней, т.е. будем предполагать тождественность понятий *объект* и *сущность*, *свойство объекта* и *свойство сущности*.

ER-модель, как описание предметной области, должна определить объекты и взаимосвязи между ними, т.е. установить связи следующих двух типов:

1. связи между объектами и наборами характеристических свойств и таким образом определить сами объекты;
2. связи между объектами, задающие характер и функциональную природу их взаимозависимости.

Как было отмечено ранее, ER-моделирование предметной области базируется на использовании графических диаграмм, как простого (привычного), наглядного и, в то же время, информативного и многоаспектного способа отображения компонентов проекта. Поэтому изложение основных положений ER-модели будет иллюстрироваться материалом примера ER-диаграммы, приведенной на рис. 5.5.



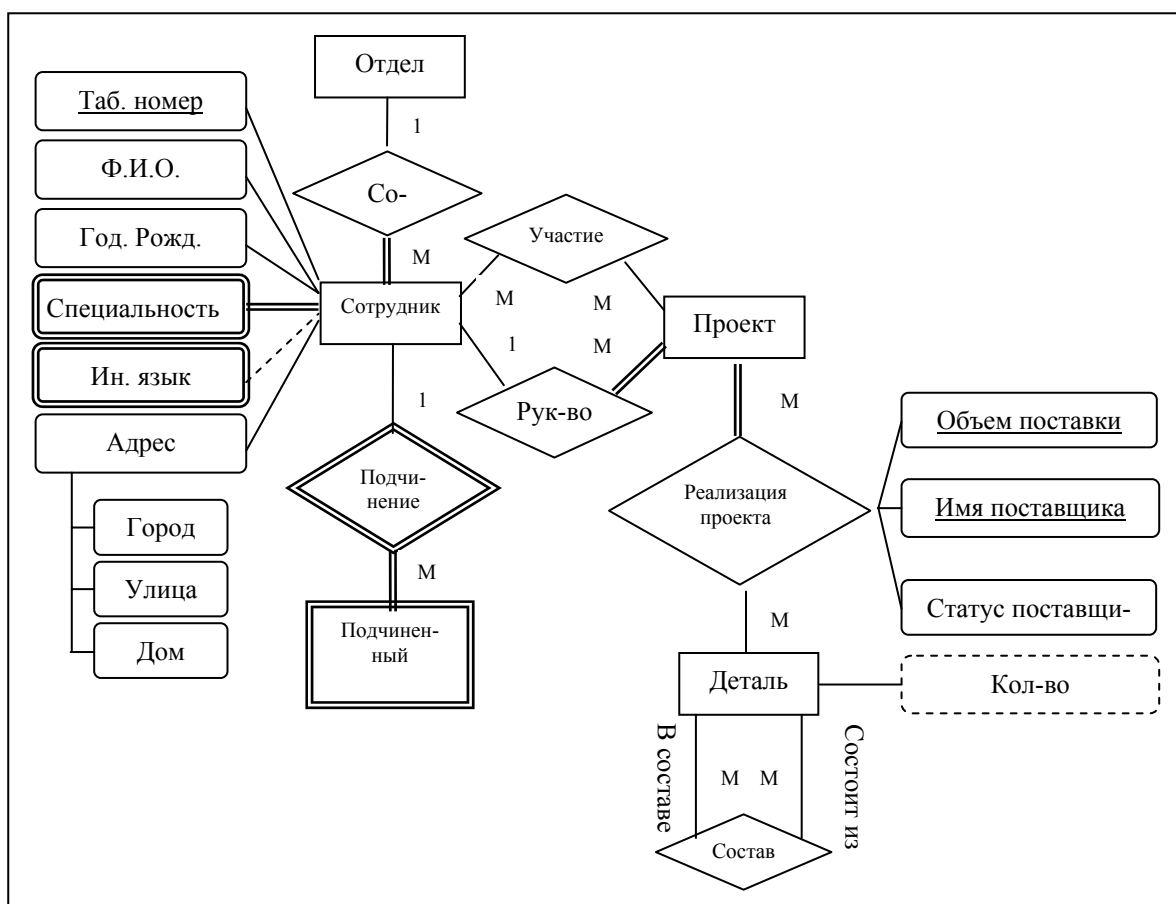


Рис. 5.5. Пример ER-диаграммы

**Сущность.** Сущность, с помощью которой моделируется класс однотипных объектов, определяется в [21] как «предмет, который может быть четко идентифицирован». Так же, как каждый объект уникально характеризуется набором значений свойств, сущность должна *определяться* таким *набором атрибутов*, который позволял бы различать отдельные экземпляры сущности. Каждый экземпляр сущности должен быть отличим от любого другого экземпляра той же сущности (это требование аналогично требованию отсутствия кортежей-дубликатов в реляционных таблицах). Например, для однозначной идентификации каждого экземпляра сущности «Сотрудник» вводится атрибут «Таб.номер», который вследствие своей природы будет всегда иметь уникальное значение в рамках предприятия. Т.е., уникальным идентификатором сущности может являться атрибут, комбинация атрибутов, комбинация связей или комбинация связей и атрибутов, однозначно отличающая любой экземпляр сущности от других экземпляров сущности того же типа.

Сущность имеет *имя*, уникальное в пределах модели. При этом *имя сущности* - это *имя типа*, а не некоторого конкретного экземпляра.

Сущности подразделяются на *сильные* и *слабые*. Сущность является слабой, если ее существование зависит от другой сущности – сильной, по отношению к ней. Например, сущность «Подчиненный» являет-

ся слабой по отношению к сущности «Сотрудник»: если будет удалена запись, соответствующая некоторому сотруднику, имеющему подчиненных, то сведения о подчинении также должны быть удалены.

**Свойства.** Природа свойства, как *характер связи* свойства с сущностью (объектом), может быть различной. Рассмотрим основные виды свойств.

Свойство может быть *множественным* или *единичным* – т.е. атрибут, задающий свойство, может одновременно иметь несколько значений или, соответственно, только одно. Например, сотрудник может иметь несколько специальностей, но единственное значение «Таб. номер».

Свойство может быть *простым* (не подлежащих дальнейшему делению с точки зрения прикладных задач) или *составным* – если его значение составляется из значений простых свойств. Например, свойство «Год рождения» является простым, а свойство «Адрес» – составным, т.к. включает значения простых свойств «Город», «Улица», «Дом».

В некоторых случаях полезно различать *базовые* и *производные* свойства. Например, «Поставщик» может иметь свойство «Общее количество поставляемых деталей», которое вычисляется суммированием количества деталей, поставляемых им по проекту.

Если наличие некоторого свойства для всех экземпляров сущности не является обязательным, то такое свойство называется *условным*. Например, не все сотрудники обладают свойством «ученая степень».

Значения свойств могут быть постоянными – *статическими*, или *динамическими*, т.е. меняться со временем. Например, свойство «Таб. номер» является статическим, а «Адрес» – динамическим. Свойство может быть *неопределенным*, если оно является динамическим, но его текущее значение еще не задано.

Свойство может рассматриваться как *ключевое*, если его значение уникально и, возможно, в определенном контексте, однозначно идентифицирует сущность. Например, подчиненный некоторого определенного сотрудника.

**Связи.** Кроме связей между объектом и его свойствами, инфологическая модель отражает связи между объектами разных классов. В [21] *связь* определяется как «ассоциация, объединяющая несколько сущностей». Эта ассоциация всегда может существовать между разными сущностями или между сущностью и ей же самой (рекурсивная связь).

Как и сущность, связь является *типовым* понятием, т.е. все экземпляры связываемых сущностей подчиняются правилам связывания типов. Принципиальность различия типов связей между типами и экземплярами иллюстрируется ER-диаграммами для типов и экземпляров, представленными на рис. 5.6.

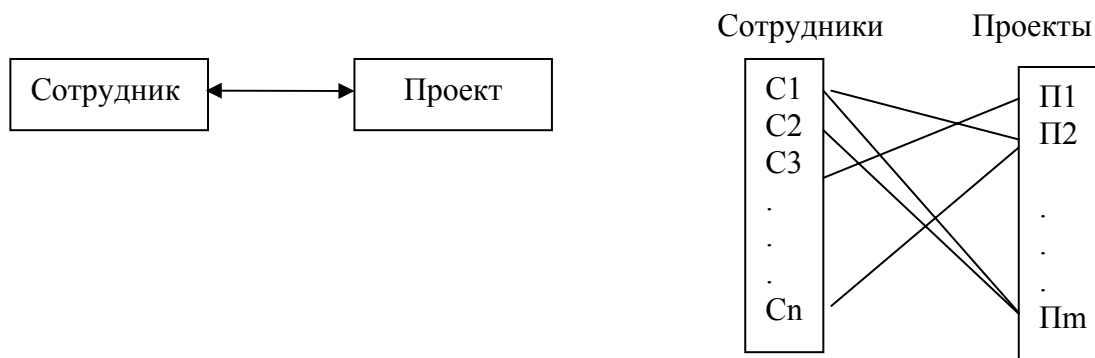


Рис. 5.6. Примеры ER-диаграмм для типов и экземпляров сущностей

Сущности, объединяемые связью, называются *участниками*. *Степень связи* определяется количеством участников связи<sup>40</sup>.

Если каждый экземпляр сущности участвует, по крайней мере, в одном экземпляре связи, то такое участие этой сущности называется *полным* (или *обязательным*); в противном случае – *неполным* (или *необязательным*).

Количественный характер участия экземпляров сущностей (один или многие) задается *типом связи* (или *мощностью связи*). Возможны следующие типы: «один к одному» (1:1), «один ко многим» (1:M), «многие к одному» (M:1), «многие ко многим» (M:M)<sup>41</sup>.

Следует отметить, что инструмент связей - это средство представления *сложных объектов*, каждый из которых может рассматриваться как множество некоторым образом взаимосвязанных *простых объектов*. Деление на простые и сложные объекты, также как и характер взаимосвязи, является условным и определяется особенностям анализа предметной области, т.е. в конце концов – характером использования данных о предметах в решаемых прикладных задачах. При этом с точки зрения, например, конструктора, ДЕТАЛЬ является сложным объектом, а с точки зрения поставщика – простым.

Среди многих разновидностей взаимосвязей наиболее частыми являются такие отношения иерархического типа, как «часть-целое», «род-вид».

Отношение «часть-целое» используется для представления *составных объектов*. Например, МАШИНЫ состоят из УЗЛОВ, УЗЛЫ состоят из ДЕТАЛЕЙ. Здесь возможны как отношения «один ко многим», так и «многие ко многим».

<sup>40</sup> В большинстве CASE-систем принята упрощенная форма: эта ассоциация всегда **должна быть бинарной** и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь).

<sup>41</sup> Согласно положениям реляционной модели «правильной» связью является только связь типа «многие ко многим», поскольку для ее реализации создается отдельная переменная-отношение. Связи «один к одному», «один ко многим» всегда могут быть представлены с помощью механизма внешнего ключа одной из переменных-отношения.

Отношение «род-вид» - для представления *обобщенных объектов*. Например, СОТРУДНИКИ подразделяются по профессии на КОНСТРУКТОРОВ, ПРОГРАММИСТОВ, РАБОЧИХ; ПРОГРАММИСТЫ – на ПРИКЛАДНЫХ ПРОГРАММИСТОВ и СИСТЕМНЫХ ПРОГРАММИСТОВ. Иерархические отношения, и, в частности – «родо-видовые», обычно используются как основа классификации объектов по наборам характеристических признаков. Причем «видовые» объекты *наследуют* свойства «родовых».

Другой широко используемой разновидностью взаимосвязи является агрегирование – объединение простых объектов в сложный по принципу их принадлежности *агрегату* или их совместного участия в некотором процессе. Агрегирование, рассматриваемое здесь как более общий случай иерархических отношений, объединяет объекты разной природы с единственным общим свойством «совместное участие». Агрегированные объекты именуются обычно отглагольными существительными, например, «Состав»: ПОДРАЗДЕЛЕНИЕ *состоит из* СОТРУДНИКОВ; «Поставка»: ПОСТАВЩИК *поставляет* ДЕТАЛИ.

**Супертипы и подтипы.** Сущность может быть расщеплена на два или более взаимно исключающих *подтипов*, каждый из которых включает общие атрибуты и/или связи. Эти общие атрибуты и/или связи явно определяются один раз на более высоком уровне. В подтипах могут определяться собственные атрибуты и/или связи. В принципе выделение подтипов может продолжаться на более низких уровнях, но в большинстве случаев оказывается достаточно двух-трех уровней.

Сущность, на основе которой определяются подтипы, называется *супертипом*. Подтипы должны образовывать полное множество, т.е. любой экземпляр супертипа должен относиться к некоторому подтипу. Иногда для полноты множества надо определять дополнительный подтип, например ПРОЧИЕ.

Подтип наследует свойства и связи супертипа. Например, тип сущности ПРОГРАММИСТ является подтипом сущности СОТРУДНИК. Программисты обладают всеми свойствами сотрудников и участвуют во всех связях, однако обратные утверждения неверны.

Тип сущности, его подтипы, подтипы этих подтипов и т.д. образуют *иерархию типов сущности*, пример которой приведен на рис. 5.7.



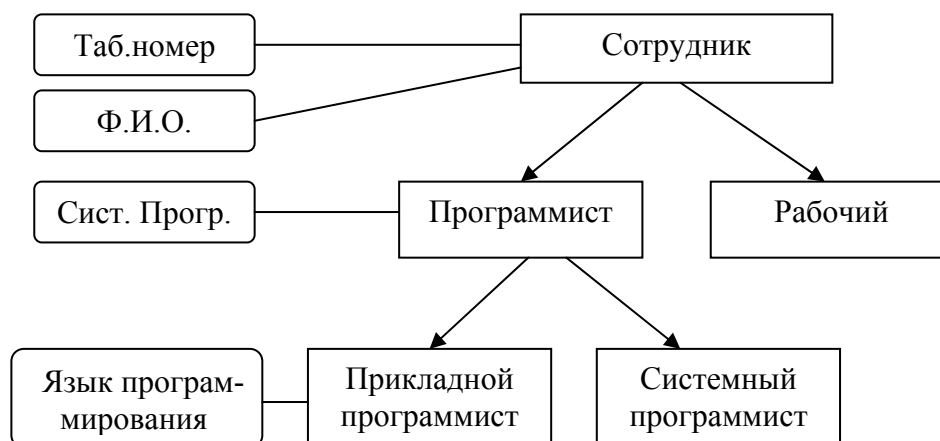


Рис. 5.7. Пример иерархии типов сущности

#### 5.4.3. ER- диаграмма

Как отмечалось ранее, одна из основных целей семантического моделирования состоит в том, чтобы результаты анализа предметной области были отражены в достаточно простом, наглядном но, в то же время, формализованном и достаточно информативном виде.

В этом смысле ER-диаграмма является очень удачным решением. В ней сочетаются функциональный и информационный подходы, что позволяет представлять как совокупность выполняемых функций, так и отношения между элементами системы, задаваемые структурами данных. При этом графическая форма позволяет отобразить в компактном виде (за счет наглядных условных обозначений) типологию и свойства сущностей и связей, а формализмы, положенные в основу ER-диаграмм, позволяют использовать на следующем шаге проектирования логической структуры базы данных строгий аппарат нормализации.

**Сущности.** Каждый тип сущности в ER-диаграммах представляется в виде прямоугольника, содержащего имя сущности. В качестве имени обычно используются существительные (или обороты существительного) в единственном числе. Для отражения сущностей слабых типов используются прямоугольники, стороны которых рисуются двойными линиями. Например, в рассматриваемой далее ER-диаграмме, приведенной на рис 5.4, ПОДЧИНЕННЫЙ – сущность слабого типа.

**Свойства.** Свойства служат для уточнения, идентификации, характеристики или выражения состояния сущности или связи. Свойства



отображаются в виде эллипсов, содержащих имя свойства<sup>42</sup>. Эллипс соединяется с соответствующей сущностью или связью линией.

Имена ключевых свойств подчеркиваются. Например, свойство «Таб.номер» сущности СОТРУДНИК.

Контур эллипса рисуется двойной линией, если свойство многозначное. Например, свойство «специальность» сущности СОТРУДНИК.

Контур эллипса рисуется штриховой линией, если свойство производное. Например, свойство «кол-во» сущности ПОСТАВЩИК.

Эллипс соединяется пунктирной линией, если свойство условное. Например, свойство «Ин. язык» сущности СОТРУДНИК.

Если свойство составное, то составляющие его свойства отображаются другими эллипсами, соединенными с эллипсом составного. Например, свойство «Адрес» сущности СОТРУДНИК состоит из простых свойств «Город», «Улица», «Дом».

**Связи.** Связь - это графически изображаемая ассоциация, устанавливаемая между сущностями. Каждый тип связи на ER-диаграмме отображается в виде ромба с именем связи внутри<sup>43</sup>. В качестве имени обычно используются отглагольные существительные.

Стороны ромба рисуют двойными линиями, если это связь сущности слабого типа с сущностью от которой она зависит. Например, связь «Подчинение», связывающая сущность слабого типа ПОДЧИНЕННЫЙ с сущностью СОТРУДНИК, от которой она зависит.

Участники связи соединены со связью линиями. Двойная линия обозначает полное участие сущности в связи с данной стороны. Например, связь «Подчинение», со стороны сущности ПОДЧИНЕННЫЙ.

Связь может быть модифицирована указанием роли. Например, для рекурсивной связи «Состав», указаны роли: «Деталь *состоит из* ...» и «Деталь *входит в состав* ...».

Тип связи указывается индексами «1» или «М» над соответствующей линией. Например, связь «Руководство» имеет тип «один ко многим»: один сотрудник может руководить многими проектами; связь «Участие» имеет тип «многие ко многим»: один сотрудник может участвовать во многих проектах, и в проекте могут участвовать многие сотрудники.

<sup>42</sup> В большинстве CASE-систем имена свойств заносятся в прямоугольник, изображающий сущность, под именем сущности и изображаются малыми буквами, возможно, с примерами.

<sup>43</sup> В большинстве CASE-систем связь всегда должна быть бинарной или рекурсивной. В этом случае в ER-диаграмме связь представляется в виде **линии**, связывающей две сущности или ведущей от сущности к ней же самой. В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи). При этом в месте "стыковки" связи с сущностью используются трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут использоваться много экземпляров сущности, и одноточечный вход, если в связи может участвовать только один экземпляр сущности. Обязательный конец связи изображается сплошной линией, а необязательный - прерывистой линией.

#### 5.4.4. Нормальные формы ER-диаграмм

Как и в реляционных схемах баз данных, в ER-диаграммах вводится понятие нормальных форм, причем их смысл очень близко соответствует смыслу реляционных нормальных форм. Приведем краткие и неформальные определения трех первых нормальных форм.

*В первой нормальной форме* ER-диаграммы устраняются повторяющиеся атрибуты или группы атрибутов, т.е. производится выявление неявных сущностей, "замаскированных" под атрибуты.

*Во второй нормальной форме* устраняются атрибуты, зависящие только от части уникального идентификатора. Эта часть уникального идентификатора определяет отдельную сущность.

*В третьей нормальной форме* устраняются атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор. Эти атрибуты являются основой отдельной сущности.

На рис. 5.8 представлена ER-диаграмма рис. 5.5 в третьей нормальной форме.

#### 5.5. Даталогические модели

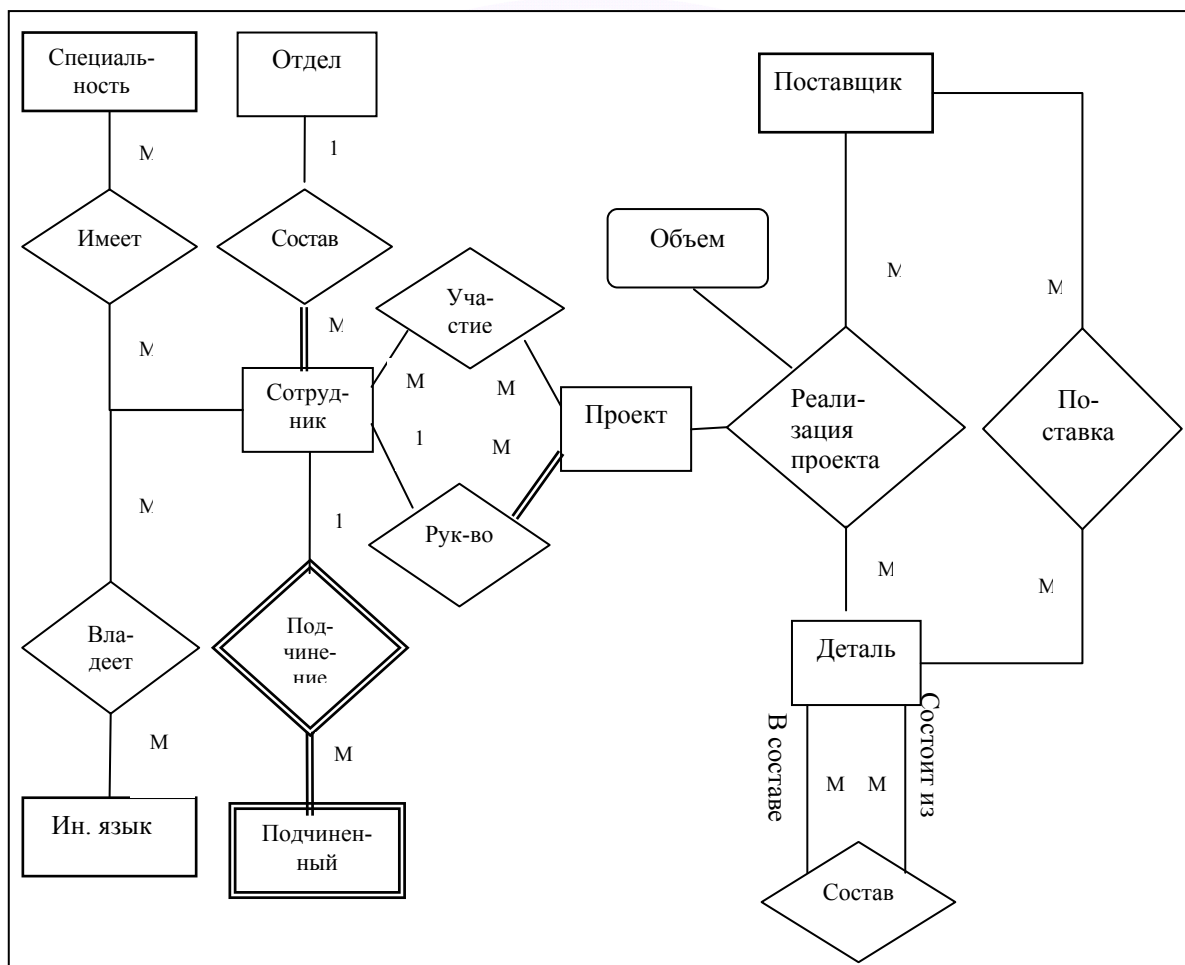
Задачей следующей стадии проектирования системы базы данных является выбор подходящей СУБД и отображение в ее среду (структур данных) спецификаций инфологической модели предметной области. Другими словами, модель предметной области разрабатываемой системы должна быть представлена в терминах модели данных концептуального уровня выбранной конкретной СУБД. Эту стадию называют логическим (или даталогическим) проектированием базы данных, а ее результатом является концептуальная схема базы данных, включающая определение всех информационных элементов (единиц) и связей, в том числе задание типов, характеристик и имен.

Хотя даталогическое проектирование оперирует не физическими записями, а логическими понятиями, связанными со структурой базы данных, тем не менее, особенности представления данных, правила и языки агрегирования и манипулирования данными имеют определяющее влияние. Не все виды связей, например, «многие ко многим», могут быть непосредственно отображены в логической модели.

Кроме того, может быть много вариантов отображения инфологической модели предметной области в даталогическую модель базы. Здесь следует учитывать влияние двух следующих значимых факторов, связанных с практикой разработки базы данных.

Во-первых, связи предметной области могут отображаться двумя путями, как декларативным - в логической схеме, так и процедурным - отработкой связей через программные модули, обрабатывающие (связывающие) соответствующие хранимые данные.

Рассмотрим по шагам общий подход к построению реляционной базы данных на основе инфологической модели, представленной ER-диаграммой.



### 5.5.1. Получение реляционной схемы из ER-диаграммы

- 140

3. Компоненты уникального идентификатора сущности превращаются в первичный ключ. Если имеется несколько возможных уникальных идентификаторов, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, то к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

4. Связи «многие к одному» и «один к одному» становятся внешними ключами. Т.е. создается копия уникального идентификатора с конца связи "один", и соответствующие столбцы составляют внешний ключ.

5. Индексы создаются для первичного ключа (уникальный индекс), а также внешних ключей и тех атрибутов, которые будут часто использоваться в запросах.

6. Если в концептуальной схеме присутствуют подтипы, то возможны два варианта.

Все подтипы хранятся в одной таблице, которая создается для самого внешнего супертипа, а для подтипов создаются представления. В таблицу добавляется, по крайней мере, один столбец, содержащий код ТИПА, и он становится частью первичного ключа.

Во втором случае для каждого подтипа создается отдельная таблица и для каждого подтипа первого уровня (для более нижних - представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы - столбцы супертипа).

7. Если остающиеся внешние ключи все принадлежат одному домену, т.е. имеют общий формат, то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи.

Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей.

## ***5.6. Физические модели***

Стадия физического проектирования базы данных в общем случае включает:

- выбор способа организации базы данных, основные из которых были рассмотрены ранее в главе 4;
- разработку спецификации внутренней схемы средствами модели данных ее внутреннего уровня;
- описание отображения концептуальной схемы во внутреннюю.

Важно заметить, что в отличие от ранних СУБД, многие современные системы не предоставляют разработчику какого-либо выбора на



этой стадии. Реально к вопросам проектирования физической модели можно отнести выбор схемы размещения данных (разделение по файлам или тип RAID-массива) и определение числа и типа индексов (например, кластеризованный или некластеризованный в случае MS SQL Server).

Способ хранения базы данных определяется механизмами СУБД автоматически “по умолчанию” на основе спецификаций концептуальной схемы базы данных, и внутренняя схема в явном виде в таких системах не используется.

Следует также отметить, что внешние схемы базы данных обычно конструируются на стадии разработки приложений.

Примеры типичных способов реализаций физических моделей, воплощенных в промышленных СУБД, приведены в Приложении.

### *Контрольные вопросы*

1. Перечислите основные этапы проектирования БД
2. Перечислите задачи этапа системного анализа
3. Определите понятие «парадигма моделирования»
4. Что называется инфологической моделью
5. Какие бывают типы сложных объектов
6. Определите типологию связей
7. Определите соотношение понятий «сущность» и «связь»
8. Какими свойствами должен обладать идентификатор объекта
9. Проанализируйте соотношение ER-модели и ER-диаграммы
10. В чем состоит сходство и отличие логической и физической модели
11. Как отображается сложный объект в реляционной модели.
12. Как отображаются в реляционной модели отношения типа М:М
13. В чем заключается логическое проектирование для конкретной СУБД.



## Глава 6. Проектирование реляционной базы данных

Задача проектирования БД для предметной области состоит в том, чтобы обеспечить поддержку не только любых ныне используемых, но и будущих приложений. Таким образом, БД создают основу для обработки неформализованных, изменяющихся и неизвестных запросов и создания приложений, для которых невозможно заранее определить требования к данным. Это позволяет в дальнейшем строить на основе предметных БД достаточно стабильные информационные системы, т.е. системы, в которых большинство изменений можно осуществить без переписывания старых приложений.

С другой стороны, основывая проектирование БД на реализации текущих и видимых задач, можно существенно ускорить создание информационной системы, структура которой учитывает наиболее часто встречающиеся пути доступа к данным. Однако по мере количества таких информационных систем быстро увеличивается число прикладных БД и, соответственно, резко возрастает уровень дублирования данных и повышается стоимость их ведения.

Желание достичь одновременно гибкости и эффективности приводит к тому, что в общем случае предметный подход используется для построения первоначальной информационной структуры, а прикладной – для ее совершенствования с целью повышения эффективности обработки данных.

При проектировании информационной системы (см. гл. 5) необходимо провести анализ целей этой системы и выявить требования к ней отдельных пользователей. Сбор данных начинается с выявления и изучения объектов информационной среды и процессов, в которых эти объекты участвуют. Объекты (сущности) группируются по типу и по мощности связей между ними (студент – сессия, преподаватель – дисциплина и т.д.).

Дальнейшая задача проектирования БД – это сокращение избыточности хранимых данных, а следовательно, экономия объема используемой памяти, уменьшение затрат на многократные операции обновления избыточных копий и устранение возможности возникновения противоречий из-за хранения в разных местах сведений об одном и том же объекте. Такой проект БД можно создать, используя методологию нормализации отношений.

### 6.1. Универсальное отношение

Рассмотрим задачу проектирования БД на базе сводной таблицы, пример которой приведен на рис. 6.1. Предложенная таблица отражает результаты сдачи сессии (шкала оценок: 0 – незачет; 1 – зачет; 2, 3, 4, 5 – экзаменационная оценка).

Этот вариант таблицы «Сессия» не является отношением, так как большинство ее столбцов не атомарны. Атомарными являются лишь значения столбцов «ФИО студента», «Семестр». Остальные столбцы таблицы – множественные.

Для преобразования данных в отношение необходимо реконструировать таблицу, например, с помощью простого процесса вставки, результат которой показан на рис. 6.2.

Однако очевидно, что такое преобразование приводит к возникновению большого объема избыточных данных.

ФИО студента	Семестр	Дисциплина	Форма отчетности	Оценка	Кол-во часов	ФИО преподавателя
Иванов В. П.	1	Английский язык	зачет	1	60	Цветкова А. Ю.
		Математический анализ	зачет	1	28	Рыбин К. К.
		Математический анализ	экзамен	5	32	Раков И. И.
		Программирование	зачет	1	36	Незабудкина З. П.
		Программирование	экзамен	5	32	Зайчиков А. А.
		Линейная алгебра	зачет	1	24	Волков Г. И.
		Линейная алгебра	экзамен	4	28	Волков Г. И.
		История Отечества	экзамен	5	24	Москвин А. П.
Петрова А.П.	1	Английский язык	зачет	1	60	Цветкова А. Ю.
		Математический анализ	зачет	1	28	Рыбин К. К.
		Математический анализ	экзамен	3	32	Раков И. И.
		Программирование	зачет	1	36	Незабудкина З. П.
		Программирование	экзамен	4	32	Зайчиков А. А.
		Линейная алгебра	зачет	1	24	Волков Г. И.
		Линейная алгебра	экзамен	4	28	Волков Г. И.
		История Отечества	экзамен	5	24	Москвин А. П.
Сидоров К.К.	3	Английский язык	зачет	1	60	Цветкова А. Ю.
		Математический анализ	зачет	1	20	Карпов К. Ю.

	Математический анализ	экзамен	5	28	Раков И. И.
	Алгоритмы и структуры данных	экзамен	5	32	Зайчиков А. А.
	Теория вероятностей и мат. статистика	экзамен	4	32	Соболев И. Г.
	Операционные системы, среды и оболочки	зачет	1	36	Незабудкина З. П.
	Операционные системы, среды и оболочки	экзамен	4	32	Незабудкина З. П.
	Экономическая теория	зачет	1	24	Лабиринтов Е. Н.

Рис. 6.1. Исходные данные для создания БД «Сессия»

Таблица на рис. 6.2 представляет собой корректное отношение. Такое отношение называют универсальным отношением проектируемой БД. В одно универсальное отношение включаются все представляющие интерес атрибуты, и оно может содержать все данные, которые предполагается размещать в БД в будущем. При проектировании некоторых БД универсальное отношение может использоваться в качестве отправной точки.

ФИО студента	Семестр	Дисциплина	Форма отчетности	Оценка	Кол-во часов	ФИО преподавателя
Иванов В. П.	1	Английский язык	зачет	1	60	Цветкова А.Ю.
Иванов В. П.	1	Математический анализ	зачет	1	28	Рыбин К.К.
Иванов В. П.	1	Математический анализ	экзамен	5	32	Раков И.И.
Иванов В. П.	1	Программирование	зачет	1	36	Незабудкина З.П.
Иванов В. П.	1	Программирование	экзамен	5	32	Зайчиков А.А.
Иванов В. П.	1	Линейная алгебра	зачет	1	24	Волков Г.И.
Иванов В. П.	1	Линейная алгебра	экзамен	4	28	Волков Г.И.
Иванов В. П.	1	История Отечества	экзамен	5	24	Москвин А.П.
Петрова А. П.	1	Английский язык	Зачет	1	60	Цветкова А.Ю.
Петрова А. П.	1	Математический анализ	Зачет	1	28	Рыбин К.К.
Петрова А. П.	1	Математический анализ	экзамен	3	32	Раков И.И.

Петрова А. П.	1	Программирование	Зачет	1	36	Незабудкина З.П.
Петрова А. П.	1	Программирование	экзамен	4	32	Зайчиков А.А.
Петрова А. П.	1	Линейная алгебра	Зачет	1	24	Волков Г.И.
Петрова А. П.	1	Линейная алгебра	экзамен	4	28	Волков Г.И.
Петрова А. П.	1	История Отечества	экзамен	5	24	Москвин А.П.
Сидоров К. К.	3	Английский язык	Зачет	1	60	Цветкова А.Ю.
Сидоров К. К.	3	Математический анализ	Зачет	1	20	Карпов К.Ю.
Сидоров К. К.	3	Математический анализ	экзамен	5	28	Раков И.И.
Сидоров К. К.	3	Алгоритмы и структуры данных	экзамен	5	32	Зайчиков А.А.
Сидоров К. К.	3	Теория вероятностей и мат. статистика	экзамен	4	32	Соболев И.Г.
Сидоров К. К.	3	Операционные системы, среды и оболочки	зачет	1	36	Незабудкина З.П.
Сидоров К. К.	3	Операционные системы, среды и оболочки	экзамен	4	32	Незабудкина З.П.
Сидоров К. К.	3	Экономическая теория	зачет	1	24	Лабиринтов Е.Н.

*Рис. 6.2. Универсальное отношение «Сессия»*

Однако при использовании универсального отношения возникают, по крайней мере, две проблемы:

1. *Избыточность данных.* Значения столбцов таблицы многократно повторяются. Повторяются также и некоторые наборы значений столбцов, например, данные о дисциплине.

2. *Потенциальная противоречивость.* Если при вводе данных, например, количество часов для дисциплины «Английский язык», была допущена ошибка, то для ее исправления необходимо найти все строки, содержащие сведения об этой дисциплине, и во всех этих строках провести изменения. Более того, при заполнении такой таблицы могут быть использованы различные формы записи одного и того же значения – на-

пример: «Англ. язык» и «Английский язык»; «Мат. анализ» и «Математический анализ».

Решение этих проблем состоит в разделении данных и связей, т.е. в выделении в отдельные таблицы сведений о студентах, преподавателях, дисциплинах и результатах сдачи экзаменов (рис. 6.3).

Студенты		Преподаватели		Дисциплины	
№	ФИО студента	№	ФИО преподавателя	№	Дисциплина
1	Иванов В. П.	1	Волков Г. И.	1	Алгоритмы и структуры данных
2	Петрова А. П.	2	Зайчиков А. А.	2	Английский язык
3	Сидоров К. К.	3	Карпов К. Ю.	3	История Отечества
		4	Лабиринтов Е. Н.	4	Линейная алгебра
		5	Москвин А. П.	5	Математический анализ
		6	Незабудкина З. П.	6	Операционные системы, среды и оболочки
		7	Раков И. И.	7	Программирование
		8	Рыбин К. К.	8	Теория вероятностей и мат. статистика
		9	Соболев И. Г.	9	Экономическая теория
		10	Цветкова А. Ю.		

Учебный план					
№	Дисциплина	Семестр	Кол-во часов	Форма отчетности	Преподаватель
1	2	1	60	зачет	10
2	3	1	24	экзамен	5
3	4	1	24	зачет	1
4	4	1	28	экзамен	1
5	5	1	28	зачет	8
6	5	1	32	экзамен	7
7	7	1	36	зачет	6
8	7	1	32	экзамен	2
9	2	3	60	зачет	10
10	5	3	20	зачет	3
11	5	3	28	экзамен	7
12	1	3	32	экзамен	2
13	8	3	32	экзамен	9
14	6	3	36	зачет	6
15	6	3	32	экзамен	6
16	9	3	24	зачет	4

Результаты сессии		
Студент	Учебный план	Оценка
1	1	1
1	2	5
1	3	1
1	4	4
1	5	1
1	6	5
1	7	1
1	8	5
2	1	1
2	2	5
2	3	1
2	4	4
2	5	1
2	6	3
2	7	1
2	8	4

Рис. 6.3. Разделение универсального отношения «Сессия»



Заменим в таблицах «Результаты сессии» и «Учебный план» конкретные значения на их номера в других таблицах и получим, помимо значительного упрощения процедуры модификации текстовых значений, дополнительные возможности по включению строк в таблицы «Студенты», «Преподаватели», «Дисциплины», что значительно расширяет возможности БД.

Теперь при изменении названия «Математический анализ» на «Мат. анализ» исправляется единственное значение в таблице «Дисциплины». И даже если оно вводится с ошибкой, то это не может повлиять на связь между дисциплиной, преподавателем и студентом (в связующей таблице «Результаты сессии» используются номера дисциплин учебного плана, а не их названия).

## **6.2. Функциональная и многозначная зависимости**

Процесс нормализации – это разбиение таблицы на две или более с целью ликвидации дублирования данных и потенциальной их противоречивости. Окончательная цель нормализации сводится к получению такого проекта базы данных, в котором «каждый факт появляется лишь в одном месте».

Каждая таблица в реляционной модели удовлетворяет условию, в соответствии с которым на пересечении любой строки и столбца таблицы всегда находится единственное атомарное значение, и никогда не может быть множества таких значений. Говорят, что таблица, удовлетворяющая такому условию, находится в *первой нормальной форме*, сокращенно 1НФ.

Теперь в дополнение к 1НФ можно определить дальнейшие уровни нормализации – вторую нормальную форму (2НФ), третью нормальную форму (3НФ) и т.д. По существу, таблица находится во 2НФ, если она находится в 1НФ и удовлетворяет, кроме того, некоторому дополнительному условию, суть которого будет рассмотрена ниже. Таблица находится в 3НФ, если она находится в 2НФ и, помимо этого, удовлетворяет еще другому дополнительному условию и т.д.

Теория нормализации основывается на наличии той или иной зависимости между столбцами таблицы. Рассмотрим два вида таких зависимостей: *функциональные* и *многозначные*.

*Функциональная зависимость*, по сути, является связью типа «многие к одному» между множествами атрибутов (столбцов) рассматриваемого отношения.

Например, в таблице «Учебный план» (см. рис. 6.3) столбцы Дисциплина, Семестр и Форма отчетности функционально зависят от ключа № (порядковый номер) в учебном плане, а в таблице «Результаты сессии» столбец Оценка функционально зависит от составного ключа (Студент, Учебный план).

*Многозначная зависимость.* Говорят, что один атрибут таблицы многозначно определяет другой атрибут той же таблицы, если для каждого значения первого атрибута существует хорошо определенное множество соответствующих значений второго атрибута.

В качестве примера рассмотрим фрагмент таблицы «Прием экзаменов (зачетов)», изображенный на рис. 6.4. Таблица отражает связь дисциплины и формы отчетности с фамилией преподавателя. В этой таблице существует многозначная зависимость «Дисциплина - Преподаватель»: дисциплину «Математический анализ» ведут несколько преподавателей (Раков И. И., Рыбин К. К., Карпов К. Ю.) и, соответственно, все они могут участвовать в приеме экзаменов (зачетов). Другая многозначная зависимость – «Дисциплина – Форма отчетности»: по одной и той же дисциплине может проводиться и экзамен, и зачет. При этом Форма отчетности и Преподаватель не связаны функциональной зависимостью, что приводит к появлению избыточности (чтобы добавить фамилию еще одного преподавателя, придется ввести в таблицу две новых строки).

Дисциплина	Преподаватель	Форма отчетности
Математический анализ	Раков И. И.	экзамен
Математический анализ	Рыбин К. К.	экзамен
Математический анализ	Карпов К. Ю.	экзамен
Математический анализ	Раков И. И.	зачет
Математический анализ	Рыбин К. К.	зачет
Математический анализ	Карпов К. Ю.	зачет

Рис. 6.4. Фрагмент таблицы «Прием экзаменов (зачетов)»

### 6.3. Нормальные формы

Мы уже говорили о первой нормальной форме (1НФ). Теперь приведем ее более строгое определение, а также определения других нормальных форм.

*Таблица находится в первой нормальной форме (1НФ)* тогда и только тогда, когда в любом допустимом значении этой таблицы каждая ее строка содержит только одно значение для каждого атрибута (столбца).

Из таблиц, рассмотренных ранее, не удовлетворяет этим требованиям (т.е. не находится в 1НФ) только таблица рис. 6.1.

*Таблица находится во второй нормальной форме (2НФ)*, если она удовлетворяет определению 1НФ и все ее атрибуты (столбцы), не входящие в первичный ключ, связаны полной функциональной зависимостью с первичным ключом.

Не удовлетворяют этим требованиям таблицы, представленные на рис. 6.1 и на рис. 6.2. Таблица 6.2 имеет составной первичный ключ (ФИО студента, Семестр, Дисциплина, Форма отчетности) и содержит множество не ключевых атрибутов (Оценка, Количество часов, ФИО преподавателя), зависящих лишь от той или иной части первичного ключа. Так, атрибуты Количество часов и ФИО преподавателя зависят только от атрибутов Семестр, Дисциплина, Форма отчетности. Следовательно, эти атрибуты не связаны с первичным ключом полной функциональной зависимостью.

Ко второй нормальной форме приведены все таблицы рис. 6.3.

*Таблица находится в третьей нормальной форме (3НФ)*, если она удовлетворяет определению 2НФ и ни один из ее не ключевых атрибутов не связан функциональной зависимостью с любым другим не ключевым атрибутом.

Таблица «Учебный план» (рис. 6.3), очевидно, не находилась бы в третьей нормальной форме, если включала бы в себя столбец Должность преподавателя. В этом случае необходимо было бы провести декомпозицию таблицы «Учебный план» и в результате получить таблицы дополнительную таблицу «Кадровый состав» с атрибутами №, ФИО преподавателя, Должность преподавателя.

Следует отметить, что в таблице «Учебный план» на самом деле существует функциональная зависимость между атрибутами «Кол-во часов» и «ФИО преподавателя», с одной стороны, и совокупностью атрибутов «Семестр», «Дисциплина» и «Форма отчетности» - с другой. Однако тройка атрибутов («Семестр», «Дисциплина» и «Форма отчетности») в свою очередь может выступать в качестве первичного ключа, который представлен в таблице атрибутом «Порядковый номер». Чтобы избежать в процессе нормализации подобных противоречий, Кодд и Бойс обосновали и предложили более строгое определение для 3НФ, которое учитывает, что в таблице может быть несколько первичных ключей.

Таблица находится в *нормальной форме Бойса-Кодда (НФБК)* тогда и только тогда, когда любая функциональная зависимость между ее атрибутами сводится к полной функциональной зависимости от *возможного* первичного ключа.

В соответствии с этой формулировкой таблица «Учебный план» находится в НФБК или в 3НФ.

В следующих нормальных формах (4НФ и 5НФ) учитываются не только функциональные, но и многозначные зависимости между атри-

бутами. Для того чтобы привести определения этих нормальных форм, введем понятие полной декомпозиции таблицы.

*Полной декомпозицией таблицы* называют такую совокупность произвольного числа ее проекций, соединение которых полностью совпадает с содержимым таблицы.

Например, применив операцию соединения (см. п. 3.6.2) к таблицам, приведенным на рис. 6.3, можно получить таблицу, приведенную на рис. 6.2. Следовательно, совокупность таблиц рис. 6.3 является полной декомпозицией таблицы "Сессия", приведенной на рис. 6.2.

Далее дадим определения высших нормальных форм.

Таблица находится в *пятой нормальной форме (5НФ)* тогда и только тогда, когда в каждой ее полной декомпозиции все проекции содержат возможный ключ. Таблица, не имеющая ни одной полной декомпозиции, также находится в 5НФ.

*Четвертая нормальная форма (4НФ)* является частным случаем 5НФ, когда полная декомпозиция должна быть соединением ровно двух проекций. На практике не просто подобрать реальную таблицу, которая находилась бы в 4НФ, но не была бы в 5НФ.

#### **6.4. Процедура нормализации**

В соответствии с определениями нормальных форм можно дать и другое определение нормализации: *нормализация* - это процесс последовательной замены таблицы ее полными декомпозициями до тех пор, пока все они не будут находиться в 5НФ. Однако оказывается, что достаточно привести таблицы к НФБК и с большой гарантией считать, что они находятся в 5НФ (это утверждение нуждается в проверке, но пока не существует эффективного алгоритма такой проверки).

Рассмотрим процедуру приведения таблиц к НФБК.

Такая процедура основывается на том, что единственными функциональными зависимостями в любой таблице должны быть зависимости вида  $A \rightarrow K$ , где  $K$  - первичный ключ, а  $A$  - некоторый атрибут. Принцип "Один факт в одном месте" говорит о том, что не должно существовать в рамках таблицы никаких других функциональных зависимостей. Цель нормализации и состоит в удалении этих "других" функциональных зависимостей.

Рассмотрим два возможных случая.

1. Таблица имеет составной первичный ключ вида, скажем,  $(K1, K2)$ , и включает также атрибут  $A$ , который функционально зависит от части этого ключа (например, от  $K2$ ), но не от полного ключа. В этом случае рекомендуется сформировать другую таблицу, содержащую атрибуты  $K2$  и  $A$  (первичный ключ -  $K2$ ), и удалить атрибут  $A$  из первоначальной таблицы:



2. Таблица имеет первичный (возможный) ключ  $K$ , атрибут  $A_1$ , который не является возможным ключом, но функционально зависит от  $K$ , и другой не ключевой атрибут  $A_2$ , который функционально зависит от  $A_1$ . Решение здесь, по существу, то же самое, что и прежде - формируется другая таблица, содержащая атрибуты  $A_1$  и  $A_2$ , с первичным ключом  $A_1$ , а атрибут  $A_2$  удаляется из первоначальной таблицы.

Таким образом, повторяя применение двух рассмотренных правил, для любой заданной таблицы почти во всех реальных практических ситуациях можно получить в конечном счете множество таблиц, которые находятся в НФБК и не содержат каких-либо функциональных зависимостей вида, отличного от  $A \rightarrow K$ .

Применим приведенные правила для полной нормализации универсального отношения «Сессия» (рис. 6.2).

*1. Определение первичного ключа таблицы.*

Предположим, что каждый студент сдает один раз экзамен (зачет) по дисциплине учебного плана и получает оценку. Дисциплина учебного плана однозначно характеризуется наименованием, номером семестра, за который отчитывается студент, и формой отчетности (т.к. учебный план предусматривает сдачу и экзамена, и зачета по одной и той же дисциплине в рамках одного семестра). Тогда в качестве первичного ключа отношения «Сессия» можно использовать следующий набор атрибутов:

ФИО студента, Дисциплина, Семестр, Форма отчетности.

*2. Выявление атрибутов, функционально зависящих от части составного ключа.*

Каждый из атрибутов - ФИО преподавателя и Количество часов - функционально зависит только от атрибутов Дисциплина, Семестр и Форма отчетности, т.е. этот атрибут вместе с совокупностью атрибутов первичного ключа составит вторую таблицу:

Учебный план (Дисциплина, Семестр, Форма отчетности, Количество часов, ФИО преподавателя).

Из исходной таблицы при этом удаляются атрибуты Количество часов и ФИО преподавателя:

Результаты сессии (ФИО студента, Дисциплина, Семестр, Форма отчетности, Оценка).

Составной первичный ключ, повторяющийся в обеих таблицах, приводит к избыточности при дублировании информации сразу трех столбцов, поэтому кажется целесообразным ввести дополнительный атрибут - № (порядковый номер) – в таблицу «Учебный план» и использовать именно его в качестве первичного ключа. Тогда таблицы примут следующий вид:

Учебный план (№ Уч. план, Дисциплина, Семестр, Форма отчетности, Кол-во часов, ФИО преподавателя).

Результаты сессии (ФИО студента, № Уч. план, Оценка).



Такой декомпозиции на самом деле достаточно для того, чтобы преобразовать исходную таблицу к совокупности нормализованных таблиц (обе полученные таблицы приведены к НФБК), однако полученный проект может быть улучшен путем введения дополнительных таблиц «Дисциплины», «Студенты» и «Преподаватели». Далее мы приведем пример проектирования БД не на основе декомпозиции универсального отношения, а путем построения инфологической модели предметной области и преобразования ее на логическом уровне в реляционную модель данных.

### **6.5. Пример проектирования реляционной БД**

Рассмотрим следующую задачу: пусть необходимо обеспечить сбор и обработку данных по результатам сдачи экзаменов и зачетов студентами факультета. Организация данных должна обеспечивать:

- выполнение текущего учебного плана;
- формирование ведомостей по отдельным дисциплинам для групп студентов;
- формирование листов зачетных книжек студентов;
- формирование сводной ведомости курса;
- расчет среднего балла по дисциплинам и т.п.

Приведем этапы построения инфологической и даталогической моделей (ER-диаграммы и реляционной схемы) для решения такой задачи.

#### **6.5.1. Построение ER-диаграммы**

Представим предметную область как взаимодействие двух сущностей - «Дисциплина учебного плана» и «Студент»: каждый студент сдает экзамен или зачет по некоторой дисциплине учебного плана и получает оценку, которая должна быть зафиксирована в модели данных.

«Дисциплина учебного плана» с точки зрения решаемой задачи должна быть представлена группой свойств, позволяющих характеризовать дисциплину в рамках каждого отдельного семестра: наименование дисциплины, семестр, количество часов, форма отчетности (экзамен или зачет) и данные о преподавателе, читающем дисциплину. Необходимость задания таких свойств обусловлена, с одной стороны, задачей организации хранения результатов сдачи экзаменов и зачетов (наименование дисциплины, семестр и форма отчетности), и с другой стороны – задачей формирования листов зачетных книжек (количество часов и данные о преподавателе). Отдельный экземпляр такой сущности однозначно идентифицируется тройкой свойств – наименование дисциплины, семестр и форма отчетности.

Сущность «Студент» для обеспечения выполнения объявленных функций должна характеризоваться следующими свойствами: фамилия, имя, отчество и номер группы. Однако следует отметить, что даже набор значений всех этих свойств не может однозначно характеризовать экземпляр сущности, т.к. можно предполагать наличие в одной группе полных однофамильцев. Таким образом, для идентификации отдельного экземпляра сущности необходимо ввести дополнительное (ключевое) свойство – идентификационный номер студента.

Определим для сущности «Студент» еще два дополнительных свойства, которые не будут непосредственно обеспечивать решение поставленной задачи, но могут служить для реализации дополнительных (сервисных) функций (например, организации почтовой или телефонной связи): домашний адрес и номер телефона. Свойство «Домашний адрес», являясь по сути составным, будет на самом деле рассматриваться в контексте решаемых задач как простое, а свойство «Номер телефона» - как условное.

Взаимодействие сущностей реализуется связью «Сводная ведомость», т.е. Студент сдает экзамен (зачет) по Дисциплине учебного плана. Мощность связи – «многие ко многим» (М:М). Для идентификации связи отдельных экземпляров сущностей в этом случае необходимо наличие у связи следующих дополнительных свойств: оценка и дата сдачи экзамена (зачета).

ER-диаграмма рассматриваемой задачи представлена на рис 6.5.

Построенная ER-диаграмма находится в *первой нормальной форме*, т.к. сущности не имеют повторяющихся групп свойств (см. п. 5.4.4). Однако при рассмотрении свойств сущности «Дисциплина учебного плана» можно заметить, что свойство «Преподаватель» зависит только от части ключевых свойств – а именно от свойств «Наименование дисциплины» и, возможно, «Форма отчетности». Следовательно, для того, чтобы привести ER-диаграмму ко второй нормальной форме, необходимо выделить свойство «Преподаватель» в отдельную сущность.

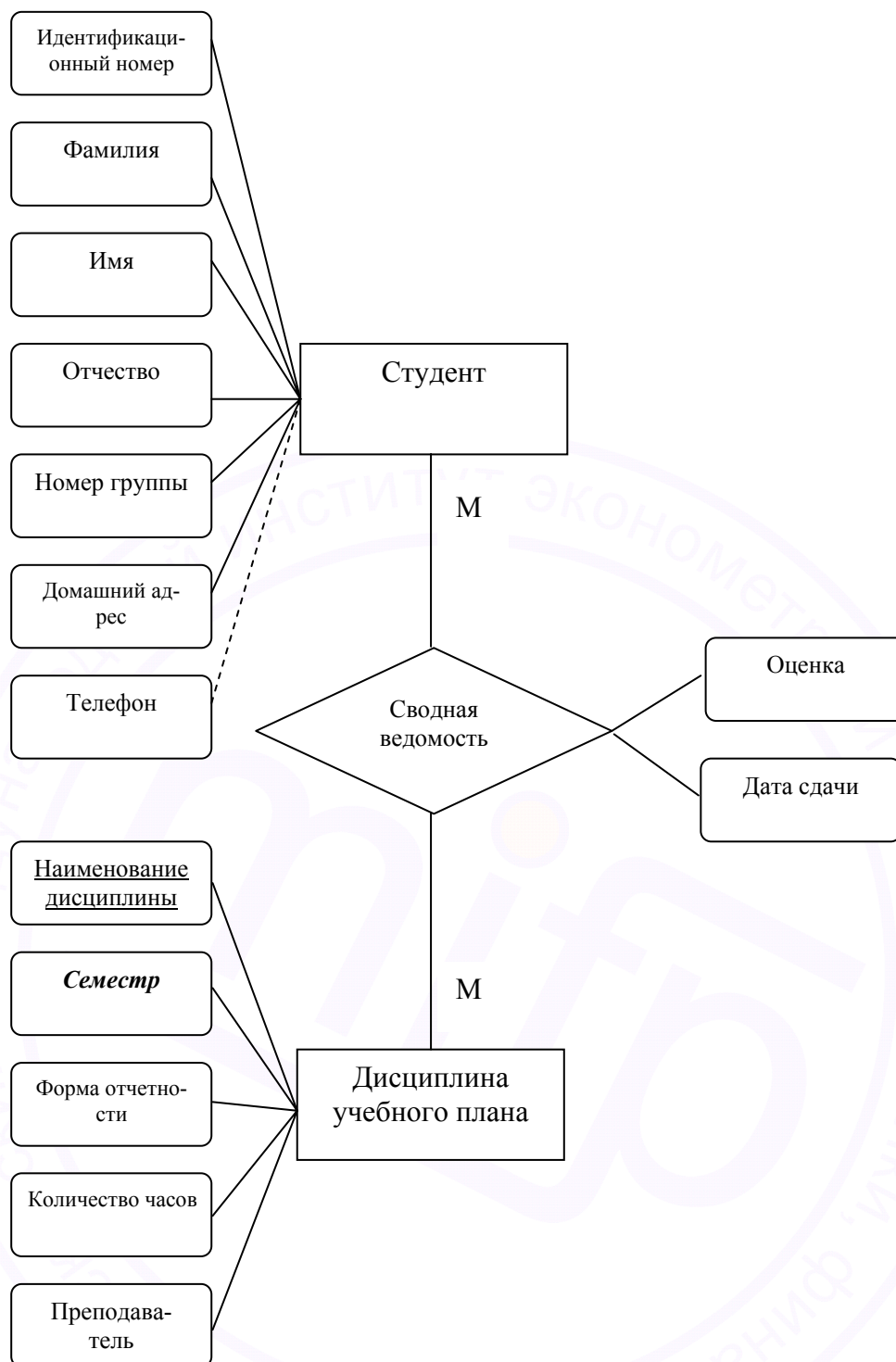


Рис. 6.5. ER-диаграмма рассматриваемой задачи.

Новая сущность «Преподаватель» характеризуется группой основных свойств - фамилия, имя, отчество, и группой дополнительных свойств – кафедра, должность, домашний адрес и телефон. Так же, как и для сущности «Студент», для сущности «Преподаватель» необходимо ввести дополнительное (ключевое) свойство – идентификационный номер преподавателя.

Взаимодействие новой сущности с сущностью «Дисциплина учебного плана» осуществляется посредством новой связи «Читает». Мощность связи – «Многие к одному» (M:1), т.е. несколько дисциплин учебного плана может читать один преподаватель.

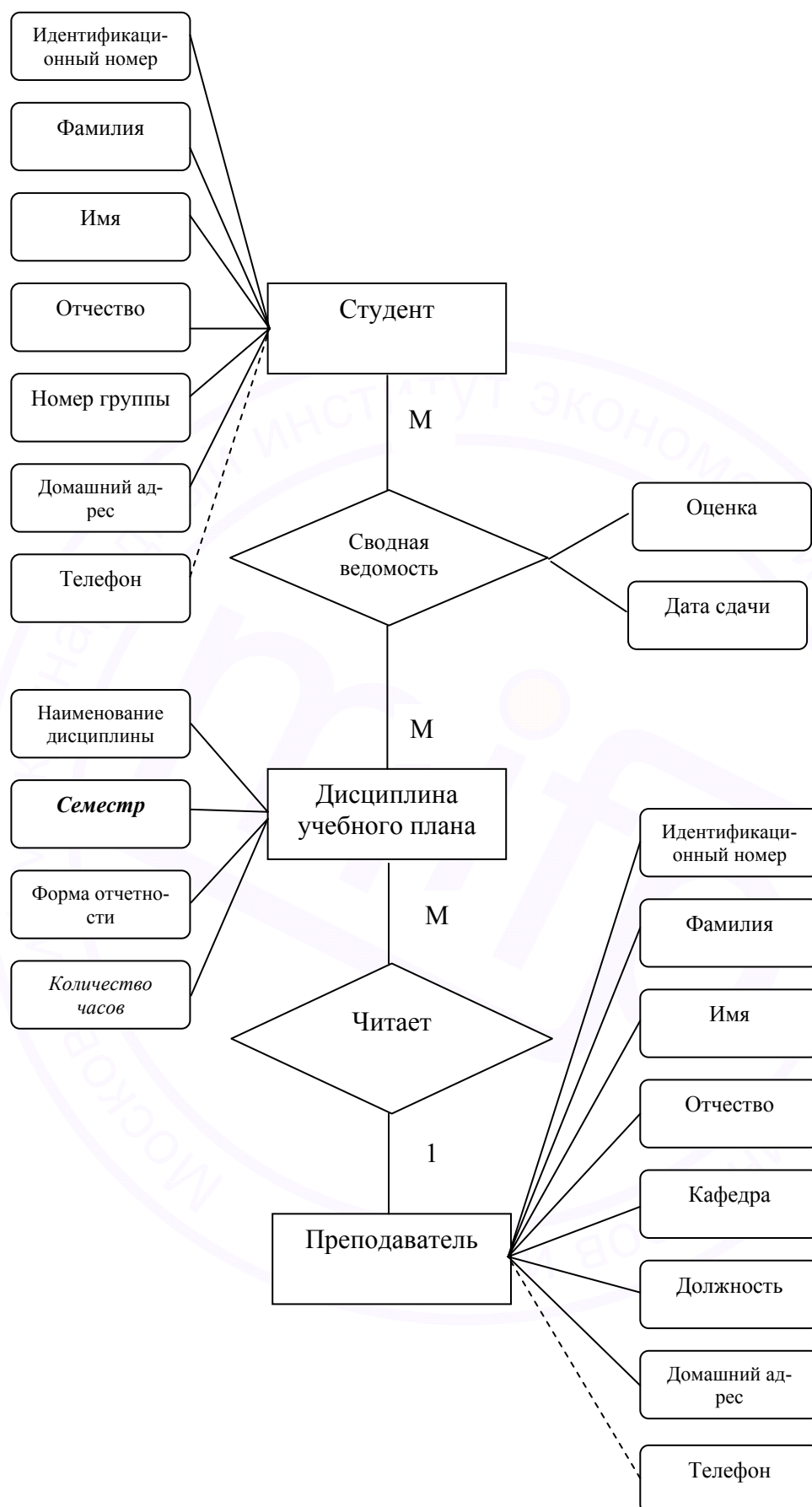


Рис. 6.6. Нормализованная ER-диаграмма

Измененная ER-диаграмма представлена на рис. 6.6. Новый вариант ER-диаграммы находится в *третьей нормальной форме*, т.к. сущности не имеют свойств, зависящих от неключевых.

### 6.5.2. Построение реляционной схемы

Следующий этап проектирования – построение даталогической модели. В рассматриваемом случае задача этого этапа – преобразование ER-диаграммы в реляционную схему (см. п. 5.5.1).

Реляционный подход, в основе которого лежит принцип разделения данных и связей, обеспечивает с одной стороны независимость данных, а с другой – более простые способы хранения и обновления.

Первые шаги преобразования состоят в превращении каждой сущности в отношение (таблицу). Связь типа M:M, которую называют «сущность-связь», тоже превращается в отдельное отношение. Каждое свойство становится атрибутом - столбцом соответствующей таблицы.

После реализации этих шагов получаем реляционную схему, изображенную на рис. 6.7, где представлены таблицы «Студенты», «Сводная ведомость», «Учебный план» и «Кадровый состав», отображающие соответственно сущности «Студент», «Сводная ведомость», «Дисциплина учебного плана» и «Преподаватель».

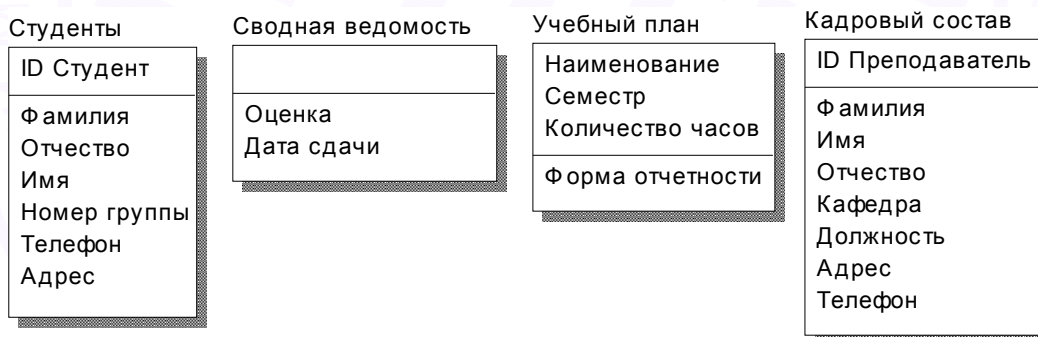


Рис. 6.7. Реляционная схема после первого этапа преобразования

Далее необходимо преобразовать связи во внешние ключи. Связь «многие ко многим», реализуемая отношением «Сводная ведомость», должна содержать уникальные идентификаторы сущностей – участников связи. При этом, если для однозначной идентификации студента достаточно добавить в таблицу столбец *ID\_Студент*, то однозначная идентификация дисциплины потребует добавления в таблицу столбцов *Наименование*, *Семестр* и *Форма\_отчетности*. Хранение всей этой информации явно приведет к *избыточности* данных и их потенциальной *противоречивости* (например, если при переносе дисциплины на другой семестр обновить только строку таблицы «Учебный план», то содержимое таблицы «Сводная ведомость» станет не актуальным).



Для ликвидации избыточности и потенциальной противоречивости данных добавим в таблицу «Учебный план» столбец *ID\_План*, содержимое которого будет однозначно идентифицировать каждую строку таблицы. Теперь этот новый столбец станет первичным ключом, и одноименный столбец должен быть добавлен в таблицу «Сводная ведомость».

Связь «Читает» предполагает добавление в таблицу «Учебный план» столбца *ID\_Преподаватель*. Реляционная схема со связями представлена на рис. 6.8.



Рис. 6.8. Реляционная схема со связями

### 6.5.3. Нормализация таблиц

Все построенные таблицы находятся в *первой нормальной форме*, т.к. каждый столбец таблицы неделим и в рамках одной таблицы нет столбцов с одинаковыми по смыслу значениями.

Таблица «Сводная ведомость» через столбцы *ID\_Студент* и *ID\_План* связывает информацию о студенте с информацией о конкретной дисциплине и фиксирует оценку, полученную студентом. Оценка и дата сдачи экзамена (зачета) однозначно зависят от содержимого столбцов *ID\_Студент* и *ID\_План*, которые представляют собой составной первичный ключ. Таким образом, все таблицы имеют первичные ключи, которые однозначно определяют строки и не избыточны, и можно говорить о том, что таблицы находятся *во второй нормальной форме*.

Рассмотрим подробнее таблицу «Учебный\_план», которая содержит перечень дисциплин текущего учебного плана. Первичным ключом таблицы служит столбец *ID\_План*, который однозначно характеризует каждую дисциплину учебного плана с точностью до семестра, т.е. для дисциплин, протяженность изучения которых более одного семестра, в таблице будет отведено столько строк, сколько семестров длится изуче-

ние дисциплины. Тогда хранение наименований дисциплин в таблице «Учебный\_план» становится избыточным: например, если изучение английского языка длится 6 семестров, то наименование «Английский язык» будет повторено в 6 записях и есть вероятность сделать 6 различных ошибок при вводе одного и того же наименования.

Чтобы избежать этого, проведем декомпозицию отношения «Учебный план», выделив наименования дисциплин в отдельное отношение. В результате получим дополнительную таблицу «Дисциплины» со столбцами *ID\_Дисциплина* и *Наименование*, а столбец *Наименование* в таблице «Учебный\_план» заменим столбцом *ID\_Дисциплина*, сформировав тем самым вторичный ключ, связывающий новую таблицу с таблицей «Учебный\_план».

Теперь можно говорить о базе данных «Сессия», реляционная схема которой представлена следующими пятью таблицами:

«Студенты» - содержит по одной строке для каждого из студентов;

«Учебный\_план» - содержит по одной строке для отдельной дисциплины отдельного семестра;

«Дисциплины» - содержит по одной строке для наименования дисциплины;

«Сводная\_ведомость» - содержит по одной строке для каждого результата сдачи отдельным студентом отдельной дисциплины;

«Кадровый\_состав» - содержит по одной строке для каждого из преподавателей.

На рис. 6.9. в графической форме изображены перечисленные таблицы, их столбцы, первичные и внешние ключи. Задание первичных и внешних ключей сопровождается построением дополнительных структур – индексов, обеспечивающих быстрый доступ к данным через значение ключа.

Все таблицы базы данных «Сессия» находятся в третьей нормальной форме:

каждый столбец таблицы неделим и в рамках одной таблицы нет столбцов с одинаковыми по смыслу значениями (1НФ);

первичные ключи однозначно определяют запись и не избыточны, все поля каждой из таблиц зависят от ее первичного ключа (2НФ);

значение любого поля, не входящего в первичный ключ, не зависит от значения другого поля, тоже не входящего в первичный ключ (3НФ).

Следующий этап проектирования – определение доменов (типов) данных, хранящихся в столбцах таблиц. Параллельно с заданием типа необходимо сформулировать ограничения целостности, связанные с типом - перечень допустимых значений типа.

Исходя из особенностей данных и их функционального назначения, требуется задать способ представления и границы возможных изменений для каждого из столбцов таблиц. При этом необходимо ответить на вопрос, данные каких типов должны храниться в столбцах и какова их максимальная длина (например, если в столбце предполагается хранить процентные значения, то достаточно будет целого типа данных длиной 1 байт, так как диапазон возможных значений от 0 до 255; если для данных столбца выбирается тип «строка символов», то желательно указать максимальный размер данных столбца и т.п.).

Далее, в каждой таблице должны быть выделены столбцы, которые *обязательно должны быть заполнены* при создании отдельной строки таблицы. Задание такого ограничения целостности не позволит, например, ввести в таблицу «Студенты» строку, в которой не указан номер группы. Если подобные ограничения целостности не будут заданы, в таблице могут появиться строки, которые не будут учтены при выполнении функций по обработке данных: появление в таблице «Студенты» строки без номера группы приведет к ошибке при формировании ведомости.

Следующий важный момент - задание для столбцов значений по умолчанию. Значение по умолчанию впоследствии будет автоматически вводиться в указанный столбец для каждой строки таблицы. Например, в столбец *Дата\_сдачи* таблицы «Сводная ведомость» при заполнении очередной строки может автоматически заноситься текущая дата.

Ниже представлены таблицы базы данных «Сессия» с типами данных столбцов и предлагаемыми ограничениями целостности.

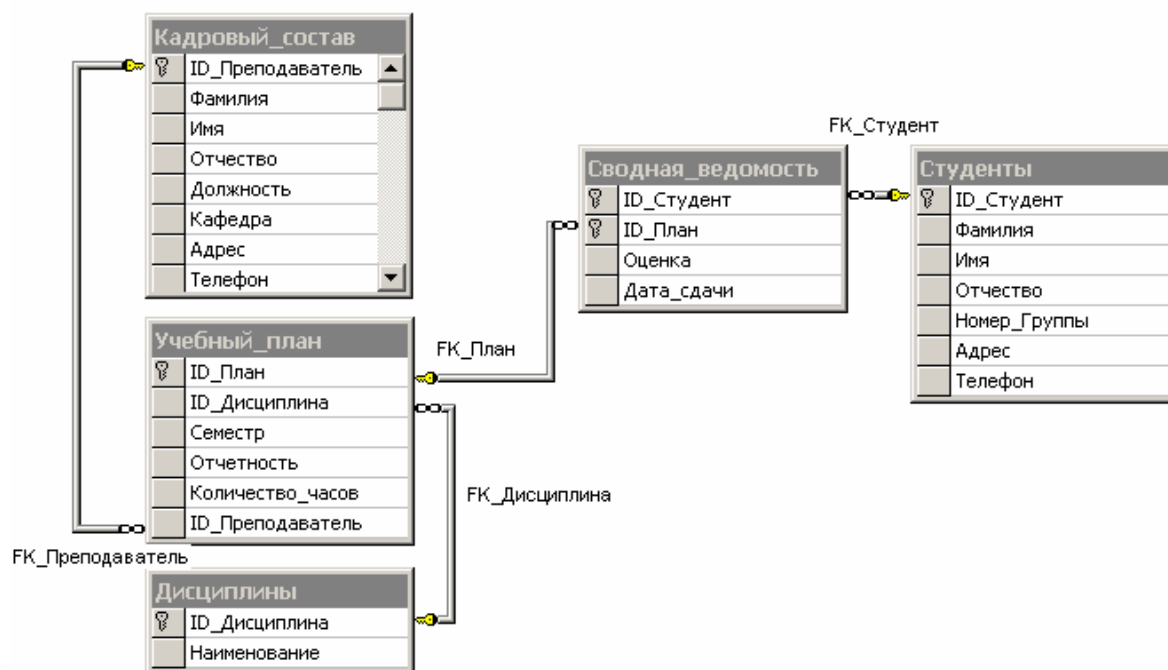


Рис. 6.9. Структура базы данных «Сессия»

**Таблица «Студенты»**

<b>Наименование столбца</b>	<b>Тип данных</b>	<b>Ограничения</b>
ID_Студент	Целое число	Значение уникально
Фамилия	Строка символов размером 30	Значение не должно быть пустым
Имя	Строка символов размером 15	Значение не должно быть пустым
Отчество	Строка символов размером 20	Значение не должно быть пустым
Номер группы	Целое число	Значение не должно быть пустым
Адрес	Строка символов размером 30	
Телефон	Строка символов размером 8	

**Таблица «Дисциплины»**

<b>Наименование столбца</b>	<b>Тип данных</b>	<b>Ограничения</b>
ID_Дисциплина	Целое число	Значение уникально
Наименование	Строка символов размером 20	Значение уникально

**Таблица «Кадровый\_состав»**

<b>Наименование столбца</b>	<b>Тип данных</b>	<b>Ограничения</b>
ID_Преподаватель	Целое число	Значение уникально
Фамилия	Строка символов размером 30	Значение не должно быть пустым
Имя	Строка символов размером 15	Значение не должно быть пустым
Отчество	Строка символов размером 20	Значение не должно быть пустым
Должность	Строка символов размером 20	Значение не должно быть пустым
Кафедра	Строка символов размером 3	Значение не должно быть пустым
Адрес	Строка символов размером 30	
Телефон	Строка символов размером 8	

**Таблица «Учебный план»**

Наименование столбца	Тип данных	Ограничения
ID_План	Целое число	Значение уникально
ID_Дисциплина	Целое число	Значение не должно быть пустым
Семестр	Целое число	Значение не должно быть пустым и находится в интервале от 1 до 10
Количество часов	Целое число	
ID_Преподаватель	Целое число	

**Таблица «Сводная ведомость»**

Наименование столбца	Тип данных	Ограничения
ID_Студент	Целое число	Значение не должно быть пустым
ID_План	Целое число	Значение не должно быть пустым
Оценка	Целое число	Значение не должно быть пустым и должно находиться в интервале от 0 до 5
Дата сдачи	Дата-время	Значение не должно быть пустым, по умолчанию – текущая дата

Все примеры использования языка SQL (Structured Query Language), рассматриваемые в следующей главе, будут построены на основе этой учебной базы данных «Сессия».

#### *Контрольные задания*

1. Приведите примеры дополнительных функций, которые могут быть реализованы с помощью таблиц БД «Сессия».
2. Проведите декомпозицию отношения «Кадровый состав», выделив отношение «Штатное расписание».
3. Проведите декомпозицию отношения «Кадровый состав», выделив отношение «Структура факультета».
4. Какие изменения должны быть внесены в структуру БД «Сессия» для реализации функции назначения стипендии?



## Глава 7. Введение в SQL

Изначально создаваемый как инструмент для выборки и представления данных, содержащихся в базе данных<sup>44</sup>, SQL сегодня представляет собой нечто гораздо большее. Несмотря на то, что выборка данных по-прежнему остается одной из наиболее важных функций SQL, сейчас этот язык используется для реализации всех функциональных возможностей, необходимых для управления БД, в том числе и для:

- *организации данных* - SQL позволяет определять и изменять структуру представления данных, а также устанавливать отношения;
- *обработки данных* - SQL позволяет изменять содержимое базы данных: добавлять новые данные, удалять или обновлять уже имеющиеся в ней данные;
- *управления доступом* - SQL позволяет ограничивать возможности пользователя по чтению и изменению данных (защита данных от несанкционированного доступа) и координировать их совместное использование пользователями, работающими параллельно.

Таким образом, хотя SQL и не объявляется как полноценный язык программирования<sup>45</sup>, он является достаточно полным и мощным языком для управления взаимодействием с СУБД. SQL является *подъязыком* баз данных, предназначенным для управления базами данных. Несмотря на не совсем точное название, SQL на сегодняшний день является *единственным* стандартным языком для работы с реляционными базами данных.

Операторы SQL *встраиваются* в базовый язык, например PASCAL, FORTRAN или C, и дают возможность получать доступ к базам данных из прикладных программ. Кроме того, из многих языков программирования операторы SQL можно посылать СУБД в явном виде, используя *интерфейс вызовов функций*.

Официальный стандарт языка SQL был опубликован в 1986 году Американским институтом национальных стандартов (ANSI) и Международной организацией по стандартам (International Standards Organization — ISO), а в 1992 году значительно расширен. Стандарт X/OPEN для переносимой среды программирования на основе операционной системы UNIX также включает в себя SQL в качестве языка для доступа к базам данных. Консорциум поставщиков компьютерного оборудования и баз данных (SQL Access Group) определил для SQL стандартный интерфейс вызовов функций, который является основой протокола ODBC

---

<sup>44</sup> Практика сегодняшнего дня показывает, что стандартный SQL реализуется в полном объеме только для реляционных СУБД.

<sup>45</sup> В стандартном SQL89, реализованном в полном объеме для реляционных СУБД, нет оператора проверки условий и ветвления, нет оператора перехода, нет операторов циклов и т.д. Однако, SQL большинства промышленных СУБД содержит эти и многие другие операторы, позволяющие создавать полноценные процедуры обработки данных.

компании Microsoft и входит также в стандарт X/OPEN. Эти стандарты de facto являются официальным одобрением SQL, и именно они ускорили завоевание им рынка.

Многие из членов комитетов по стандартизации ANSI и ISO представляли фирмы-поставщики различных СУБД, в каждой из которых был реализован собственный диалект SQL. Как и диалекты человеческого языка, диалекты SQL были в основном похожи друг на друга, однако несовместимы в деталях. Во многих случаях комитет просто игнорировал существующие различия и не стандартизировал некоторые части языка, определив, что они реализуются по усмотрению разработчика. Этот подход позволил объявить большое число реализаций SQL совместимыми со стандартом, однако сделал сам стандарт относительно слабым.

Чтобы заполнить эти пробелы, комитет ANSI продолжил свою работу и создал проект нового, более жесткого стандарта SQL2. В отличие от стандарта 1989 года, проект SQL2 предусматривал возможности, выходящие за рамки таковых, уже существующих в реальных коммерческих продуктах.

Перечислим эти отличия SQL2.

*Коды ошибок.* В стандарте SQL2 определены стандартные коды ошибок, которые возвращают операторы SQL при возникновении ошибок.

*Типы данных.* В стандарте SQL2 упомянуты многие стандартные типы данных (например, символьные строки переменной длины, дата и время, а также денежные единицы), однако отсутствуют "новые" типы данных, такие как графические и мультимедийные объекты.

*Системные таблицы.* В стандарте SQL-89 умалчивается о системных таблицах, в которых содержится информация о структуре самой базы данных. Поэтому каждый поставщик создавал собственные системные таблицы, и их структура отличается даже в четырех реализациях SQL компании IBM. В SQL2 системные таблицы стандартизированы.

*Интерактивный SQL.* В стандарте SQL-89 определен только *программный SQL*, используемый прикладной программой, но не *интерактивный SQL*. Например, оператор SELECT, предназначенный для выполнения запросов к базе данных в интерактивном режиме, в стандарте отсутствует.

*Программный интерфейс.* В стандарте SQL2 определен интерфейс встроенного SQL для некоторых языков программирования, но не интерфейс вызова функций.

*Динамический SQL.* В стандарте SQL-89 не описаны элементы SQL, необходимые для разработки приложений общего назначения, таких как генераторы отчетов и программы создания и выполнения запросов. Однако эти элементы, известные под названием *динамический SQL*, имеются почти во всех СУБД и в различных системах значительно отличаются. В стандарт SQL2 входит раздел динамического SQL.

*Семантические отличия.* Поскольку некоторые элементы определены в стандартах как зависящие от реализации, может возникнуть ситуация, когда в результате выполнения одного и того же запроса в двух совместимых СУБД будут получены два различных набора результатов. Отличия результатов обусловлены различиями в обработке значений NULL, разными агрегатными функциями и несовпадением процедур удаления повторяющихся строк.

*Последовательность сравнения.* Стандарт SQL2 позволяет программе или пользователю указывать требуемую последовательность сортировки результатов запроса.

*Структура базы данных.* В стандарте SQL-89 определен язык, операторы которого используются уже после того, как база данных была открыта и подготовлена к работе. Детали именования баз данных и первоначального подключения к ним в разных реализациях сильно отличаются или несовместимы. Стандарт SQL2 в некоторой степени унифицирует этот процесс, хотя и не может полностью ликвидировать все отличия.

Основными направлениями развития SQL2 (и принятие SQL3) являются:

- стандартизация интерфейсов вызова функций;
- стандартизация хранимых процедур;
- добавление объектно-ориентированных возможностей.

## **7.1. Основные понятия и компоненты<sup>46</sup>**

### **7.1.1. Инструкции и имена**

SQL представлен множеством инструкций, каждая из которых предписывает СУБД выполнить определенное действие: создать таблицу, извлечь данные, добавить в таблицу новые данные и т.п. Инструкция SQL начинается с *команды* – ключевого слова, описывающего действие, выполняемое инструкцией. Типичными являются команды CREATE (создать), INSERT (добавить), SELECT (выбрать), DELETE (удалить). Следом за командой указывается одно или несколько *предложений*. Предложение описывает данные, с которыми должна работать инструкция, или уточняют действие, выполняемое инструкцией. Предложения в инструкции делятся на обязательные и необязательные. Каждое предложение начинается с ключевого слова, например – WHERE (где), FROM (откуда), INTO (куда). Многие предложения в качестве параметров содержат имена таблиц или столбцов; некоторые из них могут содержать дополнительные ключевые слова, константы и выражения.

---

<sup>46</sup> Синтаксис приведенных в этой главе примеров соответствует MS SQL Server 7

У каждого объекта в базе данных есть уникальное *имя*. Имена используются в инструкциях SQL и указывают, над каким объектом базы данных инструкция должна выполнить действие. В соответствии со стандартом ANSI/ISO имена в SQL могут содержать от 1 до 18 символов, начинаться с буквы и не должны включать пробелов или специальных символов пунктуации. В стандарте SQL2 максимальное число символов в имени увеличено до 128. На практике в различных СУБД поддержка именования реализована по-разному: в DB2, например, имена пользователей не могут превышать 8 символов, а имена таблиц и столбцов могут быть более длинными. В различных СУБД также существуют и различные подходы к использованию в именах специальных символов.

В инструкциях SQL могут использоваться как полные имена объектов, так и короткие. Полное имя таблицы (в отличие от короткого) содержит имя пользователя и короткое имя таблицы, разделенные точкой: <Имя\_пользователя>.<Имя\_таблицы>

При этом уникальность именования таблицы сохраняется в случае, если в рамках одной базы данных разные пользователи создают таблицы с одинаковыми именами.

Полное имя столбца в свою очередь состоит из полного или короткого имени таблицы, которой принадлежит столбец, и короткого имени столбца, разделенных точкой:

<Имя\_пользователя>.<Имя\_таблицы>.<Имя\_столбца> или  
<Имя\_таблицы>.<Имя\_столбца>

В рамках одной таблицы не может быть определено двух столбцов с одинаковыми именами, но в разных таблицах это возможно. При этом в инструкциях SQL необходимо использовать полное именование столбцов.

### 7.1.2. Типы данных

Современные СУБД позволяют обрабатывать данные разнообразных типов, среди которых наиболее распространенными можно назвать следующие:

Целые числа (INT, SMALLINT). В столбцах, имеющих такой тип данных, обычно хранятся данные о количестве и возрасте сотрудников, идентификаторы.

Десятичные числа (NUMERIC, DECIMAL). В столбцах данного типа хранятся числа, имеющие дробную часть с фиксированным количеством знаков после запятой, например курсы валют и проценты.

Числа с плавающей запятой (REAL, FLOAT). Числа с плавающей запятой представляют больший диапазон действительных значений, чем десятичные числа.

Строки символов постоянной длины (CHAR). В столбцах, имеющих этот тип данных, хранятся имена и фамилии, географические названия, адреса и т.п.



Строки символов переменной длины (VARCHAR). Столбцы этого типа позволяют хранить символьные строки, длина которых изменяется в заданном диапазоне.

Денежные величины (MONEY, SMALLMONEY). Наличие отдельного типа данных для хранения денежных величин позволяет правильно форматировать их и снабжать признаком валюты перед выводом на экран.

Дата и время (DATETIME, SMALLDATETIME). Поддержка особого типа данных для значений дата/время широко распространена в различных СУБД. Как правило, с этим типом данных связаны особые операции и процедуры обработки.

Булевы величины (BIT). Столбцы такого типа данных позволяют хранить логические значения True (1) и False (0).

Длинный текст (TEXT). Многие СУБД поддерживают хранение в столбцах текстовых строк длиной до 32КБ или 64КБ символов, а в некоторых случаях и больше. Это позволяет хранить в базе данных целые документы.

Неструктурированные потоки байтов (BINARY, VARBINARY, IMAGE). Современные СУБД позволяют хранить и извлекать неструктурированные потоки байтов переменной длины. Такой тип данных обычно используется для хранения графических и видео изображений, исполняемых файлов и других неструктурированных данных.

### *7.1.3. Встроенные функции*

Язык SQL содержит так называемые встроенные функции, которые реализуют некоторые наиболее распространенные алгоритмы. Основной особенностью этих функций является возможность их использования при построении выражений.

Встроенные функции, доступные при работе с SQL, можно условно разделить на следующие группы:

- математические функции;
- строковые функции;
- функции для работы с величинами типа дата-время;
- функции конфигурирования;
- системные функции;
- функции системы безопасности;
- функции управления метаданными;
- статистические функции.

В табл. 7.1 приведены наиболее часто используемые функции первых трех групп.



Таблица 7.1

Функция	Назначение
ABS(число)	Вычисляет абсолютную величину числа
ISNUMERIC(выражение)	Определяет, имеет ли выражение числовой тип данных
SIGN(число)	Определяет знак числа
RAND(целое число)	Вычисляет случайное число с плавающей запятой в интервале от 0 до 1
ROUND(число, точность)	Выполняет округление числа с указанной точностью
POWER(число, степень)	Возводит число в степень
SQRT(число)	Извлекает квадратный корень из числа
SIN(угол)	Вычисляет синус угла, указанного в радианах
COS(угол)	Вычисляет косинус угла, указанного в радианах
EXP(число)	Вычисляет экспоненту числа
LOG(число)	Вычисляет натуральный логарифм числа
LEN(строка)	Вычисляет длину строки в символах
LTRIM(строка)	Удаляет пробелы в начале строки
RTRIM(строка)	Удаляет пробелы в конце строки
LEFT(строка, количество)	Возвращает указанное количество символов строки, начиная с самого левого символа
RIGHT(строка, количество)	Возвращает указанное количество символов строки, начиная с самого правого символа
LOWER(строка)	Приводит символы строки к нижнему регистру
UPPER(строка)	Приводит символы строки к верхнему регистру
STR(число)	Выполняет конвертирование числового значения в символьный формат
SUBSTRING(строка, индекс, длина)	Возвращает для строки подстроку заданной длины, начиная с символа заданного индекса
GETDATE()	Возвращает текущее системное время
ISDATE(строка)	Проверяет строку на соответствие одному из форматов даты и времени
DAY(дата)	Возвращает число указанной даты
MONTH(дата)	Возвращает месяц указанной даты
YEAR(дата)	Возвращает год указанной даты
DATEADD(тип, число, дата)	Прибавляет к дате указанное число единиц заданного типа (год, месяц, день, час и т.п.)

#### 7.1.4. Значения NULL

При заполнении таблиц базы данных отдельные элементы в них могут отсутствовать. Например, при заполнении таблицы «Студенты» или «Кадровый\_состав» может быть не задан для некоторых строк номер телефона, тем не менее, строка должна быть введена в таблицу и должна участвовать в запросах на выдачу информации.

SQL поддерживает обработку не определенных (не заданных) данных с помощью использования так называемого отсутствующего значения (NULL). Это значение показывает, что в конкретной строке конкретный элемент данных отсутствует. При этом NULL не является значением данных и в связи с этим не имеет определенного типа. Это всего лишь признак, показывающий, что значение элемента данных не задано.

Правила обработки значений NULL в различных инструкциях и предложениях включены в синтаксис языка.

### 7.2. Ограничения целостности

#### 7.2.1. Первичный ключ таблицы

Всякая таблица обычно содержит один или несколько столбцов, значение или совокупность значений которых *уникально идентифицируют* каждую строку в таблице. Этот столбец (или столбцы) называется *первичным ключом* (Primary Key, PK) таблицы.

Если в первичный ключ входит более одного столбца, значения в пределах одного столбца могут дублироваться, но любая совокупность значений всех столбцов первичного ключа при этом должна быть уникальна. Например, в таблице «Дисциплины» один столбец (*ID\_Дисциплина*) определен как первичный ключ (рис.7.1), а для таблицы «Сводная ведомость» задан составной первичный ключ — в него входят значения столбцов *ID\_Студент* и *ID\_Дисциплина*.

Таблица может иметь *только один* первичный ключ, причем никакой столбец, входящий в первичный ключ, не может хранить значения NULL.

Еще одним назначением первичного ключа является обеспечение ссылочной целостности данных в нескольких таблицах. Естественно, это может быть реализовано только при наличии соответствующих *внешних ключей* (FOREIGN KEY) в других (дочерних) таблицах.


Key	Id	Name	Data Type	Size	Nulls	Default
		ID_План	int	4	<input type="checkbox"/>	
		ID_Дисциплина	int	4	<input type="checkbox"/>	
		Семестр	int	4	<input type="checkbox"/>	
		Отчетность	char	1	<input type="checkbox"/>	
		Количество_часов	int	4	<input checked="" type="checkbox"/>	
		ID_преподавателя	int	4	<input checked="" type="checkbox"/>	

Рис. 7.1. Первичный ключ таблицы «Учебный\_план»

Если по столбцу строится первичный ключ, столбцу должен быть приписан атрибут PRIMARY KEY (ограничение целостности на уровне столбца), например, описание столбца *ID\_План* для таблицы «Учебный\_план» (см. рис. 7.1) может выглядеть следующим образом:

ID\_Дисциплина INTEGER NOT NULL PRIMARY KEY

Первичный ключ может быть также построен и с помощью отдельного предложения PRIMARY KEY (ограничение целостности на уровне таблицы) - путем включения имени (имен) ключевого столбца (столбцов) в качестве параметров. Например, первичный ключ для таблицы «Сводная\_ведомость» (рис. 7.2) может быть задан следующим образом:

PRIMARY KEY (ID\_Дисциплина, ID\_Студент)



Key	Id	Name	Data Type	Size	Nulls	Default
		ID_Студент	int	4	<input type="checkbox"/>	
		ID_Дисциплина	int	4	<input type="checkbox"/>	
		Оценка	int	4	<input type="checkbox"/>	
		Дата_сдачи	datetime	8	<input type="checkbox"/>	

Рис. 7.2. Первичный ключ таблицы «Сводная\_ведомость»

### 7.2.2. Внешний ключ таблицы

Внешний ключ строится в дочерней (зависимой) таблице для соединения родительской (главной) и дочерних таблиц БД.

Это ограничение целостности предназначено для организации ссылочной целостности данных. Внешний ключ связывается с потенциальным первичным ключом в другой таблице. Внешний ключ при этом может ссылаться либо на столбец (или столбцы) с ограничением целост-

ности PRIMARY KEY, либо на столбец (столбцы) с ограничением целостности UNIQUE.

Таблицу, в которой определен внешний ключ, будем называть *зависимой*, а таблицу с первичным ключом — *главной*. Ссылочная целостность данных двух таблиц обеспечивается следующим образом: в зависимую таблицу нельзя вставить строку, если внешний ключ не имеет соответствующего значения в главной таблице, а из главной таблицы нельзя удалить строку, если значение первичного ключа используется в зависимой таблице.

Например, если строка наименования дисциплины удалена из таблицы «Дисциплины», а идентификатор этой дисциплины (*ID\_Дисциплина*) используется в таблице «Учебный\_план», то относительная целостность между этими двумя таблицами будет нарушена — строки таблицы «Учебный\_план» с удаленным идентификатором останутся «осиротевшими». Ограничение FOREIGN KEY предотвращает возникновение подобных ситуаций — удаление строки первичного ключа не состоится.

Столбцы внешнего ключа (в отличие от столбцов первичного ключа) могут содержать значения типа NULL, однако при этом проверка на ограничение FOREIGN KEY будет пропускаться. Задать внешний ключ можно как при создании, так и при изменении таблиц.

Синтаксис определения внешнего ключа следующий:

```
FOREIGN KEY (<список столбцов внешнего ключа>)  
REFERENCES <имя родительской таблицы>  
[[<список столбцов родительской таблицы>]  
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]  
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
```

*Список столбцов внешнего ключа* определяет столбцы дочерней таблицы, по которым строится внешний ключ.

*Имя родительской таблицы* определяет таблицу, в которой описан первичный ключ (или столбец с атрибутом UNIQUE). На этот ключ (столбец) должен ссылаться внешний ключ дочерней таблицы для обеспечения ссылочной целостности.

*Список столбцов родительской таблицы*, определяющий ссылочную целостность, необязателен при ссылке на первичный ключ родительской таблицы. При ссылке в родительской таблице на столбец с атрибутом UNIQUE этот список лучше привести.

Параметры ON DELETE, ON UPDATE задают способы изменения подчиненных записей дочерней таблицы при удалении (ON DELETE) или изменении (ON UPDATE) поля связи в записи родительской таблицы. Перечислим эти способы:

NO ACTION - запрещает удаление/изменение родительской записи при наличии подчиненных записей в дочерней таблице;

CASCADE - при удалении записи родительской таблицы (исполь-

зуется совместно с ON DELETE) происходит удаление всех подчиненных записей в дочерней таблице; при изменении поля связи в записи родительской таблицы (используется совместно с ON UPDATE) происходит изменение на то же значение поля внешнего ключа у всех подчиненных записей в дочерней таблице;

SET DEFAULT - в поле внешнего ключа записей дочерней таблицы заносится значение этого поля по умолчанию, указанное при определении поля (параметр DEFAULT);

SET NULL - в поле внешнего ключа записей дочерней таблицы заносится значение NULL.

Установим связь между таблицами «Студенты», «Учебный\_план» и «Сводная\_ведомость»:

```
ALTER TABLE Сводная_ведомость  
ADD FOREIGN KEY (ID_План)  
REFERENCES Учебный_план
```

```
ALTER TABLE Сводная_ведомость  
ADD FOREIGN KEY (ID_Студент)  
REFERENCES Студенты
```

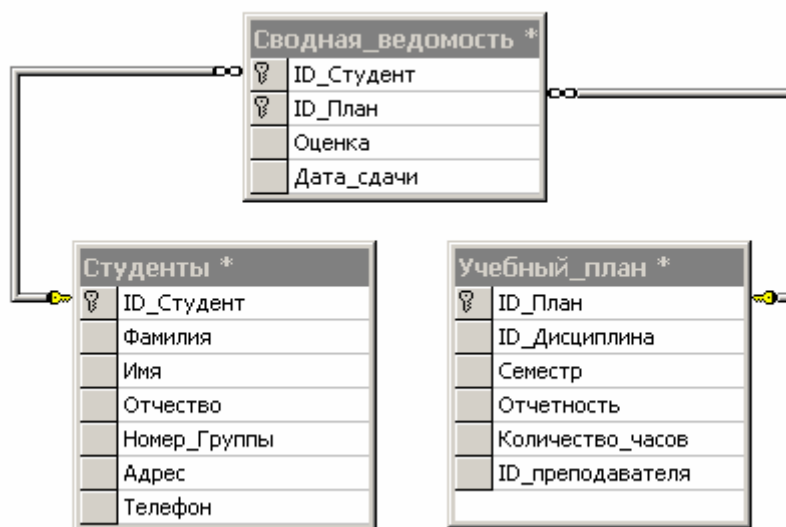


Рис. 7.3. Связь внешнего и первичного ключей

Хотя в рассмотренном примере имена столбцов первичного и внешнего ключей в обеих таблицах совпадают, это не является обязательным. Первичный ключ может быть определен для столбца с одним именем, в то время как столбец, на который наложено ограничение FOREIGN KEY, может иметь совершенно другое имя. Однако лучше давать таким столбцам идентичные названия, чтобы показать связь между ними (рис. 7.3).



### 7.2.3. Определение уникального столбца

Ограничение целостности UNIQUE предназначено для того, чтобы обеспечить уникальность значений в столбце (или нескольких столбцах). Если столбцу приписан атрибут UNIQUE, это означает, что в столбце не могут содержаться два одинаковых значения.

Для ограничения целостности PRIMARY KEY автоматически гарантируется уникальность значений. Однако в каждой таблице можно определить всего один первичный ключ. Если же необходимо дополнительно обеспечить уникальность значений еще в одном или более столбцов помимо первичного ключа, то нужно использовать ограничение целостности UNIQUE.

Ограничение целостности UNIQUE, в отличие от PRIMARY KEY, допускает существование значения NULL. При этом к значению NULL также предъявляется требование уникальности, поэтому в столбце с ограничением целостности UNIQUE допускается существование лишь единственного значения NULL.

Таким образом, ограничение UNIQUE используется в том случае, когда столбец не входит в состав первичного ключа, но, тем не менее, его значение должно всегда быть уникальным. Например, для таблицы «Дисциплины» первичный ключ строится по номеру дисциплины *ID\_Дисциплина*, введенному для сокращения объема первичного ключа и времени поиска по нему (объем ключа по столбцу типа INTEGER много меньше объема ключа по символьному полю). Однако и название дисциплины (столбец *Наименование*) должно быть уникальным, для чего ему приписан атрибут UNIQUE:

```
CREATE TABLE Дисциплины  
(ID_Дисциплина INTEGER NOT NULL PRIMARY KEY,  
Наименование VARCHAR(20) NOT NULL UNIQUE)
```

Уникальность может быть определена и на уровне таблицы:

```
CREATE TABLE Дисциплины  
(ID_Дисциплина INTEGER NOT NULL,  
Наименование VARCHAR(20) NOT NULL,  
PRIMARY KEY (ID_Дисциплина),  
UNIQUE (Наименование))
```

### 7.2.4. Определение проверочных ограничений

Ограничение целостности CHECK задает диапазон возможных значений для столбца. Например, если в столбце хранится процентное значение, то необходимо гарантировать, что оно будет лежать в пределах от 0 до 100. Для этого можно использовать тип данных, допускающий хранение целых значений в диапазоне от 0 до 255, совместно с ог-

раничением целостности CHECK, которое будет обеспечивать соответствующую проверку значений.

Преимуществом ограничения целостности CHECK является возможность определения для одного столбца множества правил контроля значений.

В основе ограничения целостности CHECK лежит проверка логического выражения, которое возвращает значение TRUE (истина) либо значение FALSE (ложь). Если возвращается значение TRUE, то ограничение целостности выполняется, и операция изменения или вставки данных разрешается. Когда же возвращается значение FALSE, то операция изменения или вставки данных отменяется.

Например, для обеспечения правильности задания значения для столбца *Семестр* в таблице «Учебный\_план» (оно должно находиться в диапазоне от 1 до 10) можно использовать следующее логическое выражение:

((Семестр >= 1) OR (Семестр <= 10)))

Ограничение целостности при этом может быть задано на уровне столбца:

Семестр INTEGER NOT NULL CHECK ((Семестр >= 1) OR (Семестр <= 10)),

Или на уровне таблицы:

CHECK ((Семестр >= 1) OR (Семестр <= 10)))

Как уже было сказано, допускается применение нескольких ограничений CHECK к одному и тому же столбцу. В этом случае они будут применены в той последовательности, в какой они указаны в инструкции.

#### 7.2.5. Определение значения по умолчанию

При вводе записи (строки) в таблицу каждый столбец должен содержать какое-либо значение. Если значение для столбца не указано, то столбец заполняется значениями NULL (конечно, если для него разрешено хранение значений NULL). Однако это нежелательно. Наилучшим решением в подобных ситуациях может быть определение для столбца *значений по умолчанию*. Например, часто ноль определяется как значение по умолчанию для числовых столбцов, а «n/a» (не определено) — как значение по умолчанию для символьных столбцов. Таким образом, определение для столбца значения по умолчанию гарантирует автоматическую подстановку этого значения, если при вставке новых строк значение для столбца не указано.

Использование значений по умолчанию довольно удобно, поскольку позволяет ускорить процесс ввода информации. Значительно расширяет область применения значений по умолчанию возможность вызова встроенных функций. Например, если в столбце необходимо ука-

зять дату поступления на работу, то по умолчанию можно воспользоваться функцией GETDATE(). В этом случае, если не указана другая дата, при вводе строки в столбец дат поступления на работу будет помещаться текущая дата.

### **7.3. Управление таблицами**

#### **7.3.1. Команда создания таблицы – CREATE TABLE**

Создание таблицы выполняется при помощи команды CREATE TABLE. Обобщенный синтаксис команды следующий:

```
CREATE TABLE имя_таблицы  
( {<определение_столбца>|<определение_ограничения_таблицы>}  
[,...,{<определение_столбца>|<определение_ограничения_таблицы>} ] )
```

Т.е. после задания имени таблицы через запятую в круглых скобках должны быть перечислены все предложения, определяющие отдельные элементы таблицы – столбцы или ограничения целостности:

имя\_таблицы — идентификатор создаваемой таблицы, который в общем случае строится из имени базы данных, имени владельца таблицы и имени самой таблицы. При этом комбинация имени таблицы и ее владельца должна быть уникальной в пределах базы данных. Если таблица создается не в текущей базе данных, в ее идентификатор необходимо включить имя базы данных.

определение\_столбца — задание имени, типа данных и параметров отдельного столбца таблицы. Названия столбцов должны соответствовать правилам для идентификаторов и быть уникальными в пределах таблицы.

определение\_ограничения\_таблицы – задание некоторого ограничения целостности на уровне таблицы.

#### **Описание столбцов**

Как видно из синтаксиса команды CREATE TABLE, для каждого столбца указывается предложение <определение\_столбца>, с помощью которого и задаются свойства столбца. Предложение имеет следующий синтаксис:

```
<Имя_столбца> <тип_данных>  
[<ограничение_столбца> ] [,...,<ограничение_столбца>]
```

Рассмотрим назначение и использование параметров.

Имя\_столбца — идентификатор, задающий имя столбца таблицы.

тип\_данных — задает тип данных столбца. Если при определении столбца явно не указано ограничение на хранения значений NULL, то будут использованы свойства типа данных, т.е. если выбранный тип

данных позволяет хранить значения NULL, то и в столбце можно будет хранить значения NULL. Если же при определении столбца в команде CREATE TABLE явно будет разрешено или запрещено хранение значений NULL, то свойства типа данных будут перекрыты установленным на уровне столбца ограничением. Например, если тип данных позволяет хранить значения NULL, а на уровне столбца будет установлен запрет, то попытка вставки значения NULL в столбец закончится ошибкой.

ограничение\_столбца — с помощью этого предложения указываются ограничения, которые будут определены для столбца. Синтаксис предложения следующий:

```
<ограничение_столбца>::=[ CONSTRAINT <имя_ограничения > ]  
{ [ DEFAULT <выражение> ]  
| [ NULL | NOT NULL ]  
| [ PRIMARY KEY | UNIQUE ]  
| [ FOREIGN KEY  
REFERENCES <имя_главной_таблицы>[(<имя_столбца> [...,n])]  
[ ON DELETE { CASCADE | NO ACTION } ]  
[ ON UPDATE { CASCADE | NO ACTION } ]  
]  
| [ CHECK (<логическое_выражение>)]  
}
```

Рассмотрим назначение параметров.

CONSTRAINT — необязательное ключевое слово, после которого указывается название ограничения на значения столбца (имя\_ограничения). Имена ограничений должны быть уникальны в пределах базы данных.

DEFAULT — задает значение по умолчанию для столбца. Это значение будет использовано при вставке строки, если для столбца явно не указано никакое значение.

NULL|NOT NULL — ключевые слова, разрешающие (NULL) или запрещающие (NOT NULL) хранение в столбце значений NULL. Если для столбца не задано значение по умолчанию, то при вставке строки с неизвестным значением для столбца будет предприниматься попытка вставки в столбец значения NULL. Если при этом для столбца указано ограничение NOT NULL, то попытка вставки строки будет отклонена, и пользователь получит соответствующее сообщение об ошибке.

PRIMARY KEY — определение первичного ключа на уровне одного столбца (т.е. первичный ключ будет состоять только из значений одного столбца). Если необходимо сформировать первичный ключ на базе двух и более столбцов, то такое ограничение целостности должно быть задано на уровне таблицы. При этом следует помнить, что для каждой таблицы может быть создан только один первичный ключ.



UNIQUE — указание на создание для столбца ограничения целостности UNIQUE, что позволит гарантировать уникальность каждого отдельного значения в столбце в пределах этого столбца. В таблице может быть создано несколько ограничений целостности UNIQUE.

FOREIGN KEY ... REFERENCES — указание на то, что столбец будет служить внешним ключом для таблицы, имя которой задается с помощью параметра <имя\_главной\_таблицы>.

(имя\_столбца [...,n]) — столбец или список перечисленных через запятую столбцов главной таблицы, входящих в ограничение FOREIGN KEY. При этом столбцы, входящие во внешний ключ, могут ссылаться только на столбцы первичного ключа или столбцы с ограничением UNIQUE таблицы.

ON DELETE {CASCADE | NO ACTION} — эти ключевые слова определяют действия, предпринимаемые при удалении строки из главной таблицы. Если указано ключевое слово CASCADE, то при удалении строки из главной (родительской) таблицы строка в зависимой таблице также будет удалена. При указании ключевого слова NO ACTION в подобном случае будет выдана ошибка. Значением по умолчанию является вариант NO ACTION.

ON UPDATE {CASCADE | NO ACTION} — эти ключевые слова определяют действия, предпринимаемые при модификации строки главной таблицы. Если указано ключевое слово CASCADE, то при модификации строки из главной (родительской) таблицы строка в зависимой таблице также будет модифицирована. При использовании ключевого слова NO ACTION в подобном случае будет выдана ошибка. Значением по умолчанию является вариант NO ACTION.

CHECK — ограничение целостности, инициирующее контроль вводимых в столбец (или столбцы) значений.

логическое\_выражение — логическое выражение, используемое для ограничения CHECK.

### Ограничения на уровне таблицы

Синтаксис команды CREATE TABLE предусматривает использование предложения <ограничение\_таблицы>, с помощью которого определяются ограничения целостности на уровне таблицы. Синтаксис предложения следующий:

```
<ограничение_таблицы> ::= [ CONSTRAINT <имя_ограничения> ]  
{ [ { PRIMARY KEY | UNIQUE }  
{(<имя_колонки> [ASC | DESC] [...,n] )} ]  
| FOREIGN KEY  
[ ( <имя_колонки> [..., n ] ) ]  
REFERENCES <внешняя_таблица> [ ( <имя_колонки_внешней_таблицы> [ , ...,  
n ] ) ]  
[ ON DELETE { CASCADE | NO ACTION } ]
```



```
[ ON UPDATE { CASCADE | NO ACTION } ]
| CHECK (<логическое_выражение> )
}
```

Назначение параметров совпадает с назначением аналогичных параметров предложения <ограничение\_столбца>. Тем не менее, в предложении <ограничение\_таблицы> имеются некоторые новые параметры:

имя\_колонки — столбец (или список столбцов), на которые необходимо наложить какие-либо ограничения целостности.

[ASC | DESC] — метод упорядочивания данных в индексе. Индекс создается при указании ключевых слов PRIMARY KEY, UNIQUE. При указании значения ASC данные в индексе будут упорядочены по возрастанию, при указании значения DESC — по убыванию. По умолчанию используется значение ASC.

### Примеры создания таблиц

В качестве примера рассмотрим инструкции создания таблиц базы данных «Сессия»:

Таблица «Студенты» состоит из следующих столбцов:

ID\_Студент – тип данных INTEGER, уникальный ключ;

Фамилия – тип данных CHAR, длина 30;

Имя - тип данных CHAR, длина 15;

Отчество - тип данных CHAR, длина 20;

Номер\_группы - тип данных CHAR, длина 6;

Адрес - тип данных CHAR, длина 30;

Телефон - тип данных CHAR, длина 8.

Создание таблицы выполнялось с помощью следующей команды:

```
CREATE TABLE Студенты
(ID_Студент    INTEGER NOT NULL,
Фамилия       CHAR(30) NOT NULL,
Имя           CHAR(15) NOT NULL,
Отчество      CHAR(20) NOT NULL,
Номер_группы  INTEGER NOT NULL,
Адрес         CHAR(30),
Телефон       CHAR(8),
PRIMARY KEY   (ID_Студент))
```

На все столбцы таблицы, кроме столбцов *Адрес* и *Телефон*, наложены ограничения NOT NULL, запрещающие ввод строки при неопределенном значении столбца.

Для создания таблицы «Дисциплины» была использована команда:

```
CREATE TABLE Дисциплины
(ID_Дисциплина INTEGER NOT NULL,
Наименование VARCHAR(40) NOT NULL,
PRIMARY KEY (ID_Дисциплина),
UNIQUE (Наименование))
```

Таблица содержит 2 столбца (*ID\_Дисциплина*, *Наименование*).

На столбцы *ID\_Дисциплина*, *Наименование* наложены ограничения NOT NULL, запрещающие ввод строки при неопределенном значении столбца.

Столбец *ID\_Дисциплина* объявлен первичным ключом, а на значения, вводимые в столбец *Наименование*, наложено условие уникальности.

Таблица «Учебный\_план» включает в себя следующие столбцы:

ID\_План – тип данных INTEGER, столбец уникального ключа;

ID\_Дисциплина – тип данных INTEGER;

Семестр - тип данных INTEGER;

Количество\_часов - тип данных INTEGER;

ID\_Преподаватель - тип данных INTEGER.

Создание таблицы выполнялось с помощью следующей команды:

```
CREATE TABLE Учебный_план
(ID_План INTEGER NOT NULL,
ID_Дисциплина INTEGER NOT NULL,
Семестр INTEGER NOT NULL,
Количество_часов INTEGER,
ID_Преподаватель INTEGER,
PRIMARY KEY (ID_План),
CHECK ((Семестр >= 1) OR (Семестр <= 10)))
```

Для значений столбца *Семестр* сформулировано логическое выражение, разрешающее вводить только значения от 1 до 10.

Таблица «Сводная\_ведомость» состоит из следующих столбцов:

ID\_Студент – тип данных INTEGER, столбец уникального ключа;

ID\_План – тип данных INTEGER, столбец уникального ключа;

Оценка - тип данных INTEGER;

Дата\_сдачи - тип данных DATETIME;

ID\_Преподаватель - тип данных INTEGER.

Создание таблицы выполнялось с помощью следующей команды:

```
CREATE TABLE Сводная_ведомость
(ID_Студент INTEGER NOT NULL,
ID_План INTEGER NOT NULL,
Оценка INTEGER NOT NULL,
Дата_сдачи DATETIME NOT NULL,
```

PRIMARY KEY (ID\_Студент, ID\_Дисциплина),  
CHECK ((Оценка >= 0) OR (Оценка <= 5)))

На все столбцы таблицы наложены ограничения NOT NULL, запрещающие ввод строки при неопределенном значении столбца.

Для значений столбца *Оценка* сформулировано логическое выражение, разрешающее вводить только значения от 0 до 5: 0 – незачет, 1 – зачет, 2 – неудовлетворительно, 3 – удовлетворительно, 4 – хорошо, 5 – отлично.

И, наконец, перечислим столбцы «Кадровый\_состав»:

ID\_Преподаватель – тип данных INTEGER, уникальный ключ;

Фамилия – тип данных CHAR, длина 30;

Имя - тип данных CHAR, длина 15;

Отчество - тип данных CHAR, длина 20;

Должность - тип данных CHAR, длина 20;

Кафедра - тип данных CHAR, длина 3;

Адрес - тип данных CHAR, длина 30;

Телефон - тип данных CHAR, длина 8.

Создание таблицы выполнялось с помощью следующей команды:

```
CREATE TABLE Кадровый_состав
(ID_Преподаватель    INTEGER NOT NULL,
Фамилия              CHAR(30) NOT NULL,
Имя                  CHAR(15) NOT NULL,
Отчество             CHAR(20) NOT NULL,
Должность            CHAR(20) NOT NULL,
Кафедра              CHAR(3) NOT NULL,
Адрес                 CHAR(30),
Телефон              CHAR(8),
PRIMARY KEY          (ID_Преподаватель))
```

На все столбцы таблицы, кроме столбцов *Адрес* и *Телефон*, наложены ограничения NOT NULL, запрещающие ввод строки при неопределенном значении столбца.

Для таблиц «Учебный\_план» и «Сводная\_ведомость» должны быть построены внешние ключи, связывающие таблицы базы данных «Сессия»:

FK\_Дисциплина – внешний ключ, связывающий таблицы «Учебный\_план» и «Дисциплины» по столбцу *ID\_Дисциплина*;

FK\_Кадровый\_состав – внешний ключ, связывающий таблицы «Учебный\_план» и «Кадровый\_состав» по столбцу *ID\_Преподаватель*;

FK\_Студент – внешний ключ, связывающий таблицы «Сводная\_ведомость» и «Студенты» по столбцу *ID\_Студент*;

FK\_План – внешний ключ, связывающий таблицы «Сводная\_ведомость» и «Учебный\_план» по столбцу *ID\_План*.

Добавление внешних ключей в таблицы будет описано при рассмотрении возможностей команды ALTER TABLE.

### 7.3.2. Изменение структуры таблицы – команда ALTER TABLE

Как бы тщательно ни планировалась структура таблицы, иногда возникает необходимость внести в нее некоторые изменения. Предположим, что в уже сформированную таблицу «Преподаватели» необходимо добавить номер домашнего телефона и домашний адрес. Эту операцию можно выполнять различными путями. Например, можно удалить таблицу со старой структурой и создать вместо нее новую таблицу с нужной структурой. Недостатком этого метода является то, что необходимо будет куда-то скопировать имеющиеся в таблице данные и переписать их в новую таблицу после ее создания.

Специальная команда ALTER TABLE предназначена для *модификации* структуры таблицы. С ее помощью можно изменять свойства существующих столбцов, удалять или добавлять в таблицу столбцы, а также управлять ограничениями целостности как на уровне столбца, так и на уровне таблицы, т.е. выполнять следующие функции:

- добавить в таблицу определение нового столбца;
- удалить столбец из таблицы;
- изменить значение по умолчанию для какого-либо столбца;
- добавить или удалить первичный ключ таблицы;
- добавить или удалить внешний ключ таблицы;
- добавить или удалить условие уникальности;
- добавить или удалить условие на значение.

Рассмотрим обобщенный синтаксис команды ALTER TABLE:

```
ALTER TABLE <имя_таблицы>
[ALTER COLUMN <имя_столбца> [SET DEFAULT <выражение>]]
[DROP DEFAULT]]
[[ADD <определение_столбца>]
[[DROP COLUMN <имя_столбца> [CASCADE]][RESTRIC]]
[[ADD[<определение_первичного_ключа>]][<определение_внешнего_ключа>]]
[<условие_уникальности>]][<условие_на_значение>]]
[[DROP CONSTRAINT <имя_ограничения> [CASCADE]][RESTRIC]]
```

Команда ALTER TABLE берет на себя все действия по копированию данных во временную таблицу, удалению старой таблицы, созданию вместо нее новой таблицы с нужной структурой и последующим переписыванием в нее данных.

Назначение многих параметров и ключевых слов команды ALTER TABLE аналогично назначению соответствующих параметров и ключе-



вых слов команды CREATE TABLE (например, синтаксис конструкции <определение\_столбца> совпадает с синтаксисом аналогичной конструкции команды CREATE TABLE).

Основные режимы использования команды ALTER TABLE следующие:

- добавление столбца;

- удаление столбца;

- модификация столбца;

- изменение, добавление и удаление ограничений (первичных и внешних ключей, значений по умолчанию).

### **Добавление столбца**

Для добавления нового столбца следует использовать ключевое слово ADD, после которого должно стоять определение столбца.

Добавим, например, в таблицу «Студенты» столбец «Год\_поступления» следующим образом:

```
ALTER TABLE Студенты  
ADD Год_поступления INTEGER NOT NULL DEFAULT YEAR(GETDATE())
```

После выполнения этой команды в структуру таблицы «Студент» будет добавлен еще один столбец со значением по умолчанию, равным текущему году (значение по умолчанию вычисляется с помощью двух встроенных функций - YEAR() и GETDATE()).

### **Модификация столбца**

Для модификации существующего столбца таблицы служит ключевое слово ALTER COLUMN. Изменение свойств столбца невозможно, если:

- столбец участвует в ограничениях PRIMARY KEY или FOREIGN KEY;

- на столбец наложены ограничения целостности CHECK или UNIQUE (исключение составляют столбцы, имеющие тип данных переменной длины, т.е. типы данных, начинающиеся на var);

- если со столбцом связано значение по умолчанию (в этом случае допускается изменение длины, общего количества цифр или количества цифр после десятичной точки при неизменном типе данных).

Определяя для столбца новый тип данных, следует помнить о том, что старый тип данных должен конвертироваться в новый.

Пример модификации столбца «Номер\_группы» таблицы «Студенты» (тип данных INTEGER заменяется на CHAR):

```
ALTER TABLE Студенты  
ALTER COLUMN Номер_группы CHAR(6) NOT NULL
```



## Удаление столбца

Для удаления столбца из таблицы используется предложение `DROP COLUMN <имя_столбца>`. При удалении столбцов следует учитывать, что нельзя удалять столбцы с ограничениями целостности `CHECK`, `FOREIGN KEY`, `UNIQUE` или `PRIMARY KEY`, а также столбцы, для которых определены значения по умолчанию (в виде ограничения целостности на уровне столбца или на уровне таблицы).

Рассмотрим, например, команду удаления из таблицы «Студент» столбца «Год\_поступления»:

```
ALTER TABLE Студенты  
DROP COLUMN Год_поступления
```

Эта команда выполнена не будет, т.т. при добавлении столбца было определено значение по умолчанию.

## Добавление ограничений на уровне таблицы

Для добавления ограничений на уровне таблицы используется предложение `ADD CONSTRAINT <имя_ограничения>`.

В качестве примера рассмотрим команды добавления внешних ключей в таблицы базы данных «Сессия».

Добавление внешних ключей в таблицу «Учебный\_план» (создание связи с именем `FK_Дисциплина` и связи с именем `FK_Кадровый_состав`):

```
ALTER TABLE Учебный_план  
ADD CONSTRAINT FK_Дисциплина  
FOREIGN KEY (ID_Дисциплина)  
REFERENCES Дисциплины  
  
ALTER TABLE Учебный_план  
ADD CONSTRAINT FK_Кадровый_состав  
FOREIGN KEY (ID_Преподаватель)  
REFERENCES Кадровый_состав
```

Добавление внешних ключей в таблицу «Сводная\_ведомость» (создание связи с именем `FK_Студент` и связи с именем `FK_План`):

```
ALTER TABLE Сводная_ведомость  
ADD CONSTRAINT FK_Студент  
FOREIGN KEY (ID_Студент)  
REFERENCES Студенты  
  
ALTER TABLE Сводная_ведомость  
ADD CONSTRAINT FK_План  
FOREIGN KEY (ID_План)  
REFERENCES Учебный_план
```

С помощью конструкции ADD CONSTRAINT создается *поименованное* ограничение. Необходимо отметить, что удаление любого ограничения на уровне таблицы происходит только по его имени, поэтому ограничение должно быть поименовано (чтобы его можно было удалить).

Рассмотрим еще один пример – добавление значения по умолчанию для столбца *Номер\_группы*:

```
ALTER TABLE Студент  
ADD CONSTRAINT DEF_Номер_группы DEFAULT 1 FOR Номер_группы
```

В результате выполнения этой команды на уровне таблицы будет создано ограничение целостности с именем DEF\_Номер\_группы.

### Удаление ограничений

Для удаления из таблицы ограничения целостности используется предложение DROP CONSTRAINT <имя\_ограничения>.

Удаление ограничения целостности возможно только в том случае, когда оно поименовано (т.е. предложение <определение\_ограничения> содержит именование ограничения CONSTRAINT).

Команда удаления построенного внешнего ключа FK\_Дисциплина из таблицы «Учебный\_план» выглядит следующим образом:

```
ALTER TABLE Учебный_план  
DROP CONSTRAINT FK_Дисциплина
```

Удалить же построенное ограничение DEF\_Номер\_группы можно с помощью следующей команды:

```
ALTER TABLE Студент  
DROP CONSTRAINT DEF_Номер_группы
```

#### 7.3.3. Удаление таблиц – команда DROP TABLE

Удаление таблицы выполняется при помощи команды DROP TABLE:

```
DROP TABLE <имя_таблицы>
```

Единственный аргумент команды задает имя таблицы, которую необходимо удалить.

Операция удаления таблицы в некоторых случаях требует определенного внимания. Невозможно удалить таблицу, если на нее с помощью ограничения целостности FOREIGN KEY ссылается другая таблица: попытка удаления таблицы «Дисциплины» вызовет сообщение об ошибке, т.к. на таблицу дисциплины ссылается таблица «Учебный\_план».

Например, в ответ на использование команды:

DROP TABLE Дисциплины

будет выдано сообщение об ошибке, гласящее о невозможности удаления таблицы вследствие наличия ограничения целостности FOREIGN KEY, ссылающегося на таблицу «Дисциплины».

#### **7.4. Управление данными**

Целью любой системы управления базами данных в конечном счете является ввод, изменение, удаление и выборка данных. Рассмотрим методы управления данными с помощью языка SQL.

##### *7.4.1. Извлечение данных – команда SELECT*

Основным инструментом выборки данных в языке SQL является команда SELECT. С помощью этой команды можно получить доступ к данным, представленным как совокупность таблиц практически любой сложности.

Чаще всего используется упрощенный вариант команды SELECT, имеющий следующий синтаксис:

```
SELECT <Список_выбора>
[ INTO <Новая_таблица> ]
FROM <Исходная_таблица>
[ WHERE <Условие_отбора> ]
[ GROUP BY <Ключи_группировки> ]
[ HAVING <Условие_отбора> ]
[ ORDER BY <Ключи_сортировки> [ ASC | DESC ] ]
```

Инструкция SELECT разбивается на отдельные разделы, каждый из которых имеет свое назначение. Из приведенного синтаксического описания видно, что обязательными являются только разделы SELECT и FROM, а остальные разделы могут быть опущены. Полный список разделов следующий:

```
SELECT
INTO
FROM
WHERE
GROUP BY
HAVING
UNION
ORDER BY
COMPUTE
FOR
OPTION
```

### 7.4.1.1. Раздел SELECT

Основное назначение раздела SELECT (одного из двух обязательных разделов, которые должны указываться в любом запросе) - задание набора столбцов, возвращаемых после выполнения запроса, т.е. внешнего вида результата. В простейшем случае возвращается столбец одной из таблиц, участвующих в запросе. В более сложных ситуациях набор значений в столбце формируется как результат вычисления выражения. Такие столбцы называются *вычисляемыми* и по умолчанию им не присваивается никакого имени.

При необходимости пользователь может указать для столбца, возвращаемого после выполнения запроса, произвольное имя. Такое имя называется *псевдонимом* (alias). В обычной ситуации назначение псевдонима не обязательно, но в некоторых случаях требуется явное его указание. Наиболее часто это требуется при работе с разделом INTO, в котором каждый из возвращаемых столбцов должен иметь имя, и это имя должно быть уникально.

Помимо сказанного, с помощью раздела SELECT можно ограничить количество строк, которое будет включено в результат выборки. Синтаксис раздела SELECT следующий:

```
SELECT [ ALL | DISTINCT ]  
[ TOP n [ PERCENT ] [ WITH TIES ] ]  
<Список_выбора>
```

Рассмотрим назначение параметров.

#### Ключевые слова ALL | DISTINCT

При указании ключевого слова ALL в результат запроса выводятся все строки, удовлетворяющие сформулированным условиям, тем самым разрешается включение в результат одинаковых строк (одинаковость строк определяется на уровне результата отбора, а не на уровне исходных данных). Параметр ALL используется по умолчанию.

Если в запросе SELECT указывается ключевое слово DISTINCT, то в результат выборки не будет включаться более одной повторяющейся строки. Таким образом, каждая возвращенная строка будет уникальной. Уникальность строки при этом определяется на уровне строк результата выборки, а не на уровне исходных данных. Если в результат выборки включаются два столбца, уникальность будет определяться по значениям обоих этих столбцов. В отдельности значения в первом и втором столбцах могут повторяться, но комбинация значений в обоих столбцах должна быть уникальна. Аналогичные правила действуют и в отношении большего количества столбцов.

	Семестр	Отчетность
▶	1	з
	2	з
	3	з
	4	з
	8	з
	1	з
	4	з
	1	з
	2	з
	3	з
	2	з
	3	з
	2	з
	1	з
	2	з
	1	з
	1	з
	1	з
	1	з

*а*

	Семестр	Отчетность
▶	1	з
	1	з
	2	з
	2	з
	3	з
	3	з
	4	з
	4	з
	5	з
	5	з
	6	з
	6	з
	7	з
	7	з
	8	з
	8	з
	9	з
	9	з
*		

*б*

*Рис. 7.4. Действие ключевых слов а - ALL и б – DISTINCT*

Рассмотрим результат использования ключевых слов ALL и DISTINCT на примере выборки столбцов *Семестр* и *Отчетность* из таблицы «Учебный\_план» базы данных «Сессия» (рис. 7.4). Сначала выполним запрос с указанием ключевого слова ALL:

```
SELECT ALL Семестр, Отчетность
FROM Учебный_план
```

Фрагмент результата представлен на рис. 7.4 а.

Теперь заменим ключевое слово ALL на DISTINCT:

```
SELECT DISTINCT ALL Семестр, Отчетность
FROM Учебный_план
```

В этом случае результат запроса, представленный на рис. 7.4 б - это строки, содержащие одинаковые значения в столбцах, включенные только один раз. Этот результат должен свидетельствовать только о наличии различных форм отчетности в семестрах.

### **Ключевое слово TOP n [PERCENT] [WITH TIES]**

Использование ключевого слова TOP n, где n – числовое значение, позволяет отобрать в результат не все строки, а только n первых. При этом выбираются первые строки результата выборки, а не исходных данных. Поэтому набор строк в результате выборки при указании ключевого слова TOP может меняться в зависимости от порядка сортировки. Если в запросе используется раздел WHERE, то ключевое слово TOP ра-



ботает с набором строк, возвращенных после применения логического условия, определенного в разделе WHERE.

Продemonстрируем использование ключевого слова TOP:

SELECT TOP 5 \* FROM Студенты

В этом примере из таблицы *Студенты* базы данных «Сессия» было выбрано 5 первых строк:

	ID_Студент	Фамилия	Имя	Отчество	Номер_Группы	Год_поступления
	1	Агапов	Иван	Иванович	2002\1	2002
	2	Агунов	Петр	Александрович	2000\1	2000
	3	Агуреев	Дмитрий	Александрович	2002\1	2002
	4	Акатьева	Мария	Алексеевна	2000\1	1999
	5	Акулов	Алексей	Юрьевич	2000\2	2000
*						

Можно также выбирать не фиксированное количество строк, а определенный процент от всех строк, удовлетворяющих условию. Для этого необходимо добавить ключевое слово PERCENT:

SELECT TOP 10 PERCENT \* FROM Студенты

Всего в таблице было 115 строк, следовательно, 10% будет составлять 11,5 строк. В результате будут выданы 12 строк:

	ID_Студент	Фамилия	Имя	Отчество	Номер_Группы	Год_поступления
	1	Агапов	Иван	Иванович	2002\1	2002
	2	Агунов	Петр	Александрович	2000\1	2000
	3	Агуреев	Дмитрий	Александрович	2002\1	2002
	4	Акатьева	Мария	Алексеевна	2000\1	1999
	5	Акулов	Алексей	Юрьевич	2000\2	2000
	6	Алексеев	Иван	Александрович	2002\2	2002
	7	Амаев	Тамерлан	Джабраилович	2001\2	2000
	8	Аюбова	Оксана	Аюбова	2001\2	2001
	9	Барыкин	Юрий	Владимирович	2000\2	2000
	10	Басов	Константин	Викторович	2000\2	2000
	11	Бибчук	Мария	Борисовна	2000\1	2000
	12	Белова	Ирина	Владимировна	2002\2	2002
*						

Если указанное количество процентов строк представляет собой нецелое число, то сервер всегда выполняется округление в большую сторону.

Приведем также пример, демонстрирующий влияние порядка сортировки на возвращаемый набор строк:

SELECT TOP 10 PERCENT \* FROM Студенты ORDER BY Номер\_Группы

В результате выполнения такого запроса будут выданы следующие 12 строк:

ID_Студент	Фамилия	Имя	Отчество	Номер_Группы	Год_поступления
2	Агупов	Петр	Александрович	2000\1	2000
4	Акатьева	Мария	Алексеевна	2000\1	1999
11	Бибчук	Мария	Борисовна	2000\1	2000
26	Голдобин	Михаил	Александрович	2000\1	2000
30	Гулько	Александр	Юрьевич	2000\1	2000
31	Гулько	Екатерина	Сергеевна	2000\1	2000
55	Кривченков	Михаил	Юрьевич	2000\1	2000
58	Лазаренко	Екатерина	Владимировна	2000\1	2000
61	Лебедев	Андрей	Евгеньевич	2000\1	2000
64	Лейпунская	Анна	Михайловна	2000\1	2000
73	Маслова	Анна	Владимировна	2000\1	2000
76	Митина	Светлана	Олеговна	2000\1	2000
*					

При указании вместе с предложением ORDER BY ключевого слова WITH TIES в результат будут включены еще и строки, совпадающие по значению колонки сортировки с последними выведенными строками запроса SELECT TOP n [PERCENT].

Использование ключевого слова WITH TIES в предыдущем примере позволит обеспечить выдачу в ответ на запрос информации обо *всех* студентах первой по порядку группы:

```
SELECT TOP 10 PERCENT WITH TIES *
FROM Студенты
```

После выполнения запроса получаем следующий результат:

ID_Студент	Фамилия	Имя	Отчество	Номер_Группы	Год_поступления
2	Агупов	Петр	Александрович	2000\1	2000
4	Акатьева	Мария	Алексеевна	2000\1	1999
11	Бибчук	Мария	Борисовна	2000\1	2000
26	Голдобин	Михаил	Александрович	2000\1	2000
30	Гулько	Александр	Юрьевич	2000\1	2000
31	Гулько	Екатерина	Сергеевна	2000\1	2000
55	Кривченков	Михаил	Юрьевич	2000\1	2000
58	Лазаренко	Екатерина	Владимировна	2000\1	2000
61	Лебедев	Андрей	Евгеньевич	2000\1	2000
64	Лейпунская	Анна	Михайловна	2000\1	2000
73	Маслова	Анна	Владимировна	2000\1	2000
76	Митина	Светлана	Олеговна	2000\1	2000
79	Никольская	Анастасия	Александровна	2000\1	2000
81	Панков	Дмитрий	Геннадьевич	2000\1	2000
83	Пасенова	Медея	Герасимовна	2000\1	2000
84	Петрова	Алина	Николаевна	2000\1	2000
92	Сибгатуллина	Таисия	Насимовна	2000\1	2000
95	Соловьев	Антон	Алексеевич	2000\1	2000
99	Стеценко	Илья	Владимирович	2000\1	2000
104	Франтова	Елена	Владимировна	2000\1	2000
111	Шваб	Кирилл	Юрьевич	2000\1	2000
*					

Предложение <Список\_выбора>

Синтаксис предложения <Список\_выбора> следующий:

```

<Список_выбора> ::=
{ *
| { <Имя_таблицы> | <Псевдоним_таблицы> }.*
| { <Имя_столбца> | <Выражение> }
[ [ AS ] <Псевдоним_столбца>]
| <Псевдоним_столбца> = <Выражение>
} [ ,...,n ]

```

Символ «\*» означает включение в результат всех столбцов, имеющих в списке таблиц раздела FROM.

Если в результат не нужно включать все столбцы *всех* таблиц, то можно явно указать имя объекта, из которого необходимо выбрать все столбцы (<Имя\_таблицы>.\* или <Псевдоним\_таблицы>.\*).

Отдельный столбец таблицы в результат выборки включается явным указанием имени столбца (параметр <Имя\_столбца>). Столбец должен принадлежать одной из таблиц, указанных в разделе FROM. Если столбец с указанным именем имеется более чем в одном источнике данных, перечисленных в разделе FROM, то необходимо явно указать имя источника данных, к которому принадлежит столбец в формате <Имя\_таблицы>.<Имя\_столбца>. В противном случае будет выдано сообщение об ошибке.

Например, попробуем выбрать данные из столбца ID\_Дисциплина, который имеется в таблицах «Дисциплина» и «Учебный\_план»:  
 SELECT ID\_Дисциплина, Наименование, Семестр  
 FROM Дисциплина, Учебный\_план

В ответ будет выдано сообщение об ошибке, указывающее на некорректное использование имени 'ID\_Дисциплина'.

Т. е., в этом случае необходимо явно указать имя источника данных, которому принадлежит столбец, например:

```

SELECT Дисциплина.ID_Дисциплина, Наименование, Семестр
FROM Дисциплина, Учебный_план

```

Столбцам, возвращаемым как результат выполнения запроса, могут быть присвоены *псевдонимы*. Псевдонимы позволяют изменить имя исходного столбца или поименовать столбец, содержимое которого получено как результат вычисления выражения. Имя псевдонима указывается с помощью параметра [AS] <Псевдоним\_столбца>. Ключевое слова AS необязательно при задании псевдонима. В общем случае сервер не требует уникальности имен столбцов результата выборки, поэтому разные столбцы могут иметь одинаковые имена или псевдонимы.

Столбцы в результате выборки могут быть не только копией столбца одной из исходных таблиц, но и формироваться на основе вычисления выражения. Такой столбец в списке выбора задается с помо-

щью конструкции <Выражение> [[AS] <Псевдоним\_столбца>]. Выражение при этом может содержать константы, имена столбцов, функции, а также их комбинации. Дополнительно столбцу, формируемому на основе вычисления выражения, можно присвоить псевдоним, указав его с помощью параметра [AS] <Псевдоним\_столбца>. По умолчанию вычисляемый столбец не имеет имени.

Другой способ формирования вычисляемого столбца состоит в использовании конструкции со знаком равенства: <Псевдоним\_столбца> = <Выражение>. Единственным отличием этого способа от предыдущего является необходимость *обязательного* задания псевдонима. В простейшем случае выражение является именем столбца, константой, переменной или функцией. Если в качестве выражения выступает имя столбца, то получаем еще один способ задания псевдонима для столбца.

Рассмотрим следующий пример. Пусть для таблицы «Студенты» необходимо построить запрос, представляющий фамилию, имя и отчество в одной колонке. Используя операцию конкатенации (сложения) символьных строк и значение ФИО в качестве псевдонима столбца, построим запрос:

```
SELECT TOP 10 Фамилия + ' ' + Имя + ' ' + Отчество as ФИО, Номер_Группы
FROM Студенты
```

Результат запроса имеет следующий вид:

	ФИО	Группа
►	Агапов Иван Иванович	2002\1
	Агупов Петр Александрович	2000\1
	Агуреев Дмитрий Александрович	2002\1
	Акатьева Мария Алексеевна	2000\1
	Акулов Алексей Юрьевич	2000\2
	Алексеев Иван Александрович	2002\2
	Амаев Тамерлан Джабраилович	2001\2
	Аюбова Оксана Аюбовна	2001\2
	Барыкин Юрий Владимирович	2000\2
	Басов Константин Викторович	2000\2
*		

#### 7.4.1.2. Раздел FROM

С помощью раздела FROM определяются источники данных, с которыми будет работать запрос.

Синтаксис раздела FROM следующий:

```
FROM { <Источник_данных> } [...,n]
```

На первый взгляд конструкция раздела выглядит простой. Однако при ближайшем рассмотрении он оказывается довольно сложным. В основном работа с разделом FROM это перечисление через запятую источников данных, с которыми должен работать запрос. Собственно ис-

точник данных указывается с помощью предложения <Источник\_данных>, синтаксис которого следующий:

<Источник\_данных> ::= <имя\_таблицы> [ [AS] <псевдоним\_таблицы> ]  
<связка\_таблиц>

С помощью параметра <имя\_таблицы> указывается имя обычной таблицы. Параметр <псевдоним\_таблицы> используется для присвоения таблице псевдонима, под которым на нее нужно будет ссылаться в запросе. Часто псевдонимы таблиц применяют, чтобы ссылку на нужную таблицу сделать более удобной и короткой. Например, если в запросе часто упоминается имя таблицы «Учебный\_план», то можно воспользоваться псевдонимом, например, tp1. Указание ключевого слова AS не является при этом обязательным.

Конструкция <связка\_таблиц> реализует один из наиболее сложных методов задания источника данных. С помощью нее можно связать данные двух и более таблиц в единый набор данных, указав критерии связывания. Синтаксис конструкции <связка\_таблиц> следующий:

<связка\_таблиц> ::= <левая\_таблица> <тип\_связывания> <правая\_таблица>  
ON <условие\_связывания>

Конструкция <тип\_связывания> описывает тип связывания двух таблиц. Исходная таблица указывается слева от конструкции <тип\_связывания> (<левая\_таблица>), а справа указывается зависимая таблица (<правая\_таблица>).

Общий синтаксис конструкции <тип\_связывания> следующий:

<тип\_связывания> ::= [INNER | { {LEFT | RIGHT | FULL } [OUTER] } ] JOIN

Как видно, обязательным в конструкции является ключевое слово JOIN.

Конструкция ON <условие\_связывания> задает логическое условие связывания двух таблиц. Допустимы операторы сравнения (например, =, <, >, <=, >=, !-, <>). Чаще всего используется оператор равенства, например:

ON Учебный\_план.ID\_Дисциплина = Дисциплины.ID\_Дисциплина

В этом примере устанавливается связь между таблицами «Учебный\_план» и «Дисциплина» по столбцу *ID\_Дисциплина*, имеющемуся в каждой из таблиц.

### Ключевое слово INNER

Этот тип связи используется по умолчанию. Указание сочетания INNER JOIN равносильно указанию только ключевого слова JOIN. В ка-



честве кандидатов на включение в результат запроса рассматриваются пары строк, удовлетворяющие критерию связывания в обеих таблицах. Затем строки из левой таблицы, для которых не имеется пары в связанной таблице, в результат не включаются. Также не включаются в результат и строки правой таблицы, для которых нет соответствующей строки в левой таблице.

В приведенном ниже примере выполняется выборка данных из таблиц «Дисциплины» и «Учебный\_план» с помощью запроса SELECT. Таблицы связаны по ключевому полю *ID\_Дисциплина*, имеющемуся в каждой из них. Для каждой строки таблицы «Учебный\_план» ищется строка с совпадающим значением поля *ID\_Дисциплина* в таблице «Дисциплины». Все строки таблицы «Учебный\_план», для которых нет строк с соответствующим значением поля *ID\_Дисциплина*, игнорируются и не включаются в конечный результат. Аналогично, не включаются в результат все строки таблицы «Дисциплины», для которых нет соответствующей строки в таблице «Учебный\_план» (что, однако, невозможно для данного примера, так как столбец *ID\_Дисциплина* таблицы «Учебный\_план» связан внешним ключом со столбцом *ID\_Дисциплина* таблицы «Дисциплины»).

```
SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план INNER JOIN Дисциплины
ON Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина
WHERE Количество_часов > 60
```

В результате выполнения этой команды будет возвращен следующий набор строк:

	Наименование	Семестр	Количество_часов
►	Английский язык	1	90
	Английский язык	2	110
	Английский язык	3	90
	Английский язык	4	100
	Английский язык	8	80
	История России	1	72
	Физическая культура	1	138
	Физическая культура	2	130
	Физическая культура	3	140
	Информатика	1	90
*			

### Ключевое слово LEFT [OUTER]

При использовании ключевого слова LEFT в результат будут включены все строки левой таблицы, независимо от того, есть для них соответствующая строка в правой таблице или нет. В случае отсутствия строки в правой таблице для столбцов правой таблицы, включенных в результат выборки, устанавливается значение NULL. В приведенном

ниже примере иллюстрируется использование ключевого слова LEFT [OUTER] для выборки данных.

```
SELECT Наименование, Семестр, Отчетность
FROM Дисциплины LEFT OUTER JOIN Учебный_план
ON Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина
WHERE (Наименование LIKE '%информатик%')
```

Будет возвращен следующий набор строк:

	Наименование	семестр	Отчетность
	Информатика и программирование	<NULL>	<NULL>
	Информатика	1	3
	Высокоуровневые методы информатики и программиров	4	3
	Высокоуровневые методы информатики и программиров	4	3
*			

Как видно, по сравнению с использованием ключевого слова INNER, в результат запроса добавлена строка из таблицы «Дисциплины», которая удовлетворяет сформулированному условию отбора, но для которой не существует соответствующей строки в таблице «Учебный\_план». В столбцах *Семестр* и *Отчетность* (относящихся к таблице «Учебный\_план») для этих строк установлено значение NULL.

### Ключевое слово RIGHT [OUTER]

При использовании этого ключевого слова в результат будут включены все строки правой таблицы, независимо от того, есть ли для них соответствующая строка в левой таблице. Для соответствующих столбцов левой таблицы, включенных в запрос, устанавливается значение NULL. Приведем пример такого запроса:

```
SELECT Отчетность, Семестр, Наименование
FROM Учебный_план RIGHT OUTER JOIN Дисциплины
ON Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина
WHERE (Наименование LIKE '%информатик%')
```

Этот пример основывается на тех же данных, что и предыдущий, но связь таблиц устанавливается в обратном порядке. После выполнения приведенной инструкции будет получен следующий результат:

	Отчетность	Семестр	Наименование
	<NULL>	<NULL>	Информатика и программирование
	3	1	Информатика
	3	4	Высокоуровневые методы информатики и программиров
	3	4	Высокоуровневые методы информатики и программиров
*			

## Ключевое слово FULL [OUTER]

При использовании ключевого слова FULL в результат будут включены все строки как правой, так и левой таблицы. Применение ключевого слова FULL [OUTER] можно рассматривать как одновременное применение ключевых слов LEFT [OUTER] и RIGHT[OUTER].

### 7.4.1.3. Раздел WHERE

Раздел WHERE предназначен для наложения вертикальных фильтров на данные, обрабатываемые запросом. Другими словами, с помощью раздела WHERE можно сузить набор строк, включаемых в результат выборки. Для этого указывается логическое условие, от которого зависит, будет ли строка включена в выборку по запросу или нет. Строка включается в результат выборки, только если логическое выражение возвращает значение TRUE.

В общем случае логическое выражение содержит имена столбцов таблиц, с которыми работает запрос. Для каждой строки, возвращенной запросом, вычисляется логическое выражение путем подстановки вместо имен столбцов конкретных значений из соответствующей строки. Если при вычислении выражения возвращается значение TRUE, то есть выражение истинно, то строка будет включена в конечный результат. В противном случае строка в результат не включается. При необходимости можно указать более одного логического выражения, объединив их с помощью логических операторов OR и AND.

Рассмотрим синтаксис раздела WHERE.

```
WHERE <условие_отбора>  
| <имя_столбца> {= | *= | =*} <имя_столбца>
```

В конструкции <условие\_отбора> можно определить любое логическое условие, при выполнении которого строка будет включена в результат. Хотя и было сказано, что обычно логическое условие содержит имена столбцов, оно может быть и произвольным, в том числе и совсем не связанным с данными. Например, в следующей команде условие WHERE никогда не выполнится и ни одна строка не будет возвращена:

```
SELECT * FROM Дисциплины WHERE 3=5
```

Приведенный пример демонстрирует логику работы раздела WHERE. Более удачное использование логического условия приведено в следующем примере:

```
SELECT Фамилия, Имя, Отчество, Номер_Группы, Год_поступления  
FROM Студенты  
WHERE Год_поступления < 2000
```

В результате будет возвращен список всех студентов, поступивших на факультет ранее 2000 года:

	Фамилия	Имя	Отчество	Номер_Группы	Год_поступления
►	Акатьева	Мария	Алексеевна	2000\1	1999
	Козлова	Нина	Сергеевна	2000\2	1999
*					

Помимо операций сравнения (=, >, <, >=, <=) и логических операторов OR, AND, NOT при формировании условия отбора могут быть использованы дополнительные логические операторы, расширяющие возможности по управлению данными. Рассмотрим некоторые из этих операторов.

### Оператор BETWEEN

С помощью этого оператора можно определить, лежит ли значение указанной величины в заданном диапазоне. Синтаксис использования оператора следующий:

<выражение> [NOT] BETWEEN <начало\_диапазона> AND <конец\_диапазона>

<Выражение> задает проверяемую величину, а аргументы <начало\_диапазона> и <конец\_диапазона> определяют возможные границы ее изменения. Использование оператора NOT совместно с оператором BETWEEN позволяет задать диапазон, *вне* которого может изменяться проверяемая величина.

При выполнении оператор BETWEEN преобразуется в конструкцию из двух операций сравнения:

(<выражение> >= <начало\_диапазона>) AND (<выражение> <= <конец\_диапазона>)

Рассмотрим пример использования оператора BETWEEN:

```
SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план INNER JOIN
Дисциплины ON
Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина
WHERE Количество_часов BETWEEN 50 AND 100
```

В результате выполнения инструкции получим список дисциплин учебного плана с количеством часов от 50 до 100:

	Наименование	Семестр	Количество_часов
►	Английский язык	1	90
	Английский язык	3	90
	Английский язык	4	100
	Английский язык	8	80
	История России	1	74
	Концепции современного естествознания	3	56
	Математический анализ	2	56
	Дискретная математика	4	54
	Основы алгебры и геометрии	1	52
	Основы алгебры и геометрии	2	52
	Информатика	1	92
	Основы программирования	1	52
	Теория вероятности и математическая статистика	3	54
	Операционная система UNIX	7	56
	Финансы и кредит	5	50
	Менеджмент	5	50
	Статистика	8	50
	Предметно-ориентированные информационные системы	8	50
*			

## Оператор IN

Оператор позволяет задать в условии отбора множество возможных значений для проверяемой величины. Синтаксис использования оператора следующий:

<выражение> [NOT] IN (<выражение1>, ..., <выражениеN>)

<Выражение> указывает проверяемую величину, а аргументы <выражение1>, ..., <выражениеN> задают перечислением через запятую набор значений, которые может принимать проверяемая величина. Ключевое слово NOT выполняет логическое отрицание.

Рассмотрим пример применения оператора IN.

```
SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план INNER JOIN
Дисциплины ON
Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина
WHERE Наименование IN ('Английский язык', 'Физическая культура')
```

В результате выполнения инструкции получим строки учебного плана для дисциплин «Английский язык» и «Физическая культура»:

	Наименование	Семестр	Количество_часов
►	Английский язык	1	90
	Английский язык	2	110
	Английский язык	3	90
	Английский язык	4	100
	Английский язык	8	80
	Физическая культура	1	138
	Физическая культура	2	130
	Физическая культура	3	140
*			



## Оператор LIKE

С помощью оператора LIKE можно выполнять сравнение выражения символьного типа с заданным шаблоном. Синтаксис оператора следующий:

<Символьное\_выражение> [NOT] LIKE <образец>

<Образец> задает символьный шаблон для сравнения и заключается в кавычки. Шаблон может содержать символы-разделители. Допускается использование следующих символов-разделителей (табл. 7.2):

Таблица 7.2.

Символы-разделители	Значение
%	Может быть заменен в символьном выражении любым количеством произвольных символов. Например, образец '%кош%' позволяет отобрать слова: 'кошка', 'окошко', 'лукошко', 'кошма' и т.п.
_	Может быть заменен в символьном выражении любым, но только одним символом. Например, образец 'программ_' позволяет отобрать слова: 'программа', 'программ', 'программы', но не 'программис' или 'программой'.
[ABC0-9]	Может быть заменен в символьном выражении только одним символом из указанного в квадратных скобках набора. Дефис используется для указания диапазона. Например, образец любой последовательности символов, начинающейся с буквы латинского алфавита, может быть задан следующим образом: '[A-Z]%'
[^ABC0-9]	Может быть заменен в символьном выражении только одним символом, кроме тех, что указаны в квадратных скобках. Дефис используется для указания диапазона. Например, образец любой последовательности символов, которая не должна заканчиваться цифрой, может быть задан следующим образом: '%[^0-9]'

Рассмотрим пример использования оператора:

```
SELECT Фамилия, Имя, Отчество, Должность  
FROM Кадровый_состав  
WHERE Должность LIKE '%пр%'
```

Результат выполнения инструкции:

	Фамилия	Имя	Отчество	Должность
►	Сидоров	Самуил	Савельевич	Проф.
	Гиацинтова	Галина	Григорьевна	Проф.
	Петров	Павел	Петрович	Ст.преп.
	Китов	Кирилл	Кириллович	Проф.
	Воробьева	Вера	Васильевна	Ст.преп.
*				

Применение образца для значения столбца *Должность* в данном случае позволило отобразить строки со значениями «Ст.преп.» и «Проф».

### Связывание таблиц

Раздел WHERE может быть использован для связывания таблиц. В этом случае условие связывания должно присоединяться к логическому выражению с помощью логической операции AND (логическое умножение).

Рассмотрим пример, уточняющий один из представленных выше:

```
SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план INNER JOIN Дисциплины
ON Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина
WHERE (Количество_часов > 60) AND (Семестр = 1)
```

Перенесем условие связывания в логическое выражение:

```
SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план, Дисциплины
WHERE (Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина) AND
(Количество_часов > 60) AND (Семестр = 1)
```

Результат выполнения обоих запросов одинаков:

	Наименование	Семестр	Количество_часов
►	Английский язык	1	90
	История России	1	72
	Физическая культура	1	138
	Информатика	1	90
*			

Использование только условия связывания в разделе WHERE аналогично связыванию ключевым словом INNER в разделе FROM. Например, результаты следующих запросов одинаковы:

```
SELECT TOP 10 Наименование, Семестр, Количество_часов
FROM Учебный_план, Дисциплины
WHERE (Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина)
```

```
SELECT TOP 10 Наименование, Семестр, Количество_часов
FROM Учебный_план INNER JOIN Дисциплины
ON (Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина)
```

	Наименование	Семестр	Количество_часов
▶	Английский язык	1	90
	Английский язык	2	110
	Английский язык	3	90
	Английский язык	4	100
	Английский язык	8	80
	История России	1	72
	Правоведение	4	32
	Физическая культура	1	138
	Физическая культура	2	130
	Физическая культура	3	140
*			

Содержимое обеих таблиц можно посмотреть с помощью следующих запросов:

```
SELECT TOP 10 *
FROM Учебный_план
```

	ID_План	ID_Дисциплина	Семестр	Отчетность	Количество_часов	ID_преподавателя
▶	1	1	1	з	90	<NULL>
	2	1	2	з	110	<NULL>
	3	1	3	з	90	<NULL>
	4	1	4	з	100	<NULL>
	5	1	8	з	80	<NULL>
	6	2	1	з	72	<NULL>
	7	3	4	з	32	<NULL>
	8	4	1	з	138	<NULL>
	9	4	2	з	130	<NULL>
	10	4	3	з	140	<NULL>
*						

```
SELECT TOP 10 *
FROM Дисциплины
```

	ID_Дисциплина	Наименование
▶	1	Английский язык
	2	История России
	3	Правоведение
	4	Физическая культура
	5	Философия
	6	Русский язык и культура речи
	7	Экономическая теория
	8	История мировых цивилизаций
	9	Культурология
	10	Социология
*		

Аналогом использования ключевых слов LEFT OUTER JOIN является указание в разделе WHERE условия с помощью символов \*=. Приведенные примеры возвращают одинаковый набор данных:

```
SELECT Наименование, Семестр, Отчетность
FROM Дисциплины LEFT OUTER JOIN Учебный_план
ON Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина
```

```
WHERE (Наименование LIKE '%информатик%')  
  
SELECT Наименование, Семестр, Отчетность  
FROM Дисциплины, Учебный_план  
WHERE (Учебный_план.ID_Дисциплина *= Дисциплины.ID_Дисциплина)  
AND (Наименование LIKE '%информатик%')
```

Аналогом использования ключевых слов RIGHT OUTER JOIN является указание условия с помощью символов =\*. Приведенные примеры возвращают одинаковый набор данных:

```
SELECT Отчетность, Семестр, Наименование  
FROM Учебный_план RIGHT OUTER JOIN Дисциплины  
ON Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина  
WHERE (Наименование LIKE '%информатик%')
```

```
SELECT Отчетность, Семестр, Наименование  
FROM Учебный_план, Дисциплины  
WHERE (Учебный_план.ID_Дисциплина *= Дисциплины.ID_Дисциплина)  
AND (Наименование LIKE '%информатик%')
```

Следует отметить, что при использовании специальных ключевых слов INNER | {LEFT | RIGHT | FULL } [OUTER ] данные представляются по-иному, чем при указании условия WHERE. Скорость выполнения запроса в первом случае оказывается выше, поскольку организуется *связывание* данных, тогда как при использовании конструкции WHERE происходит их *фильтрация*. При выполнении запросов на небольших наборах данных это не играет существенной роли, поэтому удобнее обращаться к конструкции WHERE из-за наглядности и простоты синтаксиса этого варианта, но при построении сложных запросов, выполняющих обработку тысяч строк, все же лучше использовать конструкцию связывания.

#### 7.4.1.4. Раздел ORDER BY

Раздел ORDER BY предназначен для упорядочения набора данных, возвращаемого после выполнения запроса. Рассмотрим пример упорядочения данных таблицы «Дисциплины» по столбцу *Наименование* в алфавитном порядке:

```
SELECT TOP 10 *  
FROM Дисциплины  
ORDER BY Наименование
```

Результат сортировки будет выглядеть следующим образом:

	ID_Дисциплина	Наименование
▶	1	Английский язык
	28	Базы данных
	46	Бухгалтерский учет
	36	Введение в специальность
	33	Высокоуровневые методы информатики и программирования
	32	Вычислительные системы, сети и телекоммуникаций
	16	Дискретная математика
	53	Имитационное моделирование
	48	Интеллектуальные информационные системы
	19	Информатика
*		

Полный синтаксис раздела ORDER BY следующий:

ORDER BY {<условие\_сортировки> [ ASC | DESC ] } [,...,n]

Параметр <условие\_сортировки> требует задания выражения, в соответствии с которым будет осуществляться сортировка строк. В простейшем случае это выражение представляет собой имя столбца одного из источников данных запроса.

Следует отметить, что в выражении, в соответствии с которым осуществляется сортировка строк, могут использоваться и столбцы, не указанные в разделе SELECT, то есть не входящие в результат выборки.

Раздел ORDER BY разрешает использование ключевых слов ASC и DESC, с помощью которых можно явно указать, каким образом следует упорядочить строки. При указании ключевого слова ASC данные будут отсортированы по возрастанию. Если необходимо отсортировать данные по убыванию, указывается ключевое слово DESC. По умолчанию используется сортировка по возрастанию.

Данные можно отсортировать по нескольким столбцам. Для этого необходимо ввести имена столбцов через запятую по порядку сортировки. Сначала данные сортируются по столбцу, имя которого было указано первым в разделе ORDER BY. Затем, если имеется множество строк с одинаковыми значениями в первом столбце, выполняется дополнительная сортировка этих строк по второму столбцу (внутри группы с одинаковым значением в первом столбце) и т.д.

Приведем пример сортировки по двум столбцам:

```
SELECT TOP 20 Наименование, Семестр, Количество_часов
FROM Учебный_план, Дисциплины
WHERE (Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина)
ORDER BY Семестр, Количество_часов DESC
```

Будет возвращен следующий набор строк:



Наименование	Семестр	Количество_часов
► Физическая культура	1	138
Английский язык	1	90
Информатика	1	90
История России	1	72
Основы алгебры и геометрии	1	52
Основы программирования	1	52
Культурология	1	32
Социология	1	32
Политология	1	32
Психология и педагогика	1	32
Математический анализ	1	26
Математический анализ	1	26
История мировых цивилизаций	1	20
Введение в специальность	1	12
Физическая культура	2	130
Английский язык	2	110
Математический анализ	2	56
Основы алгебры и геометрии	2	52
Экономическая теория	2	34
Философия	2	32
*		

Добавим в раздел SELECT столбец *Отчетность* и получим пример сортировки по трем столбцам:

```
SELECT TOP 20 Наименование, Семестр, Количество_часов, Отчетность
FROM Учебный_план, Дисциплины
WHERE (Учебный_план.ID_Дисциплина = Дисциплины.ID_Дисциплина)
ORDER BY Семестр, Отчетность, Количество_часов
```

Будет возвращен следующий набор строк:

Наименование	Семестр	Количество_часов	Отчетность
► Введение в специальность	1	12	э
История мировых цивилизаций	1	20	э
Математический анализ	1	26	э
Культурология	1	32	э
Социология	1	32	э
Политология	1	32	э
Психология и педагогика	1	32	э
Основы алгебры и геометрии	1	52	э
Основы программирования	1	52	э
Английский язык	1	90	э
Физическая культура	1	138	э
Математический анализ	1	28	э
История России	1	74	э
Информатика	1	92	э
Основы программирования	2	26	э
Философия	2	32	э
Экономическая теория	2	34	э
Английский язык	2	110	э
Физическая культура	2	130	э
История мировых цивилизаций	2	20	э
*			

#### 7.4.1.5. Раздел GROUP BY

Раздел GROUP BY позволяет выполнять группировку строк таблиц по определенным критериям. Для каждой группы можно выполнить специальные функции агрегирования, которые применяются ко всем строкам в группе. Одним из примеров использования раздела GROUP BY является суммирование однотипных значений.

Синтаксис раздела GROUP BY следующий:

GROUP BY [ALL] <условие\_группировки> [...,n]

При использовании группировки (раздела GROUP BY) на раздел SELECT накладываются дополнительные ограничения. В непосредственном виде разрешается указание только имен столбцов, перечисленных в разделе GROUP BY, то есть тех столбцов, по которым осуществляется группировка. Значения других столбцов не могут быть выведены в непосредственном виде, так как обычно каждая группа содержит множество строк, а в результате выборки для каждой группы должно быть указано единственное значение. Поэтому, чтобы вывести значения столбцов, не задающих критерии группировки, необходимо использовать функции агрегирования.

Аргумент <условие\_группировки> определяет условие группировки. Обычно в качестве условия группировки указывается имя столбца, однако в общем случае разрешается использование и выражений, включающих ссылки на столбцы.

Функции агрегирования позволяют выполнять статистическую обработку данных, подсчитывая количество, сумму, среднее значение и другие величины для всего набора данных. Во многих функциях агрегирования допускается использование ключевых слов ALL и DISTINCT. Ключевое слово ALL выполняет агрегирование всех строк исходного набора данных. При указании ключевого слова DISTINCT будет выполняться агрегирование только уникальных строк. Все повторяющиеся строки будут проигнорированы. По умолчанию выполняется агрегирование всех строк, то есть используется ключевое слово ALL. Далее приведены описания некоторых функций агрегирования.

Функция	Назначение																						
AVG()	<p>Эта функция вычисляет <i>среднее значение</i> для указанного столбца</p> <p>Функция имеет следующий синтаксис:</p> <p>AVG([ALL   DISTINCT] &lt;выражение&gt;)</p> <p>При выполнении группировки (GROUP BY) вычисляет среднее значение для каждой группы. Если группировка не используется, то вычисляет среднее по всему столбцу. Например:</p> <p>SELECT AVG(Количество_часов) FROM Учебный_план</p> <p>Результат запроса:</p> <pre> ----- 41 (1 row(s) affected) </pre> <p>Теперь рассмотрим пример использования функции AVG совместно с разделом GROUP BY при выполнении группировки по столбцу <i>Семестр</i>:</p> <p>SELECT Семестр, AVG(Количество_часов) FROM Учебный_план GROUP BY Семестр</p> <p>Результат:</p> <table> <thead> <tr> <th>Семестр</th><th></th></tr> <tr> <th>-----</th><th>-----</th></tr> </thead> <tbody> <tr><td>1</td><td>50</td></tr> <tr><td>2</td><td>54</td></tr> <tr><td>3</td><td>46</td></tr> <tr><td>4</td><td>39</td></tr> <tr><td>5</td><td>37</td></tr> <tr><td>6</td><td>27</td></tr> <tr><td>7</td><td>34</td></tr> <tr><td>8</td><td>44</td></tr> <tr><td>9</td><td>32</td></tr> </tbody> </table> <p>(9 row(s) affected)</p>	Семестр		-----	-----	1	50	2	54	3	46	4	39	5	37	6	27	7	34	8	44	9	32
Семестр																							
-----	-----																						
1	50																						
2	54																						
3	46																						
4	39																						
5	37																						
6	27																						
7	34																						
8	44																						
9	32																						
COUNT()	<p>Функция подсчитывает количество строк в группе (при выполнении группировки) или количество строк результата запроса. Синтаксис функции COUNT следующий:</p> <p>COUNT([ALL   DISTINCT] &lt;выражение&gt;   *)</p>																						

Функция	Назначение
---------	------------

Параметр <выражение> в простейшем случае представляет собой имя столбца. Если обрабатываемая строка в соответствующем столбце содержит значение не NULL, то счетчик будет увеличен на единицу. Указание символа (\*) предписывает считать *общее количество строк* независимо от того, содержат они значения NULL или нет.

Пример использования функции COUNT:

```
SELECT COUNT(*) AS 'Всего сотрудников',
COUNT(Телефон) AS 'С домашним телефоном'
FROM Кадровый_состав
```

Этот запрос подсчитывает общее количество строк в таблице, а также количество ненулевых значений в столбце *Телефон*.

Результат выполнения запроса:

Всего сотрудников	С домашним телефоном
-----	-----
14	10

(1 row(s) affected)

Warning: Null value eliminated from aggregate.

Пример использования функции COUNT() при выполнении группировки:

```
SELECT Должность, COUNT(*)
FROM Кадровый_состав
GROUP BY Должность
```

Данный запрос возвращает количество строк в каждой группе столбца *Должность*:

Должность	-----
-----	-----
Ассистент	3
Доцент	4
Зав. каф.	2
Проф.	3
Ст.преп.	2

(5 row(s) affected)

Функция	Назначение
MAX()	<p>Функция возвращает <i>максимальное</i> значение в указанном диапазоне. Эта функция может использоваться как в обычных запросах, так и в запросах с группировкой. Синтаксис функции следующий:</p> <p>MAX([ALL   DISTINCT] &lt;выражение&gt;)</p> <p>Пример использования функции:</p> <pre>SELECT MAX(Количество_часов),        MAX(Количество_часов/2) FROM Учебный_план</pre> <p>Результат выполнения запроса:</p> <pre>----- 1  140          70 (1 row(s) affected)</pre>
MIN()	<p>Функция возвращает минимальное значение в указанном диапазоне. Синтаксис функции следующий:</p> <p>MIN([ALL   DISTINCT] &lt;выражение&gt;)</p> <p>Пример использования функции:</p> <pre>SELECT MIN(Количество_часов) FROM Учебный_план</pre> <p>Результат выполнения запроса:</p> <pre>----- 1  12 (1 row(s) affected)</pre>
SUM()	<p>Функция выполняет обычное суммирование значений в указанном диапазоне. В качестве такого диапазона может рассматриваться группа или весь набор строк (без использования раздела GROUP BY). Синтаксис функции следующий:</p> <p>SUM([ALL   DISTINCT] &lt;выражение&gt;)</p> <p>В качестве примера просто суммируем значения в столбце <i>Количество_часов</i>:</p> <pre>SELECT SUM(Количество_часов), COUNT(*),        SUM(Количество_часов)/COUNT(*),        AVG(Количество_часов) FROM Учебный_план</pre>



Функция	Назначение
Результат выполнения запроса:	
-----	-----
694	89 41 41
(1 row(s) affected)	

Теперь вновь обратимся к разделу SELECT и приведем пример группировки значений таблицы «Учебный\_план». Произведем группировку строк по семестрам (столбец *Семестр*) и подсчитаем общую нагрузку в часах за каждый семестр:

```
SELECT Семестр, SUM(Количество_часов) AS 'Нагрузка'
FROM [Учебный_план]
GROUP BY Семестр
```

В результате выполнения запроса будет получен следующий результат:

	Семестр	Нагрузка
►	1	706
	2	486
	3	562
	4	474
	5	484
	6	334
	7	312
	8	266
	9	64
*		

В первом столбце выведен номер семестра. Это единственный столбец исходной таблицы, который можно включать в запрос непосредственно, т.к. по нему осуществляется группировка. Во втором столбце с помощью функции агрегирования SUM была получена сумма значений столбца *Количество\_часов*. Функции агрегирования работают со всеми строками группы, возвращая единственное значение для всех этих строк.

Рассмотрим теперь запрос, подсчитывающий количество экзаменов в каждом семестре:

```
SELECT Семестр, COUNT(*) AS 'Экзамены'
FROM [Учебный_план]
WHERE Отчетность = 'э'
GROUP BY Семестр
```

Результат выполнения запроса:

	Семестр	Экзамены
►	1	3
	2	4
	3	5
	4	6
	5	4
	6	5
	7	3
	8	3
	9	1
*		

Предложение группировки может содержать ключевое слово ALL. Назначение этого слова следующее. Нередко при выполнении группировки используется раздел WHERE, то есть группировка должна выполняться не над всеми строками, а лишь над определенной частью строк. Результатом такого подхода может явиться то, что одна или более групп не будет содержать ни одной строки. Если группа не содержит ни одной строки, то по умолчанию эта группа не включается в результат выборки. Однако в некоторых ситуациях все же требуется, чтобы были выведены все группы, в том числе и не содержащие ни одной строки. Для этого и необходимо указывать в разделе GROUP BY ключевое слово ALL. В этом случае будет выводиться список всех групп, но для групп, не содержащих строк, не будут выполняться функции агрегирования.

Рассмотрим это на примере. Для начала выполним группировку без использования ключевого слова ALL, но с вертикальной фильтрацией (с помощью раздела WHERE) – в таблице «Учебный\_план» посчитаем для каждого семестра количество дисциплин с нагрузкой более 60 часов:

```
SELECT Семестр, COUNT(*) AS 'Количество часов > 60'
FROM [Учебный_план]
WHERE Количество_часов > 60
GROUP BY Семестр
```

Результат запроса:

	Семестр	Количество часов > 60
►	1	4
	2	2
	3	2
	4	1
	8	1
*		

В полученной таблице отсутствуют данные для 5, 6 и 7 семестров. Это означает, что дисциплин, удовлетворяющих поставленному условию, в семестрах нет.

Добавим в раздел GROUP BY ключевое слово ALL:

```

SELECT Семестр, COUNT(*) AS 'Количество часов > 60'
FROM [Учебный_план]
WHERE Количество_часов > 60
GROUP BY ALL Семестр

```

Будет получен следующий результат:

	Семестр	Количество часов > 60
▶	1	4
	2	2
	3	2
	4	1
	5	0
	6	0
	7	0
	8	1
	9	0
*		

#### 7.4.1.6. Раздел COMPUTE

Этот раздел предназначен для выполнения групповых операций над содержимым столбцов выборки. Групповые операции задаются с помощью функций агрегирования. Результат агрегирования выводится в отдельной строке после всех данных столбца.

Синтаксис раздела COMPUTE следующий:

```

COMPUTE <Функция_агрегирования>(<столбец_агрегирования>){[,..., n]
[ BY <столбец_группировки> [,...,n ] ]

```

Аргумент <столбец\_агрегирования> должен содержать имя агрегируемого столбца. Этот столбец должен быть включен в результат выборки. Ключевое слово BY указывает, что результат вычисления следует сгруппировать. Следующий за этим ключевым словом аргумент <столбец\_группировки> содержит имя столбца, по которому будет производиться группировка. Результат необходимо предварительно отсортировать по этому столбцу, то есть столбец должен быть указан в разделе ORDER BY. Приведем простой пример применения раздела COMPUTE для вычисления количества дисциплин, читаемых в семестре, и общей суммы часов:

```

SELECT Наименование, Семестр, Количество_часов
FROM Учебный_план, Дисциплины
WHERE (Учебный_план.ID_Дисциплина =Дисциплины.ID_Дисциплина) AND
      (Семестр = 2)
COMPUTE SUM(Количество_часов), COUNT(Семестр)

```

Будет получен следующий результат:

Наименование	Семестр	Количество_часов
Английский язык	2	110
Физическая культура	2	130
Философия	2	32
Экономическая теория	2	34
История мировых цивилизаций	2	20
Математический анализ	2	56
Основы алгебры и геометрии	2	52
Основы программирования	2	26
		sum
		=====
		460
		cnt
		=====
		8

(9 row(s) affected)

Рассмотрим пример группировки при использовании раздела COMPUTE (составление списков групп и вычисление количества студентов в группе):

```
SELECT Фамилия, Имя, Отчество, Номер_Группы
FROM Студенты
ORDER BY Номер_Группы
COMPUTE COUNT(Номер_Группы) BY Номер_Группы
```

Будет получен следующий результат:

Фамилия	Имя	Отчество	Номер_Группы
Агапов	Иван	Иванович	2002\1
Агуреев	Дмитрий	Александрович	2002\1
Гогешвили	Серго	Тамазович	2002\1
Григорьева	Мария	Александровна	2002\1
Желтов	Олег	Игоревич	2002\1
Жуков	Виктор	Владимирович	2002\1
Жучков	Сергей	Сергеевич	2002\1
Захаркин	Николай	Владимирович	2002\1
Иванов	Олег	Геннадиевич	2002\1
Кадаков	Антон	Дмитриевич	2002\1
Леонтьев	Богдан	Вадимович	2002\1
Миняйло	Евгений	Николаевич	2002\1
Нечаева	Ольга	Николаевна	2002\1
Парфенова	Светлана	Витальевна	2002\1
Потапкин	Александр	Александрович	2002\1
Соловьева	Полина	Сергеевна	2002\1
Федянин	Александр	Алексеевич	2002\1
			cnt
			=====
			17

Фамилия	Имя	Отчество	Номер_Группы
Алексеев	Иван	Александрович	2002\2
Белова	Ирина	Владимировна	2002\2
Бородкина	Анна	Михайловна	2002\2
Братченко	Елена	Анатолевна	2002\2
Волков	Иван	Александрович	2002\2
Гончаров	Иван	Андреевич	2002\2
Калинин	Андрей	Евгеньевич	2002\2
Кондрашкина	Ольга	Игоревна	2002\2
Ларина	Евгения	Валерьевна	2002\2
Малкова	Дарья	Дмитриевна	2002\2
Поспелов	Игорь	Григорьевич	2002\2
Тюрина	Юлия	Александровна	2002\2
Филоненко	Петр	Алексеевич	2002\2
Юртанов	Сергей	Михайлович	2002\2

cnt

=====

14

(33 row(s) affected)

#### 7.4.1.7. Раздел UNION

Раздел UNION служит для объединения результатов выборки, возвращаемых двумя и более запросами.

Рассмотрим синтаксис раздела UNION:

```
<Спецификация_Запроса_1>
UNION [ALL]
<Спецификация_Запроса_2>
...
[UNION [ALL]]
<Спецификация_Запроса_n>
```

Чтобы к результатам запросов можно было применить операцию объединения, они должны соответствовать следующим требованиям:

запросы должны возвращать одинаковый набор столбцов (причем необходимо гарантировать одинаковый порядок следования столбцов в каждом из запросов);

типы данных соответствующих столбцов второго и последующих запросов должны поддерживать неявное преобразование или совпадать с типом данных столбцов первого запроса;

ни один из результатов не может быть отсортирован с помощью раздела ORDER BY (однако общий результат может быть отсортирован, как будет показано ниже)



Указание ключевого слова ALL предписывает включать в результат повторяющиеся строки. По умолчанию повторяющиеся строки в результат не включаются.

Продemonстрируем применение раздела UNION. Рассмотрим таблицы «Кадровый\_Состав» и «Студенты» и попробуем построить, например, общий список и учащихся, и преподавателей, номер телефона которых начинается на 120.

Сначала построим запрос для таблицы «Кадровый\_Состав»:

```
SELECT Фамилия, Имя, Отчество, Должность, Телефон
FROM Кадровый_состав
WHERE Телефон LIKE '120%'
```

Результат действия запроса следующий:

	Фамилия	Имя	Отчество	Должность	Телефон
►	Цветкова	Светлана	Сидоровна	Доцент	120-1716
	Китов	Кирилл	Кириллович	Проф.	120-4668
*					

Затем построим запрос для таблицы «Студенты»:

```
SELECT Фамилия, Имя, Отчество, Телефон
FROM Студенты
WHERE Телефон LIKE '120%'
```

В результате выполнения запроса получим следующую выборку:

	Фамилия	Имя	Отчество	Телефон
►	Барыкин	Юрий	Владимирович	120-5448
	Гулько	Александр	Юрьевич	120-7787
	Кессель	Глеб	Юрьевич	120-1112
	Козлова	Нина	Сергеевна	120-3335
*				

Теперь объединим два запроса, чтобы в результате получить единую таблицу. Заметим, что столбец *Должность* отсутствует в таблице «Студенты». Чтобы в общей таблице выделить студентов, введем в запрос для таблицы «Студенты» столбец, содержащий строку-константу «Студент» для всех записей, и объединим два запроса с помощью раздела UNION:

```
SELECT Фамилия, Имя, Отчество, Должность, Телефон
FROM Кадровый_состав
WHERE Телефон LIKE '120%'
UNION
SELECT Фамилия, Имя, Отчество, Новый_столбец = 'Студент', Телефон
FROM Студенты
WHERE Телефон LIKE '120%'
```

После выполнения запроса получим следующую таблицу:

	Фамилия	Имя	Отчество	Должность	Телефон
►	Барыкин	Юрий	Владимирович	Студент	120-5448
	Гулько	Александр	Юрьевич	Студент	120-7787
	Кессель	Глеб	Юрьевич	Студент	120-1112
	Козлова	Нина	Сергеевна	Студент	120-3335
	Цветкова	Светлана	Сидоровна	Доцент	120-1716
	Китов	Кирилл	Кириллович	Проф.	120-4668
*					

При объединении таблиц столбцам итогового набора данных всегда присваиваются те же имена, что были указаны в первом из объединяемых запросов.

Упорядочим полученный список по алфавиту, добавив предложение ORDER BY:

```
SELECT Фамилия, Имя, Отчество, Должность, Телефон
FROM Кадровый_состав
WHERE Телефон LIKE '120%'
UNION
SELECT Фамилия, Имя, Отчество, Новый_столбец = 'Студент', Телефон
FROM Студенты
WHERE Телефон LIKE '120%'
ORDER BY Фамилия
```

Получим следующий результат:

	Фамилия	Имя	Отчество	Должность	Телефон
►	Барыкин	Юрий	Владимирович	Студент	120-5448
	Гулько	Александр	Юрьевич	Студент	120-7787
	Кессель	Глеб	Юрьевич	Студент	120-1112
	Китов	Кирилл	Кириллович	Проф.	120-4668
	Козлова	Нина	Сергеевна	Студент	120-3335
	Цветкова	Светлана	Сидоровна	Доцент	120-1716
*					

#### 7.4.1.8. Раздел INTO. Использование команды SELECT...INTO

При указании этой конструкции результат выполнения запроса будет сохранен в новой таблице. Синтаксис раздела INTO следующий:

INTO <имя\_новой\_таблицы>

Аргумент <имя\_новой\_таблицы> определяет имя таблицы, в которую будут вставлены результаты.

При выполнении запроса SELECT...INTO автоматически создается новая таблица с нужной структурой и в нее заносится полученный набор строк. При этом в базе данных не должно существовать таблицы, имя которой совпадает с именем таблицы, указанной в команде SELECT...INTO. Если необходимо быстро создать таблицу со структурой, позволяющей сохранить результат выполнения запроса, то лучшим выходом будет использование команды SELECT...INTO.

Синтаксис команды SELECT...INTO следующий:

```
SELECT {<имя_столбца> [[AS] <псевдоним_столбца>] [, ..., n] }  
INTO <имя_новой_таблицы> FROM {<имя_исходной_таблицы> [, ..., n]}
```

Приведенный вариант синтаксиса далеко не исчерпывает все возможности вставки данных с помощью команды SELECT...INTO. Допускаются практически все варианты синтаксиса запроса SELECT, то есть можно выполнять группировку, сортировку, объединение и т. д.

Рассмотрим назначение аргументов команды.

<имя\_столбца> [[AS] <псевдоним\_столбца>]. Аргумент <имя\_столбца> задает имя столбца таблицы, который будет включен в результат. Указанный столбец должен принадлежать одной из таблиц, перечисленных в списке FROM {<имя\_исходной\_таблицы> [, ..., n]}. Если столбцы, принадлежащие разным таблицам, имеют одинаковые имена, то для столбцов необходимо использовать псевдонимы. В противном случае произойдет попытка создать таблицу со столбцами, имеющими одинаковые имена, что приведет к ошибке, и выполнение запроса будет прервано. Указание псевдонимов также обязательно для столбцов, значения в которых формируются на основе вычисления выражений (по умолчанию такие столбцы не имеют никакого имени, что недопустимо для таблицы) и когда пользователь хочет задать столбцам в создаваемой таблице новые имена (отличные от исходных). Имя псевдонима задается с помощью параметра <псевдоним\_колонки>.

INTO <имя\_новой\_таблицы>. Аргумент <имя\_новой\_таблицы> содержит имя создаваемой таблицы. Это имя должно быть уникальным в пределах базы данных.

FROM {<имя\_исходной\_таблицы> [, ..., n]}. В простейшем случае конструкция FROM содержит список исходных таблиц. В более сложных запросах с помощью этой конструкции определяются условия связывания двух и более таблиц.

С помощью команды SELECT...INTO, например, можно разделить таблицу «Студенты» на две, выделив в отдельную таблицу «Контакты» адреса и телефоны, а затем удалив эти столбцы из таблицы «Студенты»:

```
SELECT ID_Студент, Адрес, Телефон  
INTO Контакты  
FROM Студенты
```

Будет создана новая таблица со следующей структурой:

Key	Id	Name	Data Type	Size	Nulls	Default
		ID_Студент	int	4	<input type="checkbox"/>	
		Адрес	char	30	<input checked="" type="checkbox"/>	
		Телефон	char	8	<input checked="" type="checkbox"/>	

Запрос для таблицы «Контакты»:

```
SELECT *
FROM Контакты
WHERE Телефон LIKE '120%'
```

Выдает следующий результат:

	ID_Студент	Адрес	Телефон
	9	ул. Профсоюзная, д.57, кв.31	120-5448
	30	ул. Цурюпы, д.1, кв.11	120-7787
	48	ул. Академическая, д.17, кв.76	120-1112
	52	ул.Цурюпы, д.33, кв.236	120-3335
*			

Построим внешний ключ для таблицы «Контакты», обеспечив связь с таблицей «Студенты»:

```
ALTER TABLE Контактa
ADD CONSTRAINT FK_Контакт
FOREIGN KEY (ID_Студент)
REFERENCES Студенты
```

Модифицируем запрос для таблицы «Контакты»:

```
SELECT *
FROM Студенты INNER JOIN
Контакты ON
Студенты.ID_Студент = Контакты.ID_Студент
WHERE Телефон LIKE '120%'
```

	ID_Студент	Фамилия	Имя	Отчество	Номер_Группы	Год_поступления	ID_Студент	Адрес	Телефон
	9	Барыкин	Юрий	Владимирович	2000\2	2003	9	ул. Профсоюзная,	120-5448
	30	Гулько	Александр	Юрьевич	2000\1	2003	30	ул. Цурюпы, д.1,	120-7787
	48	Кессель	Глеб	Юрьевич	2000\2	2003	48	ул. Академическая,	120-1112
	52	Козлова	Нина	Сергеевна	2000\2	2003	52	ул. Нагорная	120-3335
*									

#### 7.4.2. Добавление данных – команда INSERT

Рассмотрим некоторые возможности заполнения таблиц. Данные в таблицу могут быть внесены различными способами:

С помощью команды INSERT. Используя команду INSERT, можно добавить как одну строку, так и множество строк.

С помощью команды SELECT INTO. В этом случае на основе результата выборки, возвращаемого запросом, автоматически создается новая таблица (аппарат использования команды рассмотрен выше).

Рассмотрим процесс внесения данных в таблицу с помощью команды INSERT. Как уже было сказано, эта команда может быть использована для вставки как одной, так и множества строк.

### Вставка одной строки

В простейшем случае вставка данных с помощью команды INSERT предполагает использование конструкции INSERT-VALUES:

```
INSERT [INTO] <имя_таблицы> [(<список_колонок>)]  
VALUES (<список_значений>)
```

С помощью этой команды можно добавить только одну строку.

Аргумент <имя\_таблицы> идентифицирует имя таблицы, в которую необходимо вставить строку данных. Необязательный параметр <список\_столбцов> задает имена столбцов, в которые будет производиться добавление данных.

Рассмотрим процесс добавления данных в таблицу «Сводная\_ведомость». Каждая строка этой таблицы содержит результат сдачи экзамена (зачета) по отдельной дисциплине отдельным студентом. Если студент, *ID\_Студент* которого равен 10, сдал экзамен по дисциплине со значением 3 в столбце *ID\_Дисциплина* на оценку «5», то команда добавления этих данных в таблицу «Сводная\_ведомость» выглядит следующим образом:

```
INSERT    Сводная_ведомость  
VALUES (10, 3, 5)
```

Для назначения произвольного порядка и состава столбцов в этом случае можно использовать следующую команду:

```
INSERT INTO Сводная_ведомость  
(ID_Дисциплина, ID_Студент)  
VALUES (3, 10)
```

Если для столбца *Оценка* определено значение по умолчанию или разрешено хранение значений NULL, то значение для этого столбца можно вообще не указывать:

Мы рассматривали вставку строк в таблицу, значения для которых были заданы с помощью констант. Однако вставляемые значения можно идентифицировать и с помощью переменных, функций, а также любых сложных выражений. Единственным требованием является совпадение типов данных столбца и значения, возвращаемого выражением.



## Вставка результата запроса

Приведем упрощенный синтаксис команды INSERT:

```
INSERT [ INTO]
<имя_таблицы>
{ [ (<список_колонок>) ]
{ VALUES
( { DEFAULT | NULL | <выражение> } [, ..., n] )
| <результатирующая_таблица>
}
}
| DEFAULT VALUES
```

Рассмотрим назначение каждого из аргументов:

INTO - дополнительное ключевое слово, которое может быть использовано между словом INSERT и именем таблицы для обозначения, что следующий параметр является именем таблицы, в которую будут вставлены данные;

<имя\_таблицы> - имя таблицы, в которую необходимо вставить данные;

<список\_столбцов> - содержит список столбцов, в которые будет производиться вставка данных. Если он опущен, то данные будут вставляться последовательно во все столбцы, начиная с первого. Значения для столбцов указываются после ключевого слова VALUES. Для каждого столбца должно быть задано выражение, имеющее соответствующий тип данных. Если список столбцов не указан, то количество значений VALUES должно соответствовать количеству столбцов таблицы. Если же список столбцов явно задан, то это определяет порядок значений VALUES (и, соответственно, их типы). Можно не указывать явно значения для столбцов, если для них определено значение по умолчанию или разрешено хранение значений NULL.

VALUES ( { DEFAULT | NULL | <выражение> } [, ..., n]) - определяет набор данных, которые будут вставлены в таблицу. Количество аргументов VALUES определяется количеством столбцов в таблице или количеством столбцов в списке (если таковой имеется). Для каждого столбца таблицы можно указать один из трех возможных вариантов:

DEFAULT - будет вставлено значение по умолчанию, определенное для столбца. Если для столбца разрешено хранение значений NULL, а значение по умолчанию не определено, то в столбец будет вставлено значение NULL.

NULL - в столбец будет вставлено значение NULL. Естественно, вставка таких значений будет успешной, если для столбца была разрешена возможность хранения значений NULL. Следует помнить, что для столбцов, входящих в первичный ключ, возможность хранения значений NULL не предусмотрена.

<выражение> - задает значение, которое будет вставлено в столбец таблицы. Этот параметр должен иметь тот же тип данных, что и столбец, а также удовлетворять ограничениям целостности, определенным для соответствующего столбца.

<результатирующая\_таблица> - этот параметр подразумевает указание запроса SELECT, с помощью которого будет формироваться набор данных, вставляемых в таблицу. Количество столбцов, порядок их перечисления и их типы данных должны соответствовать столбцам, указанным в списке <список\_столбцов>. Если последний отсутствует, то запрос должен возвращать значения для всех столбцов таблицы.

DEFAULT VALUES - при указании этого параметра строка будет содержать только значения по умолчанию. Если для столбца не установлено значение по умолчанию, но разрешено хранение значений NULL, то в столбец будет вставлено значение NULL. Если же для столбца не разрешено хранение значений NULL, нет значения по умолчанию и в команде INSERT не указано значение для вставки, то будет выдано сообщение об ошибке и выполнение команды прервется.

Более сложный случай вставки данных предполагает использование конструкции INSERT INTO...SELECT:

```
INSERT INTO <имя_таблицы>  
SELECT <выражение_запроса>
```

Аргумент <имя\_таблицы> содержит имя таблицы, в которую будут вставляться выбранные данные. Таблица должна иметь соответствующую структуру и быть предварительно создана.

<Выражение\_запроса> определяет тело запроса SELECT, с помощью которого производится выборка данных из одной или нескольких таблиц. Например, для выборки данных из таблицы «Студенты» обо всех студентах, поступивших в ВУЗ в 2000 году и сохранения их в таблице «Студент\_2000» можно использовать такую последовательность инструкций:

```
CREATE TABLE Студент_2000  
(ID_Студент_2000    INTEGER NOT NULL,  
  Фамилия           CHAR(30) NOT NULL,  
  Имя               CHAR(15) NOT NULL,  
  Отчество          CHAR(20) NOT NULL,  
  Адрес             CHAR(30),  
  Телефон           CHAR(8),  
  PRIMARY KEY       (ID_Студент_2000))  
  
INSERT INTO Студент_2000  
SELECT ID_Студент, Фамилия, Имя, Отчество, Адрес, Телефон  
FROM Студенты  
WHERE Год_поступления = 2000
```

После выполнения этой последовательности команд иницилируем запрос на отбор строк из новой таблицы:

```
SELECT TOP 5 Фамилия, Имя, Отчество  
FROM Студент_2000
```

Будет выдан следующий результат:

	Фамилия	Имя	Отчество
►	Агупов	Петр	Александрович
	Акулов	Алексей	Юрьевич
	Амаев	Тамерлан	Джабраилович
	Барыкин	Юрий	Владимирович
	Басов	Константин	Викторович
*			

Приведенный пример иллюстрирует вставку строк данных в таблицу на основе результата выполнения запроса, обращающегося к одной таблице. Более сложные запросы могут обращаться к множеству таблиц одной или нескольких баз данных.

В качестве еще одного примера рассмотрим помещение в новую таблицу «Преподаватель\_дисциплина» информации о том, какой преподаватель какую дисциплину ведет.

Для этого мы будем работать с тремя таблицами: «Кадровый\_состав», «Учебный\_план» и «Дисциплины». В первой таблице содержится список преподавателей, тогда как в третьей — список дисциплин. С помощью таблицы «Учебный\_план» устанавливается связь «многие ко многим» между таблицами «Кадровый\_состав» и «Дисциплины».

Прежде чем приступить к вставке данных, необходимо создать таблицу, которая будет содержать интересующие нас данные. Помимо столбцов для хранения информации об имени и фамилии преподавателя и названии дисциплины предусмотрим столбцы для хранения идентификационных номеров преподавателей и дисциплин:

```
CREATE TABLE Преподаватель_дисциплина  
(ID_Дисциплина      INTEGER NOT NULL,  
 ID_Преподаватель    INTEGER NOT NULL,  
 Наименование        CHAR(20) NOT NULL,  
 Фамилия              CHAR(30) NOT NULL,  
 Имя                  CHAR(15) NOT NULL,  
 Отчество             CHAR(20) NOT NULL,  
 Должность            CHAR(20) NOT NULL)
```

Теперь вставим в созданную таблицу нужные нам данные, выполнив для этого следующий запрос:

```
INSERT INTO Преподаватель_дисциплина  
SELECT DISTINCT Дисциплины.ID_Дисциплина,  
 Кадровый_состав.ID_Преподаватель, Наименование,  
 Фамилия, Имя, Отчество, Должность
```

```
FROM Кадровый_состав, Учебный_план, Дисциплины
WHERE Кадровый_состав.ID_Преподаватель = Учебный_план.ID_Преподаватель
AND Дисциплины.ID_Дисциплина = Учебный_план.ID_Дисциплина
```

В результате в таблицу будет вставлено 54 строки.

Проиллюстрируем результат запроса по новой таблице:

```
SELECT TOP 4 *
FROM Преподаватель_дисциплина
```

ID_Дисциплина	ID_Преподаватель	Наименование	Фамилия	Имя	Отчество	Должность
1	2	Английский язык	Сидоров	Самуил	Савельевич	Проф.
2	4	История России	Цветкова	Светлана	Сидоровна	Доцент
3	3	Правоведение	Гиацинтова	Галина	Григорьевна	Проф.
4	5	Физическая культура	Козлов	Константин	Константинович	Доцент
*						

### 7.4.3. Изменение данных – команда UPDATE

Для внесения изменений в данные таблиц служит команда UPDATE, позволяющая выполнять как простое обновление данных в столбце, так и сложные операции модификации данных во множестве строк таблицы. Рассмотрим упрощенный синтаксис этой команды:

```
UPDATE <имя_таблицы>
SET { <имя_колонки> = { <выражение> | DEFAULT | NULL } } [, ..., n]
{ [ FROM { <имя_исходной_таблицы> } [, ..., n] ]
[ WHERE <условие_отбора> ] }
```

Рассмотрим назначение каждого из аргументов.

<имя\_таблицы> - имя таблицы, в которой необходимо произвести изменение данных.

SET - с этого ключевого слова начинается блок, в котором определяется список изменяемых столбцов. За один вызов UPDATE можно изменить данные в нескольких столбцах множества строк одной таблицы.

<имя\_столбца> = {<выражение> | DEFAULT | NULL} - для каждого изменяемого столбца нужно задать значение, которое он примет после выполнения изменения. С помощью ключевого слова DEFAULT можно присвоить столбцу значение, определенное для него по умолчанию. Можно также установить для столбца значение NULL. Изменению подвергнутся все строки, удовлетворяющие критериям ограничения области действия запроса UPDATE, которые задаются с помощью раздела WHERE. При составлении выражения можно ссылаться на любые столбцы таблицы, включая изменяемые. При этом следует учитывать, что изменения в данные вносятся только после выполнения команды. Таким образом, при ссылке на изменяемые столбцы будут использоваться старые значения.

FROM {<имя\_исходной\_таблицы>} - если при изменении данных в таблице необходимо учесть состояние данных в других таблицах, то эти источники данных необходимо указать в разделе FROM. Собственно



источник данных описывается с помощью конструкции <имя\_исходной\_таблицы>.

WHERE <условие\_отбора> - назначение раздела WHERE, используемого в запросе UPDATE, полностью соответствует назначению, которое раздел имеет в запросе SELECT, т.е. с помощью раздела WHERE можно сузить диапазон строк, в которых будет выполняться изменение данных. Необходимо указать логическое условие, на основе которого будет приниматься решение об изменении данных конкретной строки. Если в контексте значений строки указанное логическое условие выполняется (т.е. возвращает значение TRUE), то данные этой строки будут изменены. В противном случае изменение не выполняется. Предполагается, что логическое условие включает имена столбцов изменяемой таблицы, однако это не обязательно.

Приведем простейший пример изменения данных. Добавим в таблицу «Учебный\_план» по 2 часа в столбец *Количество\_часов* для дисциплин 1-го семестра с формой отчетности - экзамен:

Выведем сначала исходное состояние данных:

```
SELECT *  
FROM Учебный_план  
WHERE (Отчетность= 'э') AND (Семестр = 1)
```

	ID_План	ID_Дисциплина	Семестр	Отчетность	Количество_часов	ID_преподавателя
▶	6	2	1	э	72	<NULL>
	24	15	1	э	26	<NULL>
	30	19	1	э	90	<NULL>
*						

Затем выполним изменения и снова посмотрим данные.

```
UPDATE Учебный_план  
SET Количество_часов = Количество_часов + 2  
WHERE (Отчетность= 'э') AND (Семестр = 1)  
SELECT *  
FROM Учебный_план  
WHERE (Отчетность= 'э') AND (Семестр = 1)
```

	ID_План	ID_Дисциплина	Семестр	Отчетность	Количество_часов	ID_преподавателя
▶	6	2	1	э	74	<NULL>
	24	15	1	э	28	<NULL>
	30	19	1	э	92	<NULL>
*						

#### 7.4.4. Удаление данных – команда DELETE

Удаление данных из таблицы выполняется построчно. За одну операцию можно выполнить удаление как одной строки, так и нескольких тысяч строк. Если необходимо удалить из таблицы *все* данные, то можно удалить саму таблицу. Естественно, при этом будут удалены и все хранящиеся в ней данные. Однако этот способ следует использовать



лишь в самых крайних случаях, так как помимо данных будет удалена и структура таблицы.

Чаще всего удаление данных выполняется с помощью команды DELETE, удаляющей строки таблицы.

Синтаксис команды, чаще всего использующийся на практике, следующий:

```
DELETE <Имя_таблицы>  
[WHERE <Условие_отбора> ]
```

Таким образом, в большинстве случаев требуется указание лишь имени таблицы, из которой необходимо удалить данные, и логического условия, ограничивающего диапазон удаляемых строк. Причем последнее вовсе не обязательно, и при отсутствии условия из таблицы будут удалены все имеющиеся строки. Как и при выборке и изменении строк, диапазон удаляемых строк формируется с помощью раздела WHERE, использование которого было подробно рассмотрено ранее.

Пусть из таблицы «Учебный план» необходимо удалить дисциплины первого семестра с формой отчетности «зачет», т.е. строки, у которых значение в столбце *Отчетность* равно 'з'. Команда, которая позволит выполнить эту функцию, имеет следующий вид:

```
DELETE Учебный_план  
WHERE (Отчетность = 'з') AND (Семестр = 1)
```

#### *Контрольные вопросы*

1. Сформулируйте на языке SQL запрос для формирования экзаменационной ведомости группы студентов по Дисциплине учебного плана.
2. Сформулируйте на языке SQL запрос, позволяющий сформировать листок зачетной книжки студента:
  - А) по результатам сдачи экзаменов;
  - В) по результатам сдачи зачетов.
3. Сформулируйте на языке SQL запрос, позволяющий получить сводную таблицу «Сессия» (см. Рис. 6.2).
4. Сформулируйте на языке SQL запрос для добавления в структуру БД «Сессия» таблицы «Штатное расписание» с колонками: Должность, Разряд, Оклад, Коэффициент надбавки. Установите связь по внешнему ключу с таблицей «Кадровый состав».
5. Используя новую таблицу «Штатное расписание», сформулируйте на языке SQL запрос для расчета зарплаты с учетом коэффициента надбавки.

## Глава 8. Распределенная обработка данных

### 8.1. Основные условия и требования к распределенной обработке данных

Такая отличительная особенность БД, как многоцелевое параллельное использование данных, предопределяет наличие средств, обеспечивающих практически одновременный и независимый доступ к одним и тем же данным. Причем сама база может быть размещена на одном или нескольких компьютерах.

В [3] приводятся следующие, сформулированные ведущими поставщиками СУБД, свойства «идеальной» системы управления распределенными базами данных:

- *Прозрачность относительно расположения данных:* СУБД должна представлять все данные так, как если бы они были локальными.
- *Гетерогенность системы:* СУБД должна работать с данными, которые хранятся в системах с различной архитектурой и производительностью (независимость от СУБД).
- *Прозрачность относительно сети:* СУБД должна одинаково работать в условиях разнородных сетей.
- *Поддержка распределенных запросов:* пользователь должен иметь возможность объединять данные из любых баз, даже если они размещены в разных системах.
- *Поддержка распределенных изменений:* пользователь должен иметь возможность изменять данные в любых базах, на доступ к которым у него есть права, даже если эти базы размещены в разных системах.
- *Поддержка распределенных транзакций:* СУБД должна выполнять транзакции, выходящие за рамки одной вычислительной системы, и поддерживать целостность распределенной БД даже при возникновении отказов как в отдельных системах, так и в сети.
- *Безопасность:* СУБД должна обеспечивать защиту всей распределенной БД от несанкционированного доступа.
- *Универсальность доступа:* СУБД должна обеспечивать единую методику доступа ко всем данным.

Однако, ни одна из существующих СУБД не достигает этого идеала в следствие следующих практических проблем:

- Низкая и несбалансированная производительность сетей передачи данных, что в распределенных транзакциях сильно снижает общую производительность обработки.
- Обеспечение целостности данных в распределенных транзакциях базируется на принципе «все или ничего» и требует специального протокола двухфазного завершения транзакций, что приводит к длительной блокировке изменяемых данных.

- Необходимо обеспечить совместимость данных стандартного типа, для хранения которых в разных системах используются разные физические форматы и кодировки.

- Выбор схемы размещения системных каталогов. Если каталог будет храниться в одной системе, то удаленный доступ будет замедлен. Если будет размножен – то изменения придется распространять и синхронизировать.

- Необходимо обеспечить совместимость СУБД разных типов и поставщиков.

- Увеличение потребностей в ресурсах для координации работы приложений с целью обнаружения и устранения тупиковых ситуаций в распределенных транзакциях.

Именно указанные причины определили на практике частичность и «этапность» введения в СУБД тех или иных возможностей распределенной обработки данных. В простейшем случае пользователь имеет возможность обращаться по сети к записям в БД, размещенным на других компьютерах. В других случаях СУБД сама производит аутентификацию удаленного клиента и устанавливает сетевые соединения.

В общем случае режимы работы с БД можно классифицировать по следующим признакам:

- *многозадачность* - однопользовательский или многопользовательский;
- *правило обслуживания запросов* – последовательное или параллельное;
- *схема размещение данных* – централизованная или распределенная БД.

Следует отметить, что общая тенденция развития технологий обработки данных вполне соответствует этапам развития средств вычислительной техники и информационных технологий, и в первую очередь – сетевых. В этом смысле следует выделить два класса: *системы распределенной обработки данных* и *системы распределенных баз данных*.

Системы распределенной обработки данных в основном отражают структуру и свойства многопользовательских операционных систем с базой данных, размещенной на большом центральном компьютере (мэйнфрейме). Еще до недавнего времени это был единственно возможный вариант вычислительной среды для реализации больших баз данных. Клиентские места в этом случае реализовались либо в виде терминалов или мини-ЭВМ, обеспечивающих в основном ввод-вывод данных и не имеющих собственных вычислительных ресурсов для функционально-ориентированной обработки получаемых данных.

Развитие сетевых технологий в сочетании с широким распространением персональных ЭВМ и внедрением стандартов открытых систем привело к появлению систем баз данных размещенных в сети разнотип-

ных компьютеров. Такие *системы распределенных баз данных* обеспечивают обработку распределенных запросов, когда при обработке одного запроса используются ресурсы базы, размещенные на различных ЭВМ сети. Система распределенных баз данных состоит из узлов, каждый из которых является СУБД, а узлы взаимодействуют между собой так, что база данных любого узла будет доступна пользователю, так как если бы она была локальной.

Соответственно, программы, обеспечивающие целевую (функциональную) обработку данных, могут быть организованы таким образом, чтобы обеспечить более эффективное использование совокупных вычислительных ресурсов за счет специализированного разделения функций обработки между центральным процессом СУБД и клиентскими функционально-ориентированными процедурами.

Для «типового» приложения обработки данных можно выделить следующие группы (уровни) функций:

- ввод и отображение данных: внешний (пользовательский) уровень реализации целевой функциональной обработки и представления (Presentation logic);
- функциональная обработка, реализующая алгоритм решения задач пользователя. Соответствующие «бизнес-правила» реализуются обычно средствами высокоуровневого языка программирования или расширенного языка манипулирования данными типа ADABAS Natural или 4-GL (Business logic);
- манипулирование данными БД в рамках приложения, которое обычно реализуется средствами SQL (Database logic);
- управление данными и другими ресурсами БД, реализуемое специализированными (внутренними) средствами конкретной СУБД обычно в рамках файловой системы ОС;
- управление процессами обработки: связывание и синхронизация процессов обработки данных разного уровня.

## **8.2. Архитектура распределенной обработки данных**

Почти все модели организации взаимодействия пользователя с базой данных, построены на основе модели "клиент-сервер". Т.е. предполагается, что каждое такое приложение отличается способом распределения функций ранее приведенных групп обработки данных между как минимум двумя частями:

- клиентской, которая отвечает за целевую обработку данных и организацию взаимодействия с пользователем;
- серверной, которая обеспечивает хранение данных, обрабатывает запросы и посылает результаты клиенту для специальной обработки.



В общем случае предполагается, что эти части приложения функционируют на отдельных компьютерах, т.е. к серверу БД с помощью сети подключены компьютеры пользователей (клиенты).

*Сервер* – это программа, реализующая функции собственно СУБД: определение данных, запись-чтение данных, поддержка схем внешнего, концептуального и внутреннего уровней, диспетчеризация и оптимизация выполнения запросов, защита данных.

*Клиент* – это различные программы, написанные как пользователями, так и поставщиками СУБД, внешние или «встроенные» по отношению к СУБД. Программа-клиент организована в виде приложения, работающего «поверх» СУБД, и обращающегося для выполнения операций над данными к компонентам СУБД через интерфейс внешнего уровня<sup>47</sup>.

Разделение процесса выполнения запроса на «клиентскую» и «серверную» компоненту позволяет:

- различным прикладным (клиентским) программам одновременно использовать общую базу данных;
- централизовать функции управления, такие, как защита информации, обеспечение целостности данных, управление совместным использованием ресурсов;
- обеспечивать параллельную обработку запроса в случае распределенных БД;
- высвобождать ресурсы рабочих станций и сети;
- повышать эффективность управления данными за счет использования ЭВМ, специально разработанных для работы СУБД (серверы баз данных и машины баз данных).

### *8.2.1. Базовые архитектуры распределенной обработки*

Учитывая, что одним из основных показателей эффективности сетевой обработки данных является время обслуживания запроса, рассмотрим различные модели архитектуры распределенной обработки на примере, когда прикладная программа работы с базой данных, расположенной на сервере, загружена на рабочую станцию, и пользователю необходимо получить все записи, удовлетворяющие некоторым поисковым условиям.

---

<sup>47</sup> Инструментальные средства, в том числе и утилиты, не отнесены к серверной части очень условно. Являясь не менее важной составляющей, чем ядро СУБД, они выполняются самостоятельно, как пользовательское приложение.



### 8.2.1.1. Архитектура «файл-сервер»

В архитектуре «файл-сервер», схема которой представлена на рис. 8.1, средства организации и управления базой данных (в том числе и СУБД) целиком располагаются на машине клиента, а база данных, представляющая собой обычно набор специализированных структурированных файлов, на машине-сервере. В этом случае серверная компонента представлена даже не средствами СУБД, а сетевыми составляющими операционной системы, обеспечивающими удаленный разделяемый доступ к файлам. Таким образом, «файл-сервер» представляет собой вырожденный случай клиент-серверной архитектуры.

Взаимодействие между клиентом и сервером происходит на уровне команд ввода-вывода файловой системы, которая возвращает запись или блок данных. Запрос к базе, сформулированный на языке манипулирования данными, преобразуется самой СУБД в последовательность команд ввода-вывода, которые обрабатываются операционной системой машины-сервера.

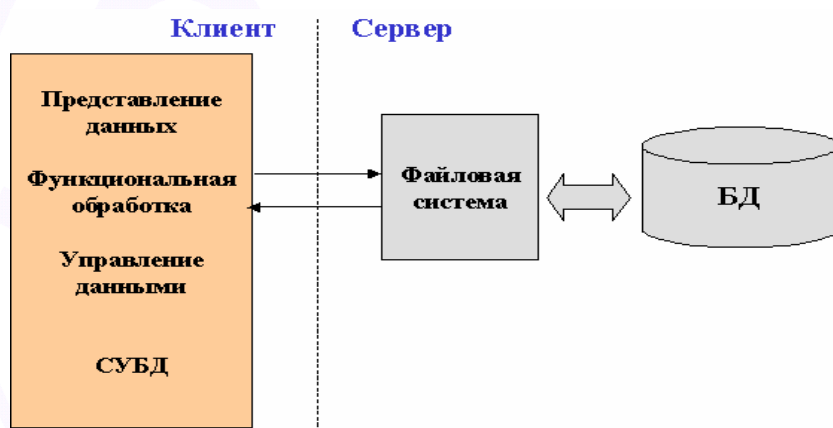


Рис.8.1. Архитектура «файл-сервер»

**Достоинство** - возможность обслуживания запросов нескольких клиентов.

**Недостатки:**

- высокая загрузка сети и машин-клиентов, т.к. обмен идет на уровне единиц информации файловой системы – физических записей, блоков или даже файлов, из которых на машине клиента будут выбраны и представлены необходимые для приложения элементы данных;
- низкий уровень защиты данных, т.к. доступ к файлам БД управляется общими средствами ОС сервера;
- низкий уровень управления целостностью и непротиворечивостью информации, т.к. бизнес-правила функциональной обработки, сосредоточенные на клиентской части, могут быть противоречивыми и несинхронизированными.

В среде файлового сервера программа управления данными, которая выполняется на машине-клиенте, должна осуществить запрос каждой записи базы, после чего она может определить, удовлетворяет ли запись поисковым условиям, лишь после этого передать для функциональной обработки. Очевидно, что для этой схемы характерно наибольшее суммарное время обработки информации.

#### 8.2.1.2. Архитектура «выделенный сервер базы данных»

В архитектуре сервера базы данных, схема которой представлена на рис. 8.2, средства управления базой данных и база данных размещены на машине-сервере.

Взаимодействие между клиентом и сервером происходит на уровне команд языка манипулирования данными СУБД (обычно SQL), которые обрабатываются СУБД на машине-сервере. Сервер базы данных осуществляет поиск записей и анализирует их. Записи, удовлетворяющие условиям, могут накапливаться на сервере и после того, как запрос будет целиком обработан, пользователю на клиентскую машину передаются все логические записи (запрашиваемые элементы данных), удовлетворяющие поисковым условиям.

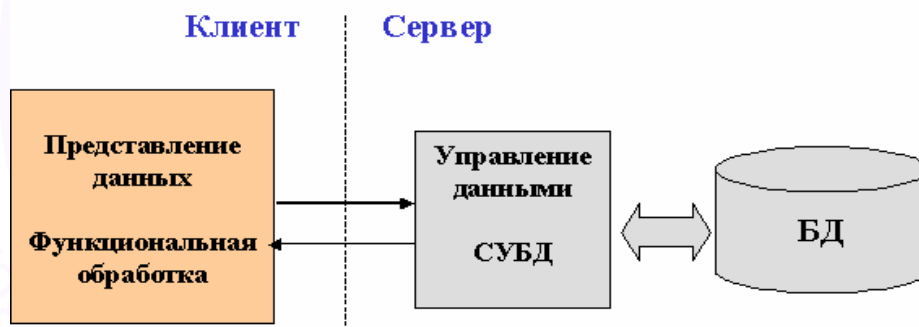


Рис. 8.2. Архитектура с выделенным сервером базы данных

##### Достоинства:

- возможность обслуживания запросов нескольких клиентов;
- снижение нагрузки на сеть и машины сервера и клиентов;
- защита данных осуществляется средствами СУБД, что позволяет блокировать неразрешенные пользователю действия;
- сервер реализует управление транзакциями и может блокировать попытки одновременного изменения одних и тех же записей.

##### Недостатки:

- бизнес-логика функциональной обработки и представление данных могут быть одинаковыми для нескольких клиентских приложений и это увеличит совокупные потребности в ресурсах при исполнении – повторение части кода программ и запросов;

- низкий уровень управления непротиворечивостью информации, т.к. бизнес-правила функциональной обработки, сосредоточенные на клиентской части, могут быть противоречивыми.

Данная технология позволяет снизить сетевой трафик и повысить общую эффективность обработки за счет оптимизации и буферизации ввода-вывода. Т.о., сервер может осуществить поиск и обрабатывать запросы даже быстрее, чем, если бы они обрабатывались на рабочей станции.

### 8.2.1.3. Архитектура «активный сервер баз данных»

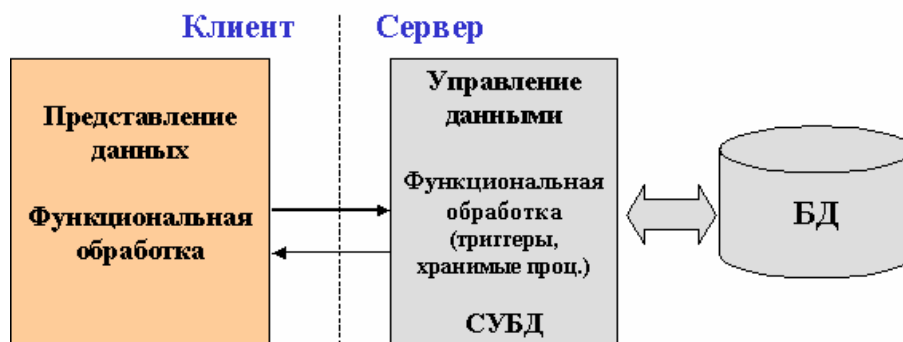
Для того чтобы устранить недостатки, свойственные архитектуре сервера базы данных необходимо, чтобы непротиворечивость бизнес-логики и изменения базы данных контролировались на стороне сервера. Причем некоторые, заранее специфицированные состояния могли бы изменять последовательность взаимодействия приложения с базой данных.

Для этого функции бизнес-логики разделяются между клиентской и серверной частью. Общие или критически значимые функции оформляются в виде *хранимых процедур*, включаемых в состав базы данных. Кроме этого, вводится механизм отслеживания событий БД – *триггеров*, также включаемых в состав базы. При возникновении соответствующего события (обычно изменения данных), СУБД вызывает для выполнения хранимую процедуру, связанную с триггером, что позволяет эффективно контролировать изменение базы данных.

Хранимые процедуры и триггеры могут быть использованы любыми клиентскими приложениями, работающими с базой данных. Это снижает дублирование программных кодов и исключает необходимость компиляции каждого запроса (рис. 8.3).

Недостатком такой архитектуры становится существенно возрастающая загрузка сервера за счет необходимости отслеживания событий и выполнения части бизнес-правил.

Такую архитектуру организации взаимодействия (а также рассматриваемый далее сервер приложений) иногда называют *моделью с «тонким клиентом»*, в отличие от предыдущих архитектур, называемых *моделью с «толстым клиентом»*, где на стороне клиента выполняется большинство функций.



### 8.3. Архитектура «активный сервер баз данных»

#### 8.2.1.4. Архитектура «сервер приложений»

Рассмотренные выше архитектуры являются *двухзвенными*: здесь все функции доступа и обработки распределены между программой клиента и сервером БД.

Дальнейшее снижение требований к ресурсам клиента достигается за счет введения промежуточного звена – *сервера приложений*, на который переносится значительная часть программных компонентов управления данными и большая часть бизнес-логики. При этом серверы баз данных обеспечивают исключительно функции СУБД по ведению и обслуживанию базы данных. Схема трехзвенной архитектуры сервера приложений приведена на рис. 8.4.

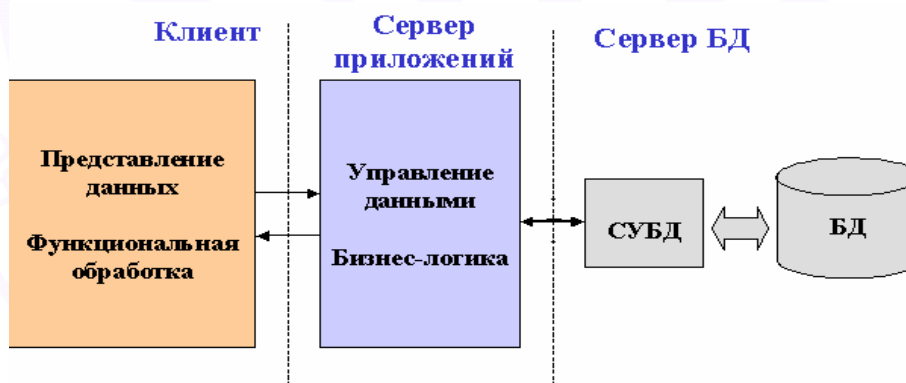


Рис. 8.4. Архитектура сервера приложений

К другим (организационно-технологическим) достоинствам трехзвенной архитектуры можно отнести:

- централизованное ведение бизнес-логики и в случае их изменения отсутствие необходимости их тиражирования в клиентских приложениях;
- отсутствие необходимости устанавливать на клиентских машинах компонент программного обеспечения управления доступом к данным;

- возможность отложенного обновления БД в случае изменения данных, запрошенных с сервера, в автономном режиме. Данные будут обновлены в базе после следующего соединения клиентской программы с сервером приложений.

### 8.2.2. Архитектура сервера баз данных

Повышение эффективности и оперативности обслуживания большого числа клиентских запросов, помимо простого увеличения ресурсов и вычислительной мощности серверной машины, может быть достигнуто двумя путями:

- снижением суммарного расхода памяти и вычислительных ресурсов за счет буферизации (кэширования) и совместного использования (разделяемые ресурсы) наиболее часто запрашиваемых данных и процедур;
- распараллеливанием процесса обработки запроса – использованием разных процессоров для параллельной обработки изолированных подзапросов и/или для одновременного обращения к частям базы данных, размещенным на отдельных физических носителях.

Рассмотрим архитектуры, реализующие следующие модели совместной обработки клиентских запросов.

#### 8.2.2.1. Архитектура «один к одному»

В этом случае (рис. 8.4) для обслуживания каждого запроса запускается отдельный серверный процесс.

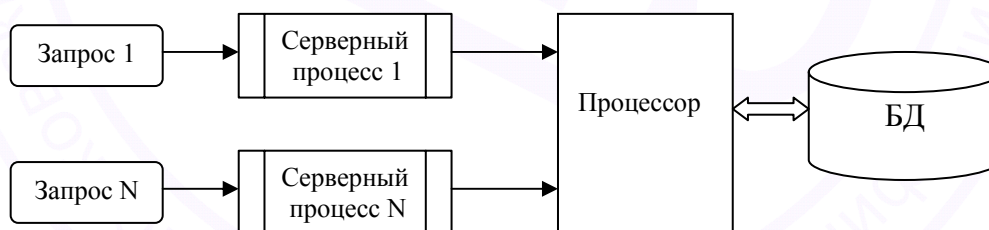


Рис. 8.5. Архитектура сервера «один к одному»

Таким образом, даже если от клиентов поступят совершенно одинаковые запросы, для обработки каждого из них будет запущен отдельный процесс, каждый из которых будет выполнять одинаковые действия и использовать одни и те же ресурсы.

#### 8.2.2.2. Многопоточная односерверная архитектура

Обработку всех клиентских запросов выполняет один серверный процесс (использующий один процессор), взаимодействующий со всеми клиентами и монополююще управляющий ресурсами (рис. 8.6). При этом



для отдельного клиентского процесса создается поток, (thread) в рамках которого локализуется обработка запроса.

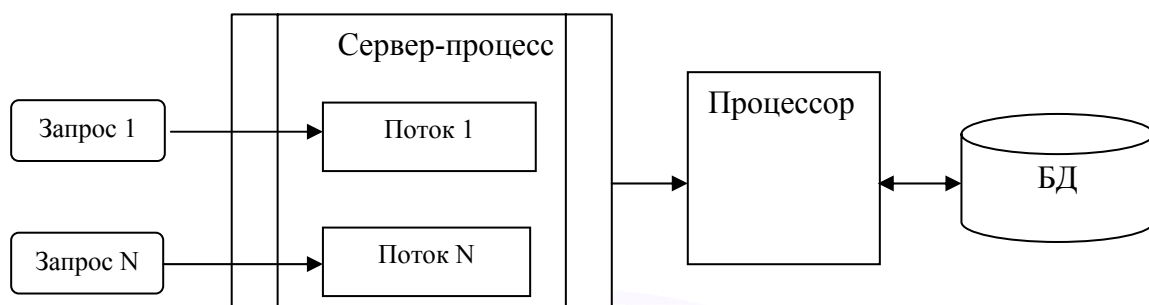


Рис. 8.6. Многопоточковая односерверная архитектура

### 8.2.2.3. Мультисерверная архитектура

В том случае, когда для работы СУБД используются многопроцессорные платформы, обслуживание запросов может быть физически распределено для параллельной обработки между процессорами. Такое решение (рис. 8.7) требует введения дополнительного звена, в задачи которого входит диспетчеризация запросов для обеспечения сбалансированной загрузки процессоров.

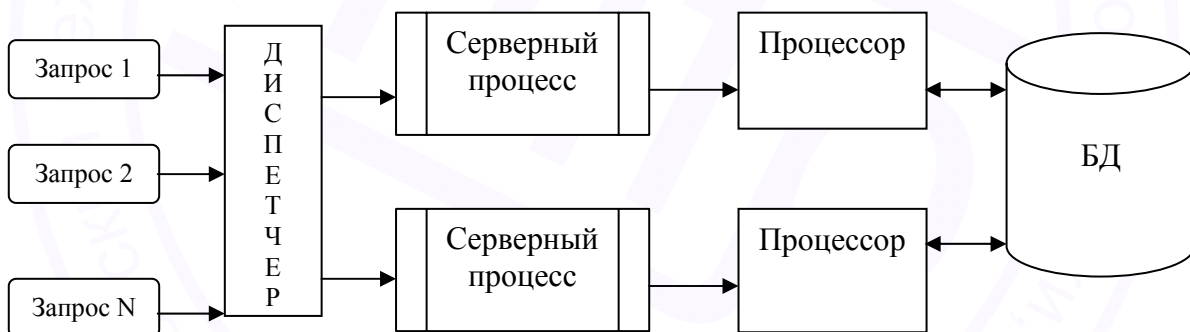


Рис. 8.7. Многопоточковая односерверная архитектура

В том случае, когда серверный процесс реализуется как многопоточное приложение, говорят, что СУБД имеет *мультисерверную многопоточковую архитектуру*.

Следует отметить, что характер распределения запросов в значительной степени зависит от того, поддерживает ли операционная система потоковую обработку, а также от возможностей средств управления приоритетами задач.

#### 8.2.2.4. Серверные архитектуры с параллельной обработкой запроса

Для повышения оперативности за счет распараллеливания процесса обработки отдельного клиентского запроса в мультисерверной архитектуре можно использовать следующие подходы:

1) Размещение хранимых данных БД на нескольких физических носителях (сегментирование базы). Для обработки запроса в этом случае запускаются несколько серверных процессов (использующих обычно отдельные процессоры), каждый из которых независимо от других выполняет одинаковую последовательность действий, определяемую существом запроса, но с данными, принадлежащими разным сегментам базы. Полученные таким образом результаты объединяются и передаются клиенту. Такой тип распараллеливания называют *моделью горизонтального параллелизма*.

2) Запрос обрабатывается по конвейерной технологии. Для этого запрос разбивается на взаимосвязанные по результатам подзапросы, каждый из которых может быть обслужен отдельным серверным процессом независимо от обработки других подзапросов. Получаемые результаты объединяются согласно схеме декомпозиции запроса и передаются клиенту. Такой тип распараллеливания называют *моделью вертикального параллелизма*.

Примерная схема обработки клиентского запроса, построенная с использованием обеих моделей параллелизма (гибридная модель) приведена на рис. 8.8.

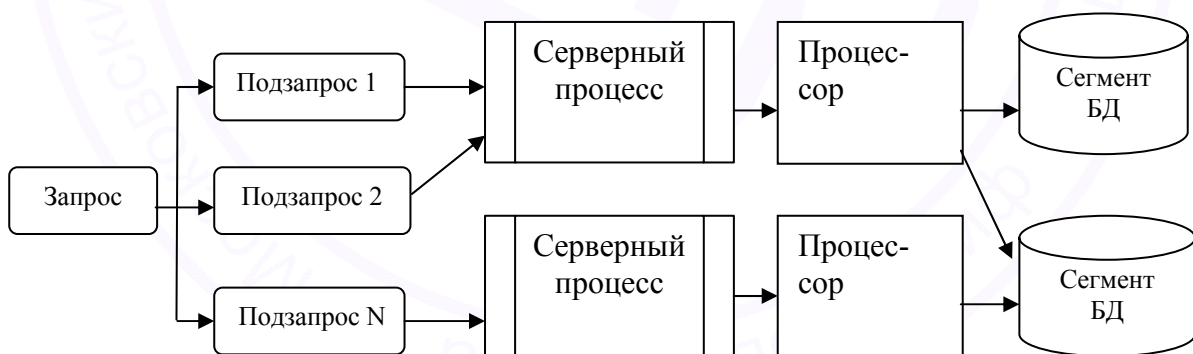


Рис. 8.8. Архитектура сервера обработки запроса при гибридном параллелизме

Использование моделей параллельной обработки позволяет существенно сократить общее время обслуживания запроса, что особенно важно в случае работы с большими базами данных и аналитической обработки (OLAP-приложений).

### 8.3. Технологии и средства доступа к удаленным БД

#### 8.3.1. Программное обеспечение распределенных приложений

Распределенные корпоративные приложения все более усложняются, интегрируя в себя унаследованные приложения, разрабатываемые и вновь приобретаемые готовые программные средства. Кроме того, разные подсистемы решают разные бизнес-задачи, однако одна из главных целей создания корпоративной системы – получить «единый образ» общего состояния системы, что обеспечит пользователям доступ к нужным операциям и ресурсам.

Основа такой инфраструктуры – так называемое *промежуточное программное обеспечение*, позволяющее, не вникая в тонкости сетевых реализаций, создавать и эксплуатировать взаимодействующие между собой приложения с разными требованиями к межмодульным коммуникациям.

Промежуточное ПО эволюционировало вместе с архитектурой клиент-сервер. Ранние, но достаточно эффективные как с точки зрения разработки, так и эксплуатации, частные решения предназначались для упрощения доступа к базам данных в двухзвенной модели, где «толстый» клиент реализует всю логику обработки информации, предоставляемой сервером базы данных. Такие системы вполне удовлетворяли потребностям небольших корпоративных подразделений с ограниченным числом пользователей и невысокой интенсивностью обмена.

Однако, по мере того, как клиент-серверная архитектура стала проникать в сферу высококритичных корпоративных приложений, обслуживающих уже не десятки, а сотни пользователей и работающих со значительными массивами данных, стали очевидны недостатки двухзвенного подхода. Этот способ реализации клиент-серверной схемы доступа ограничивал возможности масштабирования, поскольку увеличение числа обращений к одной базе данных непомерно увеличивало нагрузку на сервер и делало доступ к данным «узким местом» в общей производительности системы. Кроме того, всякая модификация логики приложения требовала внесения изменений во все экземпляры клиентских приложений.

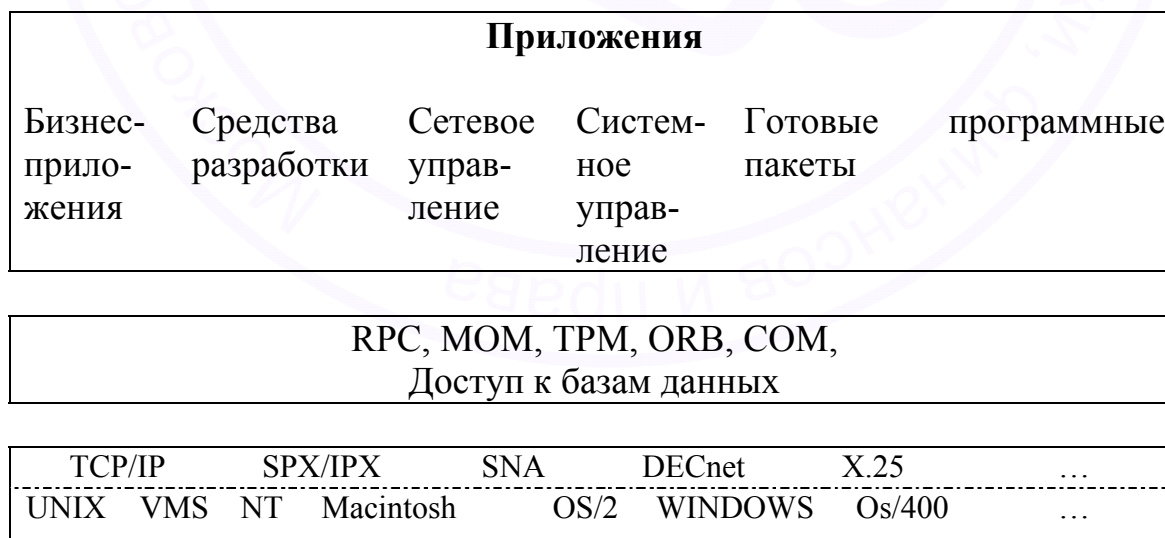
Чтобы избежать таких проблем, для разработки корпоративных приложений используют трехзвенную модель, которая переносит логику приложения на отдельный уровень сервера приложений. В результате клиентская часть приложения становится «тоньше» и в основном отвечает за предоставление удобного пользовательского интерфейса. Как правило, сервер баз данных также освобождается от необходимости поддерживать бизнес-логику, которая в двухзвенной модели реализуется с помощью специальных расширений СУБД, например, хранимых процедур. Перенос основных операций приложения на отдельный уровень позволяет с максимальной эффективностью распределить нагрузку на

аппаратные средства (трехзвенная модель на самом деле может быть многозвенной с разделением нагрузки на несколько серверов приложений) и обеспечивает безболезненное наращивание как функциональности приложения, так и числа обслуживаемых пользователей.

Развитие этого среднего звена клиент-серверной модели идет в сторону усложнения. Ограничиваясь вначале построением более высокого уровня абстракции для взаимодействия приложения с ресурсами данных, разработчик приложения получал возможность использовать общие API (Application Program Interface), которые скрывали различия специфических интерфейсов коммуникационных протоколов более низкого уровня, например, TCP/IP, Sockets или DECNet. Однако теперь этого уже явно недостаточно для построения сложных распределенных приложений. Современные решения не только обеспечивают межпрограммное взаимодействие, но и являются платформой для реализации сервера приложений, обеспечивая обширный набор необходимых служб: управления транзакциями, именования, защиты и т.д.

Вычислительная среда распределенных приложений может включать в себя различные операционные системы, аппаратные платформы, коммуникационные протоколы и разнообразные средства разработки. Соответственно, формат представления данных в различных узлах будет различаться.

Таким образом, в распределенной неоднородной среде программное обеспечение промежуточного уровня играет роль «информационной шины», надстроенной над сетевым уровнем и обеспечивающей доступ приложения к разнородным ресурсам, а также независимую от платформ взаимосвязь различных прикладных компонентов, изолирующую логику приложений от уровня сетевого взаимодействия и ОС (рис. 8.9).



*Рис. 8.9. Структура компонент поддержки удаленного доступа*



ПО промежуточного уровня можно разделить на две категории:

1. ПО доступа к базам данных (например, ODBC-интерфейсы и SQL-шлюзы);
2. ПО межмодульного взаимодействия - системы, реализующие вызов удаленных процедур (RPC – Remote Procedure Call); мониторы обработки транзакций (TP-мониторы); средства интеграции распределенных объектов.

При этом следует отметить, что различия прикладных задач не позволяют построить универсальное ПО, реализовав в одном продукте все необходимые возможности.

### *8.3.2. Доступ к базам данных в двухзвенных моделях клиент-сервер*

В простых двухзвенных моделях клиент-сервер, где несколько баз данных обслуживают ограниченное число пользователей настольных ПК, в роли встроенного ПО доступа к данным могут выступать обычные ODBC-драйверы.

Необходимость в более сложных решениях возникает в больших, разнородных многозвенных системах, где множество приложений в параллельном режиме осуществляет доступ к разнообразным источникам данных, включая разнотипные СУБД и хранилища данных. В таких системах между клиентами и серверами баз данных размещается промежуточное звено – SQL-шлюз, который представляет собой набор общих API, позволяющих разработчику строить унифицированные запросы к разнородным данным (в формате SQL или с помощью ODBC-интерфейса). SQL-шлюз выполняет синтаксический разбор такого запроса, анализирует и оптимизирует его и в конце концов выполняет преобразование в SQL-диалект нужной СУБД. ПО этого типа реализует синхронный механизм связи, когда выполнение приложения, сделавшего запрос, блокируется до момента получения данных.

Примером такого приложения может быть система анализа статистических данных о деятельности компаний, которая отбирает соответствующую информацию из расположенных в различных регионах баз данных с разными СУБД. Подобные решения достаточно просты, не требуют сложных механизмов управления транзакциями и способны обеспечить постепенную миграцию важных приложений с унаследованных платформ в архитектуру клиент-сервер.

Каждое приложение, построенное на основе архитектуры "клиент-сервер", включает, как минимум, две части:

- клиентскую часть, которая отвечает за целевую обработку данных и организацию взаимодействия с пользователем;
- серверную часть, которая собственно хранит данные, обрабатывает запросы и посылает результаты клиенту для специальной обработки.



В общем случае предполагается, что эти части приложения функционируют на отдельных компьютерах, т.е. к выделенному серверу БД с помощью сети подключены узлы - компьютеры пользователей (клиенты). При этом узел-клиент сам может быть СУБД.

Создается такое приложение обычно с использованием средств языков высокого уровня (например, C++, Pascal, Visual Basic), позволяющих реализовать эффективную целевую обработку данных и дружелюбный пользовательский интерфейс. В исходный текст программы включаются SQL-выражения, специфицирующие условия выборки или изменения данных в базе. Во время исполнения приложения эти выражения передаются серверу, который собственно и манипулирует данными. Данные, полученные в результате выполнения сервером SQL-запросов, возвращаются прикладной программе и размещаются в заранее определенных структурах для дальнейшей обработки в том числе корректировки записей.

Рассмотрим различные способы организации доступа прикладной программы к серверу базы данных в двухзвенной архитектуре.

#### 8.3.2.1. Использование библиотек доступа и встраиваемого SQL

Каждая СУБД помимо интерактивной SQL-утилиты обязательно имеет библиотеку процедур доступа и набор драйверов СУБД для различных операционных систем. Схема взаимодействия клиентского приложения с сервером базы данных в этом случае представлена на рис 8.10:

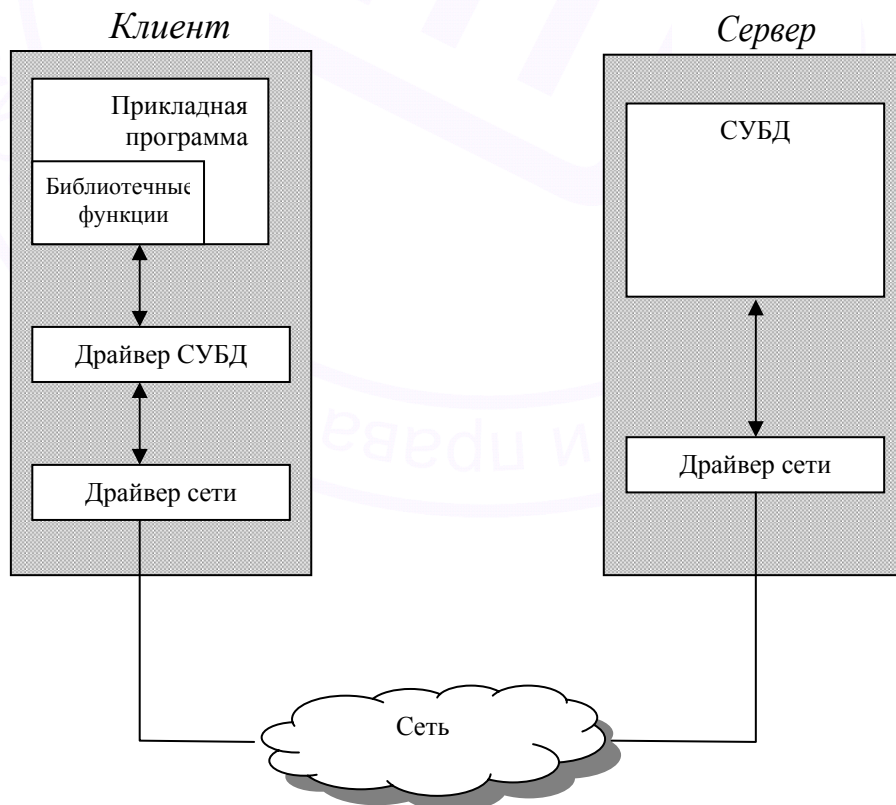


Рис. 8.10. Схема взаимодействия с использованием библиотек процедур доступа

Библиотека доступа содержит набор функций, позволяющих клиентскому приложению соединяться с базой данных, передавать запросы серверу и получать данные – результаты обработки запроса. Типичный набор функций такой библиотеки включает:

- соединение с базой данной;
- запрос к базе данных на выполнение SQL-выражения;
- запрос на извлечение данных;
- запрос на изменение данных;
- закрытие соединения с базой данных.

Обычно в библиотеке присутствуют также функции, позволяющие определить характеристики структуры набора результата (число, порядок и имена столбцов, число строк, номер текущей строки), передвигаться по этой структуре не только вперед, но и назад и т.д.

Библиотечные вызовы преобразуются драйвером базы данных в сетевые вызовы и передаются сетевым программным обеспечением на сервер. На сервере происходит обратный процесс преобразования сетевых пакетов в SQL-запросы, которые обрабатываются СУБД. Результаты обработки передаются клиенту.

Такой способ создания приложений достаточно гибок и позволяет реализовать практически любое приложение, однако имеет и недостатки:

- разработка клиентской программы возможна только для той операционной системы и на том языке программирования, в которых поддерживается библиотека;
- драйвер базы данных определяет допустимые типы сетевых интерфейсов;
- библиотечные функции обычно не унифицированы.

Некоторой модификацией данного способа является использование "встроенного" языка SQL. В этом случае текст программы на языке третьего поколения вместо вызовов функций библиотеки включает непосредственно предложения SQL, которые предваряются выражением "EXEC SQL". Перед компиляцией в машинный код такая программа обрабатывается препроцессором, который транслирует смесь операторов "собственного" языка СУБД и SQL-предложений в промежуточный "чистый" исходный код, а затем коды SQL замещаются вызовами соответствующих процедур из библиотек, поддерживающих конкретную СУБД. Такой подход позволяет несколько снизить степень привязанности к СУБД, например, при переключении прикладной программы на работу с другим сервером базы данных - достаточно указать новый сервер и заново перекомпилировать программу.

### **8.3.2.2. Программный интерфейс уровня вызовов**

Стандарт SQL2 определил интерфейс уровня вызова (CLI - Call Level Interface), в котором стандартизован общий набор рабочих процедур, обеспечивающий совместимость со всеми основными типами серверов баз данных.

Технологическая основа CLI – размещаемая на компьютере клиента специальная библиотека, в которой хранятся вызовы процедур и сетевых компонентов для организации связи с сервером. Это программное обеспечение поставляется обычно в составе среды разработки и поддерживает разнообразные сетевые протоколы.

Использование программных вызовов позволяет свести к минимуму операции на компьютере-клиенте. В общем случае клиент формирует оператор языка SQL в виде строки и пересылает ее на сервер посредством процедуры исполнения (execute). Когда же сервер в качестве ответа возвращает несколько строк данных, клиент считывает результат последовательным вызовом процедуры выборки данных. Далее данные из столбцов полученной таблицы могут быть связаны с соответствующими переменными приложения. Вызов специальной процедуры позволяет клиенту определить число полученных строк, столбцов и типы данных в каждом столбце.

### 8.3.2.3. Открытый интерфейс доступа к базам данных

Спецификация открытого интерфейса баз данных (ODBC - Open Database Connectivity), предназначена для унификации доступа к данным, размещенным на удаленных серверах. ODBC опирается на спецификации CLI.

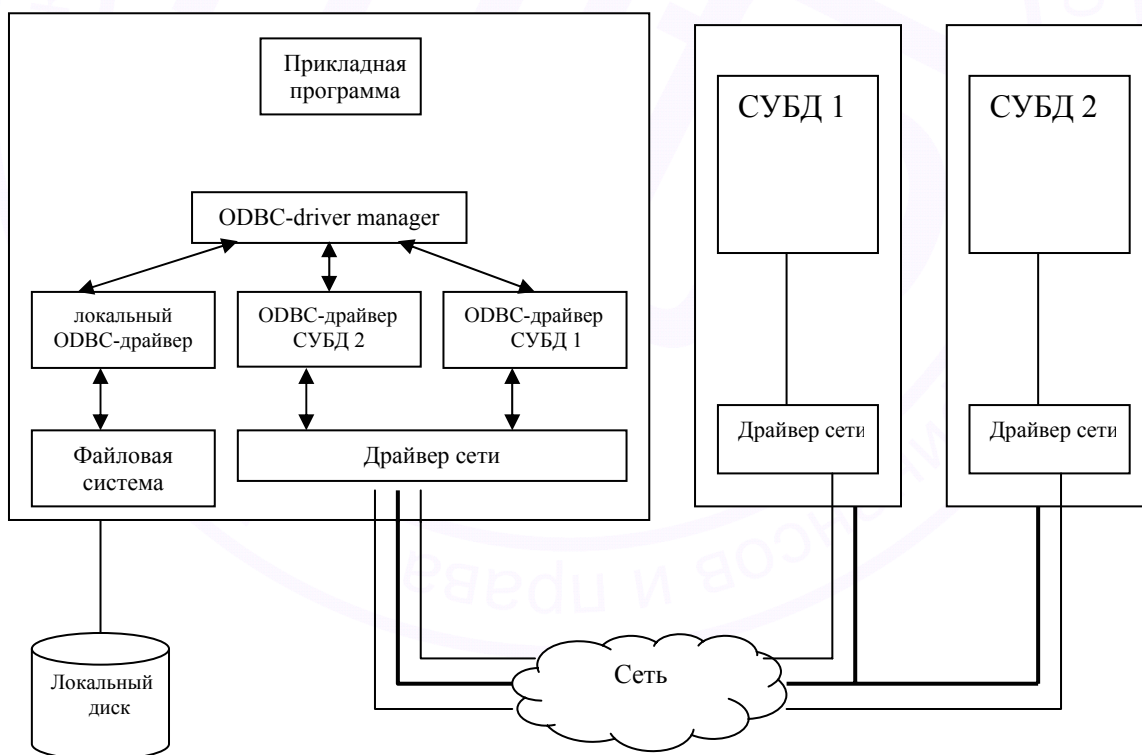


Рис.8.11. Структурная схема доступа к данным с использованием ODBC

ODBC представляет собой программный слой, унифицирующий интерфейс взаимодействия приложений с базами данных. За реализацию

особенностей доступа к каждой отдельной СУБД отвечает соответствующий специальный ODBC-драйвер. Пользовательское приложение этих особенностей не видит, т.к. взаимодействует с универсальным программным слоем более высокого уровня. Таким образом, приложение становится в значительной степени независимым от СУБД. Вместо создания в каждом отдельном случае СУБД-приложения с обращениями через «родной», но быстро устаревающий интерфейс, можно использовать один общий стандартизированный программный интерфейс.

В архитектуре ODBC используется один ODBC Driver Manager и несколько ODBC-драйверов, обеспечивающих доступ к конкретным СУБД. Driver Manager связывает приложение и интерфейсные объекты, которые выполняют обработку SQL-запросов к конкретной СУБД.

Такой подход является достаточно универсальным, стандартизируемым, что и позволяет использовать ODBC-механизмы для работы практически с любой системой.

Однако этот способ также не лишен недостатков:

- увеличивается время обработки запросов (как следствие введения дополнительного программного слоя);
- необходимы предварительная инсталляция и настройка ODBC-драйвера (указание драйвера СУБД, сетевого пути к серверу, базы данных и т.д.) на каждом рабочем месте. Параметры этой настройки являются статическими, т.е. приложение их изменить самостоятельно не может.

#### *8.3.2.4. Мобильный интерфейс к базам данных на платформе Java*

JDBC (Java Data Base Connectivity) – это интерфейс прикладного программирования (API) для выполнения SQL-запросов к базам данных из программ, написанных на платформенно-независимом языке Java, позволяющем создавать как самостоятельные приложения (standalone application), так и апплеты, встраиваемые в web-страницы.

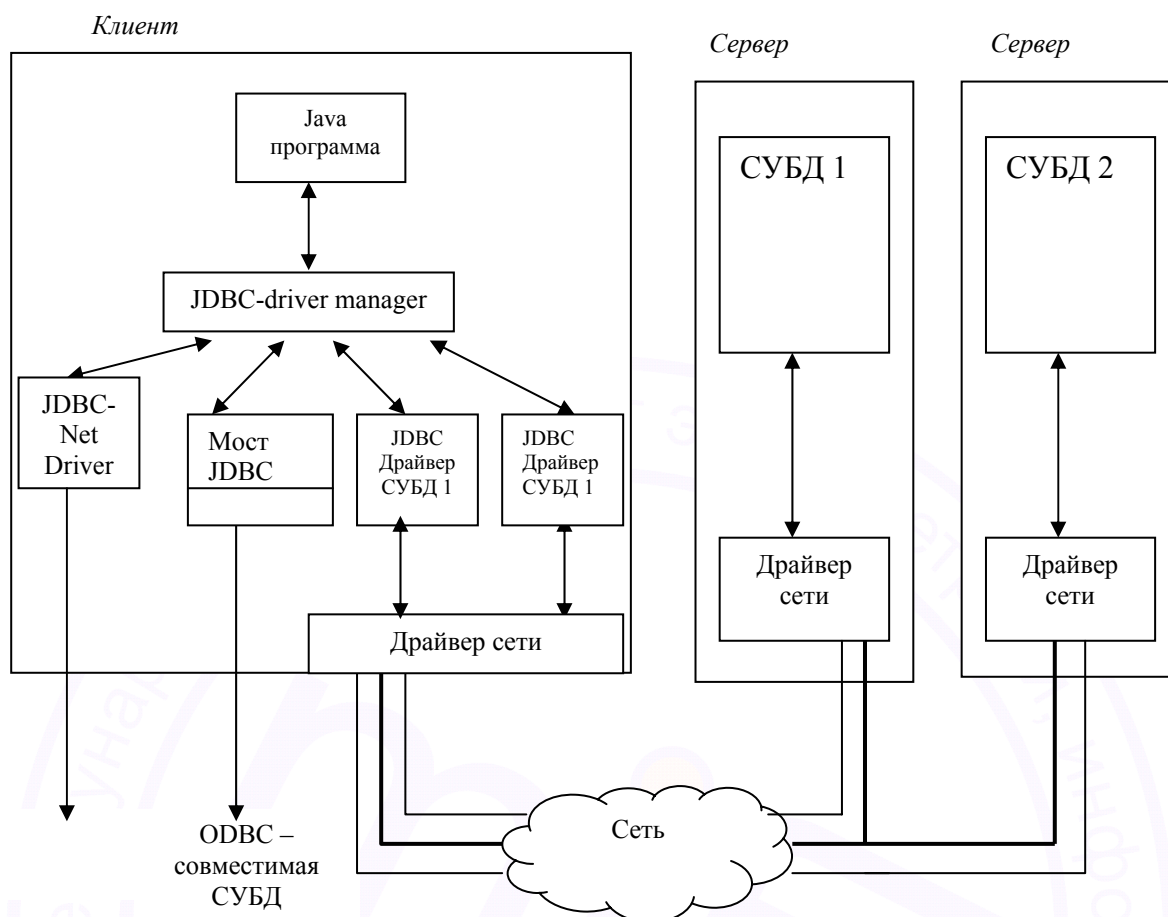


Рис. 8.12. Структурная схема доступа к данным с использованием JDBC

JDBC во многом подобен ODBC, он также построен на основе спецификации CLI, однако имеет ряд следующих отличий.

- приложение загружает JDBC-драйвер динамически, следовательно, администрирование клиентов упрощается, более того, появляется возможность переключаться на работу с другой СУБД без перенастройки клиентского рабочего места.

- JDBC, как и Java в целом, не привязан к конкретной аппаратной платформе, следовательно проблемы с переносимостью приложений практически снимаются.

- использование Java-приложений и связанной с ними идеологии "тонких клиентов" обещает снизить требования к оборудованию клиентских рабочих мест.

Обобщенная структурная схема доступа к данным с использованием JDBC приведена на рис. 8.12.



### 8.3.2.5. Прикладные интерфейсы OLE DB и ADO

OLE DB (Object Linking and Embedding Data Base), как и ODBC – это прикладные интерфейсы доступа к данным с использованием SQL.

OLE DB специфицирует взаимодействие, обеспечивая единый интерфейс доступа к данным через провайдеров – поставщиков данных не только из реляционных БД. В отличие от ODBC, OLE DB предоставляет общее решение обеспечения COM-приложениям доступа к информации независимо от типа источника данных.

OLE DB включает два базовых компонента: *провайдер данных* и *потребитель данных*. Потребитель (клиент) – это приложение или COM-компонент, обращающийся посредством API-вызовов к OLE DB. Провайдер (сервер) - это приложение отвечающее на вызовы OLE DB и возвращающее запрашиваемый объект – обычно это данные в табличном виде.

ADO (Active Data Object) – это универсальный интерфейс высокого уровня к OLE DB. Модель объекта ADO не содержит таблиц, среды или машины БД. Здесь основными объектами являются следующие: объект *Соединение*, создающий связь с провайдером данных; объект *Набор данных* и объект *Команда* – выполнение процедуры, SQL-строки

В общем случае ADO можно рассматривать как язык программирования с БД, позволяющий выбирать, модифицировать и удалять записи. И поскольку он опирается на универсальный OLE DB, то может использоваться практически в любых приложениях Microsoft.

### 8.3.2.6. Взаимосвязь механизмов доступа к данным

Рассмотренные технологии построения приложения ориентированы на извлечение данных непосредственно из статического источника (хранилища данных) и не могут обращаться за данными к другому прикладному модулю.

Один из способов организации доступа к данным заключается в непосредственном использовании API. Однако это означает полную зависимость создаваемого приложения от используемой СУБД. В этом случае переход к другой системе (например, для перехода от настольной системы к системе типа клиент/сервер) влечет за собой переписывание большей части программного кода клиентского приложения.

Таким образом, следующим этапом в обеспечении доступа клиентского приложения к данным является создание универсального механизма доступа к БД, обеспечивающего для клиентского приложения стандартный набор функций, классов или сервисов (служб), необходимых для работы с различными системами управления базами данных. Эти стандартные функции (классы или сервисы) должны размещаться в библиотеках, именуемых *драйверами* или *провайдерами баз данных* (data base drivers (providers)). Каждая такая библиотека реализует набор

стандартных функций, классов или сервисов, используя обращения API к конкретной СУБД.

Наиболее популярными механизмами доступа к данным (Universal Data Access, UDA) в настоящий момент являются:

- **ODBC**
- **OLE DB**
- **ADO**
- **BDE**

Первые три являются фактически промышленными стандартами. Последний долгое время был единственным механизмом доступа к данным, реализованным в инструментальных средствах разработки компании Borland (например, Delphi, C++Builder).

На рис. 8.13 схематически представлены различные механизмы доступа к данным, включая непосредственные вызовы клиентской частью API системы управления базой данных.

#### ***8.4. Технологии межмодульного взаимодействия***

Следующий тип промежуточного ПО ориентирован на архитектуру приложения, в которой один прикладной модуль, используя специальные протоколы, получает данные из другого модуля.

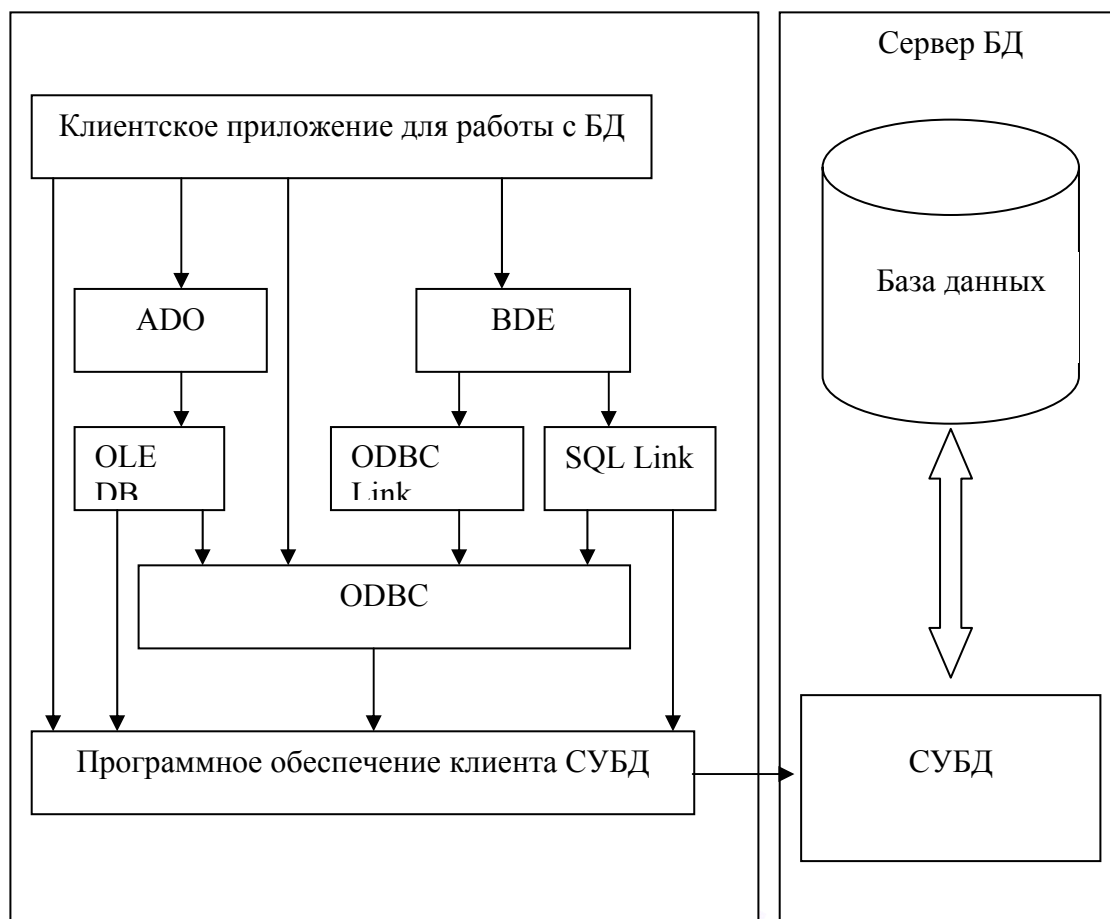


Рис. 8.13. Взаимосвязь механизмов доступа к данным

#### 8.4.1. Спецификация вызова удаленных процедур

Средства вызова удаленных процедур (RPC) поддерживает синхронный режим коммуникаций между двумя прикладными модулями (клиентом и сервером).

Для установки связи, передачи вызова и возврата результата клиентский и серверный процессы обращаются к специальным процедурам – клиентскому и серверному суррогатам (*client stub* и *server stub*). Эти процедуры не реализуют никакой прикладной логики и предназначены только для организации взаимодействия удаленных прикладных модулей.

Каждая функция на сервере, которая может быть вызвана удаленным клиентом, должна иметь такой суррогатный процесс. Если клиент вызывает удаленную процедуру, вызов вместе с параметрами передается клиентскому суррогату. Он упаковывает эти данные в сетевое сообщение и передает его серверному суррогату. Тот, в свою очередь, распаковывает полученные данные и передает их реальной функции сервера и затем предельывает обратную процедуру с результатами. Таким образом

изолируются прикладные модули клиента и сервера от уровня сетевых коммуникаций.

По существу, RPC реализует в распределенной среде принципы традиционного структурного программирования. Клиент обращается к процессу-суррогату так, как будто он и есть реальный серверный процесс, и этот вызов ничем не отличается от вызова локальной функции. Как и в случае нераспределенной программы, вызов процедуры на удаленном компьютере влечет за собой передачу управления этой процедуре, то есть блокирует выполнение клиентской программы на время обработки вызова.

В общем случае механизм RPC создает статические отношения между компонентами распределенного приложения – привязка клиентского процесса к конкретным серверным суррогатам происходит на этапе компиляции и не может быть изменена во время выполнения. Этим RPC отличается от таких более выгодных решений, как TP-мониторы, которые поддерживают возможности оптимального распределения нагрузки на серверы и средства восстановления при сбоях.

Ключевым компонентом RPC является язык описания интерфейсов (interface definition language – IDL), предназначенный для определения интерфейсов, которые задают контрактные отношения между клиентом и сервером. Интерфейс содержит определение имени функции и полное описание передаваемых параметров и результатов выполнения.

Язык IDL обеспечивает независимость механизма RPC от языков программирования – вызывая удаленную процедуру, клиент может использовать свои языковые конструкции, которые IDL-компилятор преобразует в свои описания. На сервере IDL-описания обратно преобразуются в конструкции языка программирования, на котором реализован серверный процесс.

#### *8.4.2. Мониторы обработки транзакций*

Первоначально основной задачей мониторов обработки транзакций (TP-мониторов) в среде клиент-сервер было сокращение числа соединений клиентских систем с базами данных. При непосредственном обращении клиента к серверу базы данных для каждого клиента устанавливается соединение с СУБД, которое порождает запуск отдельного процесса в рамках операционной системы. TP-мониторы брали на себя роль концентратора таких соединений, становясь посредником между клиентом и сервером базы данных.

Постепенно, с развитием трехзвенной архитектуры клиент-сервер функции TP-мониторов расширились, и они превратились в платформу для транзакционных приложений в распределенной среде с множеством баз данных под различными СУБД.

TP-мониторы представляют собой одну из самых сложных и многофункциональных технологий в мире промежуточного ПО. Основное

их назначение – автоматизированная поддержка приложений, оформленных в виде последовательности транзакций. Каждая транзакция – это законченный блок обращений к ресурсу (как правило, базе данных) и некоторых действий над ним, для которого гарантируется выполнение четырех условий:

- *атомарность* – операции транзакции образуют неразделимый, атомарный блок с определенным началом и концом. Этот блок либо выполняется от начала до конца, либо не выполняется вообще. Если в процессе выполнения транзакции произошел сбой, происходит *откат* (возврат) к исходному состоянию;

- *согласованность* – по завершении транзакции все задействованные ресурсы находятся в согласованном состоянии;

- *изолированность* – одновременный доступ транзакций различных приложений к разделяемым ресурсам координируется таким образом, чтобы эти транзакции не влияли друг на друга;

- *долговременность* – все изменения данных (ресурсов), осуществленные в процессе выполнения транзакции, не могут быть потеряны.

В системе без TP-монитора, обеспечение этих свойств берут на себя серверы распределенной базы данных, использующие двухфазный протокол (2PC- two-phase commit). Протокол 2PC описывает двухфазный процесс, в котором перед началом распределенной транзакции все системы опрашиваются о готовности выполнить необходимые действия. Если каждый из серверов баз данных дает утвердительный ответ, транзакция выполняется на всех задействованных источниках данных. Если хотя бы в одном месте происходит какой-либо сбой, будет выполнен откат для всех частей транзакции.

Однако в системе с распределенными базами данных выполнение протокола 2PC можно гарантировать только в том случае, если все источники данных принадлежат одному поставщику. Поэтому для сложной распределенной среды, которая обслуживает тысячи клиентских мест и работает с десятками разнородных источников данных, без монитора транзакций не обойтись. TP-мониторы способны координировать и управлять транзакциями, которые обращаются к серверам баз данных от различных поставщиков благодаря тому, что большинство этих продуктов помимо протокола 2PC поддерживают *транзакционную архитектуру* (XA), которая определяет интерфейс для взаимодействия TP-монитора с менеджером ресурсов, например, СУБД Oracle или Sybase. Спецификация XA является частью общего стандарта распределенной обработки транзакций (distributed transaction processing – DTP), разработанного X/Open (рис. 8.14).

Функции современных TP-мониторов не ограничиваются поддержкой целостности прикладных транзакций. Большинство продуктов этой категории способны распределять, планировать и выделять приоритеты запросам нескольких приложений одновременно, тем самым,



сокращая процессорную нагрузку и время отклика системы. Обработка запросов организуется в виде «нитей» ОС, а не полновесных процессов, тем самым значительно снижая загруженность системы.

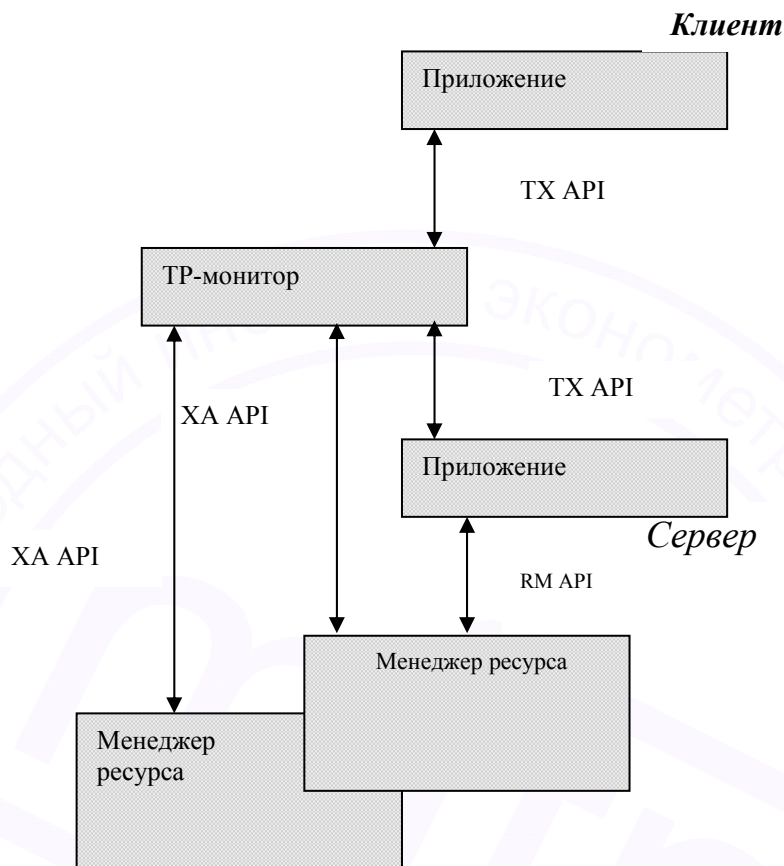


Рис. 8.14. Архитектура распределенной обработки транзакций

Таким образом снимается одно из серьезных ограничений производительности и масштабируемости клиент-серверной среды – необходимость поддержки отдельного соединения с базой данных для каждого клиента.

#### 8.4.3. Корпоративные серверы приложений

Появление серверов приложений как отдельных готовых решений связано и с бурным вторжением Web-технологий в сферу корпоративных высоко-критичных систем. Однако возможности протокола HTTP ограничены функциями связи без каких-либо средств сохранения информации о состоянии, поэтому он не подходит для поддержки мощных корпоративных систем

На рис. 8.15 приведен «идеальный» состав сервера приложений с максимальным набором необходимых служб и средств связи с клиентскими системами и информационными ресурсами.

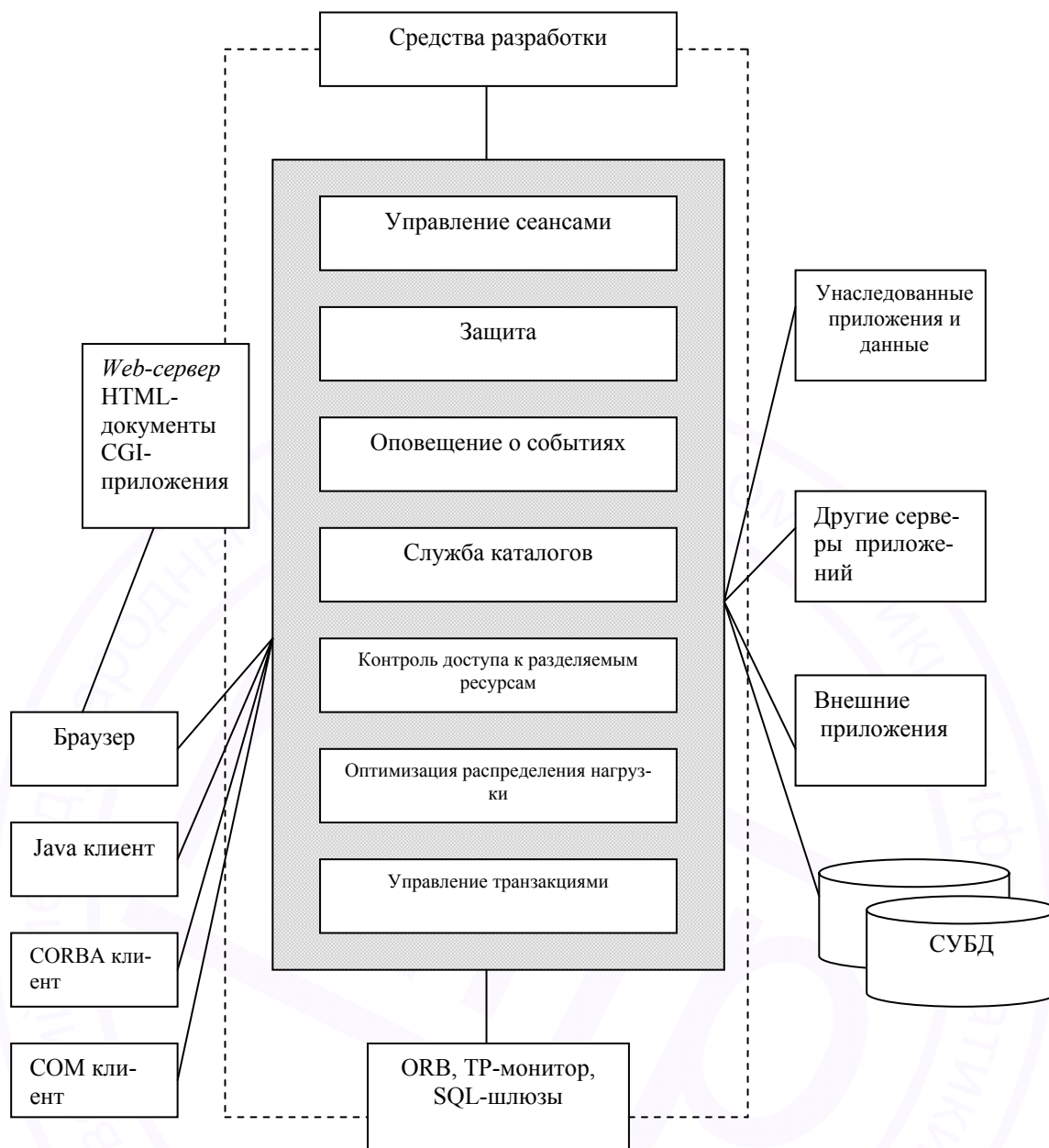


Рис. 8.15. Обобщенная структура сервера приложений

Сегодня прикладные разработки базируются на одной из двух компонентных моделей – MTS/DCOM и CORBA, способных интегрировать объекты на удаленных платформах.

Обе модели распространяют принципы вызова удаленных процедур на объектные распределенные приложения и обеспечивают прозрачность реализации и физического размещения серверного объекта для клиентской части приложения; поддерживают возможность взаимодействия объектов, созданных на различных объектно-ориентированных языках и скрывают от приложения детали сетевого взаимодействия.

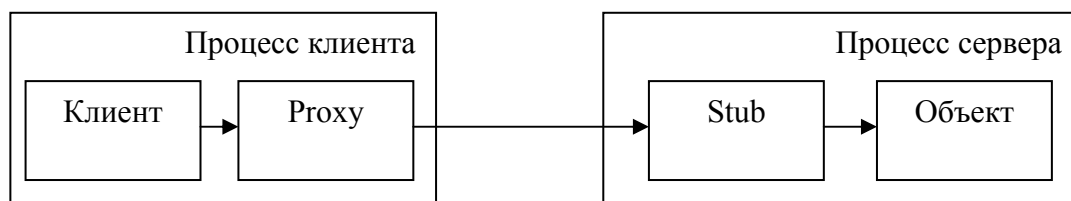


Рис. 8.16. DCOM-технология взаимодействия «клиент-сервер»

В DCOM взаимодействие удаленных объектов, представленное на рис. 8.16, базируется на спецификации DCE RPC, а CORBA использует брокер объектных запросов (ORB), синхронный механизм которого во многом схож с RPC.

В DCOM-технологии взаимодействие между клиентом и сервером осуществляется через двух посредников. Клиент помещает параметры вызова в стек и обращается к методу интерфейса объекта. Это обращение перехватывает посредник Proxy, упаковывает параметры вызова в COM-пакет и адресует его в Stub, который в свою очередь распаковывает параметры в стек и инициирует выполнение метода объекта в пространстве сервера.

CORBA-технология также использует интерфейс объекта, но в этом случае, схема взаимодействия объектов (рис. 8.17) включает промежуточное звено (*Smart agent*), реализующее доступ к удаленным объектам. Smart agent установленный на машинах сетевого окружения (сервере локальной сети или Internet-узле), моделирует сетевой каталог известных ему серверов объектов. При создании сервера происходит автоматическая регистрация его объектов в каталоге одного или нескольких Smart agent.

Связи между брокерами осуществляется в соответствии с требованиями специального протокола General Inter ORB Protocol, определяющего низкоуровневое представление данных и множество форматов сообщений.

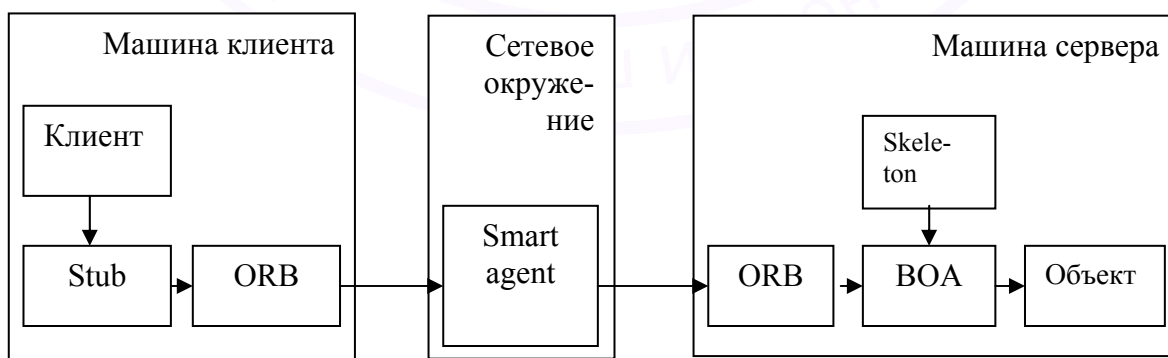


Рис. 8.17. CORBA-технология взаимодействия клиент-сервер

На машине клиента создаются два объекта-посредника: Stub и ORB (Object Required Broker – брокер вызываемого объекта). Также как и в DCOM-технологии, Stub передает перехваченный вызов брокеру, который посылает широковещательное сообщение в сеть. Smart agent, получив сообщение, отыскивает сетевой адрес сервера и передает запрос брокеру, размещенному на машине сервера. Вызов требуемого объекта производится через специальный базовый объектный адаптер (BOA). При этом данные в стек пространства вызываемого объекта помещает особый объект сервера (Skeleton), который вызывается адаптером.

Кроме того, CORBA помимо механизма взаимодействия с помощью ORB, включает в себя ряд общих служб CORBA Services (служба каталогов, защиты, оповещения о событиях, поддержки транзакций и ряд других), а также реализаций объектов для разных прикладных областей.

Ключевым компонентом архитектуры CORBA является язык описания интерфейсов IDL, на уровне которого поддерживаются «контрактные» отношения между клиентом и сервером и обеспечивается независимость от конкретного объектно-ориентированного языка. CORBA IDL поддерживает основные понятия объектно-ориентированной парадигмы (инкапсуляцию, полиморфизм и наследование).

В модели DCOM также может использоваться разработанный Microsoft язык IDL, который, однако, играет вспомогательную роль и используется в основном для удобства описания объектов. Реальная интеграция объектов в DCOM происходит не на уровне абстрактных интерфейсов, а на уровне бинарных кодов, и это одно из основных различий этих двух объектных моделей.

И DCOM, и CORBA, в отличие от процедурного RPC, дают возможность динамического связывания удаленных объектов: клиент может обратиться к серверу-объекту во время выполнения, не имея информации об этом объекте на этапе компиляции. В CORBA для этого существует специальный интерфейс динамического вызова DII, а COM использует механизм OLE-Automation. Информацию о доступных объектах сервера на этапе выполнения клиентская часть программы получает из специального хранилища метаданных об объектах – репозитория интерфейсов Interface Repository в случае CORBA, или библиотеки типов (Type Library) в модели DCOM. Эта возможность очень важна для больших распределенных приложений, поскольку позволяет менять и расширять функциональность серверов, не внося существенных изменений в код клиентских компонентов программы. Например, банковское приложение, основная бизнес-логика которого поддерживается сервером в центральном офисе, а клиентские системы разбросаны по филиалам в разных городах.

#### 8.4.4. Доступ к данным с помощью ADO.NET

ADO.NET является преемником Microsoft ActiveX Data Objects (ADO). Это W3C-стандартизированная модель программирования для создания распределенных прикладных программ, нацеленных на совместное использование данных.

ADO.NET является программным интерфейсом (API) для прикладного программного обеспечения, позволяющим обращаться к данным и другой информации. ADO.NET поддерживает такие современные требования, как создание клиентского интерфейса к базам данных на фронтальном уровне и на уровне промежуточного слоя объектов клиентских приложений, инструментальных средств, языков программирования или Internet-браузера.

ADO.NET, подобно ADO, обеспечивает интерфейс доступа к OLE DB-совместимым источникам данных, таким, как Microsoft SQL Server 2000. Прикладные программы, позволяющие пользователям совместно использовать данные, могут использовать ADO.NET для подключения к источникам данных, а также для поиска, и модификации этих данных. Прикладные программы также могут использовать OLE DB для управления данными, хранящимися в неструктурированных форматах, таких, как Microsoft Excel.

В решениях, требующих автономного или удаленного доступа к данным, ADO.NET использует XML для обмена данными между программами или с Web страницами. Любой компонент, который обслуживает XML, также может использовать и компоненты ADO.NET. Если передача пакетов компонентом ADO.NET подразумевает поставку набора данных в файле XML, то компонентом, способным обеспечить его получение, может быть только компонент ADO.NET. Передача данных в XML-формате даёт возможность легко отделить обработку данных от компонент пользовательского интерфейса.

Для распределенных приложений использование наборов данных XML в ADO.NET обеспечивает лучшую эффективность, чем использование COM для офлайнового обслуживания данных в ADO. Поскольку передача наборов данных происходит через файлы XML, описанные в достаточно простом стандартном языке, и являющиеся обычными текстовыми файлами, компоненты ADO.NET не имеют архитектурных ограничений, свойственных COM. Фактически, любые два компонента могут совместно использовать наборы XML-данных при условии, что они оба используют ту же самую XML-схему форматирования.

ADO.NET обладает хорошей масштабируемостью, что удобно для совместно использующих данные Web-приложений. Кроме того, ADO.NET не использует длительные блокировки баз данных и активные подключения, которые на долгое время монополизировать ресурсы сервера, являющиеся, как правило, весьма ограниченными. Это позволяет увеличивать число пользователей без значительного увеличения загрузки ресурсов системы.



### *Контрольные вопросы*

1. Сформулируйте основные требования к системам управления распределенными базами данных.
2. Перечислите основные условия и предпосылки появления систем управления распределенными базами данных.
3. Перечислите основные различия системы распределенной обработки данных и системы распределенных баз данных.
4. Обоснуйте целесообразность разделения «клиентских» и «серверных» функций.
5. Проведите сравнительный анализ распределения функций для различных базовых архитектур.
6. Определите основные принципы и примерные структурные схемы сервера распределенной обработки.
7. Перечислите основные решения распределенной обработки на основе межмодульного взаимодействия.

## Глава 9. Транзакции и целостность БД

Применение СУБД для работы с интегрированными БД выявило особую важность проблемы *целостности* БД. Под целостностью БД понимают правильность и непротиворечивость ее содержимого. Нарушение целостности может быть вызвано, например, ошибками и сбоями, так как в этом случае система не в состоянии обеспечить нормальную обработку или выдачу правильных данных.

Рассмотрим два аспекта целостности – на уровне отдельных объектов и операций и на уровне базы данных в целом.

Первый аспект целостности обеспечивается на уровне структур данных и отдельных операторов языковых средств СУБД (вспомним ограничения целостности для столбцов и таблиц в языке SQL). При нарушениях такой целостности (например, ввод значения больше 10 в столбец *Семестр* таблицы «Учебный\_план» БД «Сессия») соответствующий оператор отвергается.

Некоторые ограничения целостности не нужно выражать в явном виде, поскольку они встроены в структуры данных. Например, в СУБД, поддерживающей структуры, составленные из записей, каждый экземпляр записи в БД должен отображать спецификацию типа записи. Это означает, что все поля, специфицированные в описании типа, должны быть представлены в каждом экземпляре записи, а значение, заносимое в отдельное поле, должно иметь соответствующий описанию тип данных.

Часто же база данных может иметь такие ограничения целостности, которые требуют обязательного выполнения не одной, а нескольких операций. Для иллюстрации примеров этой главы расширим функциональные возможности учебной БД «Сессия», добавив в таблицу «Кадровый\_состав» столбец *Нагрузка* для решения дополнительной задачи – расчета общей годовой нагрузки преподавателей (в часах учебной работы). Тогда любая операция по внесению изменений или по добавлению данных в столбец *ID\_Преподаватель* таблицы «Учебный\_план» должна сопровождаться соответствующими изменениями данных в столбце *Нагрузка*. Если после внесения изменений в столбец *ID\_Преподаватель* произойдет сбой, то БД окажется в нецелостном состоянии.

Для обеспечения целостности в случае ограничений на базу данных, а не на какие-либо отдельные операции, служит аппарат транзакций.

*Транзакция* – неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации) такая, что:

- либо результаты всех операторов, входящих в транзакцию, отображаются в БД;
- либо воздействие всех этих операторов полностью отсутствует.

При этом для поддержания ограничений целостности на уровне БД допускается их нарушение внутри транзакции так, чтобы к моменту завершения транзакции условия целостности были соблюдены.

Для обеспечения контроля целостности каждая транзакция должна начинаться при целостном состоянии БД и должна сохранить это состояние целостным после своего завершения. Если операторы, объединенные в транзакцию, выполняются, то происходит нормальное завершение транзакции, и БД переходит в обновленное (целостное) состояние (ситуация COMMIT на рис. 9.1). Если же происходит сбой при выполнении транзакции, то происходит так называемый откат к исходному состоянию БД (ситуация ROLLBACK на рис. 9.1).

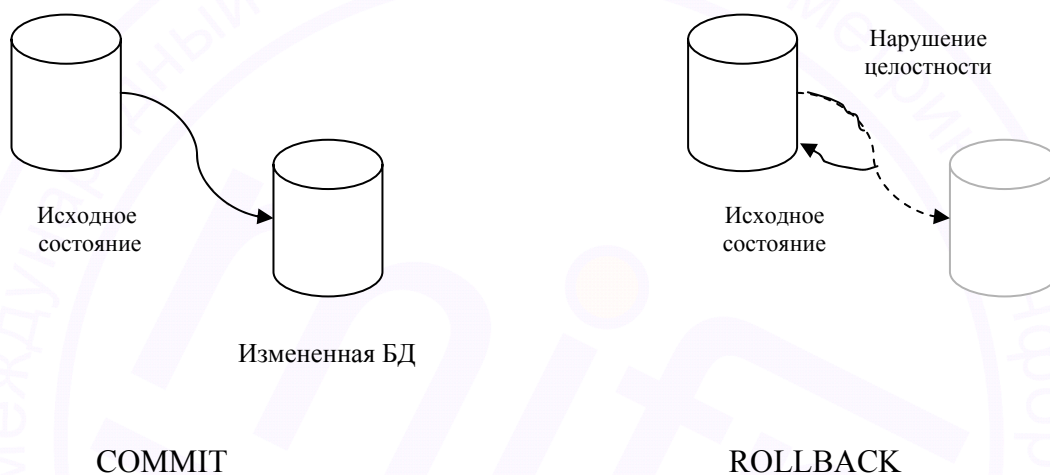


Рис. 9.1. Выполнение и откат транзакции

### 9.1. Модели транзакций

Рассмотрим две модели транзакций, используемые в большинстве коммерческих СУБД: модель автоматического выполнения транзакций и модель управляемого выполнения транзакций, обе основанные на инструкциях языка SQL – COMMIT и ROLLBACK.

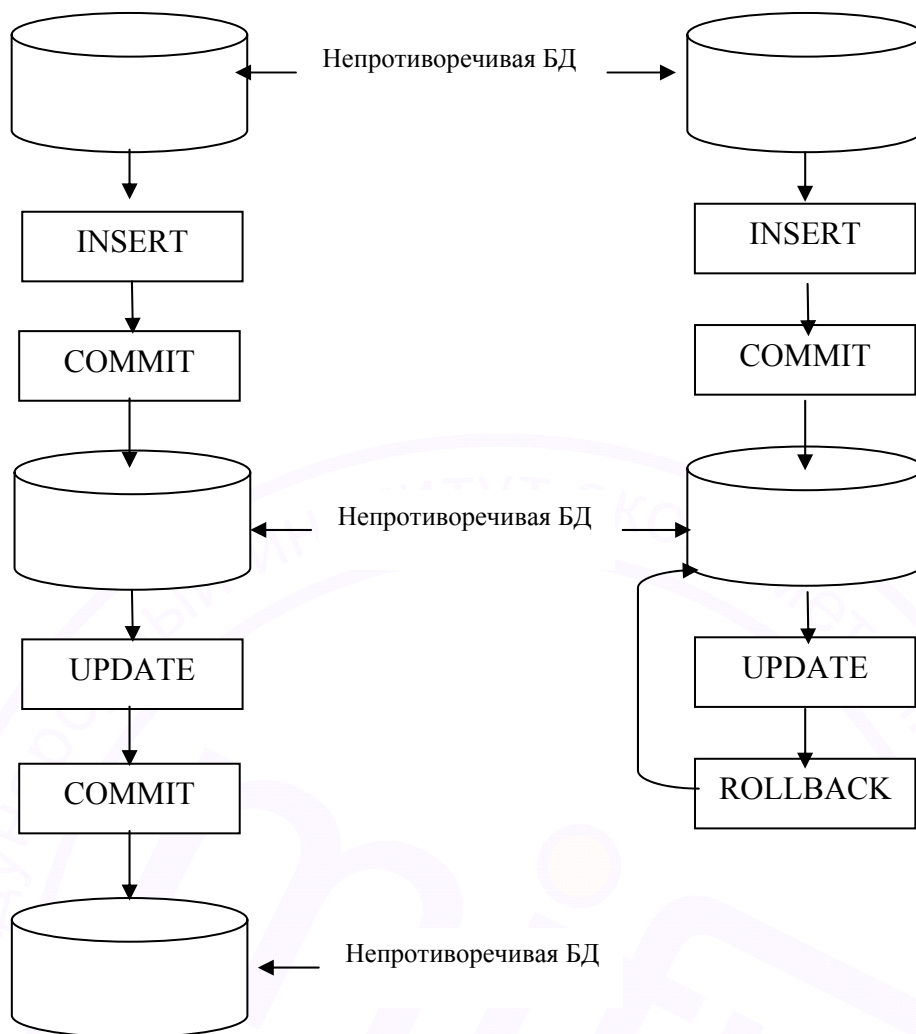


Рис. 9.2. Модель автоматического выполнения транзакций

### Автоматическое выполнение транзакций

В стандарте ANSI/ISO зафиксировано, что транзакция автоматически начинается с выполнения пользователем или программой первой инструкции SQL. Далее происходит последовательное выполнение инструкций до тех пор, пока транзакция не завершается одним из двух способов (рис. 9.2):

- инструкцией COMMIT, которая выполняет завершение транзакции: изменения, внесенные в БД, становятся постоянными, а новая транзакция начинается сразу после инструкции COMMIT;
- инструкцией ROLLBACK, которая отменяет выполнение текущей транзакции и возвращает БД к состоянию начала транзакции, новая транзакция начинается сразу после инструкции ROLLBACK.

Такая модель создана на основе модели, принятой в СУБД DB2.

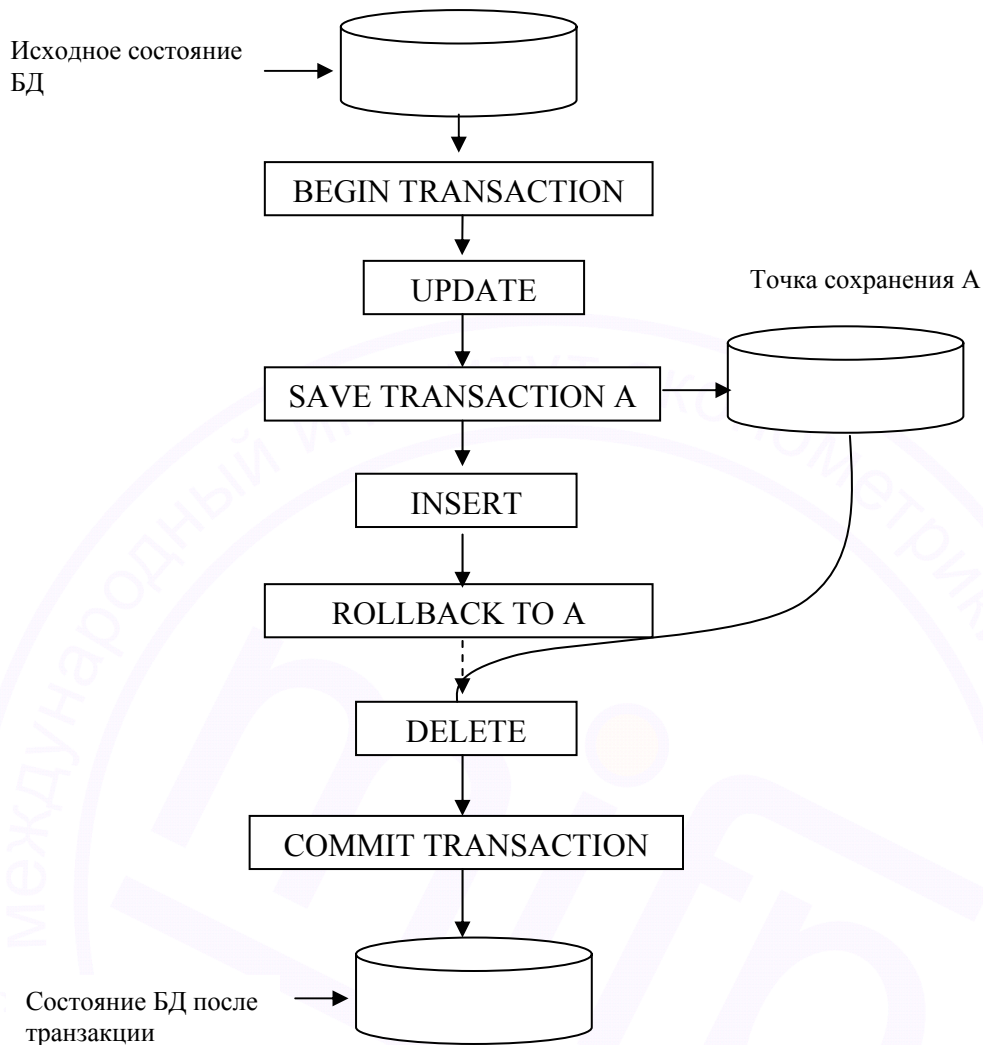


Рис. 9.3. Модель управляемого выполнения транзакций

### Управляемое выполнение транзакций

Отличная от модели ANSI/ISO модель транзакций используется в СУБД Sybase, где применяется диалект Transact-SQL, в котором для обработки транзакций служат четыре инструкции (см. рис. 9.3):

- инструкция **BEGIN TRANSACTION** сообщает о начале транзакции, т.е. начало транзакции задается явно;
- инструкция **COMMIT TRANSACTION** сообщает об успешном выполнении транзакции, но при этом новая транзакция не начинается автоматически;
- инструкция **SAVE TRANSACTION** позволяет создать внутри транзакции *точку сохранения* и присвоить сохраненному состоянию *имя точки сохранения*, указанное в инструкции;
- инструкция **ROLLBACK** отменяет выполнение текущей транзакции и возвращает БД к состоянию, где была выполнена инструкция **SAVE TRANSACTION** (если в инструкции указана точка сохранения –



ROLLBACK TO *имя\_точки\_сохранения*) или к состоянию начала транзакции.

## 9.2. Журнал транзакций

Возможность восстановления состояния базы данных после сбоя обеспечивается с помощью *журнала транзакций*. Журнализация изменений, т.е. сохранение во внешней памяти информации обо всех модификациях БД, тесно связана с управлением транзакциями.

Основным принципом согласованной политики записи изменений в журнал и непосредственно в базу данных является то, что запись об изменении объекта базы данных должна попадать во внешнюю память журнала раньше, чем измененный объект оказывается во внешней памяти базы данных. Соответствующий протокол журнализации (и управления буферизацией) называется Write Ahead Log (WAL) – «пиши сначала в журнал», и состоит в том, что если требуется сохранить во внешней памяти измененный объект базы данных, то перед этим нужно гарантировать сохранение во внешней памяти журнала записи о его изменении.

Другими словами, если во внешней памяти базы данных находится некоторый объект базы данных, по отношению к которому выполнена операция модификации, то во внешней памяти журнала обязательно находится запись, соответствующая этой операции. Каждая успешно завершившаяся транзакция должна быть реально зафиксирована во внешней памяти. Какой бы сбой не произошел, система должна иметь все данные для восстановления состояния базы данных, содержащие результаты всех зафиксированных к моменту сбоя транзакций. Минимальным требованием, гарантирующим возможность восстановления последнего согласованного состояния базы данных, является сохранение во внешней памяти журнала всех записей об изменении базы данных этой транзакцией. При этом последней записью в журнал, производимой от имени данной транзакции, является специальная запись о конце транзакции.

Иногда для восстановления последнего согласованного состояния базы данных после сбоя журнала изменений базы данных явно недостаточно. Основой восстановления в этом случае являются журнал и *архивная копия* базы данных.

Восстановление начинается с обратного копирования базы данных из архивной копии. Затем для всех закончившихся транзакций в прямом смысле повторно выполняются все операции. Более точно происходит следующее: по журналу в прямом направлении выполняются все операции; для транзакций, которые не закончились к моменту сбоя, выполняется откат.

Хотя к ведению журнала предъявляются особые требования по части надежности, реально возможна и его утрата. Тогда единственным способом восстановления базы данных является возврат к архивной ко-

пии. Конечно, в этом случае не удастся получить последнее согласованное состояние базы данных.

Рассмотрим способы производства архивных копий базы данных. Самый простой способ - архивировать базу данных при переполнении журнала. В этом случае образование новых транзакций временно блокируется. Когда все транзакции закончатся, и, следовательно, база данных придет в согласованное состояние, можно выполнять ее архивацию, после чего начинать заполнять журнал заново.

Можно выполнять архивацию базы данных реже, чем переполняется журнал. При переполнении журнала и окончании всех начатых транзакций можно архивировать сам журнал. Поскольку такой архивированный журнал, по сути дела, требуется только для воссоздания архивной копии базы данных, журнальная информация при архивации может быть существенно сжата.

В заключении сформулируем общие требования к системе восстановления данных в составе СУБД.

- Пользователь не должен осуществлять рестарт транзакций или повторный ввод данных. Восстановление должно проходить на базе транзакции с помощью отмены или изменения отдельных транзакций.
- Быстрое восстановление данных обеспечивается генерацией данных, используемых для восстановления.
- При выполнении процедур автоматизированного восстановления пользователь не должен анализировать состав данных и выбирать сами процедуры.

Для восстановления базы данных СУБД имеют в своем составе сервисные программные средства:

- *Программы ведения системного журнала* регистрируют операции над БД: описание соответствующей транзакции, код пользователя, текст входного сообщения, тип изменения БД, адреса изменяемых данных вместе с их значениями до и после изменения.
- *Программы архивации* используются для регулярного получения копий БД для последующего ее восстановления.
- *Программы восстановления* применяются для возврата БД или некоторых ее частей в состояние, предшествующее возникновению отказа. При этом используют архивную копию БД и системный журнал.
- *Программы отката* ликвидируют последствия выполнения определенной транзакции в БД.
- *Программы записи контрольных точек и повторного исполнения* позволяют ускорить восстановление.

### 9.3. Параллельное выполнение транзакций

При параллельной обработке данных (т.е. при совместной работе с БД нескольких пользователей) СУБД должна гарантировать, что пользователи не будут мешать друг другу. Средства обработки транзакций позволяют изолировать пользователей друг от друга таким образом, чтобы у каждого из них было ощущение монопольной работы с БД.

Транзакции являются подходящими единицами изолированности пользователей благодаря свойству сохранения целостности БД. Действительно, если с каждым сеансом работы с базой данных ассоциируется транзакция, то каждый пользователь начинает работу с согласованным состоянием базы данных, т.е. с таким состоянием, в котором база данных могла бы находиться, даже если бы пользователь работал с ней в одиночку.

Чтобы понять, как должны выполняться параллельные транзакции, рассмотрим проблемы, возникающие при параллельной работе с данными.

#### Пропавшие обновления

Рассмотрим пример работы двух диспетчеров с модифицированной БД «Сессия». Пусть Диспетчер 1 вносит в текущий учебный план для каждой дисциплины, читаемой на третьем курсе, сведения о преподавателях, параллельно изменяя при этом значение столбца *Нагрузка* в таблице «Кадровый\_состав», а Диспетчер 2 выполняет такую же операцию для дисциплин второго курса.

Диспетчер 1 начинает работу по изменению таблицы «Учебный\_план» для Дисциплины 1 с количеством часов, равным 50. В столбец *ID\_Преподаватель* для этой дисциплины предполагается внести значение 5. Запрос текущей нагрузки преподавателя возвращает значение 350, и Диспетчер 1 подтверждает изменение таблицы «Учебный\_план». При этом выполняются дополнительные действия по изменению столбца *Нагрузка* в таблице «Кадровый\_состав» для строки с *ID\_Преподаватель* = 5 (в столбец заносится значение 400).

До завершения операции Диспетчер 2 начинает те же действия для Дисциплины 2 с количеством часов 32, которую должен читать тот же преподаватель (*ID\_Преподаватель* = 5). Запрос текущей нагрузки преподавателя также возвращает значение 350, с которым и работает дальше Диспетчер 2. Выполнив те же операции, что и Диспетчер 1 (но после него), Диспетчер 2 помещает в столбец *Нагрузка* значение 382, отменив тем самым предыдущие изменения (рис. 9.4).

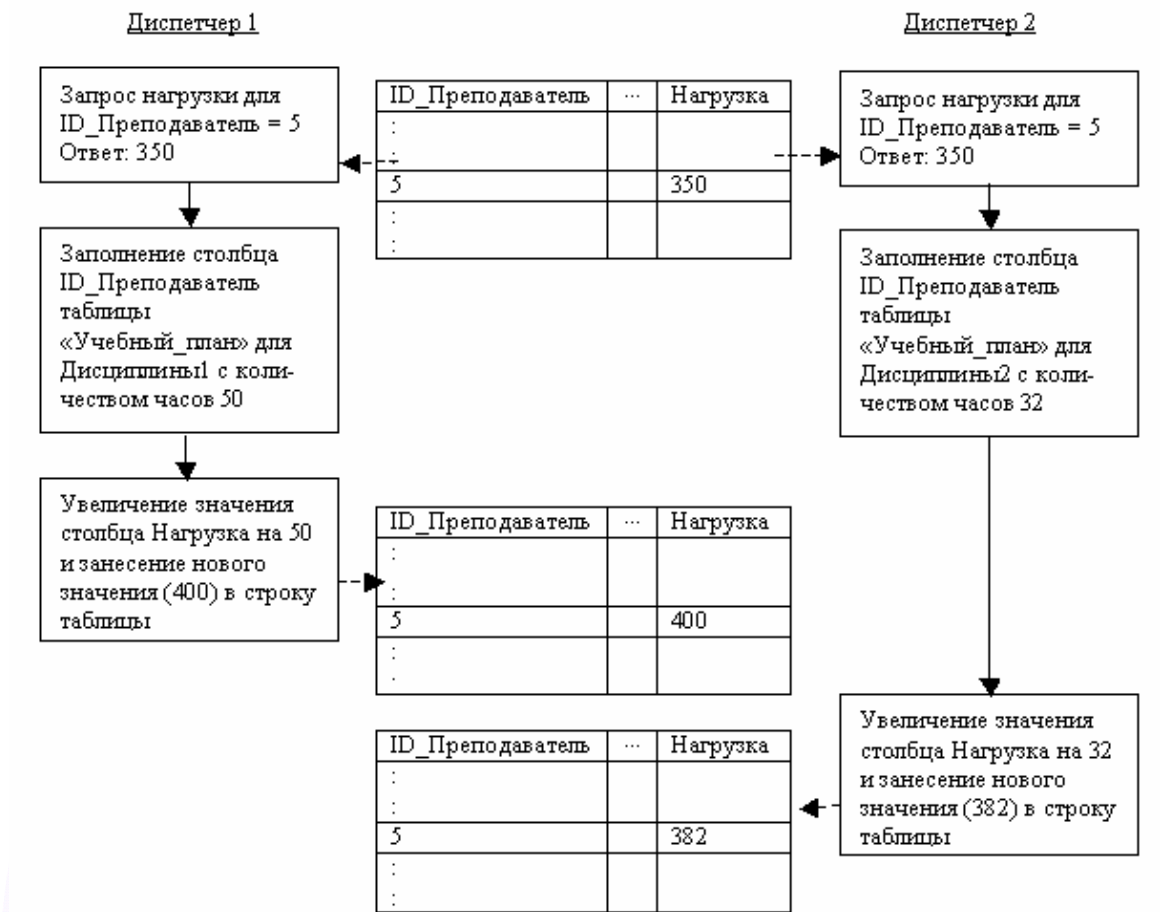


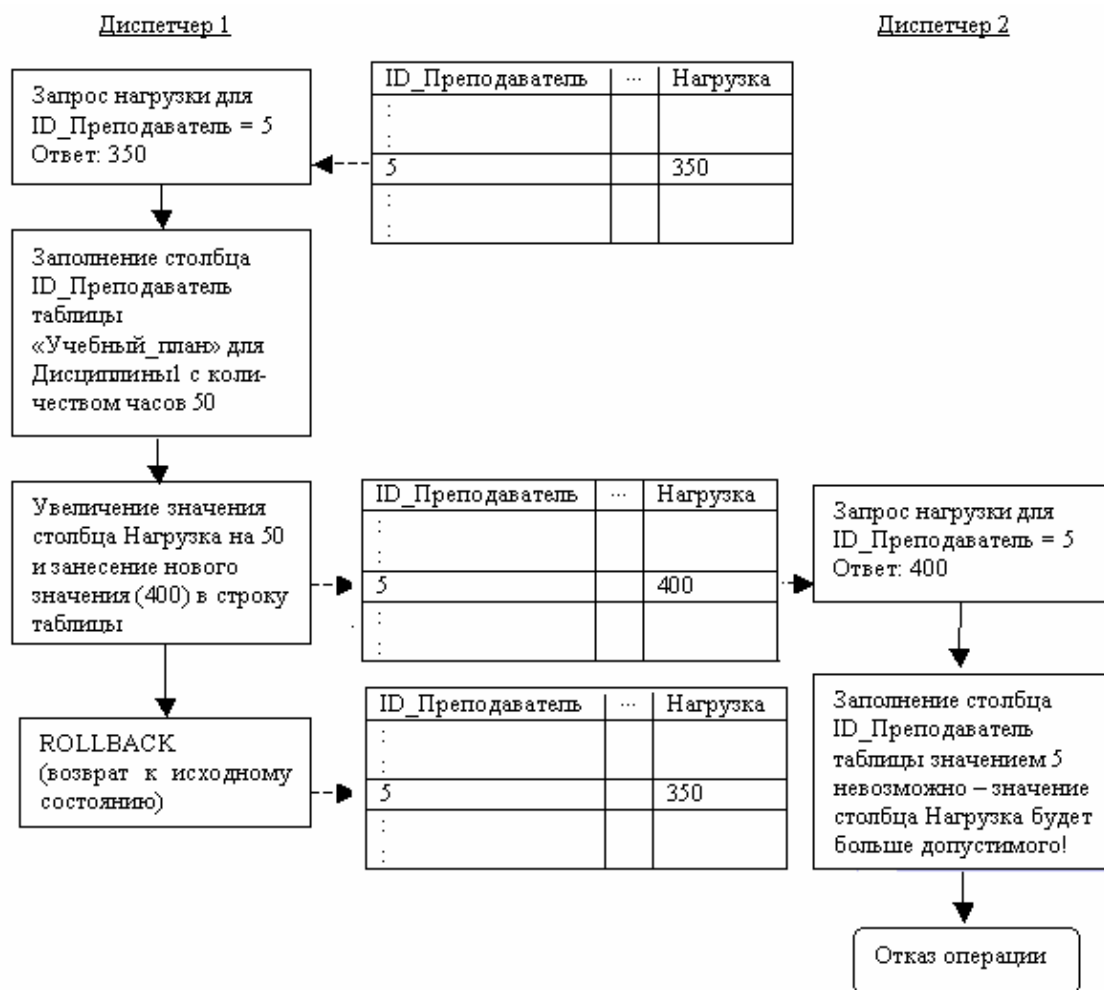
Рис. 9.4. Проблема пропавшего изменения

Для избежания подобных ситуаций к СУБД по части синхронизации параллельно выполняемых транзакций предъявляется минимальное требование - отсутствие потерянных изменений.

### Чтение «грязных» данных

Другой пример коллизий при несогласованной работе двух параллельных транзакций представлен на рис. 9.5.

Как показано на рисунке, Диспетчер 1 и Диспетчер 2 опять выполняют действия, описанные в предыдущем примере, но Диспетчер 2 начинает запрашивать данные о нагрузке преподавателя в тот момент, когда изменения, сделанные Диспетчером 1, уже зафиксированы в столбце *Нагрузка*, а транзакция еще не закончилась. Запрос Диспетчера 2 возвращает значение 400, и Диспетчер 2 вынужден отменить свои действия потому, что 400 часов – это максимальное разрешенное значение нагрузки. Между тем транзакция Диспетчера 1 закончилась возвратом к исходному состоянию, т.е. на самом деле Диспетчер 2 мог бы успешно завершить операцию.



*Рис. 9.5. Проблема чтения «грязных» данных*

Это тоже не соответствует требованию изолированности пользователей (каждый пользователь начинает свою транзакцию при согласованном состоянии базы данных и в праве ожидать увидеть согласованные данные). Чтобы избежать ситуации чтения «грязных» данных, необходимо требовать, чтобы до завершения одной транзакции, изменившей некоторый объект, никакая другая транзакция не могла читать изменяемый объект.

### Чтение несогласованных данных

Рассмотрим ситуацию, которая приводит к получению несогласованных данных при выполнении операций над БД (рис. 9.6).

По-прежнему Диспетчер 1 выполняет операцию по заполнению строки учебного плана, а Диспетчер 2 при выполнении своей операции должен сделать выбор между двумя преподавателями в соответствии с их текущей нагрузкой.

Начиная работу практически одновременно с Диспетчером 1, Диспетчер 2 получает следующие сведения: нагрузка первого преподавателя



( $ID\_Преподаватель = 5$ ) составляет 350 часов, а нагрузка второго преподавателя ( $ID\_Преподаватель = 7$ ) составляет 370 часов. Далее Диспетчер 2 принимает решение в пользу первого преподавателя, но повторный запрос нагрузки возвращает значение 400, т.к. Диспетчер 1 уже сохранил новые данные в таблице «Кадровый\_состав».

В большинстве систем обеспечение изолированности пользователей в подобных ситуациях является максимальным требованием к синхронизации транзакций.

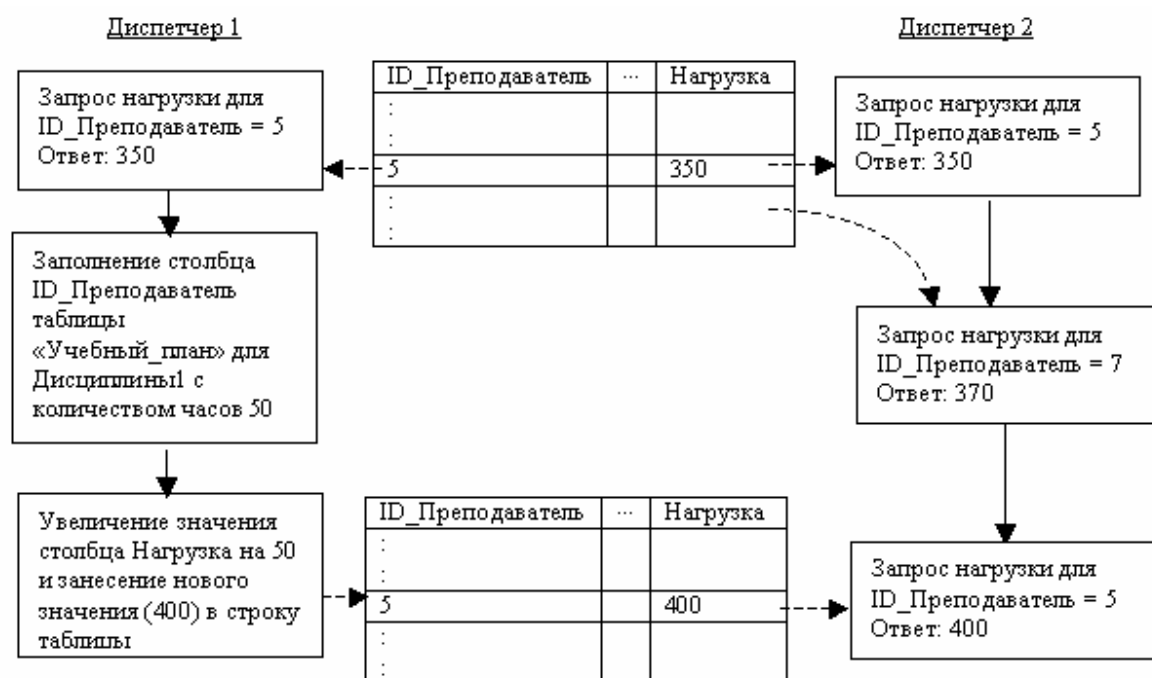


Рис. 9.6. Проблема чтения несогласованных данных

### Строки-призраки

К более тонким проблемам изолированности транзакций относится так называемая проблема строк-призраков, вызывающая ситуации, которые также противоречат изолированности пользователей. Рассмотрим следующий сценарий.

Диспетчер 1 инициирует Транзакцию 1, которая выполняет оператор выборки строк таблицы в соответствии с некоторым условием (например, формирование списка студентов, сдавших дисциплину с  $ID\_Дисциплина = N$ , по таблице «Сводная\_ведомость»). До завершения Транзакции 1 Транзакция 2, вызванная Диспетчером 2, вставляет в таблицу «Сводная\_ведомость» новую строку, удовлетворяющую условию отбора Транзакции 1 (данные о результате сдачи дисциплины N еще одним студентом), и успешно завершается. Транзакция 1 повторно выполняет оператор выборки, и в результате появляется строка, которая отсутствовала при первом выполнении оператора. Конечно, такая ситуация противоречит идее изолированности транзакций.

#### **9.4. Сериализация транзакций**

Чтобы добиться изолированности транзакций, СУБД должна использовать специальные методы регулирования совместного выполнения транзакций.

Метод *сериализации транзакций* - это механизм их выполнения по такому плану, когда результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций. Обеспечение такого механизма является основной функцией управления транзакциями. Система, в которой поддерживается метод сериализации транзакций, реально обеспечивает изолированность пользователей при работе с БД.

Основная реализационная проблема метода состоит в обеспечении такого выполнения транзакций, при котором не слишком ограничивалась бы их параллельность. Простейшим решением является действительно их последовательное выполнение, но существуют ситуации, когда можно выполнять операторы разных транзакций в любом порядке, и это не приведет к проблемным ситуациям. Примерами могут служить транзакции, выполняющие только операции чтения, или транзакции, работающие с разными объектами базы данных.

На самом деле между транзакциями могут существовать следующие виды конфликтов:

- Транзакция 2 пытается изменить объект, измененный не закончившейся Транзакцией 1 (W-W – конфликт);
- Транзакция 2 пытается изменить объект, прочитанный не закончившейся Транзакцией 1 (R-W – конфликт);
- Транзакция 2 пытается читать объект, измененный не закончившейся Транзакцией 1 (W-R – конфликт).

Практические методы сериализации транзакций основываются на учете этих конфликтов.

#### **9.5. Захват и освобождение объекта**

Для обеспечения сериализации транзакций применяются методы «захвата» и «освобождения» объектов, производимого по инициативе транзакции: транзакция «захватывает» объект, что приводит к его блокировке для других транзакций, и освобождает его только при своем завершении. При этом захваты объектов несколькими транзакциями на чтение совместимы (т.е. нескольким транзакциям разрешается читать один и тот же объект), захват объекта одной транзакцией на чтение не совместим с захватом другой транзакцией того же объекта на запись, и захваты одного объекта разными транзакциями на запись не совместимы. Тем самым, выделяются два основных режима захватов:

- совместный режим - S (Shared), означающий разделяемый захват объекта и необходимый для выполнения операции чтения объекта;
- монополюный режим - X (eXclusive), означающий монополюный захват объекта и необходимый для выполнения операций записи, удаления и модификации.

Наиболее распространенным в СУБД, основанных на архитектуре «клиент-сервер», является подход, реализующий соблюдение *двухфазного протокола* захватов объектов БД. В общих чертах протокол состоит в том, что перед выполнением любой операции над объектом базы данных от имени транзакции запрашивается захват объекта в соответствующем режиме (в зависимости от вида операции – совместном или монополюном). В соответствии с этим протоколом выполнение транзакции разбивается на две фазы: *первая фаза* транзакции - накопление захватов; *вторая фаза* (фиксация или откат) - освобождение захватов.

При соблюдении двухфазного протокола основная проблема состоит в том, что следует считать объектом для захвата?

В контексте реляционных баз данных возможны следующие варианты:

- файл - физический (с точки зрения базы данных) объект, область хранения нескольких отношений и, возможно, индексов;
- таблица - логический объект, соответствующий множеству записей данного отношения;
- страница данных - физический объект, хранящий записи одного или нескольких отношений, индексную или служебную информацию;
- запись - элементарный физический объект базы данных.

Очевидно, что чем крупнее объект захвата, тем меньше захватов будет поддерживаться в системе, и на это, соответственно, будут тратиться меньшие накладные расходы. Более того, если выбрать в качестве уровня объектов для захватов файл или отношение, то будет решена даже проблема строк-призраков. Однако, при использовании для захватов крупных объектов возрастает вероятность конфликтов транзакций и тем самым уменьшается допускаемая степень их параллельного выполнения. Фактически, при укрупнении объекта синхронизационного захвата мы умышленно огрубляем ситуацию и видим конфликты в тех ситуациях, когда на самом деле конфликтов нет.

Таким образом, можно резюмировать, что транзакция – это законченный блок обращений к базе данных и некоторых действий над ней, для которого гарантируется выполнение четырех условий, так называемых свойств ACID (Atomicity, Consistency, Isolation, Durability).

– *Атомарность* – операции транзакции образуют неразделимый атомарный блок с определенным началом и концом. Этот блок либо вы-

полняется от начала до конца, либо не выполняется вообще. Если в процессе выполнения транзакции произошел сбой, происходит откат к исходному состоянию.

– *Согласованность* – по завершении транзакции все задействованные объекты находятся в согласованном состоянии.

– *Изолированность* – одновременный доступ транзакций различных приложений к разделяемым объектам координируется таким образом, чтобы эти транзакции не влияли друг на друга.

– *Долговременность* – все изменения данных, осуществленные в процессе выполнения транзакции, не могут быть потеряны.

### *Контрольные вопросы*

1. Дайте определение транзакции
2. Охарактеризуйте модели автоматического и управляемого выполнения транзакций.
3. Назовите виды конфликтов при параллельном выполнении транзакций.
4. Что такое сериализация транзакций?
5. Охарактеризуйте методы «захвата» и «освобождения» объектов.
6. Назовите основные режимы «захвата» объектов.
7. Что такое журнал транзакций?
8. Перечислите основные сервисные программные средства восстановления базы данных в составе СУБД.

## Глава 10. Управление базами данных в СУБД

Для обеспечения эффективного контролируемого управления доступом к данным, целостности и сокращения избыточности хранимых данных большинство СУБД должны быть тесно связаны с операционной системой: многопользовательские приложения, обработка распределенных запросов, защита данных, использование многопроцессорных систем и мультитемных технологий требуют использовать ресурсы, управление которыми обычно является функцией ОС. Соответственно, средства управления доступом и обеспечения защиты также обычно интегрируются с соответствующими средствами операционной системы.

Напомним, что с точки зрения операционной системы база данных – это один или несколько обычных файлов ОС, доступ к данным которых осуществляется не напрямую, а через СУБД. При этом данные обычно располагаются на машине-сервере, а СУБД обеспечивает корректную и эффективную их обработку из приложений, выполняющихся на машинах-клиентах.

С другой стороны, в рамках СУБД, база данных – это логически структурированный набор объектов, связанных не только с хранением и обработкой прикладных данных, но и обеспечивающих целостность БД, управление доступом, представление данных и т.д. Например, в MS SQL Server база данных включает следующие объекты:

- таблицы;
- хранимые процедуры;
- триггеры;
- представления;
- правила;
- пользовательские типы данных;
- индексы;
- пользователи;
- роли;
- публикации;
- диаграммы.

Кроме того, при создании базы данных для нее всегда определяется *журнал транзакций*, который используется для восстановления состояния базы данных в случае сбоев или потери данных. Журнал размещается в одном или нескольких файлах. В журнале регистрируются все транзакции и все изменения, произведенные в их рамках. Транзакция не считается завершенной, пока соответствующая запись не будет внесена в журнал.

Управление системой баз данных производится обычно с помощью нескольких сервисных программ – отдельных приложений, выполняемых в среде операционной системы. Рассмотрим основные функции



и компоненты управления на примере сервера реляционных баз данных MS SQL Server.

Большая часть функций администрирования<sup>48</sup> работы пользователей, серверов и баз данных сосредоточена в приложении SQL Server Enterprise Manager, которое позволяет осуществлять, в том числе, следующие функции:

- запускать и конфигурировать SQL Server;
- управлять доступом пользователей к объектам БД;
- создавать и модифицировать базы данных и их объекты, такие, как таблицы, индексы, представления и т.д.
- управлять выполнением заданий «по расписанию»;
- управлять репликациями;
- создавать резервные копии баз данных и журналов транзакций.

### 10.1. Планирование БД

Использование концепции файлов и файловых групп для физического размещения хранимых данных упрощает управление базами данных и дисковой памятью, а также обеспечивает гибкость при размещении конкретных объектов на устройстве или устройствах. Причем в этом случае обеспечивается реальное распределение данных между всеми входящими в группу файлами (отдельными дисковыми устройствами или RAID-массивами): дисковые устройства действительно одновременно, а не поочередно будут заполняться поступающими данными, поскольку данные будут пропорционально «чередоваться» по файлам группы (рис. 10.1).

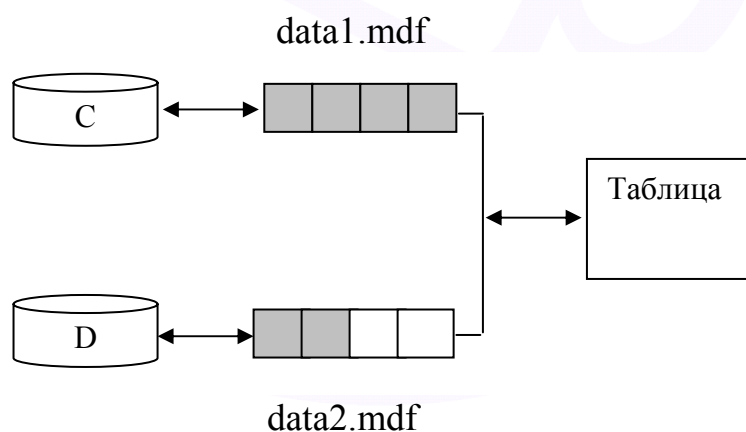


Рис. 10.1. Чередование данных в файловой группе

<sup>48</sup> Большинство функций администрирования, таких как создание, удаление или редактирование объектов БД в SQL Server можно проводить тремя способами: при помощи специализированных диалоговых «мастеров», средствами Enterprise Manager или командами языка T-SQL (надмножество языка SQL-92, включающее дополнительные функции, операторы условия и цикла, хранимые процедуры и т.д.), выполняемым из приложения или, например, из утилиты Query Analyzer.

Для повышения производительности системы в ряде случаев может использоваться индекс - отдельная физическая структура в базе данных, создаваемая на основе таблицы и предназначенная для ускорения выборки данных, поиск которых осуществляется по значению из проиндексированного столбца. Кроме того, индексы используются для обеспечения уникальности строк и столбцов таблиц, упорядочения данных таблицы в отдельном файле или группе файлов для повышения скорости доступа.

Однако наличие индекса замедляет такие операции с таблицей, как вставка, обновление и удаление данных: индексы являются *динамически поддерживаемыми* структурами, т.е. при вставке, удалении или обновлении данных информация в индексах также должна быть изменена для отражения выполненных в таблице изменений. Для такой обработки требуются дополнительные операции ввода-вывода.

**Кластерный индекс** представляет собой двоичное дерево, в котором на нулевом уровне (уровне листьев) содержатся страницы актуальных данных таблицы, а физическое расположение информации в данном индексе логически упорядочено. Такое размещение данных позволяет сократить время доступа к данным, но только при отборе по этому индексу. В других случаях это приводит к задержкам, т.к. доступ данным осуществляется только через индекс и доступ начинается всегда с корня.

Для отдельной таблицы можно построить только один кластерный индекс.

В случае *некластерных индексов* страницы уровня листа содержат не текущие данные таблицы (как в случае кластерного индекса), а указатель на строку данных, включающий номер страницы данных и порядковый номер записи на странице.

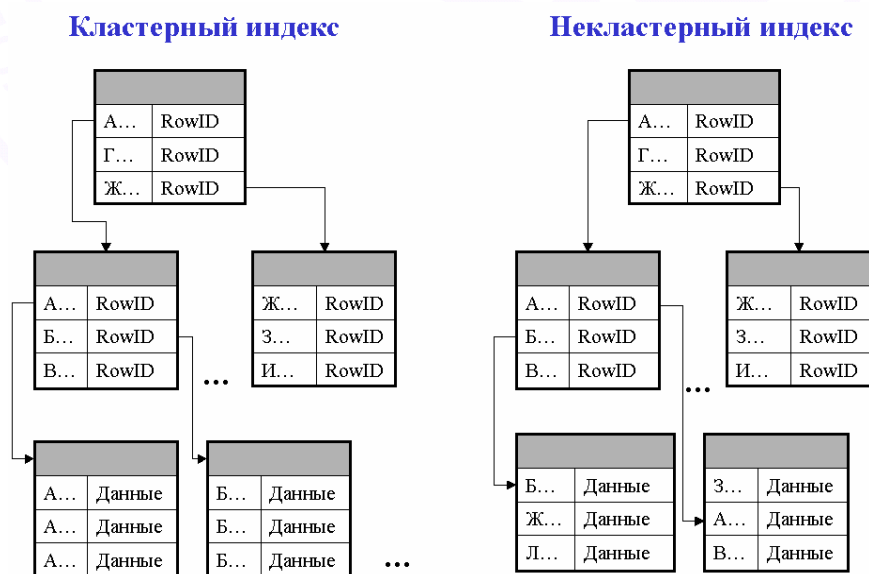


Рис. 10.2. Кластерный и некластерный индексы

Некластерный индекс позволяет быстро получить доступ к данным и не требует физического переупорядочения строк данных таблицы. Для каждой таблицы можно создавать до 249 некластерных индексов (рис. 10.2).

## **10.2. Управление доступом**

Система безопасности SQL Server имеет несколько уровней безопасности:

- Операционная система;
- SQL Server;
- База данных;
- Объект базы данных.

С другой стороны механизм безопасности предполагает существование четырех типов пользователей:

- системный администратор, имеющий неограниченный доступ;
- владелец БД, имеющий полный доступ ко всем объектам БД;
- владелец объектов БД;
- другие пользователи, которые должны получать разрешение на доступ к объектам БД.

Модель безопасности SQL Server включает следующие компоненты:

- Тип подключения к SQL Server;
- Пользователь базы данных;
- Пользователь guest;
- Роли (roles).

### **10.2.1. Тип подключения к SQL Server**

При подключении (и в зависимости от типа подключения) SQL Server поддерживает два режима безопасности:

- Режим аутентификации Windows NT;
- Смешанный режим аутентификации.

**Режим аутентификации Windows NT.** В этом режиме используется система безопасности Windows NT и ее механизм учетных записей. Этот режим позволяет SQL Server использовать имя пользователя и пароль, которые определены в Windows, и тем самым обходить процесс подключения к SQL Server. Таким образом, пользователи, имеющие действующую учетную запись Windows, могут подключиться к SQL Server, не сообщая своего имени и пароля. Когда пользователь обращается к СУБД, последняя получает информацию об имени пользователя и

пароле из атрибутов системы сетевой безопасности пользователей Windows (которые устанавливаются, когда пользователь подключается к Windows).

**Смешанный режим аутентификации.** В смешанном режиме безопасности задействованы обе системы аутентификации: Windows и SQL Server. При использовании системы аутентификации SQL Server отдельный пользователь, подключающийся к SQL Server, должен сообщить имя пользователя и пароль, которые будут сравниваться с хранимыми в системной таблице сервера. При использовании системы аутентификации Windows пользователи могут подключиться к SQL Server, не сообщая имя и пароль.

### *10.2.2. Пользователи базы данных*

Понятие *пользователь базы данных* относится к базе (или базам) данных, к которым может получить доступ отдельный пользователь. После успешного подключения сервер определяет, имеет ли этот пользователь разрешение на работу с базой данных, к которой обращается.

Единственным исключением из этого правила является пользователь guest (гость). Особое имя пользователя guest, разрешает любому подключившемуся к SQL Server пользователю получить доступ к этой базе данных. Пользователю с именем guest назначена роль public.

### **Права доступа**

Для управления правами доступа в SQL Server используются следующие команды:

- GRANT. Позволяет выполнять действия с объектом или, для команды - выполнять ее.
- REVOKE. Аннулирует права доступа для объекта или, для команды - не позволяет выполнить ее.
- DENY. Не разрешает выполнять действия с объектом (в то время, как команда REVOKE просто удаляет эти права доступа).

*Объектные права доступа* позволяют контролировать доступ к объектам в SQL Server, предоставляя и аннулируя права доступа для таблиц, столбцов, представлений и хранимых процедур. Чтобы выполнить по отношению к некоторому объекту некоторое действие, пользователь должен иметь соответствующее право доступа. Например, если пользователь хочет выполнить оператор SELECT \* FROM table, то он должен иметь права выполнения оператора SELECT для таблицы table.

*Командные права доступа* определяют тех пользователей, которые могут выполнять административные действия, например, создавать или копировать базу данных. Ниже приведены командные права доступа:

CREATE DATABASE. Право создания базы данных.  
CREATE DEFAULT. Право создания стандартного значения для столбца таблицы.  
CREATE PROCEDURE. Право создания хранимой процедуры.  
CREATE ROLE. Право создания правила для столбца таблицы.  
CREATE TABLE. Право создания таблицы.  
CREATE VIEW. Право создания представления.  
BACKUP DATABASE. Право создания резервной копии.  
BACKUP TRANSACTION. Право создания резервной копии журнала транзакций.

### *10.2.3. Роли*

Назначение пользователю некоторой роли позволяет ему выполнять все функции, разрешенные этой ролью. Фактически роли логически группируют пользователей, имеющих одинаковые права доступа. В SQL Server есть следующие типы ролей:

- роли уровня сервера;
- роли уровня базы данных.

#### **Роли уровня сервера**

С помощью этих ролей предоставляются различные степени доступа к операциям и задачам сервера. Роли уровня сервера заранее определены и действуют в пределах сервера. Они не зависят от конкретных баз данных, и их нельзя модифицировать.

В SQL Server существуют следующие типы ролей уровня сервера:

Sysadmin	– Дает право выполнить любое действие в SQL Server;
Serveradmin	– Дает право изменить параметры SQL Server и завершить его работу
Setupadmin	– Дает право устанавливать систему репликации и управлять выполнением расширенных хранимых процедур
Securityadmin	– Дает право контролировать параметры учетных записей для подключения к серверу и предоставлять права доступа к базам данных
Processadmin	– Дает право управлять ходом выполнения процессов в SQL Server
Dbcreator	– Дает право создавать и модифицировать базы данных;
Diskadmin	– Дает право управлять файлами баз данных на диске.

#### **Роли уровня базы данных**

Роли уровня базы данных позволяют назначить права для работы с конкретной базой данных отдельному пользователю или группе. Роли



уровня базы данных можно назначать учетным записям пользователей в режиме аутентификации Windows или SQL Server. Роли могут быть и вложенными, так что учетным записям можно назначить иерархическую группу прав доступа.

В SQL Server существует три типа ролей:

- заранее определенные роли;
- определяемые пользователем роли;
- неявные роли.

*Заранее определенными* являются стандартные роли уровня БД. Эти роли имеет каждая база данных SQL Server. Они позволяют легко и просто передавать обязанности.

Заранее определенные роли зависят от конкретной базы данных и не могут быть изменены. Ниже перечислены стандартные роли уровня базы данных.

db_owner	Определяет полный доступ ко всем объектам базы данных, может удалять и воссоздавать объекты, а также присваивать объектные права другим пользователям. Охватывает все функции, перечисленные ниже для других стандартных ролей уровня базы данных.
db_accessadmin	Осуществляет контроль за доступом к базе данных путем добавления или удаления пользователей в режимах аутентификации.
db_datareader	Определяет полный доступ к выборке данных (с помощью оператора SELECT) из любой таблицы базы данных. Запрещает выполнение операторов INSERT, DELETE и UPDATE для любой таблицы БД
db_datawriter	Разрешает выполнять операторы INSERT, DELETE и UPDATE для любой таблицы базы данных. Запрещает выполнение оператора SELECT для любой таблицы базы данных
db_ddladmin	Дает возможность создавать, модифицировать и удалять объекты базы данных
db_securityadmin	Управляет системой безопасности базы данных, а также назначением объектных и командных разрешений и ролей для базы данных
db_backupoperator	Позволяет создавать резервные копии базы данных
db_denydatareader	Отказ в разрешении на выполнение оператора SELECT для всех таблиц базы данных. Позволяет пользователям изменять существующие структуры таблиц, но не позволяет создавать или удалять существующие таблицы

db_denydatawriter	Отказ в разрешении на выполнение операторов модификации данных (INSERT, DELETE и UPDATE) для любых таблиц базы данных
public	Автоматически назначаемая роль сразу после предоставления права доступа пользователя к БД.

**Роли, определяемые пользователем**, позволяют группировать пользователей и назначать каждой группе конкретную функцию безопасности.

Существуют два типа ролей уровня базы данных, определяемых пользователем:

- стандартная роль;
- роль уровня приложения.

*Стандартная роль* предоставляет зависящий от базы данных метод создания определяемых пользователем ролей. Самое распространенное назначение стандартной роли – логически сгруппировать пользователей в соответствии с их правами доступа. Например, в приложениях выделяют несколько типов уровней безопасности, ассоциируемых с тремя категориями пользователей. **Опытный пользователь** может выполнять в базе данных любые операции; **обычный пользователь** может модифицировать некоторые типы данных и обновлять данные; **неквалифицированному пользователю** обычно запрещается модифицировать любые типы данных.

*Роль уровня приложения* позволяет пользователю выполнять права некоторой роли. Когда пользователь принимает роль уровня приложения, он берет на себя выполнение новой роли и временно отказывается от всех других назначенных ему прав доступа к конкретной базе данных. Роль уровня приложения имеет смысл применять в среде, где пользователи делают запросы и модифицируют данные с помощью клиентского приложения.

### **10.3. Управление обработкой. Представления, хранимые процедуры, триггеры**

Для решения типовых (часто повторяющихся) задач выборки или обновления данных, а также в значительной части для управления доступом к данным (как альтернатива механизму разрешения-запрета) и обеспечения целостности данных целесообразно использовать процедуры. Кроме того, другое преимущество, уже в части администрирования, состоит в том, что не надо специально определять пользователю права доступа к таблицам и представлениям, используемым в процедуре: достаточно определить только разрешение на выполнение процедуры.

Существуют два способа взаимодействия приложения с SQL Server. Можно создать приложение, отправляющее клиентские операторы T-SQL на сервер, либо создать хранимые процедуры непосредственно на сервере. В первом случае операторы каждый раз recompилируются сервером. Второй способ активизирует хранимые процедуры, вызывая их из приложения одним оператором. При первом вызове хранимой процедуры она компилируется и создается план ее выполнения, который сохраняется в памяти. При последующих вызовах SQL Server будет использовать этот план и процедуру повторно не компилирует. Таким образом, когда для решения определенных задач требуется многократно выполнить одну и ту же последовательность операторов SQL, применение хранимой процедуры обеспечивает более высокую производительность.

Для управления обработкой в процедурах можно использовать локальные переменные, которые создаются с помощью оператора DECLARE. Переменная доступна с момента ее объявления и до выхода из процедуры. После выхода из процедуры на переменную ссылаться нельзя. Локальные переменные можно объявлять в пакете, в сценарии, внешней программе, а также в хранимой процедуре. В операторе DECLARE необходимо указать имя переменной и ее тип.

### *10.3.1. Представления*

Представления (**View**) существуют независимо от информации в базе данных, но тесно с ней связаны. Представления используются для фильтрации и предварительной обработки данных.

Представление – это по существу некая виртуальная таблица, содержащая результаты выполнения запроса (оператора SELECT) к одной или нескольким таблицам. Для конечного пользователя представление выглядит как обычная таблица в базе данных, над которой можно выполнять операторы SELECT, INSERT, UPDATE и DELETE. В действительности представление хранится в виде предопределенного оператора SQL.

**Типы представлений.** Различные типы представлений имеют свои преимущества и недостатки. Выбор того или иного типа представлений полностью зависит от задач приложения. Выделяют следующие типы представлений:

- **Подмножество полей таблицы** – состоит из одного или более полей таблицы и считается самым простым типом представления. Обычно используется для упрощения представления данных и обеспечения безопасности;

- **Подмножество записей таблицы** – включает определенное количество записей таблицы и также применяется для обеспечения безопасности;

- **Соединение двух и более таблиц** – создается соединением нескольких таблиц и используется для упрощения сложных операций соединения;
- **Агрегирование информации** – создается группированием данных и также применяется для упрощения сложных операций.

Представления также позволяют логически объединять данные. Например, если данные хранятся в нескольких таблицах, их затем посредством представления можно объединить в более крупную виртуальную таблицу.

Еще одно преимущество представлений заключается в том, что они могут иметь более низкий уровень безопасности, чем их исходные таблицы. Запрос для представления выполняется согласно уровню безопасности вызывающего его пользователя. Таким образом, представление можно применять для сокрытия данных от определенной группы пользователей.

Представления, как и индексы, можно создавать различными способами: использовать для этого «мастер» или команду T-SQL, имеющую в общем случае следующий формат.

```
CREATE VIEW имя_представления [столбец[...]]  
AS SELECT-оператор
```

Следует отметить, что использование в операторе SELECT предложения WHERE позволяет локализовать доступ пользователя к данным даже на уровне отдельных строк и столбцов.

### *10.3.2. Хранимые процедуры*

Хранимая процедура (**stored procedure**) – это набор операторов T-SQL, которые SQL SERVER компилирует в единый план выполнения. Этот план сохраняется в кэше процедур при первом выполнении хранимой процедуры, и затем план можно повторно использовать уже без recompilation при каждом вызове. Хранимая процедура аналогична процедурам в языках программирования: она может принимать входные параметры, возвращать данные и коды завершения.

Применение хранимых процедур улучшает производительность, например, при использовании в хранимой процедуре условных операторов (таких как IF и WHILE), поскольку условие будет проверяться непосредственно на сервере и серверу не потребуется возвращать промежуточные результаты проверки условия программам-клиентам.

Хранимые процедуры также позволяют централизованно контролировать выполнение задачи, что гарантирует соблюдение бизнес-правил.



Хранимые процедуры, как и представления, можно создавать различными способами: использовать для этого «мастер» или команду T-SQL, имеющую в общем случае следующий формат

```
CREATE PROCEDURE имя_процедуры [(%параметр1 тип_данных[...])]
AS SQL-операторы
```

Существует два типа хранимых процедур: системные и пользовательские. Первые поддерживаются SQL Server и применяются для управления сервером и отображения информации о базах данных и пользователях. Вторые создаются пользователями для выполнения прикладных задач.

### 10.3.3. Триггеры

Триггер (**trigger**) - это особый тип хранимой процедуры, которая автоматически выполняется при изменении таблицы с помощью операторов UPDATE, INSERT или DELETE. Как и хранимые процедуры, триггеры содержат операторы T-SQL, но в отличие от процедур запускаются не индивидуально, а автоматически при выполнении операций изменения данных. Триггеры наряду с ограничениями обеспечивают целостность данных и соблюдение бизнес-правил, однако их следует использовать разумно. Например, не нужно создавать триггер, проверяющий наличие значения первичного ключа в одной таблице, чтобы определить можно ли вставить значение в соответствующее поле другой таблицы. Однако трудно обойтись без триггеров при выполнении каскадных изменений в дочерних таблицах.

Триггер создается на одной таблице в текущей базе данных, хотя может использовать данные других таблиц и объекты других баз данных. Триггеры нельзя создавать на представлениях, временных и системных таблицах. Таблица, для которой определен триггер, называется *таблицей триггера*.

Существуют три типа триггеров: UPDATE, INSERT и DELETE, каждый из которых инициируется при выполнении одноименной команды. Операции UPDATE, INSERT и DELETE иногда называют событиями изменения данных. Можно создать триггер, который будет срабатывать при возникновении более чем одного события, например, запускаться в ответ на операторы UPDATE или INSERT. Такие комбинированные триггеры называются UPDATE/INSERT. Возможно создание триггеров, срабатывающих при выполнении любого из трех операторов обновления данных (триггер UPDATE/INSERT/DELETE).

Триггеры, как и представления, можно создавать различными способами: использовать для этого «мастер» или команду T-SQL, имеющую следующую в общем случае формат:



**CREATE TRIGGER** *имя\_триггера*  
**ON** *имя\_таблицы*  
**FOR** [INSERT] [,] [UPDATE] [,] [DELETE]  
**AS** *SQL-операторы*

В программе-триггере нельзя использовать операторы создания, реструктуризации, удаления объектов, реконфигурации и восстановления.

Работа триггеров подчиняется следующим правилам:

- Триггеры запускаются только после завершения выполнения вызывающего их оператора. Например, триггер UPDATE не начинает работать, пока не завершится выполнение оператора UPDATE.
- Триггер не начинает работать, если при выполнении оператора происходит нарушение какого-либо ограничения таблицы или возникает другая ошибка.
- Триггер и вызывающий его оператор образуют транзакцию. В результате вызова из триггера оператора ROLLBACK отменяются изменения, выполненные триггером и оператором. При возникновении серьезной ошибки, например, при отключении пользователя, SQL-Server автоматически выполнит откат всей транзакции.
- Триггер запускается один раз для каждого оператора, независимо от количества изменяемых оператором записей.

Триггеры возвращают результаты своей работы в приложение, подобно хранимым процедурам. Как правило, пользователь не ожидает вывода после выполнения операторов UPDATE, INSERT и DELETE, вызывающих срабатывание триггеров. Если триггер возвращает данные, приложение должно содержать код, правильно интерпретирующий результаты модификации таблицы и вывод триггера.

Для каждого триггера SQL Server создает две временные таблицы, на которые можно ссылаться в описании триггера. Эти таблицы хранятся в памяти и локальны по отношению к триггеру, то есть триггер имеет доступ только к своей собственной версии таблиц. Временные таблицы применяются для сравнения состояния таблицы до и после внесения изменений.

В MS SQL Server возможно создание нескольких триггеров на таблице для каждого события изменения данных (UPDATE, INSERT или DELETE) и рекурсивный вызов триггера. Например, если для таблицы уже определен триггер UPDATE, то можно определить еще один триггер UPDATE для той же самой таблицы. В этом случае после выполнения соответствующего оператора сработают оба триггера. Кроме того, допускаются вложенные триггеры, которые срабатывают в результате выполнения других триггеров. Они отличаются от рекурсивных тем, что не запускают сами себя.

#### 10.4. Управление транзакциями

Репликация базы данных заключается в копировании, или *тиражировании*, данных из одной таблицы или базы данных в другую.

Офис с сетью региональных отделений – показательный случай для использования системы с репликацией транзакций. Каждый филиал ведет работу со своими счетами, информация о которых содержится в его базе данных. Главный офис является подписчиком на базы данных всех филиалов, что позволяет собирать в нем информацию по всей организации.

Репликация основана на метафоре «издатель-подписчик»: *издатель (публикующий сервер)* предоставляет данные; *распространитель* содержит тиражируемую базу или служебную информацию для управления репликацией; *подписчик* – получает и обрабатывает реплицированные данные. Данные передаются от публикующего или рассылающего сервера в направлении каждого из подписчиков. Данные не могут пересылаться подписчику непосредственно от другого подписчика. Если один из подписчиков перестает функционировать, это не должно оказывать никакого влияния на издателя или других подписчиков.

В схеме репликации транзакций для доставки данных от публикующего сервера на каждый из серверов-подписчиков используются три следующих компонента.

- **Агент синхронизации** (Snapshot Agent). Создает файлы данных и структуры, используемые для согласования новых подписчиков с текущим состоянием публикации.

- **Агент чтения журнала** (Log Reader Agent). Считывает из журнала транзакций публикующего сервера подлежащие репликации транзакции и помещает их в базу данных рассылки.

- **Агент рассылки** (Distributation Agent). Пересылает подлежащие репликации транзакции из базы данных рассылки всем подписчикам на публикацию.

Каждая публикация (набор реплицируемых данных - *статей*, которыми могут быть таблицы, записи, поля или хранимые процедуры) создается для выделения данных, подлежащих репликации в базе данных подписчиков. Агент синхронизации создает файл схемы, предназначенный для создания в базе данных табличных структур реплицируемых данных. Этот агент также создает файлы, содержащие данные из публикуемых статей, и обновляет содержимое базы данных рассылки для фиксации нового задания на согласование.

В журнале транзакций публикующего сервера все транзакции, подлежащие репликации в адрес одного или более подписчиков, помечаются специальным флажком. Агент чтения журнала считывает из журнала все помеченные этим флажком команды INSERT, UPDATE и

DELETE. Агент следит за появлением подлежащих репликации транзакций для каждой публикации, существующей в базе данных публикующего сервера. Любая обнаруженная транзакция копируется им в базу данных рассылки. Затем агент чтения журналов адресует рассылаемые данные каждому подписчику на публикацию.

После исходного согласования состояний подписчика и публикующего сервера весь объем данных никогда не пересылается в адрес подписчика. Поддержание актуального состояния базы данных подписчика обеспечивается агентом рассылки. Он пересылает подписчику все команды INSERT, UPDATE и DELETE, введенные пользователями на стороне публикующего сервера. Очень важно четко представлять всю последовательность действий, которые выполняются при передаче подписчику сведений об изменениях, проведенных на стороне публикующего сервера.

При создании публикации разработчик может разрешить подписчику выполнять обновление собственной локальной копии данных. В этом случае все выполненные на стороне такого подписчика изменения будут переданы в обратном направлении на публикующий сервер, а последний разошлет их в адрес всех остальных серверов-подписчиков. Отсутствие конфликтов и гарантия внесения изменений на публикующий сервер обеспечиваются благодаря использованию на сервере-подписчике протокола двухступенчатой фиксации изменений. Если публикующий сервер по какой-либо причине не сможет получить сведения о внесенных изменениях, выполненная на стороне подписчика транзакция будет отменена и восстановится исходное состояние базы данных. В такой схеме *непосредственно обновляемых подписчиков* используются триггеры, хранимые процедуры, координатор распределенных транзакций, а также средства обнаружения конфликтов и рекурсии.

Триггеры размещаются на стороне подписчика. Это гарантирует, что любая начатая на сервере-подписчике транзакция будет обязательно зафиксирована на публикующем сервере, прежде чем появится возможность зафиксировать ее на сервере-подписчике. Здесь используется протокол с двухступенчатой фиксацией изменений (Two Phase Commit - 2PC). Если транзакцию не удастся зафиксировать на публикующем сервере, она будет отменена и на сервере-подписчике, поэтому состояние обеих баз данных останется согласованным.

Хранимые процедуры размещаются на стороне публикующего сервера. Это гарантирует, что любые реплицируемые транзакции будут выполнены только в том случае, если это не приведет к конфликту. Если в результате выполнения транзакции возникает конфликт, на серверах обоих узлов для данной транзакции будет выполнен откат.

Координацию выполнения двухступенчатой фиксации изменений между публикующим сервером и сервером-подписчиком осуществляет *Координатор распределенных транзакций* (Microsoft Distributed Transac-

tion Coordinator — MS DTC), который вызывает выполнение удаленных хранимых процедур.

### ***10.5. Резервное копирование и восстановление***

Архивирование и восстановление базы данных с корректировкой целостности основаны на механизме регистрации изменений, использующем журнал транзакций и контрольные точки.

В журнале транзакций регистрируются все транзакции и все изменения базы данных, произведенные в их рамках. Транзакция не считается завершенной, пока соответствующая запись не будет внесена в журнал.

Журнал может размещаться в нескольких файлах, допускающих автоматический рост. Журнал рассматривается не как таблица, а как отдельный файл в базе данных: запись в журнал ведется блоками любого размера, не зависящего от размера страниц сервера. При обновлении журнала или его архивировании происходит усечение журнала.

***Контрольная точка*** – это операция согласования состояния базы данных в физических файлах с текущим состоянием кэша – системного буфера. С целью улучшения производительности сохраняемые в БД данные сначала помещаются в кэш, а потом система перезапишет модифицированные страницы на диск (*отложенная запись*), причем пользователь не может знать, когда эта запись производится.

Контрольная точка выполняется командой CHECKPOINT при завершении работы сервера, а также в соответствии с установленным интервалом контрольных точек и включает выполнение следующих операций:

- запись на диск всех страниц, измененных к началу контрольной точки;
- запись в журнал транзакций списка незавершенных транзакций;
- запись в журнал транзакций всех измененных страниц;
- регистрация завершения контрольной точки в базе данных (а не в журнале транзакций).

***Резервное копирование*** выполняется для каждой базы индивидуально и может производиться несколькими способами.

*Полное резервное копирование* обеспечивает архивирование всех данных базы, размещенных как в группах файлов, так и в отдельных файлах. Этот способ наиболее часто используется для архивирования баз данных не очень большого размера. В противном случае надо использовать выборочное копирование или копирование групп файлов.

*Выборочное (дифференциальное) резервное копирование* обеспечивает архивирование только тех данных базы, которые были изменены с момента последнего архивирования.



*Резервное копирование журнала транзакций* обеспечивает архивирование и усечение журнала.

В случае *резервного копирования файлов и групп файлов* их можно копировать вместе или по отдельности. Полностью восстановить базу данных с помощью резервной копии файлов и группы файлов несколько сложнее, чем с помощью обычной резервной копии. Для восстановления таблиц и индексов, которые охватывают несколько групп файлов, нужно, чтобы эти файлы и группы файлов были скопированы вместе с охватываемыми их объектами.

Для правильного восстановления базы данных на основе файлов или группы файлов, необходимо использовать резервную копию журнала транзакций.

Резервное копирование и восстановление можно выполнить с помощью Enterprise Manager, «мастера» или T-SQL. Для размещения архивных копий должно быть создано логическое устройство (которое может быть и отдельным физическим устройством).

Информация о выполнении резервного копирования сохраняется как запись в системной таблице *backupfile* базы *msdb*, что позволяет определить, когда и на какое устройство сделана копия.

Процесс восстановления базы данных зависит от типа архива. При восстановлении из дифференциального архива или из архива журнала транзакций необходимо предварительно восстановить БД из последнего полного архива.

### ***10.6. Пример администрирования базы данных в среде Microsoft SQL Server***

В среде SQL Server существует три способа создания базы данных:

- при помощи мастера Create Database Wizard
- средствами Enterprise Manager
- командами языка SQL, которые сохраняются в файлах сценарий.

Рассмотрим функции Enterprise Manager создания, резервного копирования и восстановления БД на примере базы данных «Сессия», спроектированной в главе 6.

#### ***Создание БД «Сессия»***

Для создания базы данных после запуска Enterprise Manager необходимо выполнить следующую последовательность действий.

1. *Выбрать сервер, на котором создается БД.* Выбор сервера осуществляется в левой панели открывшегося окна «Console Root». Раскрыв группу «SQL Server», пользователь получает доступ к зарегистри-



рованными серверами и может выбрать имя сервера, на котором будет создаваться база данных.

2. *Создать БД и спланировать местоположение файлов.* Для создания новой БД необходимо раскрыть папку «Databases» и с помощью контекстного меню или общего меню среды инициировать команду <New Database>. После вызова команды откроется диалоговое окно «Database Properties» с тремя вкладками: «General», «Data Files» и «Transaction Log». На первой вкладке необходимо ввести имя базы данных – «Сессия». На двух следующих – определить имена и местоположение файлов базы данных. По умолчанию все файлы располагаются в каталоге \MSSQL\DATA. Для изменения предлагаемого местоположения файлов БД необходимо в столбце Location ввести новый путь (рис. 10.3).

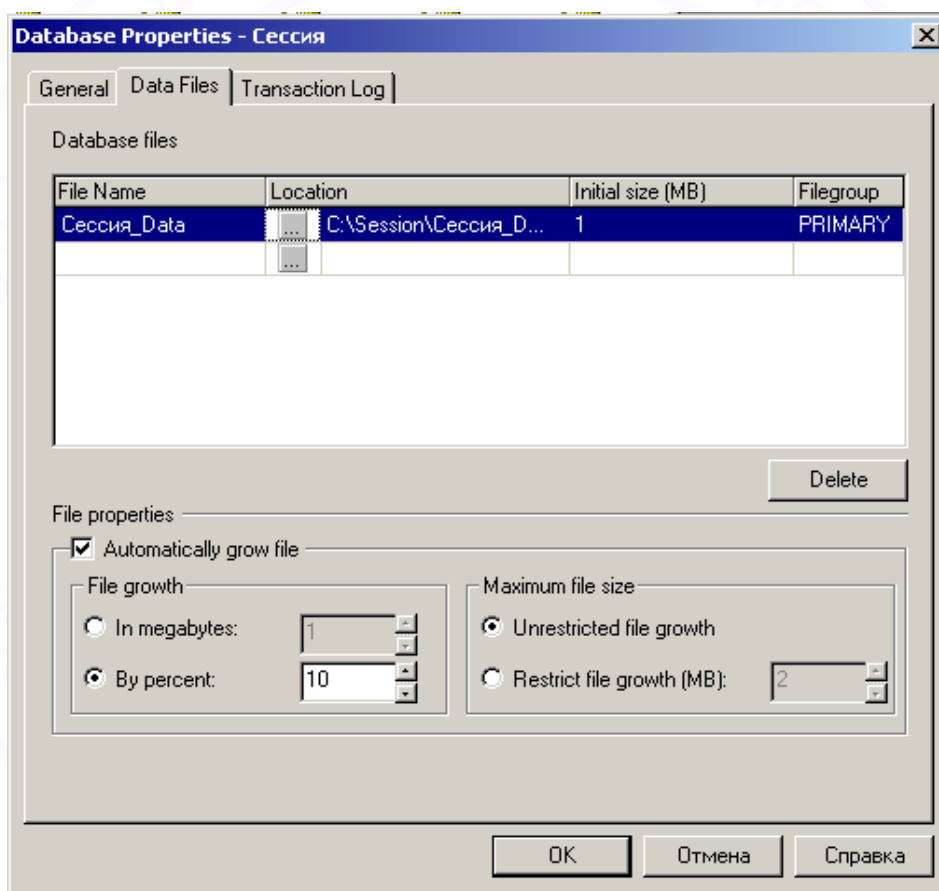


Рис. 10.3. Вкладка Data Files окна Database Properties

Аналогично настраивается файл журнала транзакций на вкладке Transaction Log. После настройки параметров необходимо закрыть диалоговое окно (кнопка <ОК>). При этом имя новой базы данных добавится в список баз данных, отображаемый в окне «Console Root» (Рис. 10.4).

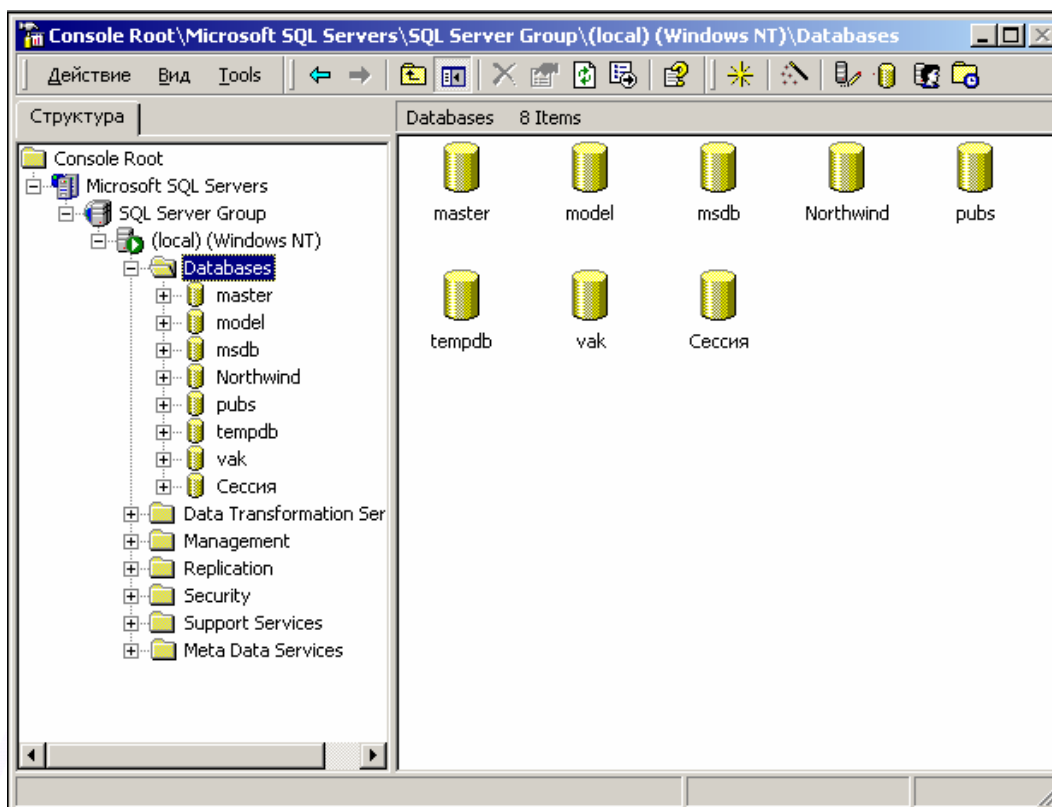


Рис. 10.4. Окно Console Root после создания базы данных «Сессия»

3. *Создать таблицы БД.* Порядок создания таблиц проиллюстрируем на примере таблицы «Студенты» в составе базы данных «Сессия». Раскрыв базу данных «Сессия» в окне Console Root, пользователь получает доступ к объектам БД. Далее, открыв папку «Tables», пользователь имеет возможность инициировать команду контекстного меню или главного меню среды <New Table>. После вызова команды откроется диалоговое окно «New Table», в котором задаются имена полей, их типы и другие параметры. При необходимости вставить новое поле между двумя уже существующими необходимо инициировать команду контекстного меню <Insert Column>. Удалить поле можно с помощью команды <Delete Column>.

Поле *ID\_Студент* в таблице является первичным ключом. Для присвоения полю (или совокупности полей) статуса первичного ключа служит пиктограмма с изображением ключа на панели инструментов (необходимо выделить поле или группу полей и активизировать пиктограмму).

Чтобы построить индекс для поля необходимо в контекстном меню выбрать команду <Indexes/Keys>. Откроется диалоговое окно «Properties» (Рис. 10.5). Окно имеет четыре вкладки, с помощью которых задаются различные свойства таблицы. Так, например, на вкладке «Relationships» определяются связи между текущей таблицей и остальными таблицами базы данных. Вкладка «Indexes/Keys» позволяет получить информацию о существующих индексах и построить новый.

Для создания индекса служит управляющая кнопка <New>. Чтобы построить, например, индекс для поля «Фамилия», необходимо активизировать кнопку и выбрать в колонке «Column name» поле «Фамилия». Строка ввода «Index name» служит для задания имени нового индекса. Остальные интерфейсные элементы служат для определения параметров создаваемого индекса.

После задания всех полей таблицы необходимо закрыть окно «New Table» и в открывшемся диалоговом окне «Choose Name» указать ее имя (Рис. 10.6).

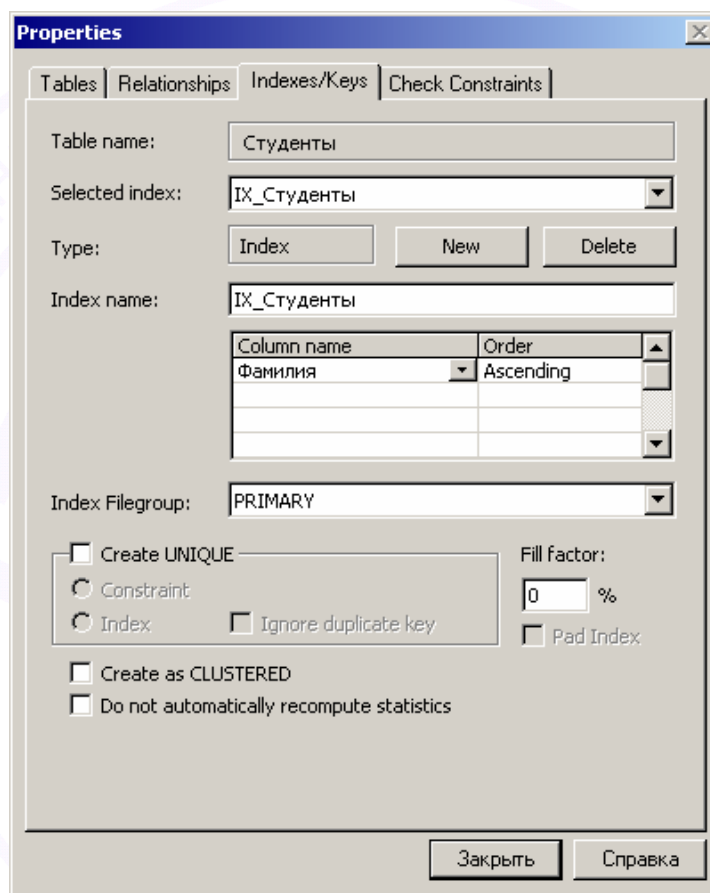


Рис. 10.5. Окно «Свойства поля»

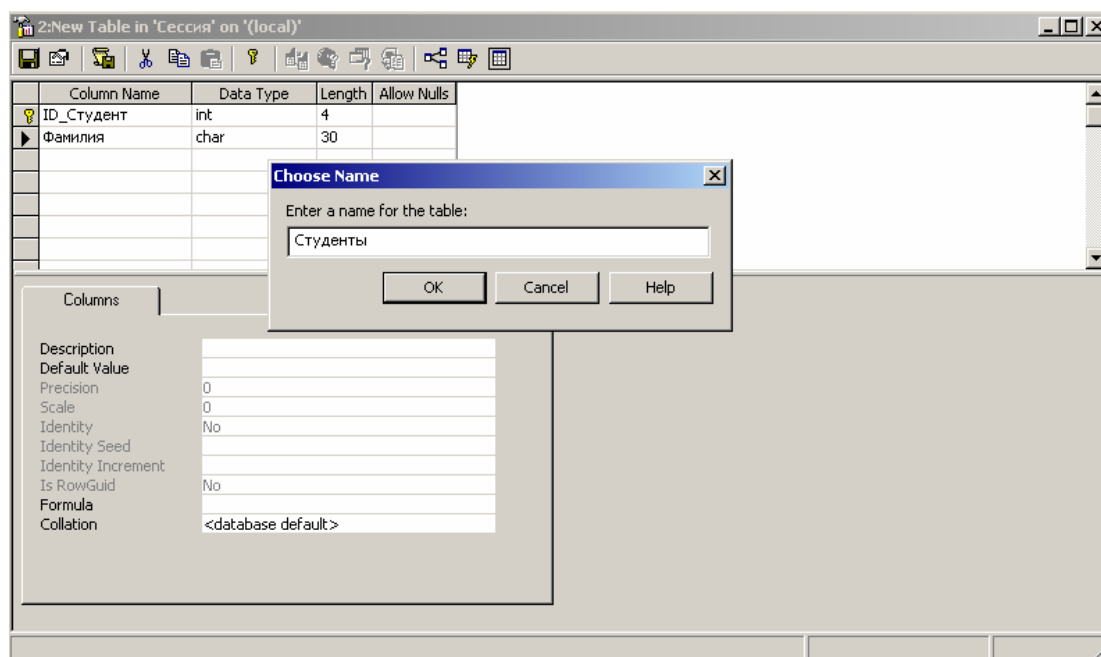


Рис. 10.6. Окно «New Table». Определение полей и имени таблицы

4. *Ввести в таблицы данные.* Чтобы ввести данные в таблицу (например, в созданную на предыдущем шаге таблицу «Студенты»), необходимо:

- в окне Console Root на левой панели раскрыть папку базы данных «Сессия»;
- выбрать объект «Tables» в открывшемся списке, чтобы просмотреть весь список таблиц на правой панели окна
- выделить таблицу «Студенты» на правой панели и активизировать команду контекстного меню *Open table->Return all rows*.

Откроется окно для ввода и отображения содержимого выбранной таблицы. Строка таблицы соответствует отдельной записи, столбец – полю. В этом режиме с помощью клавиатуры в ячейки таблицы можно вводить значения, соответствующие (по типу, длине и другим ограничениям) заданным для полей.

Данные в записях таблицы хранятся в последовательности, задаваемой порядком следования полей.

### Резервное копирование базы данных

Важным этапом ведения баз данных являются операции (backup) и восстановления (restore) БД.

Для того, чтобы выполнить функцию резервного копирования базы данных «Сессия», необходимо:

- в окне «Console Root» на левой панели открыть папку базы данных «Сессия»;

- в строке главного меню инициировать команду *Tools->Backup Database*;
- задать параметры копирования в диалоговом окне SQL Server Backup (Рис. 10.7).

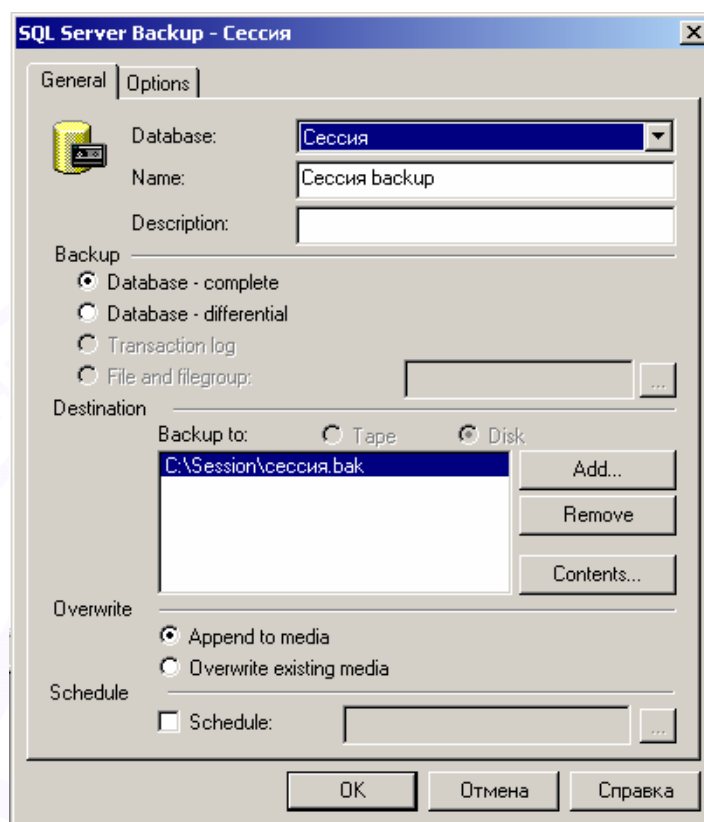


Рис. 10.7. Окно резервного копирования БД

Имя архива (строка ввода «Name») по умолчанию генерируется автоматически на основе имени базы данных и при желании может быть изменено.

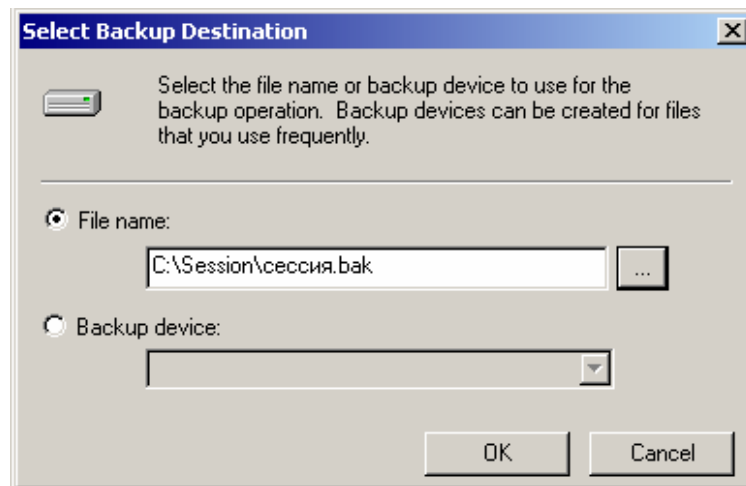
Описание архива (строка ввода «Description») является необязательным параметром.

В области «Backup» расположены переключатели режимов архивирования:

- Database Complete – полное архивирование;
- Database Differential – копируются данные только измененные с момента создания последней резервной копии.

В области «Destination» необходимо указать расположение архивной копии. Кнопка <Add> позволяет открыть окно «Select Backup Destination» (Рис. 10.8). Далее необходимо либо ввести в строку «File Name» идентификатор файла, либо выбрать устройство резервного копирования из списка «Backup Device». Кнопка <OK> возвращает в окно «SQL Server Backup».

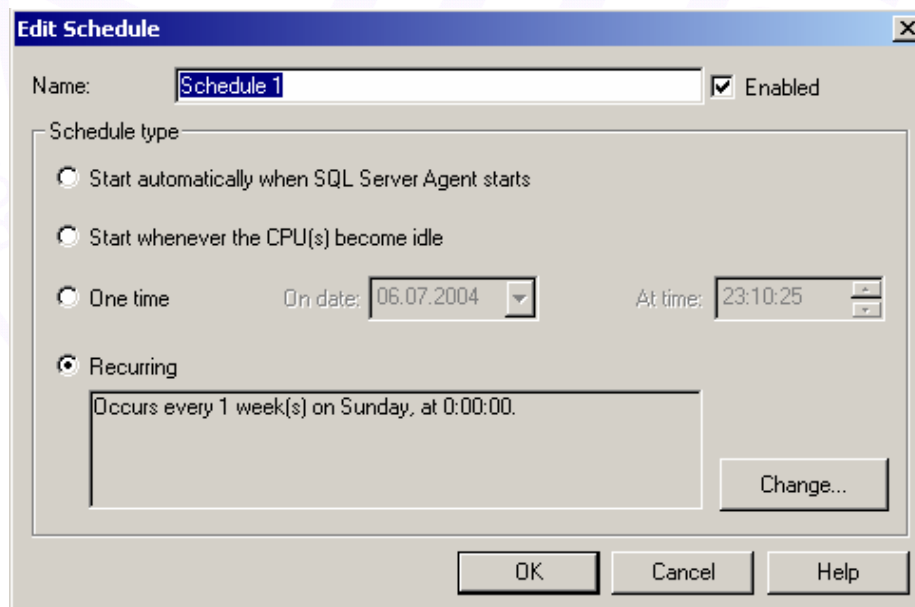




*Рис. 10.8. Диалоговое окно выбора расположения архивной копии*

В области «Overwrite» предлагается записать резервную копию на свободное место (Append to media) или вместо других копий (Overwrite Existing media).

В области «Schedule» предлагается выполнить архивирование немедленно или спланировать его на другое время. Для того, чтобы создать расписание архивирования, необходимо пометить флажок «Schedule» и раскрыть диалоговое окно «Edit Schedule» (Рис. 10.9) с помощью кнопки «...» (Browse).



*Рис. 10.9. Диалоговое окно создания расписания*

Для каждого расписания задается имя (строка ввода «Name»). В области «Schedule Type» можно указать, следует делать архивную копию автоматически при запуске SQL Server Agent или отложить до тех пор, пока не снизится нагрузка на процессор. Здесь же настраивается

частота резервного копирования. Дата и время копирования устанавливается с помощью счетчиков «On Date» и «At Time» соответственно.

Для настройки регулярного архивирования необходимо активизировать кнопку «Change». Откроется окно «Edit Recurring Job Schedule» (Рис. 10.10), с помощью которого создается расписание регулярного архивирования

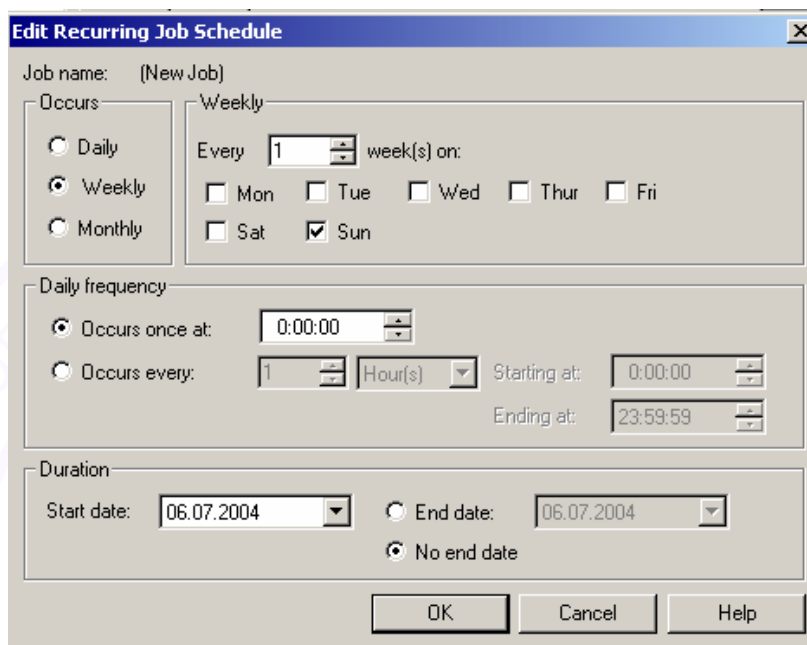


Рис. 10.10. Диалоговое окно создания расписания регулярного архивирования

### Восстановление базы данных

Чтобы восстановить базу данных из резервной копии необходимо:

- в окне Console Root на левой панели открыть папку базы данных «Сессия»;
- в строке главного меню выбрать команду *Tools->Restore Database*. Откроется диалоговое окно с таким же именем (Рис. 10.11);
- в списке «Restore as Database» диалогового окна «Restore Database» выбрать имя восстанавливаемой базы данных (в нашем примере - «Сессия»);
- в секции «Restore» установить переключатель режимов, определяющий тип операции восстановления. Режим «Database» применяется для восстановления всей базы данных, режим «Filegroups Or Files» - для восстановления файлов и групп файлов, режим «From Device» – для восстановления архива с конкретного устройства.

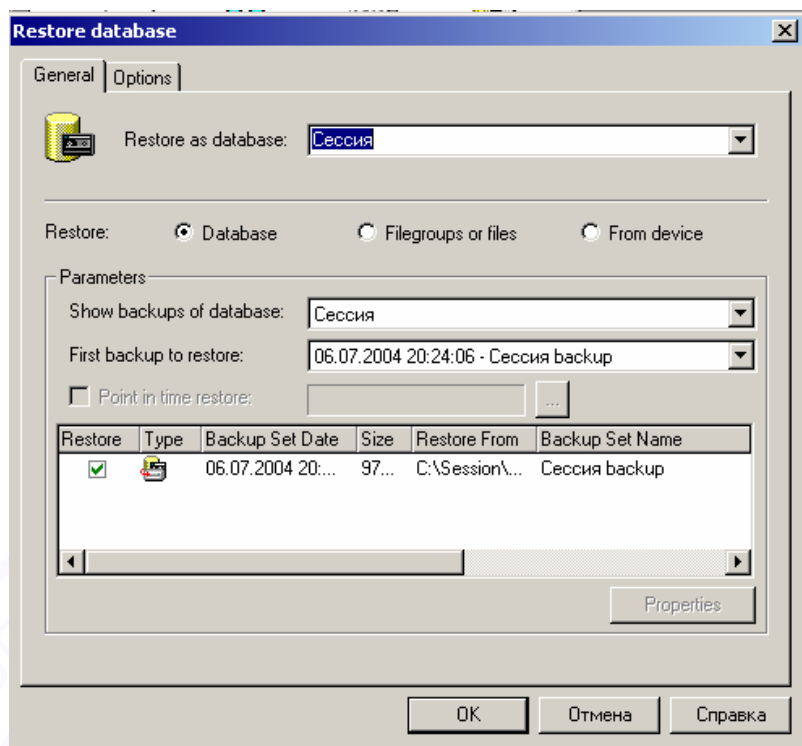


Рис. 10.11. Окно восстановления базы данных

В области «Parameters» можно отобразить архивы указанных баз и просмотреть их свойства, выбрав архив в списке и активизировав кнопку «Properties» (Рис. 10.12).

Чтобы начать восстановление, следует нажать кнопку ОК на окне Restore Database. В итоге появиться сообщение об успешном завершении операции или возникшей ошибке.

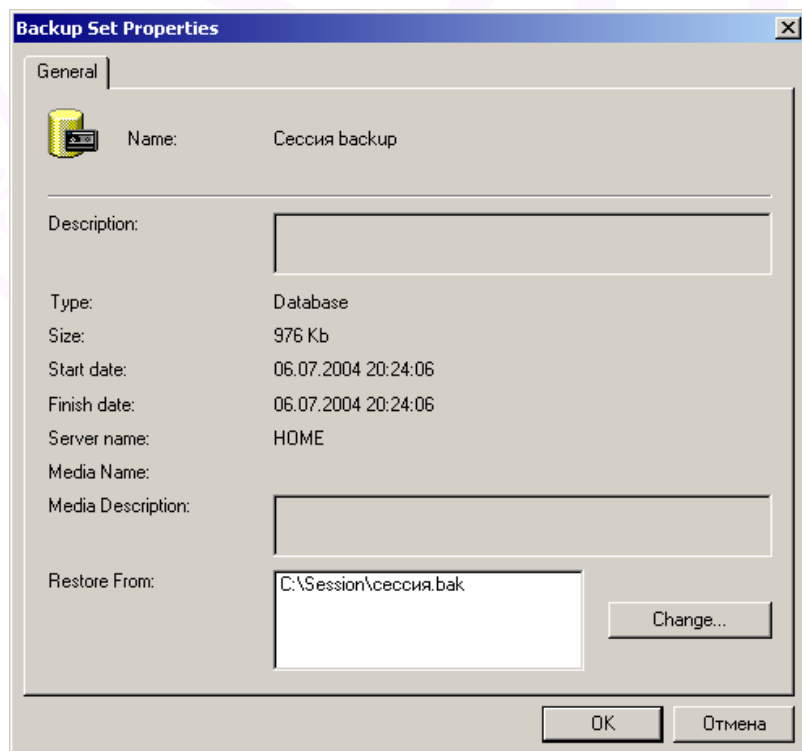


Рис. 10.12. Окно просмотра свойств архива

### *Контрольные вопросы*

1. Определите понятие «база данных» в рамках СУБД
2. В чем состоит сходство и различие кластеризованного и некластеризованного индексов.
3. Какие компоненты включает в себя модель безопасности
4. Когда нужно использовать систему аутентификации Windows NT и SQL Server
5. Дайте сравнительный анализ типов ролей уровня сервера, уровня базы данных, уровня приложений
6. Назначение и типы «ролей»
7. Назначение хранимых процедур и триггеров. В чем состоит сходство и различие хранимых процедур и триггеров.
8. Использование «представлений» для управления доступом
9. Назначение и обобщенная схема репликации баз данных
10. Назначение и использование «контрольных точек» для восстановления БД
11. Назначение и основные способы резервного копирования

## Глава 11. Направления развития концепций и систем обработки данных

### 11.1. Еще раз о проектировании и реализации систем баз данных

Приведем из [7] обзор проблем, связанных с реальностью проектирования и использования систем баз данных.

Прежде всего, необходимо отметить, что многое из классических подходов и методов на практике не используется или используется плохо. В первую очередь, это относится к использованию формализованных методов и моделей, если только они не входят в используемую модель данных непосредственно, а должны применяться проектировщиками для получения и верификации высокого качества проектных решений. Например, полная процедура нормализации высоких степеней и минимизации набора отношений не проводится или проводится редко ввиду ее громоздкости и высоких требований к квалификации проектировщика; оптимизация размещения БД на устройствах внешней памяти и по сети проводится "на глазок". В последнее время значительно меньше внимания уделяется и инструментальным средствам автоматизации физического проектирования БД, включая математическое и натурное моделирование характеристик, в том числе, распределенных БД.

Это объясняется несколькими, отчасти субъективными, причинами:

- громоздкостью методов, используемых в рамках "каскадной" схемы, в условиях практической невозможности обеспечить устойчивость больших интегрированных решений в мире с постоянно меняющимися требованиями к информационным системам;
- высокими требованиями к квалификации проектировщиков в области теоретических основ классического проектирования БД;
- относительной легкостью выполнения реорганизации логической и физической структуры БД в современных СУБД, что зачастую становится одной из ловушек для проектировщика.

Существенны и ограничения, свойственные классическим методам. Классические модели и методы ориентировались на организацию хранения и обработки хорошо структурированных данных, чему отвечало понятие "атрибута" как свойства объекта, представляемого атомарным элементом данных. Именно вследствие этого, как отмечалось ранее, документальные БД выделились в особый тип баз данных.

Спустя годы и десятилетия после объявления своих классических моделей и концепций классики в явном виде указали на их ограниченность. Так, в [25] было признано, что "при обсуждении моделирования семантики данных акцент делается на структурные аспекты в ущерб аспектам обработки. Структура без соответствующих операций или подразумеваемых методов подобна анатомии в отсутствии физиологии". А в



[26] отмечается, что "...обладание большой корпоративной БД имеет маленькое значение, если конечные пользователи не имеют возможностей легко синтезировать необходимую информацию из этих запасов данных". Проектирование БД (и ИС в целом) по классическим правилам полноты и целостности часто стало признаваться практически бессмысленным. "К 1990 году почти все аспекты стандартной процедуры работы с информационными технологиями были оспорены, и вычислительные архитектуры вырвались из-под контроля. ... Стандарты программирования размывались, а понятие неизбыточных, непротиворечивых, высококачественных данных годилось разве что для груды хлама" [16].

Вместе с тем становились значимыми новые причины появления новых требований. Глобальность компьютерных коммуникаций и падение удельной стоимости средств вычислительной техники привели ко многим новым возможностям. Список типов хранимых и обрабатываемых данных расширился до пределов, определяемых самым общим нормативным значением понятия "данное". В базы данных включаются не только неформатированные элементы и полнотекстовые фрагменты: успешно создаются и широко используются мультимедийные, геоинформационные и другие БД. Более того, новые возможности ИТ привели к увеличению рыночных возможностей и требований потребителей, как следствие - к резкому усилению конкуренции в различных отраслях промышленности и услуг. Это в свою очередь стимулировало появление технологии бизнес-реинжиниринга [27] и строительство киберкорпораций [15]. Эти подходы ориентированы, в первую очередь, на осуществление радикальных изменений в организации основной деятельности предприятий и, в частности, на резкое снижение затрат времени, числа работников и других ресурсов; глобализацию работы с клиентами и партнерами в любой точке мира, в том числе в режиме 24\*365; увеличение мобильности персонала - снабжение работника всеми возможностями для самостоятельного получения конечного результата и ориентацией на будущие потребности клиента.

Новые требования к архитектуре корпоративных ИС и, как следствие - новые требования к корпоративным БД должны рассматриваться как интеграция трех составных частей: требований бизнес-реинжиниринга, человеческого фактора и методов новых ИТ. Реальное объединение этих трех составных частей, каждая из которых приобрела в 90-е годы качественно новое наполнение, включает следующие требования:

- обеспечение максимальных возможностей для каждого работника, то есть поддержка выполнения всех бизнес-функций тем самым работником, который и получает конечный результат;
- использование архитектуры и программных средств хранилища данных, средств Оперативной Аналитической Обработки Данных

(OLAP), применение средств быстрой разработки приложений (RAD), средств поддержки принятия решений (DSS) на основе хранилища данных, OLAP и RAD/EIS; применение средств DSS на основе анализа БД прецедентов, а также методов логического вывода, нейронных сетей и др.;

- предложение единого интерфейса пользователя для работы с разными компонентами данных и приложений
- разработка концепции и структуры корпоративной базы данных для новой ИС, реализация структуры БД, предполагающая снятие (существенное уменьшение) ограничений на ее развитие, в том числе, при смене функций или функциональных компонентов обработки информации; применение методов компонентного проектирования БД
- постоянная актуализация понятийной модели деятельности предприятия для учета новых понятий, возникающих при изменении прикладных компонент на функционально сходные и при изменении видов деятельности предприятия, и построение на этой основе новых интерфейсов между компонентами ИС;
- динамическое администрирование фрагментами распределенной корпоративной БД при изменении частоты их использования, при модификации их структуры и при изменении их размещения.

Объединение требований к динамике и разнообразию типов информационных потоков, обрабатываемых в ИС, с учетом роста их объемов и требований к разнообразию методов обработки позволяет дать следующую обобщенную характеристику технологий, определяющих архитектуру БД:

- компонентная технология проектирования и представления предметно-ориентированных баз данных, допускающих работу пользователей через общие интерфейсы;
- расширенная технология Хранилища Данных, интегрирующая наследованные форматированные данные, архивные текстовые документы, звуковые и видеоархивы, и включающая средства оперативной аналитической обработки данных, необходимые виды "дружественных" интерфейсов;
- открытость БД для включения в нее и получения из нее информации с использованием принципов и средств глобальной сети Internet;
- использование архитектуры Открытых Систем, расширенной методами и средствами, обеспечивающими открытость компонентного проектирования БД, переносимость, интероперабельность, масштабируемость и др.

Еще раз напомним, что требование интеграции ресурсов появилось в связи с необходимостью комплексирования ресурсов и, в первую очередь систем БД, основанных на разных моделях данных и управляемых разными СУБД. Основной задачей такой интеграции является пре-

доставление пользователям интегрированной системы некоторой глобальной схемы БД, обеспечение метода доступа к данным, в том числе автоматическое преобразование операторов манипулирования данными в операторы, понятные соответствующим локальным СУБД, организация доступа к данным на сетевом уровне.

Среди направлений исследований и разработок последних лет, в той или иной степени ориентированных на решение этих задач, и получивших не только развитие, но также нашедших практическое воплощение и применение, можно выделить следующие:

- *объектно-ориентированные базы данных*, как развитие парадигмы построения баз данных на основе хорошо формализованных моделей, ориентированные на расширение возможностей использования атрибутивных представлений объектов и процессов предметной области, а также учет поведенческого аспекта;

- ориентированная на интеграцию данных *технология Хранилища данных* - построение федеративных систем неоднородных БД, объединяющая на логическом или физическом уровне наследованные форматированные данные, архивные текстовые документы, звуковые и видео-архивы, и включающая средства оперативной аналитической обработки данных, необходимые виды "дружественных" интерфейсов;

- интеграция с Internet-технологиями – не только включение в состав СУБД средств и интерфейсов поддержки Internet-доступа, но и создание Internet-протоколов и технологий, ориентированных на задачи информационного (документального) поиска.

### ***11.2. Объектно-ориентированные базы данных***

В объектно-ориентированной парадигме модель предметной области – это класс взаимодействующих объектов, каждый из которых представляется набором свойств (статическими характеристиками) и набором методов работы с этим объектом (что и позволяет отразить «поведение» объекта) и обладающих следующими свойствами.

1. Инкапсуляция - объекты наделяются некоторой структурой и обладают определенным набором операций (методов). Внутренняя структура объекта скрыта от пользователя; манипуляция объектом, изменение его состояния возможны лишь посредством его методов. Таким образом объекты можно рассматривать как самостоятельные сущности

2. Наследование - возможность создавать из объектов новые объекты, которые унаследуют структуру и поведение своих предшественников, добавляя к ним черты, отражающие их собственную индивидуальность.

3. Полиморфизм - различные объекты могут получать одинаковые сообщения, но реагировать на них по-разному, в соответствии с тем, как реализованы у них методы, реагирующие на сообщения.

В [1] определены следующие свойства<sup>49</sup>, положенные в основу создания объектно-ориентированных баз данных.

1. *Сложные объекты строятся из более простых при помощи конструкторов.* Простейшими объектами являются такие объекты, как целые числа, символы, и др. Минимальный набор конструкторов, который должна иметь система, - это конструкторы множеств, списков и кортежей.

2. *Идентифицируемость объектов.* В модели с идентифицируемостью объектов объект существует независимо от его значения. Таким образом, имеется два понятия эквивалентности объектов: два объекта могут быть идентичны (они представляют собой один и тот же объект) или они могут быть равны (имеют одно и то же значение). Это влечет два следствия: первое – существование совместно используемых (разделяемых) объектов, а второе - изменения объектов.

3. *Инкапсуляция.* Идея инкапсуляции связана с одной стороны, с потребностью четко различать состояния (время) спецификации и реализации операций, а с другой, для обеспечения модульности, являющейся основой для структурирования сложных приложений, разрабатываемых группой программистов. Интерпретация этого принципа для баз данных состоит в том, что объект инкапсулирует и программу, и данные.

Таким образом, имеется единая модель для данных и операций, причем информация может быть скрыта, т.е. никакие операции, кроме указанных в интерфейсе, не могут выполняться. Инкапсуляция обеспечивает "логическую независимость данных": можно изменить реализацию типа, не меняя каких-либо программ, использующих этот тип.

4. *Типы и классы.* В объектно-ориентированной системе тип обобщает общие черты множества объектов с одинаковыми характеристиками. Понятие класса отличается от понятия типа. Его спецификация совпадает со спецификацией типа, но является более динамическим понятием. Классы используются не для проверки правильности программы, а скорее для создания и манипулирования объектами. В большинстве систем классами можно манипулировать во время выполнения, т. е. изменять их и передавать как параметры.

5. *Иерархии классов или типов.* Наследование обладает двумя положительными достоинствами. Во-первых, оно является мощным средством моделирования, поскольку обеспечивает возможность краткого и точного описания предметной области. Во-вторых, эта возможность помогает факторизовать спецификации и реализации, совместно используемые в приложениях.

6. *Перекрытие, перегрузка и позднее связывание.* Чтобы исключить переписывание программ при введении нового типа и добавлении нового экземпляра существующего типа, система не должна связывать

---

<sup>49</sup> Эти характеристики, используемые как классификационные признаки, однозначно позволяют классифицировать СУБД с точки зрения используемых моделей данных.



имена операций с программами во время компиляции. Поэтому имена операций должны разрешаться во время выполнения. Эта отложенная трансляция называется *поздним связыванием*. Отметим, что хотя позднее связывание затрудняет проверку типов (а в некоторых случаях делает ее невозможной), оно не отменяет ее полностью.

7. *Вычислительная полнота*. С точки зрения языка программирования это свойство является очевидным: оно просто означает, что любую вычислимую функцию можно выразить с помощью языка манипулирования данными системы баз данных. С точки зрения базы данных это является новшеством, так как SQL, например, не является полным языком.

8. *Расширяемость*. Система базы данных поставляется с набором predefined типов, но этот набор должен быть расширяемым: должны иметься средства для определения новых типов и не должно быть различий в использовании системных и определенных пользователем типов. Способы поддержания системой системных и пользовательских типов могут значительно различаться, но эти различия должны быть невидимыми для приложения и прикладного программиста.

9. *Стабильность*. Стабильность (persistence) означает возможность обеспечить сохранность данных после завершения выполнения одного процесса для последующего использования в другом процессе.

10. *Управление вторичной памятью*. Управление вторичной памятью является классической чертой систем управления базами данных. Эта возможность обычно поддерживается с помощью набора механизмов, таких как управление индексами, кластеризация данных, буферизация данных, выбор метода доступа и оптимизация запросов.

11. *Параллелизм*. Система должна обеспечивать пользователям возможность одновременно работать с базой данных. Следовательно, система должна поддерживать стандартное понятие атомарности последовательности операций и управляемого совместного доступа.

12. *Восстановление*. В случае аппаратных или программных сбоев система должна восстанавливаться, т. е. возвращаться к некоторому согласованному состоянию данных.

13. *Средства обеспечения незапланированных запросов*. Система должна позволять обрабатывать запросы, не предусмотренные при проектировании базы данных.

С точки зрения вышеуказанных требований можно отметить следующие ограничения реляционной технологии:

1. Неестественное представление данных со сложной структурой. Реляционная модель данных не допускает «естественного» моделирования данных со сложной структурой, поскольку в ее рамках возможно моделирование лишь с помощью плоских отношений (таблиц). Так как все отношения принадлежат одному уровню, многие значимые связи



между данными либо теряются, либо их поддержку приходится осуществлять в рамках конкретной прикладной программы.

2. Затруднительно должным образом смоделировать свойства данных. Чтобы смоделировать структуру сложных данных, пользователь должен иметь возможность определять свои типы данных, не ограничиваясь типами данных, предоставляемыми определенной СУБД.

3. Реляционная модель данных не позволяет определить набор операций, связанных с данными определенного типа, что часто является естественным требованием при моделировании данных со сложной структурой. Операции приходится задавать в конкретном приложении.

4. Реляционная модель не позволяет рассматривать данные послойно, на различных уровнях абстракции, при необходимости отвлекаясь от ненужных деталей.

5. Усложненный доступ к базе данных. Интерфейс между языком программирования и языком баз данных обычно усложнен, поскольку каждый язык имеет свой набор типов и свою модель вычислений. Организуя обращение к базе данных из прикладной программы, написанной, например, на C++, приходится подвергать данные структурной трансформации при передаче их из/в базу данных.

Модель данных, созданная на этапе концептуального конструирования, не находит непосредственного выражения в структуре базы данных, поскольку модель реализации не предоставляет для этого необходимых средств.

Однако также необходимо привести ряд, в той или иной степени важных, недостатков, отмеченных, например, в [10].

Большинство современных ООБД не делают существенного шага вперед к полным возможностям обработки запросов по сравнению с реляционными БД; они обладают простыми средствами извлечения постоянных объектов.

Так или иначе, объектно-ориентированные БД (ООБД) накладывают некоторые ограничения в части определения постоянных данных. В частности, большинство систем по-разному трактуют постоянные и непостоянные данные; поэтому пользователи должны явно объявлять, постоянный объект или нет.

Второй, гораздо более серьезный, показатель незрелости большей части современных ООБД - недостаток средств, к которым привыкли и появления которых ожидают пользователи баз данных, в том числе полный непроцедурный язык запросов, автоматическая оптимизация и обработка запросов, одновременный доступ, авторизация, динамическое изменение схем, обработка вложенных подзапросов, операции над множествами (объединение, пересечение, разность), функции агрегирования.

Большинство современных ООБД предоставляют возможность только добавлять новые классы к базе данных, но не позволяют добавить к классу новый атрибут или метод.

Ведутся жаркие споры о степени однородности таких систем. Является ли тип и метод объектами? На уровне реализации необходимо решить, будет ли информация о типе храниться как объект, или будет реализована некоторая специализированная система. Это тот же самый вопрос, с которым сталкиваются проектировщики систем реляционных баз данных при принятии решения о том, как следует хранить схему: в виде таблицы или некоторым специальным образом.

Наконец, на уровне интерфейса требуется принять еще одно решение. Типы, объекты и методы могут быть представлены для пользователя как однородные, если даже в семантике языка программирования они предстают как различные по своей природе понятия. Обратно, они могут быть представлены для пользователя как различные сущности, хотя язык программирования не делает между ними различия. Это решение следует принимать с учетом человеческого фактора.

### ***11.3. Интеграция БД и хранилища данных***

Направление появилось в связи с необходимостью комплексирования систем БД, основанных на разных моделях данных и управляемых разными СУБД<sup>50</sup>.

При строгой интеграции неоднородных БД локальные системы утрачивают свою автономность. После включения локальной БД в федеративную систему все дальнейшие действия с ней, включая администрирование, должны вестись на глобальном уровне. Однако, на практике возможны два следующих способа интеграции.

В первом случае – создания мультитебаз, ресурсы объединяются локально (например, в составе конкретной организационной структуры, для решения некоторого класса задач, организации доступа определенного круга пользователей и т.д.), при этом пользователи не теряют автономность и имеют возможность работать со всеми локальными СУБД на одном языке, адресуя запросы одновременно в разные локальные БД. В системах мульти-БД не поддерживается глобальная схема интегрированной БД и применяются специальные способы идентификации объектов и методов для доступа к локальным БД. Чаще всего в таких системах на глобальном уровне допускается только выборка данных, что позволяет достаточно просто сохранить автономность локальных БД. В последнее время для внешнего представления интегрированных и мульти-БД все чаще предлагается использовать объектно-ориентированные модели, но на практике пока чаще встречаются частные решения, ориентированные на конкретные СУБД.

---

<sup>50</sup> В контексте эволюции концепции реляционных баз данных это направление иногда связывают с отказом от нормализации отношений.

Во втором случае, как правило, интегрировать приходится неоднородные БД, распределенные в вычислительной сети. Это в значительной степени усложняет реализацию. Дополнительно к собственным проблемам интеграции приходится решать все проблемы, присущие распределенным СУБД: управление глобальными транзакциями, сетевую оптимизацию запросов и т. д. В этом случае добиться эффективности решений очень трудно.

### *11.3.1. Основы технологии интеграции распределенных данных*

Одна из главных задач, которую призваны решать системы управления - это интеграция данных из различных источников, в том числе со слабоструктурированными данными. Системы интеграции данных должны обрабатывать запросы, для ответа на которые может потребоваться извлечение и обобщение данных из различных источников. При этом возможны следующие варианты:

- Регулярные источники, где представление и организация данных в той или иной степени формализованы, хотя при этом могут использоваться различные модели данных и интерфейсы доступа к ним, или данные источника могут быть не структурированными (HTML-файлы, текстовые файлы и т.д.).
- Источники уникальные, т.е. взаимодействовать с источником можно только через предоставляемый им интерфейс и нет никакой возможности повлиять на его внутренние процессы.

Теоретически и практически возможны два подхода к решению задачи интеграции данных – хранилища данных и виртуальные хранилища (или «витрины данных»).

При использовании первого подхода хранилище физически или логически объединяет данные из различных источников, и затем все запросы обрабатываются с использованием этих данных. В этом случае актуальность данных не гарантируется, поскольку никакой синхронизации с источником не происходит, но преимущество заключается в том, что объединение централизовано и, следовательно, время выполнения запроса не велико.

При использовании второго подхода, данные хранятся в источниках, а запросы к системе интеграции транслируются в запросы или операции понятные источнику. Данные, полученные в ответ на эти запросы к источникам, объединяются и предоставляются пользователю. Преимущество виртуальных хранилищ заключается в гарантии того, что пользователь получает только актуальные данные. Но поскольку источники могут значительно отличаться, возникают трудности связанные с оптимизацией запросов и необходимы дополнительные расходы на конвертирование данных, что существенно снижает общую производительность системы.

Для построения систем объединяющих большое количество источников, содержание которых часто изменяется (например, Web-серверы), наиболее предпочтительным является виртуальный подход.

Рассматривая типичную организацию виртуального хранилища, выделим два уровня логический и физический.

**Логический уровень** определяется выбором модели данных и языка запросов для этой модели. Выбранная модель используется для представления данных, извлекаемых из всех источников. Таким образом, пользователь системы интеграции получает возможность унифицированного доступа ко всем данным. Важным требованием к модели данных является обеспечение прозрачности доступа к внешним источникам, т.е. пользователь видит внешние данные как локальные и в выбранной им модели, не заботясь об управлении доступом к источнику.

**Физический уровень.** Ключевым понятием организации виртуального хранилища являются средства преобразования данных. На рисунке 11.1 приведена типичная архитектура, основанная на распространенной концепции посредников.

Основными компонентами, обеспечивающими возможность интегральной обработки распределенных данных, являются «оболочка» и «посредник».

**Оболочка** используется для хранения информации о внешнем источнике и организации доступа к нему. При получении запроса оболочка обращается к источнику через предоставляемый им интерфейс. Полученные от источника данные конвертируются во внутренний формат данных хранилища (т.е. в модель данных хранилища). Очевидно, что для каждого источника необходима своя оболочка.

**Посредник** осуществляет интеграцию данных из различных источников (используя различные оболочки). Посредник может взаимодействовать как с оболочками, так и с другими посредниками. Таким образом, предоставляется возможность построения сложной сети взаимодействующих между собой посредников, что позволит обобщать данные различными способами для удовлетворения нужд различных приложений. Важно отметить, что посредник не содержит данных - интеграция происходит, как правило, за счет использования технологии представлений.



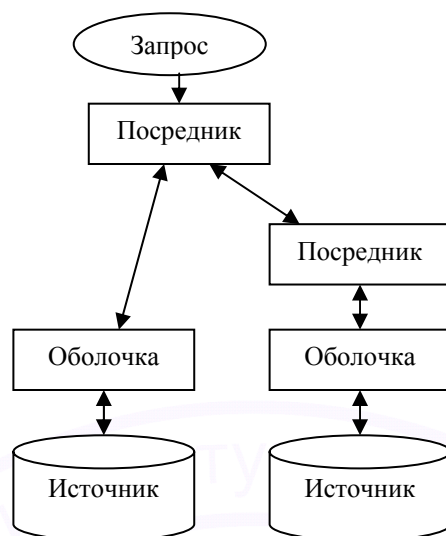


Рис. 11.1. Архитектура виртуального хранилища

Поскольку при использовании предложенной архитектуры задача построения виртуального хранилища сводится к созданию оболочек и посредников, необходимо иметь средства, позволяющие легко их генерировать. С этой целью разрабатываются специальные декларативные языки, с помощью которых описываются оболочки и посредники. По этим описаниям и происходит их генерация.

### 11.3.2. Аналитическая обработка данных

Хранилища данных становятся реальной информационной основой для принятия решений и интеллектуального анализа информации в науке и задачах управления. Особенно широко методы аналитической обработки применяются в бизнесе аналитиками и руководителями организационных структур. Инструментальные средства, основанные на использовании хранилищ данных, позволяют пользователям, не имеющим достаточной математической подготовки, решать достаточно сложные задачи обработки данных.

Системы аналитической обработки типа OLTP (Online Analytical Processing) используют многомерное *представление* (но не хранение!) агрегируемых данных. В этом случае многомерный анализ представляется как одновременный анализ по нескольким «измерениям», каждое из которых – это некоторый способ объединения элементов данных БД. Причем, каждое измерение может иметь несколько уровней обобщения. Соответственно, система предоставляет средства выбора уровня детализации информации и перемещения по каждому измерению.

Это также заставляет на новой основе ставить вопрос о разработке интегрированной совокупности интерфейсов пользователя: они должны создавать естественные условия для работы с информацией и функциями, причем вне зависимости от того, к какому классу хранимых данных разработчик должен был бы отнести сегодня его (пользователя) информацию.



#### ***11.4. Базы данных и Internet***

Задача системы интеграции информации, поддерживаемой средствами Internet, состоит в том, чтобы отвечать на запросы, которые могут потребовать извлечения данных из множества Internet-источников. Многие из проблем, с которыми связаны эти задачи, аналогичны проблемам создания систем неоднородных баз данных, но при этом мы имеем дело с большим и не постоянным множеством Internet-источников, каждый из которых, имеет большую степень автономности и характеризуется разными метаданными.

Так же, как и в ранее рассмотренном случае, интеграции может строиться на подходе, основанном на хранилищах данных или на виртуальном подходе. В первом случае данные из множества Internet-источников загружаются в хранилище, и далее все запросы будут обращены к этому хранилищу данных. При этом необходимо, чтобы данные, изменяемые в источниках, обновлялось и хранилище. Однако преимущество состоит в том, что может быть гарантирована адекватная эффективность на стадии обработки запроса.

При виртуальном подходе, когда данные остаются в Internet-источниках, запросы к системе интеграции на стадии исполнения разделяются на запросы к отдельным источникам, а результаты, соответственно, интегрируются. При таком подходе данные не тиражируются, и тем самым гарантируется их актуальность на стадии обработки запросов. С другой стороны, поскольку Internet-источники автономны, для обеспечения адекватной эффективности необходима более сложная технология обработки запросов. Виртуальный подход более уместен при построении таких систем, где число источников велико, данные изменяются часто, и имеется слабый контроль над Internet-источниками. Нужно, однако, подчеркнуть, что многие проблемы, которые возникают при виртуальном подходе, возникают также и при использовании хранилищ данных (хотя зачастую и в несколько иной форме).

Создание систем для решения любой из указанных выше задач требует, как и в случае классических баз данных, выбора для моделирования предметной области. Однако, кроме модели самих информационных объектов, нам необходимо также моделировать сам Internet (как среду доступа), структуру Web-сайтов, внутреннюю структуру Web-страниц или другого типа ресурса.

Важной особенностью моделирования Internet-ресурса является и то, что во многих случаях данные слабо структурированы: нет какой-либо фиксированной схемы, которая была бы задана заранее, а представления данных поступающих из разных источников могут различаться уже на уровне набора атрибутов или иметь различные типы.

Другая особенность Internet-ресурса – это связи между объектами. Моделирование множества Web-страниц, а также связи между ними ос-

новано на модели помеченных графов. В этой модели узлы представляют Web-страницы (или внутренние компоненты страниц), а дуги - связи между страницами. Метки на дугах могут рассматриваться как имена атрибутов.

Важный аспект языков запросов данных в Web-приложениях - это необходимость генерировать сложные структуры в результате обработки запроса. Например, результат некоторого запроса в системе управления Web-сайтом может представлять собой граф, моделирующий этот Web-сайт.

### ***11.5. Еще раз о проблемах и решениях***

В заключение, считаем целесообразным привести требования и рекомендации [7] к «новым» методам и инструментам проектирования БД практически подтверждающие, что все новое есть в той или иной степени забытое старое.

*Об исключении избыточности в данных.* Требование однократного ввода данных в БД сохраняется как разумное и, прежде всего, для защиты от возникновения противоречий (нарушении логической целостности) при актуализации хранимых данных. Однако в условиях глобального информационного пространства и компонентного проектирования контекст этих требований должен быть пересмотрен. Несомненно, в операционных БД рационально планировать "острова" нормализованных и, в классическом смысле, безизбыточных кластеров отношений или объектов. Эти "острова" чаще всего и будут являться давно известными предметными БД.

Кроме того, заранее нерегламентированный поток информации из Internet в корпоративную БД потребует разработки или увеличения возможностей "процедур отождествления" экземпляров информационных структур, т.е. выяснения того, что эти экземпляры описывают один и тот же предмет реального мира.

*Проблема консервации проблем проектирования.* Сам характер дисциплины проектирования, предусмотренный каскадной схемой, методами структурного проектирования, подталкивает проектировщиков фиксировать достаточно жестко определенные модели предметной области. Технология проектирования БД должна быть изменена таким образом, чтобы исключить консервацию существующих проблем предприятий в жестких, "цельных" структурах БД. Для этого может потребоваться изменение не только технологии, но и инструментов проектирования.

*Компонентная открытость и смысловая интероперабельность.* Замена некоторой функциональной компоненты ИС на подобную, но

спроектированную другим разработчиком, потребует структурной замены некоторой части корпоративной БД. Такая замена должна поддерживаться как постоянный процесс перепроектирования БД. При замене компоненты БД интерфейсы с ней имеющихся приложений и их пользователей должны получать точно ту же в смысловом отношении информацию, что и ранее.

Реальное компонентное проектирование БД может основываться на формировании и использовании общей для комплексизируемых компонент понятийной модели и поддержании соответствий между моделями компонент БД (и связанных с ними приложений) и общей понятийной моделью.

Необходимость использования общих понятийных моделей заставляет заново рассматривать и использование нормативно-справочной информации и систем кодирования. До сих пор часто встречается мнение, что системы классификации и кодирования (СКК) – это средство сокращенного представления информации в БД. На самом деле, отсутствие СКК или использование некорректно построенных СКК приводит к смысловой несовместимости информации, хранимой в различных БД или даже в одной БД. Таким образом, целесообразно использовать работы по проектированию БД с НСИ и проектирование СКК как начало и основу для создания понятийного пространства предметной области.

### *Контрольные вопросы*

1. Охарактеризуйте основные требования к архитектуре корпоративных ИС.
2. Дайте обобщенную характеристику технологий проектирования баз данных.
3. Приведите основные направления исследований в области баз данных.
4. Охарактеризуйте объектно-ориентированный подход к проектированию баз данных.
5. Что такое хранилища данных?
6. Охарактеризуйте особенности баз данных как Internet-ресурсов.

## Глоссарий

**Автоматизированная информационно-поисковая система (АИПС)** - совокупное название как для программных оболочек, ориентированных на ввод, хранение, поиск и выходное представление документов (структур данных сложного или неопределенного формата), так и для конкретных систем определенного наполнения и предметной ориентации, реализованных на основе таких оболочек (или иными программными методами). Примерами программных оболочек АИПС являются STAIRS, DPS, ISIS, IRBIS.

**Агрегат данных** – именованная совокупность элементов данных, представленных простой (векторной) или иерархической (группы или повторяющиеся группы) структурой. Примеры - массивы, записи, комплексные числа и пр.

**Администратор базы данных (АБД)** - лицо или группа лиц, уполномоченных для ведения БД (модификация структуры и содержания БД, активизация доступа пользователей, выполнение других административных функций, которые затрагивают всех пользователей).

**Адрес данных** – идентификация места расположения данных.

**Адресация** – способ размещения и выборки данных в памяти.

**Атрибут** – поле данных содержащее информацию об объекте.

**База данных (БД)** - именованная совокупность взаимосвязанных данных, отображающая состояние объектов и их отношений в некоторой предметной области, используемых несколькими пользователями и хранящимися с минимальной избыточностью.

**Банк данных (БнД)** - система специально организованных данных, программных, языковых, организационных и технических средств, предназначенных для централизованного накопления и коллективного многоцелевого использования данных.

**Внешняя схема** – Представление данных с точки зрения пользователя или прикладной программы

**Внутренняя схема** – физическая структура данных.

**Документ** - агрегат данных в документальных системах (АИПС), имеющий иерархическую структуру и, кроме форматных полей (элементы или агрегаты данных фиксированной длины), обычно содержащий текстовые поля, или символьные последовательности неопределенной длины, логически подразделяющиеся на параграфы (PAR, SEGM), предложения (SENT), слова (WORD).

**Запись логическая** - идентифицируемая (именованная) совокупность элементов или агрегатов данных воспринимаемая прикладной программой как единое целое при обмене информацией с внешней памятью. Запись – это упорядоченная в соответствии с характером взаимосвязей совокупность *полей* (элементов) данных, размещаемых в памяти в соответствии с их *типом*.



**Запись физическая** - совокупность данных, которая может быть считана или записана как единое целое одной командой ввода-вывода.

**Иерархическая модель данных** - использует представление предметной области БД в форме иерархического дерева, узлы которого связаны по вертикали отношением "предок-потомок". Навигация в БД представляет собой перемещение по вертикали и горизонтали в данной структуре. Одной из наиболее популярных иерархических СУБД была Information Management System (IMS) компании IBM, появившаяся в 1968 году. Преимущества IMS и реализованной в ней иерархической модели: (1) *Простота модели*. Принцип построения IMS легок для понимания. Иерархия базы данных напоминает структуру компании или генеалогическое дерево; (2) *Использование отношений предок/потомок*. СУБД IMS позволяла легко представлять отношения предок/потомок, например: "А является частью В" или "А владеет В"; (3) *Быстродействие*. В СУБД IMS отношения предок/потомок были реализованы в виде физических указателей из одной записи на другую, вследствие чего перемещение по базе данных происходило быстро. Поскольку структура данных в этой СУБД отличалась простотой, IMS могла размещать записи предков и потомков на диске рядом друг с другом, что позволяло свести к минимуму количество операций записи-чтения.

**Импорт (загрузка, download)** - утилита (функция, команда) СУБД, служащая для чтения файлов операционной системы, которые содержат данные из базы данных, представленные в некотором коммуникативном формате.

**Инвертированный файл (список)** – файл предназначенный для быстрого произвольного поиска записей по значениям ключей, организованный в виде независимых упорядоченных списков (индексов) ключей – значений определенных полей записей основного файла.

**Индекс** – таблица, используемая для определения адреса записи.

**Информационная база** - данные, отражающие состояние определенной предметной области и используемые информационной системой. Информационная база состоит из двух компонент: 1) коллекций записей собственно данных и 2) описаний этих данных — *метаданных*. Данные отделены от описаний, но в то же время данные не могут использоваться без обращения к соответствующим описаниям.

**Клиент** – программа, написанная как пользователем, так и поставщиком СУБД, внешняя или «встроенная» по отношению к СУБД. Программа-клиент организована в виде приложения, работающего «поверх» СУБД, и обращающегося для выполнения операций с данными к компонентам СУБД через интерфейс внешнего уровня.

**Ключ** – значение (элемент данных) используемый для идентификации или определения адреса записи.

**Ключ первичный (главный)** – ключ, значение которого идентифицирует запись единственным образом.



**Ключ вторичный (альтернативный)** – ключ, который идентифицирует некоторую группу записей, имеющих определенное общее свойство. Набор данных может иметь несколько вторичных ключей, необходимость введения которых определяется практической необходимостью – оптимизацией процессов нахождения записей по соответствующему ключу.

**Ключ сцепленный (составной)** – несколько элементов данных, которые в совокупности обеспечивают уникальность идентификации каждой записи набора данных.

**Кольцевая структура** – Списковая структура данных, в которой последний элемент указывает на первый, образуя тем самым кольцо.

**Контрольная точка** – Операция согласования состояния базы данных в физических файлах с текущим состоянием кэша – системного буфера в операциях обновления БД.

**Концепция баз данных** - информационная технология интегрированного хранения и обработки данных, в основе которой лежит механизм выделения обрабатывающей программе из всех хранимых данных только тех, которые ей необходимы, и в форме, требуемой именно этой программе.

**Коммуникативный формат** - способ представления данных в файле операционной системы, обеспечивающий безошибочное распознавание единиц информации (записей, агрегатов, полей) при импорте (загрузке) информации файла ОС в базу данных (обычно в одну из таблиц БД). Файлы в коммуникативных форматах (КФ) предназначены для архивного хранения и транспортировки содержимого БД. Примерами КФ могут служить основные методы, используемые в системах семейства FoxPro: (1) *SDF (системный формат данных)*, когда каждая строка таблицы БД выводится в виде записи текстового файла, где каждый элемент данных (поле) имеет длину и тип, соответствующие описанию в БД; (2) *Метод с разделителями*, где каждый элемент данных отделяется от соседнего символом-разделителем ("терминатором"). В системах, ориентированных на обработку документов (АИПС) используются более сложные форматы, например представленные в стандарте ISO 2709, использующим рекурсивное описание агрегатов и элементов данных.

**Логическая структура БД** – определение БД на физически независимом уровне.

**Логический файл** – файл в представлении прикладной задачи, состоящий из логических записей, структура которых может отличаться от структуры физических записей, представляющих информацию в памяти.

**Метод доступа** – метод (способ) организации обмена данными обычно между оперативной и внешней памятью, поддерживаемый ОС например прямой, последовательный, индексный метод доступа.

**Модель базы данных** – это комплекс средств, позволяющих определить границу между логическим и физическим аспектами управления базой данных (*независимость данных*); обеспечить конечным пользова-

телям и программистам возможность и средства общего понимания смысла данных (*коммуникабельность*); определить языковые понятия высокого уровня, обеспечивающие возможность выполнения однотипных операций над большими совокупностями записей (в общем случае разнотипных данных) как единую операцию (*обработка множеств*).

**Модель данных** – базовый инструментарий, обеспечивающий на формальном абстрактном уровне конкретные способы представления объектов и связей.

**Модель даталогическая** – описание, создаваемое по инфологической модели данных, и представленное на языке описания данных конкретной СУБД.

**Модель инфологическая (концептуальная)** – описание предметной области, выполненное с использованием естественного языка, математических выражений, таблиц, графов и других средств, понятных всем людям, работающим над проектированием базы данных.

**Модель физическая** – определяющая размещение и способы поиска данных на внешних запоминающих устройствах СУБД.

**Моделирование парадигма** – условности, определяющие способ представления взаимосвязи объектов на уровне *структур данных*. С этой точки зрения различаются реляционные, сетевые, иерархические, объектные, объектно-реляционные, документальные и другие виды моделей.

**Независимость данных логическая (физическая)** – свойство системы, обеспечивающее возможность изменять логическую (физическую) структуры данных без изменения физической (логической).

**Нормализация** – представление сложных структур данных (документов) в виде двумерных таблиц (отношений).

**Отношение (relation)** – агрегат данных, хранящийся в одной из таблиц (строка таблицы) табличной, реляционной БД, или создаваемый виртуально в процессе выполнения операция над базой данных при выполнении запросов к данным.

**Пользователь БД** – программа или человек, обращающийся к базе данных с помощью средств управления данными СУБД.

**Предметная область (ПрО)** – набор объектов, представляющих интерес для актуальных или предполагаемых пользователей, когда реальный мир отображается совокупностью конкретных и абстрактных понятий, между которыми фиксируются определенные связи.

**Представление (View)** – это средство создания виртуальной таблицы, содержащей результаты выполнения запроса (оператора SELECT) к одной или нескольким таблицам. Для конечного пользователя представление выглядит как обычная таблица в базе данных, над которой можно выполнять операторы SELECT, INSERT, UPDATE и DELETE. Представления используются для фильтрования и предварительной обработки данных.

**Проектирование базы данных** - упорядоченный формализованный процесс создания системы взаимосвязанных описаний – таких моделей предметной области, которые связывают (фиксируют) хранимые в базе данные с объектами предметной области, описываемые этими данными.

**Распределенная база данных.** Совокупность баз данных, которые обрабатываются и управляются по отдельности, а также могут разделять информацию.

**Реляционная алгебра** – алгебра (язык), включающая набор операций для манипулирования отношениями.

**Реляционная база данных** – база данных, состоящая из отношений. Здесь вся информация, доступная пользователю, организована в виде таблиц, обычно имеющих уникальные имена, состоящих из строк и столбцов, на пересечении которых содержатся значения данных, а операции над данными сводятся к операциям над этими таблицами.

**Сервер** – программа, реализующая функции СУБД: определение данных, запись-чтение-удаление данных, поддержку схем внешнего-концептуального-внутреннего уровней, диспетчирование и оптимизацию выполнения запросов, защиту данных.

**Сетевая модель данных (модель CODASYL).** Предложенная CODASYL модификация иерархической модели, в которой одна запись могла участвовать в нескольких отношениях предок/потомок. В сетевой модели такие отношения называются *множествами*. В 70-е годы независимые производители программного обеспечения реализовали сетевую модель в таких продуктах, как IDMS компании Cullinet, Total компании Cincom, которые приобрели большую популярность. Сетевые базы данных обладали рядом преимуществ: (1) *Гибкость*. Множественные отношения предок/потомок позволяют сетевой базе данных хранить данные, структура которых сложнее обычной иерархии. (2) *Стандартизация*. Появление стандарта CODASYL; (3) *Быстродействие*. Вопреки своей сложности, сетевые базы данных достигали быстродействия, сравнимого с быстродействием иерархических баз данных. Множества были представлены указателями на физические записи данных, и в некоторых системах администратор мог задать кластеризацию данных на основе множества отношений. Недостатки - жесткость БД, наборы отношений и структуру записей приходилось задавать наперёд. Изменение структуры данных означало перестройку всей базы данных.

**Система управления базами данных (СУБД)** - совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

**Словарь данных** – исчерпывающий набор таблиц или файлов, представляющий собой каталог всех описаний данных (имен, типов). Может содержать также информацию о пользователях, привилегиях и т.д., доступную только администратору базы данных. Является центральным источником информации для СУБД, АБД и всех пользователей.



**Структура данных** - атрибутивная форма представления свойств и связей предметной области, ориентированная на выражение описания данных средствами формальных языков, и таким образом учитывающая возможности и ограничения конкретных средств, с целью сведения описаний к стандартным типам и регулярным связям. Структура данных с точки зрения программирования - это способ отображения значений в памяти – размер области и порядок ее выделения (который и определит характер процедуры адресации-выборки).

**Структура данных линейная** - порядок следования элементов данных которых имеет линейный характер и соответствует порядку расположения элементов в памяти.

**Структура записей данных** – целесообразная (учитывающая особенности физической среды) реализация способов хранения данных и организации доступа к ним как на уровне отдельных записей, так и их элементов.

**Структура информации** - схематичная форма представления сложных композиционных объектов и связей реальной предметной области, выделяемых как актуально необходимые для решения прикладных задач.

**Таблица** - основная единица информации в системе управления реляционной базой данных. Состоит из одной или более единиц информации (строк), каждая из которых содержит значения некоторого вида (столбцы).

**Тип данных** - характер данных, определяющий способ представления значения в памяти и, соответственно, множество стандартных операций (примитивов) преобразования этого значения.

**Топология БД** – схема распределения компонент базы данных по физическим носителям, в том числе, различным узлам вычислительной сети.

**Точка сохранения** - момент времени, когда в БД записывается вся работа в транзакции. В транзакции могут применяться ряд точек сохранения, выступающих в роли промежуточных точек для работы.

**Транзакция** – последовательность операций над данными базы, переводящая БД из одного непротиворечивого состояния в другое, которое может быть представлено как одно «событие».

**Триггер (trigger)** - Особый тип *хранимой процедуры*, которая автоматически выполняется при изменении таблицы с помощью операторов UPDATE, INSERT или DELETE.

**Уровни представления данных** — концептуальный, внутренний и внешний. *Внутренний уровень* - глобальное представление БД, определяет необходимые условия в первую очередь для организации хранения данных на внешних запоминающих устройствах. *Представление на концептуальном уровне* представляет собой обобщенный взгляд на данные с позиций предметной области. *Внешний уровень* – представляет потребности пользователей и прикладных программ.

**Утилита СУБД** - программа, которая запускается в работу командой операционной системы главного компьютера и выполняет какую-то функцию над базой данных (обычно на физическом уровне данных), либо команда (функция ядра СУБД, доступная только АБД), реализующая аналогичную операцию.

**Файл** - именуемая единица информации, поддерживаемая операционной системой. Доступ к данным реализуется либо в рамках ОС, либо пользовательскими программами, либо в рамках СУБД, либо комбинированно. Обычно ОС может предоставить пользовательским программам не более 2-х типов файлов: (1) *запись-ориентированные*, когда при обращении к файлу из пользовательской программы считывается или выводится в файл запись (агрегат или элемент данных - логическая единица информации); и (2) *поток-ориентированные*, когда пользовательской программе предоставляется для записи или чтения физический элемент файла (очередной бит или байт данных).

**Файл ASCII, ( ASCII-File ),** Файл, содержащий символьную информацию, представленную только ASCII-кодами "левой части" (первые 128 символов кодовой таблицы, или код Latin-1) и символьную разметку

**Файл базы данных** - физический файл ОС, используемый для размещения БД. Управление данными в таком файле производится совместно ОС и СУБД. Крайние варианты размещения БД по файлам - (1) все данные БД - в одном файле (файл DATA, СУБД ADABAS); (2) каждая таблица БД - в отдельном файле ОС (DBF-файлы, системы FoxPro). Промежуточный вариант размещения, например - ORACLE - база данных состоит из одного или более табличных пространств, которые в свою очередь состоят из одного или более файлов базы данных.

**Файл бинарный, ( Binary File ),** Файл, содержащий произвольную двоичную информацию (текст с бинарной разметкой, программа, графика, архивный файл)

**Файл графический, ( Image file ),** Бинарный файл, содержащий данные, обычно полученные с помощью растрового сканера и соответствующие двумерному изображению объекта

**Файл текстовый ( Text file ),** Файл, содержащий символьную информацию в одном из соответствующих кодов и коды, управляющие режимом отображения символов на печать и экранные устройства.

**Хранимая процедура (stored procedure)** – это набор операторов SQL, которые сервер компилирует в единый план выполнения. Этот план сохраняется в кэше процедур при первом выполнении хранимой процедуры и затем может быть повторно использован уже без recompilation при каждом вызове.

**Цепочка данных** – организация данных, в которой записи связываются друг с другом посредством указателей.

**Экспорт (выгрузка, upload)** - утилита (функция, команда) СУБД, служащая для вывода информации из БД (обычно одной из таблиц) в файл(ы) операционной системы, организованные в некотором, обычно, коммуникативном формате.



**Элемент данных (элементарное данное)** - неделимое именованное данное, характеризующееся типом (например, символьный, числовой, логический, и пр.), длиной (в байтах) и, обычно, рассчитанное на размещение в одном машинном слове соответствующей разрядности. Это минимальная адресуемая (идентифицируемая) часть памяти - единица данных на которую можно ссылаться при обращении к данным. Ранние языки программирования (Алгол, Фортран) были рассчитаны на обработку элементарных данных или их простейших агрегатов - массивов (матрицы, векторы). С появлением ЯП Кобол появляется возможность представления и обработки агрегатов разнотипных данных (записей). В реляционных БД элементарное данное есть элемент таблицы. Иногда используется термин поле записи в качестве синонима.

**Язык манипулирования данными (ЯМД).** ЯМД обычно включает в себя средства запросов к базе данных и поддержания базы данных (добавление, удаление, обновление данных, создание и уничтожение БД, изменение определений БД, обеспечение запросов к справочнику БД).

**Язык описания данных (ЯОД)** средство внутрисистемного определения данных, представляющего обобщение внешних взглядов. Описание представляет собой модель данных и их отношений, т. е. структур, из которых образуется БД.

**Язык структурированных запросов (SQL).** Основной интерфейс пользователя и АБД для сохранения, обновления и поиска информации в базе данных для ряда СУБД. Включает в себя в качестве подмножеств следующие группы операторов: (1) *Язык описания данных (ЯОД)*. Эти операторы определяют или удаляют объекты базы данных. (2) *Язык управления данными (ЯУД)*. Эти операторы управляют доступом к данным и к базе данных. (3) *Язык манипулирования данными (ЯМД)*. Эти операторы позволяют искать и обновлять данные.

## Примеры организации данных фактографических и документальных БД

### III. Физическая структура данных в dBase

Dbase-подобная база данных физически может состоять из специализированных файлов следующего назначения:

- основного файла базы данных;
- мемо-файла для хранения длинных полей;
- индексного файла.

#### Структура основного файла базы данных (типа .DBF)

Файл базы данных состоит из записи заголовка и записей с данными. В записи заголовка, начинающейся с нулевой позиции, определяется структура базы данных.

Количество полей определяет число подзаписей полей. В базе данных для каждого поля существует одна подзапись поля.

#### Структура заголовка файла данных

Байты	Описание
00	Типы файлов с данными FoxBASE+/dBASE III +, без мемо - 0x03 FoxBASE+/dBASE III +, с мемо - 0x83 FoxPro/dBASE IV, без мемо - 0x03 FoxPro с мемо - 0xF5 dBASE IV с мемо - 0x8B
01-03	Последнее изменение (ГГММДД)
04-07	Число записей в файле
08-09	Положение первой записи с данными
10-11	Длина одной записи с данными (включая признак удаления)
12-27	Зарезервированы
28	1 - есть составной индексный файл (типа .CDX), 0-нет
29-31	Зарезервированы
32-n	Подзаписи полей (для каждого поля одна подзапись)
N+1	Признак завершения записи заголовка (0x01)

#### Структура подзаписи полей

Байты	Описание
00-10	Название поля (максимально - 10 символов)
11	Тип данных: С - символьное; N - числовое; L - логическое; М - типа мемо; D - дата; F - с плавающей точкой; Р - шаблон.
12-15	Расположение поля внутри записи
16	Длина поля (в байтах)
18-31	Зарезервированы

Записи с данными следуют за заголовком и включают в себя фактическое содержимое полей. Длина записи (в байтах) определяется суммированием длин полей, указанных в заголовке.

Записи данных (значений полей) в файле начинаются с позиции, указываемой в записи заголовка в байтах 08-09. Записи начинаются с байта, содержащего признак удаления. Если в этот байт занесен пробел, то запись не удалялась; если же в первом байте – звездочка, то запись удалена. За признаком удаления следуют данные из полей, названия которых находятся в подзаписях полей.

### Структура мемо-файла (тип .FPT)

Файл типа мемо содержит одну запись заголовка файла и произвольное число блоков данных.

В записи заголовка располагается указатель на следующий свободный блок и размер блока в байтах, который устанавливается командой SET BLOCKSIZE (или фиксированная длина 512 байт для файлов типа **.DBT**) при создании файла. Запись заголовка начинается с нулевой позиции файла и занимает 512 байтов.

За записью заголовка следуют блоки, в которых содержатся заголовки блока и текст мемо. В файл базы данных включены номера блоков, которые используются для ссылки на блоки мемо. Расположение блока в мемо-файле определяется умножением номера блока на размер блока (находящийся в записи заголовка). Все мемо-блоки начинаются с четных адресов границ блоков.

### Структура заголовка мемо-файла

<i>Байты</i>	<i>Описание</i>
00-03	Расположение следующего свободного блока
04-05	Не используются
06-07	Размер блока (число байтов в блоке)
08-511	Не используются

### Заголовок блока мемо и текст мемо

00-03	Сигнатура блока (тип данных в блоке): 0 - шаблон (поле типа шаблон) / 1 - текст (поле типа мемо)
04-07	Длина мемо (в байтах)
08-n	Текст мемо (n=длина)

## Структура индексного файла (*тип .IDX*)

В индексных файлах располагается одна запись заголовка и одна или больше записей вершин. В записи заголовка находится информация о корневой вершине, текущем размере файла, длине ключа, особенностях индекса и сигнатура, а также представление ключа в коде ASCII, которое можно вывести на печать, и выражения FOR. Запись заголовка начинается с нулевой позиции файла.

Во всех других записях вершин содержится атрибут, количество существующих ключей и указатели на вершины, располагающиеся слева и справа (на том же уровне) от данной вершины. Помимо этого, в них находится группа символов, представляющая значение ключа, и либо указатель на вершину нижнего уровня, либо подлинный номер записи в базе данных. Размер каждой записи равен 512 байтам.

### Запись заголовка индексного файла

Байты	Описание
00-03	Указатель на корневую вершину
04-07	Указатель на свободную в списке вершину (-1, если таковая отсутствует)
08-11	Указатель на конец файла (размер файла)
12-13	Длина ключа
14	Особенности индекса (любое из нижеследующих числовых значений либо их сумма): 1 - уникальный индекс; 8 - индекс имеет дополнительный оператор FOR.
15	Сигнатура индекса
16-235	Ключевое выражение (не компилируется; до 220 символов) <sup>51</sup>
236-455	Выражение FOR (не компилируется; до 220 символов, оканчивающееся пустым символом)
456-511	Не используются

<sup>51</sup> Тип ключа не запоминается в индексе. Он должен определяться индексным выражением. Если числа используются в качестве ключей, то они подвергаются специальной обработке. Они преобразовываются таким образом, чтобы их можно было отсортировать с помощью такой же схемы упорядочения в коде ASCII, что и символы, т.е., преобразовать число в формат с плавающей точкой IEEE, и изменить на противоположный порядок следования байтов с порядка Intel на порядок слева направо. Если число отрицательное, взять логическое дополнение числа (изменить на противоположные все 64 бита, 1 на 0 и 0 на 1), иначе инвертировать только самый левый бит.

### Запись вершины индекса

Байты	Описание
00-01	Атрибуты вершины (любое из нижеследующих числовых значений либо их сумма): 0 - вершина индекса; 1 - корневая вершина; 2 - лист.
02-03	Количество существующих ключей (0, 1 или больше)
04-07	Указатель на вершину, расположенную непосредственно слева от данной вершины (на том же уровне; -1, если отсутствует)
08-11	Указатель на вершину, расположенную непосредственно справа от данной вершины (на том же уровне; -1, если отсутствует)
12-511	До 500 символов, включающих в себя значение ключа для длины ключа с четырехбайтовым шестнадцатиричным числом (хранящемся в обычном формате слева направо). Если вершина является листом (атрибут = 02 или 03), тогда четыре байта содержат подлинный номер, номер в базе данных в шестнадцатиричном формате - иначе 4 байта содержат внутрииндексный указатель <sup>52</sup> .

### Структура компактного индексного файла (*min .IDX*)

#### Структура записи заголовка компактного индексного файла

Байты	Описание
00-03	Указатель на корневую вершину
04-07	Указатель на свободную в списке вершину (-1, если такая отсутствует)
08-11	Резервируются для внутреннего использования
12-13	Длина ключа
14	Особенности индекса (любое из нижеследующих значений либо их сумма): 1 - уникальный индекс; 8 - индекс имеет дополнительный оператор FOR; 32 - формат компактного индекса; 64 - заголовок составного индекса.
15	Сигнатура индекса
16-35	Зарезервированы для внутреннего использования
36-501	Зарезервированы для внутреннего использования
502-503	По возрастанию или убыванию: 0 - возрастание; 1 - убывание.

<sup>52</sup> В вершине-листе все, что отлично от символьных строк, числа, используемые в качестве значений ключей и четырехбайтовые номера представляются в байтах, порядок которых изменен на противоположный (в формате Intel 8086).



504-505	Зарезервированы для внутреннего использования
506-507	Длина пула выражения FOR
508-509	Зарезервированы для внутреннего использования
510-511	Длина пула выражения FOR
510-1023	Пул выражения ключа (не компилируется)

### Структура записи внутренней вершины для компактного индекса

<i>Байты</i>	<i>Описание</i>
00-01	Атрибуты вершины (любое из нижеследующих числовых значений либо их сумма): 0 - индексная вершина; 1 - корневая вершина; 2 - вершина-лист.
02-03	Число существующих ключей (0, 1 или больше)
04-07	Указатель на вершину, расположенную непосредственно слева от данной вершины (на том же уровне; -1 - если отсутствует)
08-11	Указатель на вершину, расположенную непосредственно справа от данной вершины (на том же уровне; -1 - если отсутствует)
12-511	До 500 символов, включающих в себя значение ключа для длины ключа с четырехбайтовым шестнадцатиричным числом (хранящемся в обычном формате слева направо). Эта вершина всегда содержит ключ индекса, номер записи и внутрииндексный указатель <sup>53</sup> . Комбинация из значения ключа и четырехбайтового числа будет повторена столько раз, количество которых задается в байтах 02-03.

### Структура записи внешней вершины для компактного индекса

<i>Байты</i>	<i>Описание</i>
00-01	Атрибуты вершины (любое из нижеследующих числовых значений либо их сумма): 0 - индексная вершина; 1 - корневая вершина; 2 - вершина-лист.
02-03	Число существующих ключей (0, 1 или больше)
04-07	Указатель на вершину, расположенную непосредственно слева от данной вершины (на том же уровне; -1 - если отсутствует)
08-11	Указатель на вершину, расположенную непосредственно справа от данной вершины (на том же уровне; -1 - если отсутствует)

<sup>53</sup> Каждый элемент состоит из номера записи, запасного байтового счетчика и хвостового байтового счетчика, все в сжатом виде. Текст ключа помещается в логический конец вершины, обрабатывается он в обратном направлении, что позволяет находить элементы предшествующих ключей.

12-13	Свободное для распределения пространство в вершине
14-17	Маска номера записи
18	Маска запасного байтового счетчика
19	Маска хвостового байтового счетчика
20	Количество битов, используемых для номера записи
21	Количество битов, используемых для запасного счетчика
22	Количество битов, используемых для хвостового счетчика
23	Количество байтов, содержащих номер записи, запасной счетчик и хвостовой счетчик
24-511	Ключи индексов и информация

## ***П2. Физическая структура<sup>54</sup> данных в MS SQL Server***

Файлы операционной системы в MS SQL Server представляются как *нумерованные устройства* для хранения БД. Каждое устройство разбивается на виртуальные страницы по 8 Кбайт.

MS SQL Server используется следующая иерархия понятий:

*База данных* — некоторый объем файлового физического пространства для размещения данных, принадлежащих одной логической базе.

*Файлы БД.* Каждая база данных состоит не менее чем из двух файлов. Один из них отводится под журнал транзакций. Отдельный файл данных может принадлежать только одной базе данных.

*Экстенст.* Пространство для хранения объектов выделяется блоками (*экстенстами*) по 8 следующих друг за другом страниц размером 8К. Экстенст является единицей выделения пространства. Поэтому при создании БД нужно указывать размер файла с точностью до 64Кбайт.

*Страница.* Файлы делятся на страницы размером по 8 Кбайт каждая. Логический номер страницы складывается из внутреннего номера базы данных, номера файла и номера страницы в файле. В рамках БД файлы нумеруются, начиная с 1, и так же нумеруются страницы в рамках файла.

Используется два типа экстенстов: *однородные* и *смешанные*. Однородные экстенсты всегда принадлежат только одному объекту. Смешанный экстенст может использоваться несколькими объектами.

В SQL Server существуют несколько типов страниц.

Следующие типы страниц относятся к хранению и поиску информации:

- страниц данных;
- индексных страниц;
- текстовых страниц;
- страницы журнала транзакций;

<sup>54</sup> Здесь приводится схематичное описание структур данных, характерных для MS SQL Server 7.0

Кроме этого используются также страницы размещения:

- карты распределения блоков (основная и вторичная);
- карты свободного пространства;
- индексные карты размещения.

На странице всегда (в отличие от экстенента) хранится однородная информация. Все страницы имеют *заголовок*, в котором хранится общая информация, используемая ядром СУБД для работы со страницами:

- номер страницы в формате *<номер файла, номер страницы>*;
- идентификатор объекта, которому принадлежит страница;
- индекс и уровень внутри индексного дерева, которому принадлежит страница;
- количество строк на странице;
- общий объем свободного пространства на странице;
- указатель на свободное пространство после последней строки на странице;
- минимальная длина строки на странице;
- объем зарезервированного пространства.

После заголовка следует информация о статусе страницы в картах распределения блоков и карте свободного пространства.

### *Страницы размещения*

SQL Server использует три типа страниц размещения: карты распределения экстенентов, карты свободного пространства, индексные карты размещения.

### **Карты распределения экстенентов**

Карта распределения экстенентов состоит из стандартного заголовка и битового массива в 64 000 битов. Каждый бит характеризует один экстенент. Поэтому одна страница карты распределения описывает пространство в 64 000 экстенентов или 4 Гбайт данных.

При отведении пространства используются два типа карт распределения экстенентов:

- глобальная карта распределения (Global allocation map, GAM) хранит информацию об использовании экстенентов. Если бит установлен в 0, то экстенент занят данными, если в 1 — то экстенент свободен;
- вторичная глобальная карта распределения (Secondary global allocation map, SGAM) хранит информацию о типе экстенентов. Если бит установлен в 1, то соответствующий экстенент смешанный и минимум одна страница в нем свободна, в остальных случаях бит равен 0.

## Карты свободного пространства

*Карта свободного пространства* (Page free space page, PFS) отражает степень заполнения страниц. Каждая PFS-страница хранит информацию о 8000 страницах - по одному байту на страницу. Каждый байт представляет собой битовую карту, которая сообщает о степени занятости страницы и о том, принадлежит ли она объекту.

Карта распределения размещается с первой страницы файла БД. Страницы повторяются через каждые 8000 страниц.

Первая страница PFS после стандартного *заголовка страницы* содержит *заголовок файла* (его описание), затем размещается сам блок PFS. Вторая страница — это GAM, третья — SGAM. Карты распределения экстенгов повторяются через каждые 512 000 страниц.

## Карты размещения

Для организации связи между экстентами и расположенными на них объектами используются *индексные карты размещения* (Index Allocation Map, IAM). Каждая таблица или индекс имеют одну или более страниц IAM. В каждом файле, в котором размещаются таблица или индекс, существует минимум одна карта размещения для этой таблицы или индекса. Страницы IAM размещаются произвольно внутри файла и отводятся по мере необходимости. IAM объединены друг с другом в цепочку двунаправленными ссылками. Указатель на первую карту размещения содержится в поле FirstIAM системной таблицы Sysindex.

Каждая IAM описывает некоторый диапазон экстенгов и представляет собой битовую карту: если бит установлен в 1, то в данном экстенте есть страницы, принадлежащие данному объекту, если в 0 — то нет.

Все страницы размещения не связаны напрямую с некоторым объектом БД, они соответствуют некоторой системной информации, поэтому параметр «идентификатор объекта» для всех этих страниц одинаков и равен 99.

## Страницы данных

Страницы данных используются для хранения собственно данных. Структурно страницу данных можно подразделить на три зоны: заголовок, строки данных и таблицу размещения строк (слоты). Связь между страницами и объектами реализует специальная структура — карты размещения.

Строка данных должна полностью уместиться на странице, поэтому существуют ограничения на длину строки. Размер страницы 8 Кбайт, 96 байт занимает заголовок. Кроме того, в таблице размещения каждому

слоту отводится по 4 байта для каждой строки, размещенной на странице.

Строки данных на странице не обязательно хранятся непрерывно. При удалении строки пустое пространство помечается как свободное и потом его может занять новая строка, перемещения строк не происходит. Адрес (смещение) на странице и длина строки фиксируется в *слоте* (Slot).

Если таблица не имеет кластеризованного индекса, то номер слота является идентификатором строки, пока не будет удалена соответствующая строка. Если же таблица имеет кластеризованный индекс, то слоты располагаются в порядке, задаваемом индексом.

Первые страницы данных таблиц БД расположены не подряд: сразу за первой страницей данных таблицы следует ее индексная карта размещения.

Для более эффективного управления дисковым пространством SQL Server не выделяет создаваемым таблицам сразу целый экстенст. Для новой таблицы или индекса, как правило, выделяется место на смешанном экстенсте. Когда объем таблицы или индекса увеличивается до восьми страниц, все последующие выделяемые экстенсты будут однородными. Соответственно, если на смешанных экстенстах места нет, а объем таблицы не достиг еще восьми страниц, то выделяемый новый экстенст будет объявлен смешанным. Например, таблица занимает две страницы на смешанном экстенсте, и в нее еще добавляется сразу шесть записей, то если свободных страниц на смешанных экстенстах нет, будет выделен новый смешанный экстенст, и на нем разместятся 6 записей. Потом, если добавляется еще одна запись, будет выделен полный новый однородный экстенст, и на нем размещена эта новая запись.

### Строки данных

Данные хранятся на страницах в виде строк. Каждая строка кроме собственно данных хранит дополнительную форматирующую информацию. Длина строки зависит от типов полей таблицы. Независимо от объявления, каждая строка имеет поле с количеством полей переменной длины (к ним относятся также поля фиксированной длины, допускающие неопределенные значения NULL, которые при этом резервируют пространство, указанное в определении поля).

Фиксированные поля вместе с описателями хранятся до полей переменной длины. Поля фиксированной длины всегда занимают свою полную длину, значение NULL задается специальным флагом.

В каждой строке хранится общая длина строки и текущие длины полей переменной длины. Данные считываются последовательно с начального адреса.



Вторая часть — это необязательная область, она существует только тогда, когда имеются в записи поля переменной длины, и включает:

- указатель на местоположение полей переменной длины;
- собственно значения полей переменной длины.

### Текстовые страницы

Текстовая страница может содержать несколько текстовых полей.

Строка данных содержит указатель на корневую структуру. Собственно данные хранятся в виде сбалансированного В-дерева.

Данные длиной менее 64 байт хранятся в корневой структуре.

Для данных до 32 Кбайт корневая структура может адресовать 4 блока данных (это не экстенды страниц) до 8 Кбайт каждый. Блоки наращиваются до 8 Кбайт (реально на одной текстовой странице может быть размещено до 8080 байт).

Если же длина текстового поля более 32 Кбайт, то строятся промежуточные узлы.

### Индексы

*Кластеризованный индекс* представляет собой двоичное дерево, в котором на нулевом уровне (уровне листьев) содержатся страницы актуальных данных таблицы, а физическое расположение информации в данном индексе логически упорядочено.

В случае некластеризованных индексов страницы уровня листа содержат не актуальные данные таблицы (как в случае кластеризованного индекса), а указатель на строку данных, включающий номер страницы данных и порядковый номер записи на странице. Некластеризованный индекс не требует физического переупорядочения строк данных таблицы.

Двоичные деревья являются *динамически поддерживаемыми структурами*, т.е. при вставке, удалении или обновлении строк данных информация в индексах также должна быть изменена для отражения выполненных в таблице изменений. Для обработки страниц индексов требуются дополнительные операции ввода-вывода.

Но при разбивке<sup>55</sup> страницы кластеризованного индекса не требуется вносить изменения в информацию некластеризованных индексов, так как в SQL Server 7 эти дополнительные операции ввода-вывода пол-

<sup>55</sup> При добавлении в кластеризованный индекс данные вставляются в таблицу в правильной физической последовательности, соответствующей логической упорядоченности индекса. Поэтому может потребоваться сместить остальные строки таблицы вверх или вниз, в зависимости от места вставки данных. Все это связано с выполнением дополнительных операций ввода-вывода, необходимых для обработки данных индекса. Со временем, по мере накопления данных на странице, при вставке очередной записи может потребоваться разбить информацию страницы на две части, так как для новой записи уже не будет хватать свободного места. В результате часть записей данных будет перенесена на новую страницу. В предыдущих версиях SQL Server эта ситуация сопровождалась обновлением данных во всех некластеризованных индексах таблицы с целью отражения перемещения части строк на новую страницу.

ностью исключены за счет использования в некластеризованных индексах значений ключей кластеризованного индекса вместо номеров физических страниц в случае, когда таблица имеет оба типа индексов.

Индексы таблиц хранятся в виде страниц. Каждая страница размером 8192 байт включает заголовок, имеющий длину 96 байт. Еще один фрагмент страницы используется для размещения других структур данных, например информации о переполнении строк. Вся оставшаяся часть страницы (8060 байт) предназначена для размещения данных. Каждая строка включает элемент индекса (значение индексируемого поля таблицы) и идентификатор RowID (включающий идентификатор файла, номер страницы, номер строки), указывающий на соответствующую запись в таблице.

### **Организация и оптимизация доступа к данным**

Вследствие объективно существующей разницы в скорости работы процессоров и оперативной памяти с одной стороны, и устройств внешней памяти с другой, буферизация страниц базы данных в оперативной памяти — единственно реальный способ достижения удовлетворительной эффективности СУБД. Кроме этого используется механизм *распределенного хранения информации* - расщепления данных между файлами и файловыми группам, физически размещаемыми на разных устройствах или RAID-массивах. Логически такое устройство представляется как единое целое, но на самом деле состоит из нескольких физических дисков. Данные на дисках размещаются блоками одной длины и таким образом, легко могут быть распределены по всем дискам.

Стратегия буферизации, применяемая в операционных средах, не соответствует целям и задачам СУБД, поэтому для оптимизации обработки данных одной из главных задач СУБД является создание эффективной системы управления процессом буферизации.

Память, управляемая СУБД, состоит из нескольких типов буферов:

- буфера страниц данных, с которыми работает СУБД;
- буфера страниц журнала транзакций, которые отражают процесс выполнения транзакции — последовательности операций над БД, переводящей БД из одного непротиворечивого состояния в другое непротиворечивое состояние;
- системные буферы, которые содержат общую информацию о БД, о пользователях, о физической структуре БД, о базе метаданных.

Если бы запись об изменении базы данных реально немедленно записывалась во внешнюю память, это привело бы к существенному замедлению работы системы. Поэтому записи в журнал тоже буферизуются: при нормальной работе очередная страница выталкивается во внешнюю память журнала только при полном наполнении записями.

Но поскольку имеются два вида буферов, содержащих взаимосвязанную информацию — буфер журнала и буфер страниц оперативной памяти, которые могут выталкиваться во внешнюю память, буферы выделяются не для каждого пользовательского процесса, а для всех процессов сервера. Это позволяет увеличить степень параллелизма при исполнении клиентских процессов.

### ***П3. Документальная информационно-поисковая система***

Организация данных и механизмы поиска в базах данных документальных информационных систем, построены на тех же принципах, что и фактографические системы. Однако в физической реализации есть и существенные отличия, которые обусловлены в первую очередь информационной природой элементов данных:

1. Запись базы данных — документ, который задается как набор в общем случае *необязательных* полей, для каждого из которых определены имя и тип. Допустимы большинство стандартных типов (так называемые «форматные» поля, задающие числовые, символьные и другие величины), а также текстовые. Текстовые поля имеют переменную длину и композиционную структуру, не имеющую прямых аналогов среди стандартных типов языков программирования: текстовое поле состоит из параграфов; параграф — из предложений; предложение — из слов. При этом идентифицируемым (адресуемым атомарным) элементом данных с точки зрения хранения будет *поле*, а с точки зрения поиска (атомарным семантически значимым) — *слово*. Вследствие этого поисковые структуры строятся в виде инвертированных файлов.

2. Семантическая природа текстовых полей, представляющих смысл в основном на естественном языке, определяет необходимость учитывать важнейшие свойства используемых терминов: синонимию, полисемию, омонимию, контекстную обусловленность смысла отдельного слова и возможность выразить один смысл многими способами. Вследствие этого поисковые *индексы* могут быть отличны от соответствующих словоформ поля.

На рис. п.1 приведена принципиальная схема организации данных для представления и поиска информации диалоговой системы поиска документов STAIRS (Storage and Information Retrieval System), разработанной фирмой IBM в 70-х годах. Данная структура характерна и для большинства современных АИПС.



Рис. n1. Организация данных в диалоговой системы поиска документов STAIRS

Физическая структура БД рассматриваемой системы включает в себя четыре файла операционной системы:

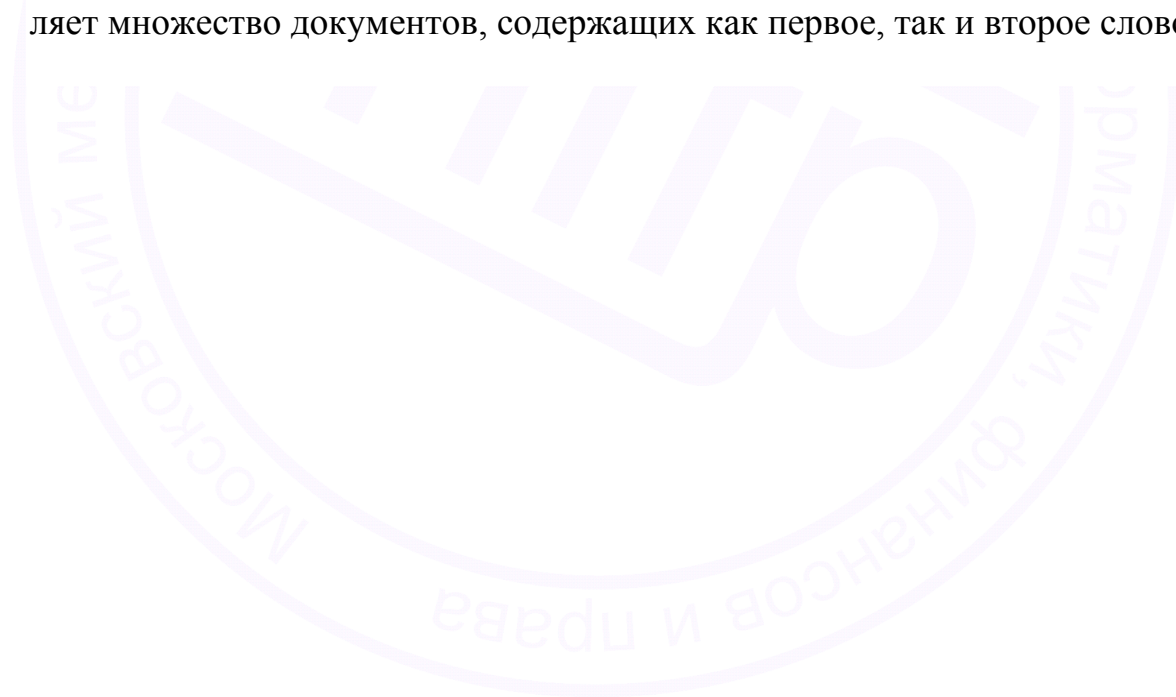
- файл частотного словаря, устанавливающий соответствие между словом, встречающимся в БД, его кодом и частотой, используется при текстовом поиске;
- инверсный (инвертированный, обратный) список, содержащий для каждого слова БД список документов, его содержащих, используется при текстовом поиске;
- текстовый файл, содержащий собственно документы, используется при выдаче (просмотре) документов;
- прямой, последовательный файл, содержащий "собранные" в одну строку фиксированной длины форматные поля и список двухбайтовых кодов слов, находящихся в тексте данного документа. При необходимости, в соответствующих местах находятся разделители сегментов и/или предложений. Файл используется при форматном поиске и при наличии в запросах конструкций *SENT*, *SEGM*, *CTX*.

На рис. п.2 детально представлен *словарь слов*, в котором содержится перечень слов, встречающихся в документах. Ввиду значительных размеров словаря его организация должна предусматривать наличие специального индекса, представленного *матрицей пар знаков*. Каждой паре знаков поставлен в соответствие указатель на *блок словаря*, содержащий группу слов, начинающихся с этих знаков. Знаками могут быть буквы, цифры, а также специальные символы. Вторым знаком может быть пробелом. Группы слов в словаре имеют переменную длину. Первые два знака слов, содержащихся в словаре, отсутствуют, но они показаны на

рисунке, чтобы облегчить понимание структуры файла. Некоторые слова в словаре могут иметь одинаковый смысл; такие слова связаны с помощью специального указателя «синоним» (на рисунке связи данного типа показаны штриховыми стрелками).

Каждому слову поставлен в соответствие указатель на *списки экземпляров*, являющихся перечнем документов, в которых встречается данное слово. Каждый список экземпляров содержит заголовок, из которого можно узнать число экземпляров слова во всем файле документов, а также число документов, в которых это слово встречается.

Система присваивает каждому документу уникальный номер. Этот номер является внутрисистемным и не связан с номерами, по которым пользователь может получить данный документ где-нибудь вне системы. В списке экземпляров, соответствующем какому-либо слову, содержатся внутрисистемные номера всех документов, в которых оно встречается. Поисковый критерий может включать требование *поиска всех документов, содержащих одновременно два специфических слова*. Например, можно осуществлять поиск документа, в котором содержится как слово ORANGUTANG, так и слово OSTRICH. В этом случае система находит множество документов, содержащих первое слово, а затем множество документов, содержащих второе слово, и путем их пересечения определяет множество документов, содержащих как первое, так и второе слово.





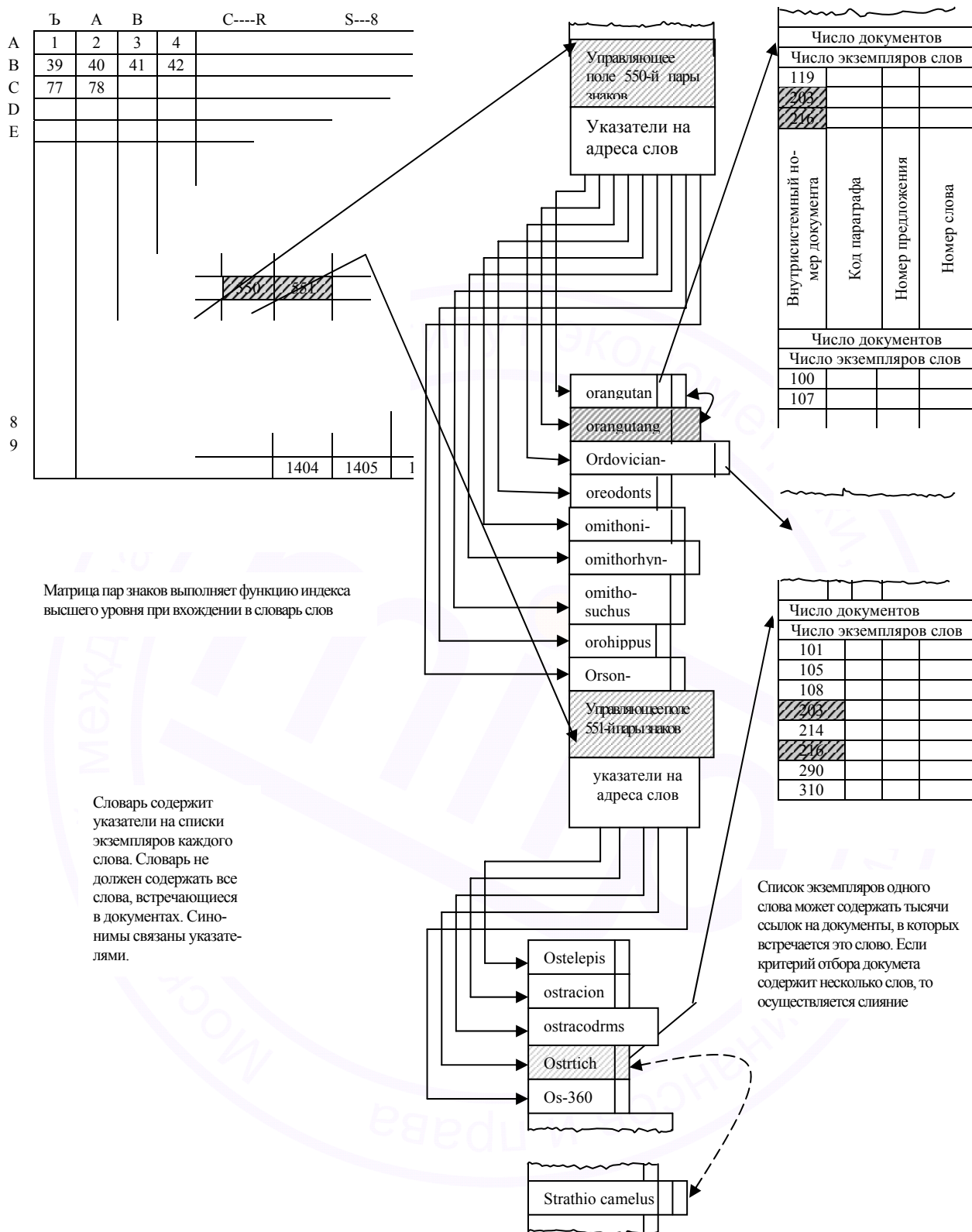


Рис. п.2. Организация поисковых индексов АИПС STAIRS

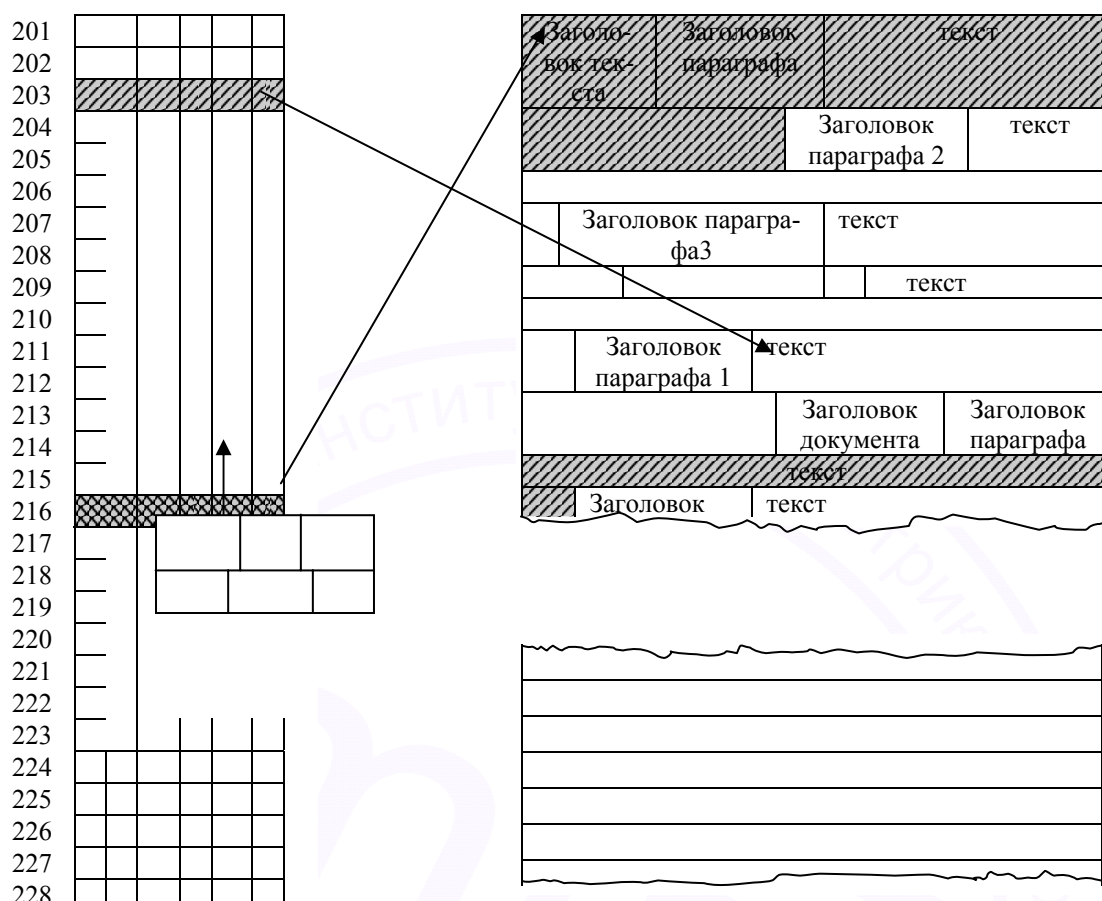


Рис. п.3. Организация поисковых файлов документов АИПС STAIRS

На рис. п.3 показан *файл документов*, каждому из которых система сама присваивает внутренний порядковый номер. Документы состоят из параграфов и текстов, причем тексты также пронумерованы. Каждому параграфу присвоен специальный код, определяющий его тип (например, заголовок, автор, аннотация и т. д.).

Внутрисистемный номер документа является ключом к *индексу документов*. Этот индекс содержит адреса соответствующих документов в памяти. В принципе *можно* хранить эти адресные указатели непосредственно в списке экземпляров, но это нецелесообразно, так как объем памяти, необходимый для хранения адреса, больше объема памяти, необходимого для хранения номера документа. Индекс документов содержит не только адреса, а также некоторые вспомогательные сведения о документах. К этим сведениям относятся внешний номер документа, признак удаления документа, указывающий, какие параграфы документа (или документ в целом) исключены из файла, а также уровень секретности.

В состав документов могут входить *параграфы различных типов*, поэтому пользователь может потребовать, чтобы заданное слово со-

держалось в названии документа, аннотации, введении или каком-либо конкретном параграфе. В критерии отбора можно указывать автора, место издания документа и дату издания. Независимо от содержания критерия отбора поиск документа осуществляется на уровне списка экземпляров без необходимости входа в файл документов.

#### ***П4. Интегральный банк юридической информации ЮРИУС***

С логической точки зрения банк имеет «стандартную» структуру и включает две компоненты: *регистрационные карты (РК)* и *полные тексты*.

*РК* представляют собой форматированные записи, содержащие относительно стандартный набор библиографических данных, а также ссылку на соответствующий полный текст (Рис. п.3).

*Полные тексты* документов состоят из страниц двух типов:

- логических, т.е. структурных единиц текста - пункт, параграф, статья;
- физических - принудительное разбиение длинного неструктурированного текста на фрагменты одинаковой длины).

Кроме этого имеется возможность отнесения документа к тому или иному *тому Свода законов* (с номерами 1 - 10). Это связано с традицией выпуска Свода законов в форме десяти томного печатного издания.



*Рис. п.4. Логическая структура БД ЮРИУС*

**Физическая структура БД ЮРИУС** (Рис. п.5.) является примером реализации документальной системы в среде реляционной СУБД.

**Файл текстовой части БД (SZDOC)**- один или несколько файлов, в которых содержатся полные тексты актов. На логическом уровне образует, представленную на рис. иерархическую структуру: БД (том), документ, страница.

**Словарный файл текстовой части (DCFRV)** - представляет собой список слов и стандартных словосочетаний, (например “статья 256”, “п. 13”, “N 1400-РП”), извлеченных из текста, сопровождаемых частотами появления в данной БД. Практика выделения словосочетаний при индексировании с целью включения их в словарь и инверсный список является достаточно известной.

**Инверсный файл текстовой части (DCIND)** - список кодов слов и словосочетаний, сопровождаемых номерами страниц. Словарный и инверсный файл используются для сквозного полнотекстового поиска.

**Справочно-поисковые файлы (СПФ)** (до 9-ти различных файлов SF1 - SF9). Стандартным является файл *регистрационных карт* нормативных актов (РК), запись которого содержит наименование, дату, номер, вид, ссылки на страницы БД и другие поля, перечень которых может изменяться для конкретной БД.

**Словарь справочно-поисковых файлов (IXFRV)** содержит значения и коды полей (например, РК) совместно с частотой появления и ссылкой на номер файла СПФ.

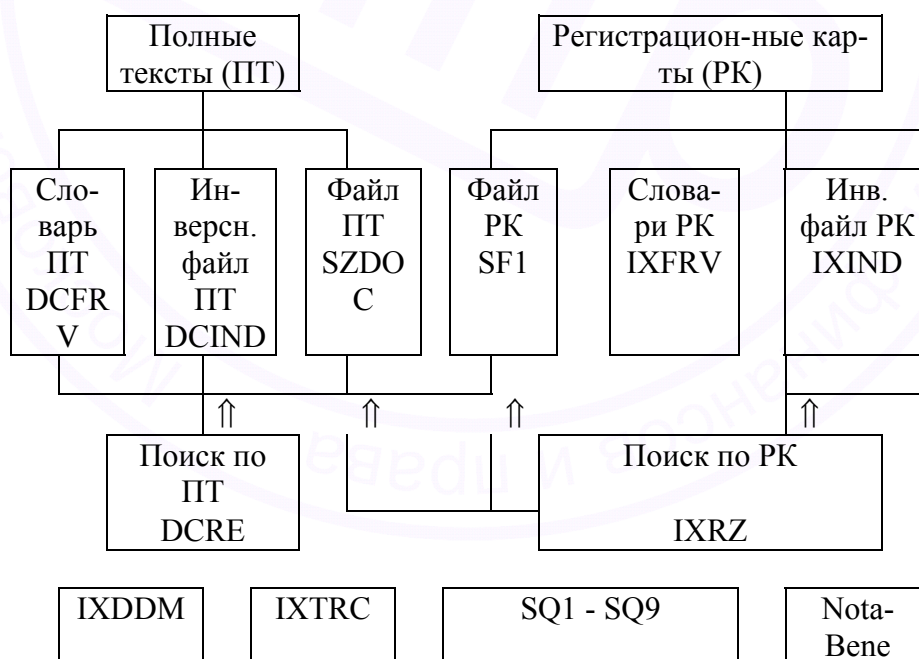


Рис. п.5. Физическая структура БД и использование файлов модулями пользовательского интерфейса (БД НЗР)

*Инверсный файл СПФ (IXIND)* содержит коды слов и словосочетаний. Словарный и инверсный файлы используются для поиска записей СПФ (РК, рубрики указателя и т.д.) с доступом к странице БД.

*Файл синонимов (IXTRC)* служит для расшифровки кодов или для обеспечения двуязычного поиска в словарных файлах.

*Файл описания СПФ (словарь данных IXDDM)* - содержит данные о полных, сокращенных и внутренних именах полей каждого СПФ, типах данных, разделителях слов, методах обработки числовых кодов и т.д. Используется при поиске через СПФ и при построении словарных и инверсных файлов.

*Файлы хранимых запросов (SQ1 - SQ9)* содержат запросы к СПФ БД, отложенные и сохраненные пользователем.

*Файл заметок (NotaBene)* позволяет пользователю дополнить СПФ собственными именованными прямыми ссылками на страницы БД.

### ***П5. Технологии индексирования текстовой информации***

В рассмотренной выше документальной АИПС STAIRS используется стратегия *свободного индексирования*. Каждое (за исключением так называемых *стоп-слов*, т.е. информационно незначимых слов, перечисленных в специальном словнике) слово загружаемого в базу данных документа может использоваться в качестве индекса – ключа поиска этого документа.

Индексирование по ключевым словам (или атрибутивное индексирование) является наиболее простой и экономичной в отношении дискового пространства технологией. Суть ее заключается в том, что для каждого вводимого или сохраняемого документа заполняются соответствующие поля в индексном файле. Заполнение осуществляется как вручную, так и с помощью программы, выделяющей в документе по какому-либо признаку значения ключей/атрибутов. Эта технология позволяет индексировать как текстовые документы (в ручном и автоматическом режимах), так и изображения (в ручном режиме). В простейшем случае ключевыми словами служат название и/или имя автора документа. В более сложных ситуациях необходимо использовать независимого эксперта для чтения документа и выделения ключевых слов.

Серьезные ограничения при использовании этих систем связаны со следующими обстоятельствами:

- Определение ключевых слов - достаточно субъективный процесс; даже при участии самого независимого эксперта трудно избежать односторонности при выборе ключевых слов.

- Определение ключевых слов - достаточно дорогостоящая процедура (по оценкам АИМ, наиболее авторитетной организации на рынке систем, связанных с управлением документами, это от 5 до 20 долларов на документ) из-за невозможности автоматической индексации и низкой производительности при определении ключевых слов вручную.



- Предполагается, что пользователи будут осуществлять поиск информации предсказуемым способом, используя predetermined ключевые слова.

- Поиск по ключевым словам - это четкий поиск, когда пользователь должен точно знать, что он ищет. Если сделана ошибка при написании ключевого слова в запросе для поиска, система никогда не найдет нужную информацию.

- Ключевые слова могут со временем меняться (понятия, которые были "ключевыми" вчера, вовсе не обязательно будут столь же важны через год).

Поиск информации в таких системах происходит с использованием механизмов полнотекстового поиска, который реализуется с помощью технологии индексирования на основе инвертированных структур: при создании индексного файла в него вносятся все значимые слова (без союзов, предлогов и т. п.) из всех документов в алфавитном порядке. Эти слова затем объединяются в пары с указателями на документы, содержащие эти слова.

Цель индексирования документов – возможность их быстрого поиска. Индекс – это набор слов документа или о документе, по которым этот поиск производится. Основными критериями качества индексирующе-поисковых подсистем являются качество поиска (процент нерелевантных документов в списке найденных), размер индекса по отношению к размеру документа и скорость поиска по нему.

Развитие индексирования в документальных системах происходило от ручного заполнения списка ключевых слов в системах первого поколения до автоматического полнотекстового индексирования сегодня, подразумевающего сохранение всех слов текста. Несмотря на большой пройденный путь говорить о полном решении проблемы, наверное, пока рано.

Индексирование документа обычно организуется через автоматическую обработку его текста и заполнение метаданных. Автоматическая обработка – полнотекстовое индексирование – заключается в преобразовании текста документа в набор слов. Причем обычно для слов сохраняется их позиция в документе, что обеспечивает возможность поиска по словосочетаниям. Существуют два принципиально различных метода такого индексирования с учетом применяемых в дальнейшем методов поиска:

- бинарное индексирование – не зависит от языка документа по причине бинарной или словарной индексации;

- морфологическое индексирование – производится с учетом морфологии и семантики языка.

При бинарном индексировании поиск ведется на основе алгоритмов “нечеткого поиска”, т.е. поиска с ошибками. В этом случае допускается неполное (с заданным количеством ошибок в начале, середине и

конце слова) совпадение слов с шаблоном. При втором методе индексации слова преобразуются в словоформы с отсечением суффиксов и окончаний, что позволяет искать склонения и спряжения шаблонов.

Заметим, что, несмотря на несомненные плюсы, полнотекстовое индексирование в любом своем виде имеет и существенный минус: большое количество “мусора” в индексе, т.е. слов никак не характеризующих документ, а связывающих “ключевые” слова.

Эти недостатки обусловлены самой концепцией такого индексирования – сохранением всего текста за исключением “стоп-слов”, под которыми подразумеваются предлоги, союзы, местоимения и т.п. Действительно, с одной стороны наличие в индексе всех слов текста гарантирует его нахождение по любому из них, но с другой стороны встает вопрос: “А насколько это корректно?”. Предположим, мы имеем текст о компьютерных технологиях, в котором приведена пословица: “За двумя зайцами погонишься, ни одного не поймаешь”. При проведении поиска по слову “заяц” система выдаст этот документ, хотя он не будет иметь ни малейшего отношения к фауне. Безусловно, можно найти и сотни менее экзотичных примеров.

Среди других “узких мест” можно назвать следующие:

1) Индекс, создаваемый такими системами, обычно составляет от 200 до 400% от объема исходного текста, что означает увеличение времени поиска и ресурсов компьютера.

2) Из-за необходимости “очистки” текста стоимость обработки документов достаточно велика.

3) Механизм четкого поиска через инвертированный файл не позволит найти информацию, если были допущены ошибки при загрузке текста или при написании запроса.

В начале 90-х годов появились технологические разработки, связанные с индексацией и поиском документов и использующие результаты, полученные в области нейронных сетей и искусственного интеллекта. Они позволили сформулировать принципиально новые концепции построения систем управления неструктурированной информацией в электронном виде.

Компания Excalibur Technologies разработала и представила на рынке технологию адаптивного распознавания образов APRP (Adaptive Pattern Recognition Processing), которая была положена в основу программного продукта - систему управления документами Excalibur EFS [8]. Технология APRP основана на нейронных сетях, что позволяет не только обойти проблемы ошибок распознавания текстов, но и предоставляет возможности автоматического индексирования и поиска различных типов неструктурированной информации.

Ядро технологии APRP “выросло” из работ основателя компании Excalibur Technologies Джеймса Дау III, посвященных изучению и разработке моделей нейронных сетей, способных идентифицировать, или

более точно, распознавать присутствие тех или иных образов в составе данных специального вида. Это позволило построить систему индексации общего назначения, которую можно применять к основным видам данных, включая устную речь (голос), сигналы, тексты и изображения. Был также создан комплекс алгоритмов, самостоятельно адаптирующихся к особенностям обрабатываемой информации и позволяющих осуществлять *нечеткий поиск* - поиск образов, составленных из двоичных символов.

В технологии APRP под нечетким поиском понимается возможность найти достаточно близкое приближение к запрошенному термину или фразе. Нечеткий поиск устраняет для пользователя необходимость знать правильное написание каждого термина, с которым он работает. Поскольку APRP работает не с ключевыми словами, а с образами, две-три ошибочные буквы в слове или фразе не могут существенно изменить базовую картину текста. Таким образом, автоматически становится исправимой ошибка, как во входных данных, так и в терминах запроса. APRP всегда в состоянии найти ближайшее приближение к терминам и фразам, заданным в качестве объектов поиска. Поясним это на примере.

Даже, если мы напишем в запросе:

ЦЦЦТЕРМАРГМАСАРИТАЭЭЭЭЭЭ,

имея в виду название романа Михаила Булгакова, мы получим правильный ответ: "Мастер и Маргарита".

Поиск происходит следующим образом:

- Запрос конвертируется в бинарную форму
- Игнорируется шум, т.е. отбрасываются ЦЦЦ и ЭЭЭЭЭЭ
- Проводится нечеткий поиск

Как реально происходит нечеткий поиск? Ранее упоминалось, что технология APRP оперирует информацией на уровне двоичных кодов, т. е. каждое слово для нее - это образ, состоящий из нулей и единиц. Например, слово "пень" для нее представляется двоичным образом 10101111 10100101 10101101 11101100; а слово "печь" имеет двоичный образ 10101111 10100101 11100111 11101100 (каждая буква в слове представляется одним байтом). Сравним двоичные образы обоих слов:

ПЕНЬ	—
10101111 10100101 10101101 11101100	
ПЕЧЬ	—
10101111 10100101 11100111 11101100	

Из 32 позиций каждого двоичного образа не совпадают только комбинации из 6-ти элементов, что составляет лишь около 20% от длины двоичного образа. С точки зрения технологии APRP образы этих слов очень близки к друг другу, и в качестве результата поиска вам могут быть предложены документы, содержащие оба слова, а вы укажете, которые из них вы имели в виду при поиске. Приведенный пример, однако, не означает, что вам будет предложен бесконечный список вариантов, в той или иной степени похожих на ваш запрос.

## Литература

1. Аткинсон М., Бансилон Ф., ДеВитт Д., Диттрих К., Майер Д., Здоник С. Манифест систем объектно-ориентированных баз данных // СУБД, №4, 1995.
2. Вирт Н. Алгоритмы и структуры данных: пер. с англ. – М.: Мир, 1989.
3. Грофф Дж, Вайнберг П. SQL: полное руководство: Пер. с англ. – 2-е изд. – К.: BHV, 2001.
4. Дейт К. Дж., Введение в системы баз данных: пер. с англ. - 7-е издание. – М.: Вильямс, 2001.
5. Диго С.М. Проектирование и использование баз данных: Учебник. – М.: Финансы и статистика, 1995.
6. Дунаев С.Б. Доступ к базам данных и техника работы в сети. Практические приемы современного программирования. – М.: ДИАЛОГ-МИФИ, 1999.
7. Зиндер Е.З. Проектирование баз данных: новые требования, новые подходы. // СУБД, N3, 1996.
8. Каменнова М. Управление электронными документами: технологии и решения. // Открытые системы, №4, 1995
9. Карпова Т.С. Базы данных: модели, разработка, реализация. – СПб.: Питер, 2001.
10. Ким Вон, Технология объектно-ориентированных баз данных. // Открытые системы, №4, 1994.
11. Когаловский М.Р. Абстракции и модели в системах баз данных. // Открытые системы, №4-5, 1998.
12. Кузнецов С.Д. Основы современных баз данных. [www.citforum.ru](http://www.citforum.ru), 2002.
13. Малкольм Г. Программирование для Microsoft SQL Server 2000 с использованием XML.: Пер. с англ. – М.: Издательско-торговый дом «Русская редакция», 2002.
14. Мартин Дж. Организация баз данных в вычислительных системах. – М.: Мир, 1980
15. Мартин Дж. Превратите вашу компанию в киберкорпорацию. // Computerworld Россия, ноябрь 14, 1995.
16. Меллинг В.П. Корпоративные информационные архитектуры: и все-таки они меняются. // СУБД, N2, 1995.
17. Озкарахан Э. Машины баз данных и управление базами данных. – М.: Мир, 1989.
18. Фаронов В.В., Шумаков П.В. Delphi 4. Руководство разработчика баз данных. – М.: «Нолидж», 1999.
19. Фокс Дж. Программное обеспечение и его разработка. – М.: Мир, 1985.
20. Цикритзис Д., Лоховски Ф. Модели данных. - М.: "Финансы и статистика", 1985.



21. Шпеник М., Следж О. Руководство администратора баз данных Microsoft SQL Server 2000: пер. с англ. – М.: Вильямс, 2001. 928с.
22. ANSI/X3/SPARC Study Group on Data Base Management Systems. Interim Report. FDT Bull. ASM-SIGMOD. v. 7, no. 2 (1975)
23. Chen P. P.-S. The Entity-Relationship Model – Toward a Unified View of Data // ACM TODS. – 1976. – 1, №1.
24. CODASYL DBTG Report. – New York: ACM, 1969.
25. Codd E.F. Extending the Database Relational Model to Capture More Meaning. // ACM Trans. Database Syst., N 4, 1979.
26. Codd E.F., Codd S.B., and Salley C.T. Providing OLAP to User-Analyst: An IT Mandate. E.F. // Codd & Associates, 1993.
27. Hammer M., Champy J. Reengineering the Corporation. A Manifesto for Business Revolutions. // HarperBusiness, 1993.
28. Sowa J.F., Zachman J.A. Extending and Formalizing the Framework for Information System Architecture. // IBM System Journal, vol. 31, no. 3, 1992.
29. Zachman. John A. A Framework for Information System Architecture. // IBM System Journal, vol. 26, no. 3, 1987.