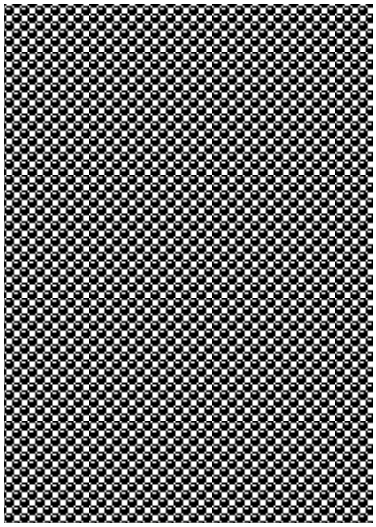
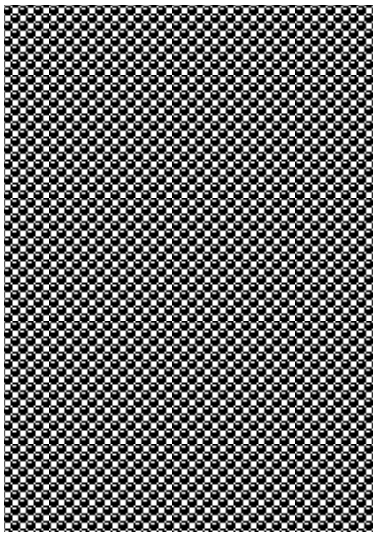
Ю.В. ЛИТОВКА
И.А. ДЬЯКОВ
А.В. РОМАНЕНКО
С.Ю. АЛЕКСЕЕВ
А.И. ПОПОВ



ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ В САПР



♦ ИЗДАТЕЛЬСТВО ТГТУ ♦

Министерство образования и науки Российской Федерации

Государственное образовательное учреждение
высшего профессионального образования
«Тамбовский государственный технический университет»

**Ю.В. ЛИТОВКА, И.А. ДЬЯКОВ, А.В. РОМАНЕНКО,
С.Ю. АЛЕКСЕЕВ, А.И. ПОПОВ**

ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ В САПР

Утверждено Ученым советом университета в качестве учебного пособия

Тамбов
Издательство ТГТУ
2005

УДК 681(075)
ББК ←973-018.3я73
О75

Рецензенты:

Доктор технических наук, профессор
Ю.Ю. Громов

Доктор физико-математических наук, профессор
В.Б. Байбурин

Литовка Ю.В., Дьяков И.А., Романенко А.В.,
Алексеев С.Ю., Попов А.И.
О75 Основы проектирования баз данных в САПР: Учеб. пособие. Тамбов: Изд-во Тамб. гос. техн. ун-та, 2005. 96 с.

Содержит сведения о теоретических основах разработки баз данных, приведена их классификация, рассмотрены подходы к организации информационного обеспечения САПР и основы проектирования баз данных.

Предназначено для студентов, обучающихся по специальности 230104.

УДК 681(075)
ББК ←973-018.3я73

ISBN 5-8265-0422-6

© Литовка Ю.В., Дьяков И.А.,
Романенко А.В., Алексеев С.Ю., Попов А.И., 2005
© Тамбовский государственный
технический университет
(ТГТУ), 2005

Учебное издание

ЛИТОВКА Юрий Владимирович,
ДЬЯКОВ Игорь Алексеевич,
РОМАНЕНКО Александр Васильевич,
АЛЕКСЕЕВ Сергей Юрьевич,
ПОПОВ Андрей Иванович

ОСНОВЫ ПРОЕКТИРОВАНИЯ
БАЗ ДАННЫХ В САПР

Учебное пособие

Редактор Т.М. Глинкина
Инженер по компьютерному макетированию Т.А. Сынкова

Подписано к печати 8.11.2005.
Формат 60 × 84/16. Гарнитура Times. Бумага офсетная. Печать офсетная.
Объем: 5,58 усл. печ. л.; 5,45 уч.-изд. л.
Тираж 100 экз. С. 748

Издательско-полиграфический центр
Тамбовского государственного технического университета
392000, Тамбов, ул. Советская, 106, к. 14

ВВЕДЕНИЕ

Автоматизация многих сфер человеческой деятельности прочно базируется на обработке, хранении и преобразовании больших объемов информации. Исключение не составляют и специализированные программные комплексы, занятые в сфере решения задач автоматизации проектирования, которые называются системами автоматизированного проектирования (САПР).

Одним из основных компонентов САПР является информационное обеспечение. Информационное обеспечение САПР объединяет различные данные, необходимые для выполнения автоматизированного проектирования. В рамках одноименной дисциплины изучаются теоретические вопросы в сфере разработки баз данных и специальных систем управления базами данных (СУБД), а также их практическое воплощение.

Настоящее учебное пособие предназначено для выработки у студентов практических навыков по обработке информационных потоков, проектированию, разработке и эксплуатации баз данных и СУБД.

Учебное пособие состоит из двух частей. В первой части описаны теоретические основы разработки баз данных: функции, структура и классификация баз данных, иерархическая, сетевая и реляционная модели данных, основы реляционной алгебры и нормализация информации при проектировании баз данных, а также инфологическое и даталогическое проектирование баз данных.

Во второй части учебного пособия приводятся описание языка запросов к базам данных SQL и основы работы с драйвером ODBC, позволяющим организовать взаимодействие прикладных программ с базами данных. Даны рекомендации по организации эффективной работы проектировщика, занятого реализацией реального проекта.

Ч а с т ь I

ТЕОРЕТИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ БАЗ ДАННЫХ

1 КЛАССИФИКАЦИЯ ДАННЫХ

1.1 ИНФОРМАЦИЯ И ДАННЫЕ

Рассмотрим понятия *информация* и *данные*. В узком смысле *информация* – это приращение знаний. Понятие "информация" связано с семантикой, т.е. с содержательной интерпретацией данных. В более широком смысле информация отождествляется с некоторыми сведениями. Таким образом, данные рассматриваются как носитель информации. Автоматизированные системы поддерживают модель некоторой части реального мира – предметной области. В зависимости от уровня семантической интерпретируемости, обеспечиваемой используемой моделью, выделяют модели данных (даталогические) и информационные (инфологические) модели.

Структурно данные могут быть представлены тремя уровнями: концептуальным, внешним и внутренним. *Концептуальный уровень* отражает объективные свойства данных, описывающих предметную область. *Внешний уровень*, напротив, отражает субъективные взгляды приложений на данные. В практических случаях внешний уровень является подмножеством концептуального представления. *Внут-*

ренний уровень представления данных определяет машинно-ориентированное, физическое представление данных. В указанной структуре концептуальный уровень находится между внешним и внутренним. Рассмотрим в качестве примера уровни представления списка или списков деталей, входящих в состав изделия. Как видно из схемы (рис. 1.1), на концептуальном уровне могут быть представлены данные, не обязательно относящиеся к списку деталей – это может быть и бухгалтерский документ. Одновременно внешний уровень отражает наше восприятие. Здесь решающим является квалификация, например инженера-конструктора. Внутренний уровень не играет в решении данной задачи главной роли и может меняться в зависимости от типа и уровня развития компьютерной техники.

Современные системы обработки информации выполняют преобразования больших массивов данных. Здесь и далее будем для простоты синонимом понятия "данные" считать термин "информация", так как это ближе для технических систем. Система автоматизированного проектирования (САПР) также оперирует большим числом данных различного типа и назначения.

Рис. 1.1 Уровни представления данных

1.2 ОСНОВНЫЕ ПОНЯТИЯ

Рассмотрим основные определения понятий, часто используемых в информационном обеспечении (ИО) и являющихся основополагающими. Термин *база данных* начал применяться с 1963 г. и записывался на английском языке как "data base". По мере развития вычислительной техники, эти два слова были объединены в одно (database). Основной смысл, вкладываемый в термин "база данных" – это база информационной системы, инструментом обработки данных в которой является ЭВМ. Информационная база или база данных представляет собой совокупность данных, предназначенных для совместного применения.

Одним из разработчиков теории баз данных, Инглисом (R. Engles), в 1972 г. дано следующее рабочее определение: база данных представляет собой совокупность хранимых операционных данных, используемых прикладными системами некоторого предприятия. Другой классик теории баз данных К. Дейт в своих работах дает более предметное определение базы данных, как совокупности данных, хранящихся во вторичной памяти ЭВМ (на дисках). Одновременно российские разработчики теории баз данных предложили понимать под термином "база данных" даталогическое представление информационной модели предметной области. Это наиболее абстрактное и емкое определение. Государственным комитетом по науке и технике СССР (ГКНТ) в 1982 г. был принят ряд документов, определяющих базу данных как именованную совокупность данных, отражающую состояние объектов и их отношений в рассматриваемой предметной области. Таким образом, единого мнения по поводу определения термина "база данных" пока не существует. На основе анализа существующих определений и истории развития данной науки в дальнейшем будем пользоваться следующими определениями.

База данных (БД) – структурированная совокупность данных. Наименьшая единица описания данных называется элементом описания. Совокупность элементов описания, объединенных отношением принадлежности к одному описываемому объекту, называется *записью*. Например, код типа микросхемы, логическая функция, мощность потребления, коэффициент разветвления в совокупности составляют запись и описывают свойства конкретного объекта – микросхемы.

Система управления базами данных (СУБД) состоит из языковых и программных средств, предназначенных для создания, ведения и эксплуатации баз данных.

Модель данных – формализованное описание, отражающее состав и типы данных, а также взаимосвязь между ними.

На каждом из рассмотренных на рис. 1.1 уровней присутствует своя модель данных. Это так называемый логический уровень моделей данных. По способам отражений связей между данными на логическом уровне различают иерархическую, сетевую и реляционную модели. Модель называется сетевой, если данные и их связи имеют структуру графа. Если структура отражаемых связей представляется в виде дерева, то модель называют иерархической. Представление данных в форме таблиц соответствует реляционной модели данных. Задание модели данных в БД осуществляется на специальном языке описания данных.

2 ОРГАНИЗАЦИЯ ИНФОРМАЦИОННОГО ОБЕСПЕЧЕНИЯ САПР

Система автоматизированного проектирования представляет собой сложную и многокомпонентную систему, процессы преобразования данных в которой разнообразны. При этом данные, являющиеся результатом одного процесса преобразования, могут быть исходными для другого процесса. Вследствие этого термин "данные" может иметь различную трактовку в САПР. Так, например, для управляющего монитора в состав данных входит совокупность программных модулей, обеспечивающих процесс проектирования. Для подсистемы математического моделирования и оптимизации к данным относится совокупность исходных и результирующих чисел, для правильной организации проектных работ пользователю САПР в качестве данных требуется иметь исходную проектную документацию, справочные данные, типовые проектные решения и т.д. Форма представления данных также может быть различной: в виде текстов, графических изображений, звука, видеороликов. Совокупность данных, используемых всеми компонентами САПР, представляет *информационный фонд САПР*.

Назначение ***информационного обеспечения САПР*** (ИО САПР) – реализация информационных потребностей всех составных компонентов САПР. Основная функция информационного обеспечения состоит в ведении информационного фонда. Таким образом, информационное обеспечение САПР имеет две составляющих – это информационный фонд и средства его ведения.

Состав информационного фонда САПР можно определить следующим образом:

1) *программные модули*, участвующие в процессе проектирования, начиная от операционных систем и заканчивая пакетами прикладных программ. Часть этих данных меняется довольно редко, другая может динамично изменяться, например, при разработке новых методик или нового математического обеспечения САПР;

2) *исходные и результирующие данные* (цифровые, текстовые, графические, видео, звуковые) для обработки программными модулями. Эти данные меняются часто в процессе проектирования, однако их тип постоянен;

3) *нормативно-справочная проектная документация* (НСПД) включает справочные данные об элементах проектируемых изделий, технологиях их изготовления и испытаний, унифицированных узлах и конструкциях. Государственные и отраслевые стандарты, руководства и указания, типовые проектные решения, регламентирующие документы также относятся к НСПД;

4) *проектная документация*, отражающая состояние и ход выполнения процесса проектирования. Изменяется в процессе проектирования и представляется в виде текстового и графического материала. Возможно звуковое сопровождение и динамическая смена графики (видеоролики). Сюда можно отнести также и готовые проектные решения.

Рассмотрим следующий пример. Пусть надо разработать программу для микропроцессорной системы, управляющей роботом. Для этого в проекте должны быть следующие данные: пояснительная записка, общее описание, руководство программиста, описание языка, описание программ, порядок и методика проведения испытаний.

Проблема организации и ведения информационного фонда решается в двух направлениях: методологическом и организационном. Первое направление полностью определено методикой автоматизированного проектирования и алгоритмами решения частных задач. Во втором направлении различают следующие способы: использование файловой системы, построение библиотек, использование банков данных, создание информационных программ-адаптеров (для организации межмодульного интерфейса).

3 БАНКИ ДАННЫХ: ОБЩИЕ ТРЕБОВАНИЯ И ТРАДИЦИОННАЯ АРХИТЕКТУРА

Рассмотрим более полное определение банков данных. ***Банк данных*** – представляет собой систему специальным образом организованных данных (баз данных), а также программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования информации.

Термин "банк данных" (БнД) не является общепризнанным. Наиболее близким к нему в англоязычной литературе является термин "система баз данных" (*data base system*). Система баз данных включает базу данных, СУБД, соответствующее оборудование и персонал. Понятие "система баз данных" уже, чем банк данных, так как "банк" обозначает то, что хранится в нем и всю инфраструктуру, но по сути они одинаковы.

Банк данных является сложной человеко-машинной системой, включающей в свой состав различные взаимосвязанные и взаимозависимые компоненты (рис. 3.1). Ядром банка данных является база данных. Информационный компонент банка данных состоит из БД, схем БД и словарей данных. Словари данных играют в САПР особо важное значение.

Программные средства банков данных представляют собой сложный комплекс, обеспечивающий взаимодействие всех частей информационной системы при ее функционировании (рис. 3.2). Основу программных средств банка данных представляет СУБД. В ней можно выделить ядро СУБД, обеспечивающее организацию ввода информации, обработки и хранения данных, средства настройки и тестирования, а также утилиты вспомогательных функций для восстановления БД, сбора статистики о функционировании БД и проч. Компиляторы и трансляторы являются важной компонентой языковых средств СУБД. Все СУБД работают под управлением операционной системы (ОС).

Для обработки запросов к БД разрабатывается соответствующее прикладное программное обеспечение.

Языковые средства банков данных основываются на языковых средствах СУБД и должны обеспечивать интерфейс пользователей разных категорий с банком данных. Их спектр достаточно широк (рис. 3.3).

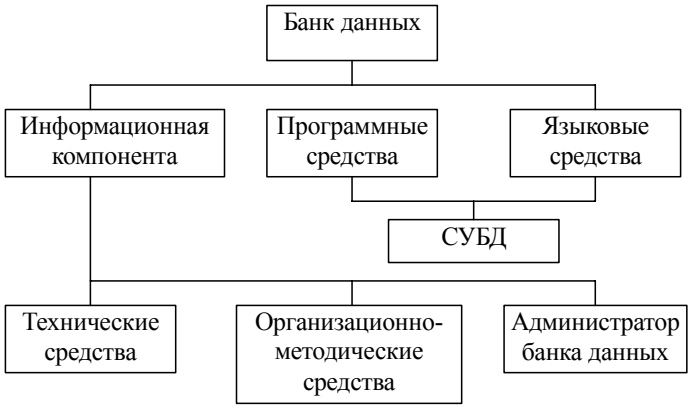


Рис. 3.1 Компоненты банков данных

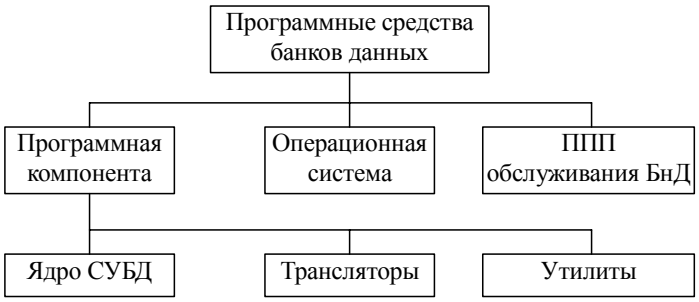


Рис. 3.2 Программные средства банков данных

Категории языковых средств различаются по функциональным возможностям:

- языки ввода данных по запросу (устаревшая компонента);
- языки запросов-обновлений (сложные запросы по нескольким взаимосвязанным записям);
- генератор отчетов для выбора данных и формирования в виде формы требуемого документа;
- графические языки (аналогичны генератору отчетов) в которых данные отображаются в виде диаграмм, графиков и т.п.;
- языки принятия решений (например, Пролог);
- генераторы приложений для автоматизированной генерации программ;
- параметризированные ППП для генерирования собственных отчетов и запросов;
- языки приложений.

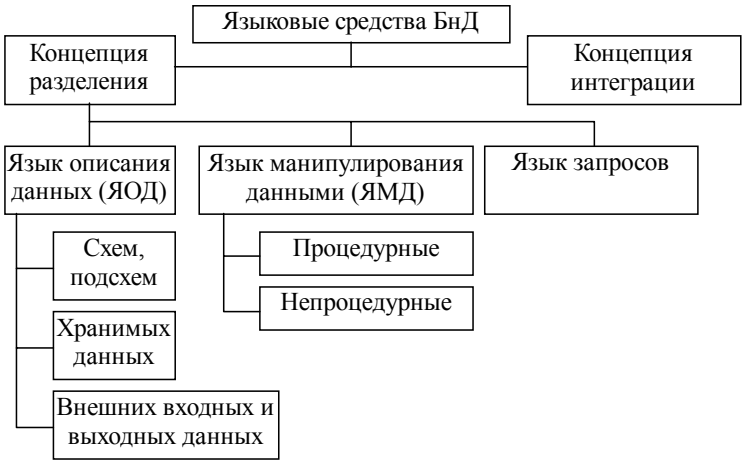


Рис. 3.3 Языковые средства банков данных

По форме представления различают аналитические, табличные и графические языковые средства. Такая классификация справедлива и для ЯОД и для ЯМД.

Современные СУБД обычно включают в свой состав несколько языковых средств разного уровня.

В качестве *технических средств* для банков данных чаще всего используются универсальные ЭВМ, стандартный набор периферийных устройств и сетевого оборудования. Для их создания и эксплуатации применяются специальные технические средства, например серверы, накопители на магнитных лентах (стримеры), накопители на оптических носителях и проч.

К *организационно-методическим средствам* банков данных относятся различные инструкции, методические и регламентирующие документы для пользователей различных категорий.

Группа специалистов, обеспечивающих создание, функционирование и развитие систем баз данных называется администратором банка данных (АБД).

3.1 СВОЙСТВА И КЛАССИФИКАЦИЯ СИСТЕМ БАЗ ДАННЫХ

Банки или системы баз данных являются сложными системами и их классификация может быть проведена по различным признакам, относящимся как к банкам данных в целом, так и к их компонентам. Большинство классификационных признаков относится к центральной компоненте банка данных – базе данных. Рассмотрим некоторые из них.

По форме представления данных разделены на видео- и аудиосистемы, мультимедиа, а также символьные. В настоящее время наибольшее применение находят базы данных, содержащие обычные символьные данные. Они в свою очередь разделены на неструктурированные, частично структурированные и структурированные (семантические сети, обычный текст и построение по модели). По типу хранимой информации БД можно разделить на документальные, фактографические и лексикографические. По характеру организации и хранения данных и обращению к ним различают локальные (однопользовательские), общие (интегрированные), распределенные и объектно-ориентированные. На рис. 3.4 приведена структурная схема классификации банков данных по различным признакам.

Банки данных классифицируют по следующим свойствам:

1) *скорость доступа* – определяет время реакции, т.е. получение ответа на запрос пользователя;

2) *доступность* определяет какие данные, содержащиеся в БД, доступны данной категории пользователей;

3) *гибкость* определяет возможность получить ответ на сложные запросы;

4) *целостность* отвечает за снижение избыточности данных, согласованность данных при упорядочении обновления.

3.2 ФУНКЦИИ И СТРУКТУРА СУБД

Система управления базой данных (СУБД) представляет собой программное обеспечение, которое управляет доступом к хранимой в базе данных информации. Этот процесс производится следующим образом:

1) пользователь выдает запрос на доступ, применяя команды определенного языка манипулирования данными (в этих целях обычно используется язык SQL);
2) СУБД перехватывает этот запрос и анализирует его;
3) СУБД просматривает внешнюю схему для пользователя, концептуальную, внутреннюю схему и определяет структуру хранения информации;
4) СУБД выполняет необходимые операции над хранимой базой данных.

Описанные действия выполняются благодаря функциям СУБД. Рассмотрим их подробнее.

– *Определение данных.* СУБД должна допускать определение данных в исходной форме и преобразовывать эти определения в форму соответствующих объектов.

– *Обработка данных.* СУБД должна уметь обрабатывать запросы пользователя по выбору, изменению или удалению существующих данных или добавлению новой информации. Программное обеспечение должно быть построено таким образом, чтобы реализовать *планируемые* и *непланируемые* запросы. К планируемым запросам относятся запросы, необходимость которых предусмотрена заранее. Планируемые запросы к базе данных обычно осуществляются из написанных заранее приложений, а непланируемые (специальные) запросы по определению производятся интерактивно.

– *Безопасность и целостность данных* достигается контролем пользовательских запросов и пресечением попыток нарушения правил безопасности и целостности данных, определяемых администратором БД.

– *Восстановление и дублирование данных* выполняется СУБД или другим программным компонентом, называемым администратором транзакций.

– *Словарь данных.* СУБД должна обеспечивать функцию словаря данных. Сам словарь данных является системой базы данных, содержащей информацию о хранимых данных, например, исходные и объектные схемы внешнего и концептуального уровня, перекрестные ссылки программ или частей БД, отчеты для различных пользователей и т.д.

– *Производительность.* Все перечисленные функции должны выполняться с максимально возможной эффективностью.

Следует отличать СУБД от ***системы управления файлами***, не учитывающей внутреннюю структуру хранимых данных (записей), имеющих особую поддержку безопасности и целостности данных. Запросы, основанные на знании структуры данных, здесь не обрабатываются.

Равноправным с СУБД программным компонентом можно считать ***систему управления передачей данных***. Применяется такая система при обработке запросов пользователей, физически удаленных от базы данных. Запросы и результат работы СУБД оформляются в виде коммуникационных сообщений. Наличие такой системы может быть отражено в определении БнД как системы баз данных и передачи данных.

Основные функции СУБД, рассмотренные выше, можно дополнить еще несколькими, входящими в уже известные.

Для функции производительности характерно применение функций *непосредственного управления данными во внешней памяти* и *управления буферами оперативной памяти*. Первая функция включает обеспечение необходимых структур внешней памяти как для хранения непосредственно данных, так и для служебных целей, например, для хранения индексов. Еще одним реальным способом увеличения скорости является буферизация данных в оперативной памяти, независимо от буферов ОС. Развитие СУБД поддерживает собственный набор буферов оперативной памяти с собственной дисциплиной их замены.

С функцией обработки данных тесно связано управление транзакциями. ***Транзакция*** – это последовательность операций над БД, рассматриваемых СУБД как единое целое. В процессе функционирования СУБД транзакция либо успешно выполняется и СУБД фиксирует (*commit*) изменения БД, произведенные ей во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. При помощи транзакций поддерживается логическая целостность базы данных за счет объединения элементарных операций над разными файлами в одну транзакцию. Например, при приеме нового сотрудника информация должна быть внесена в разные файлы "сотрудники" и "отделы" одной БД. Таким образом, поддержание механизма транзакций – обязательное условие однопользовательских, а тем более многопользовательских СУБД.

Функцию восстановления и дублирования данных в СУБД принято называть *журнализацией*. В результате аварийного выключения питания компьютера, сбоя в программе, выхода из строя носителя информации и других аварийных ситуаций может произойти потеря информации в базе данных. В первых двух случаях (мягкий сбой) восстановить данные можно ликвидацией последствий одной транзакции, в третьем (жесткий сбой) – восстановление данных возможно только копированием их из архива. Соблюдение надежности системы достигается избыточностью хранимых данных или ведением журнала изменений БД. ***Журнал*** – это особая часть БД, недоступная пользователям СУБД и поддерживаемая особенно тщательно, в которую поступают записи обо всех изменениях основной части БД.

Изменения базы данных журнализируются на разных уровнях, например, операции удаления строки из таблицы реляционной БД или операции модификации страницы внешней памяти. Самая простая ситуация восстановления – индивидуальный откат транзакции. В более сложных случаях применяют одновременно журнал и архивную копию. Тогда восстановление БД состоит в том, что исходя из архивной копии, по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя.

Для функции определения данных характерно наличие языкового процессора. В ранних СУБД поддерживалось несколько специализированных по функциям языков. Чаще всего выделялись два – язык описания данных (ЯОД) или схемы базы данных (*SDL – Schema Definition Language*) и язык манипулирования данными (ЯМД) (*DML – Data Manipulation Language*). В современных СУБД обычно поддерживается единый интегрированный язык. Примером такого языка может служить широко распространенный язык *SQL* (*Structured Query Language*).

Организация типичной СУБД и состав ее компонентов соответствуют рассмотренному выше набору функций. Логически в современной реляционной СУБД можно выделить внутреннюю часть – ядро СУБД, компилятор языка БД, подсистему поддержки времени выполнения и набор утилит (рис. 3.5). В некоторых системах эти части выполняются явно, в других нет, но логически такое разделение прослеживается во всех СУБД.

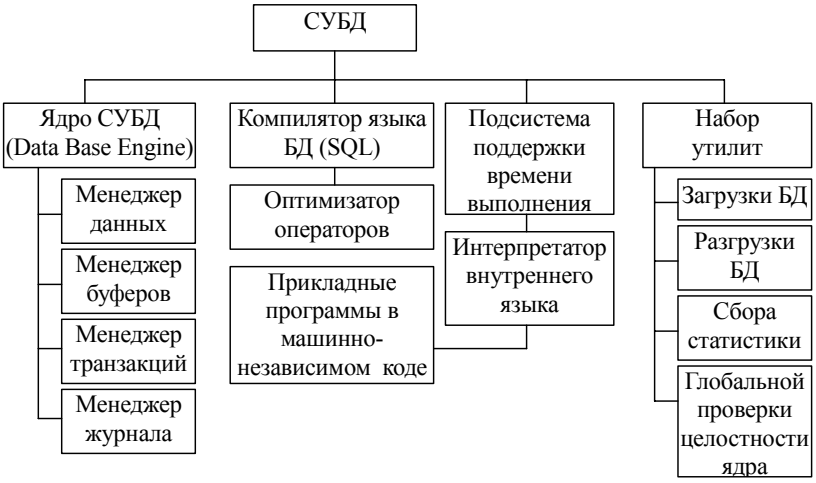


Рис. 3.5 Логическая структура СУБД

Ядро СУБД обладает собственным интерфейсом, не доступным пользователям напрямую. Этот интерфейс используется в программах, выполняемых компилятором языка SQL (или в подсистеме поддержки выполнения таких программ), и утилитах БД. Ядро СУБД является основной резидентной частью СУБД. Компилятор непроцедурного языка SQL решает проблему выполнения текущего оператора, его оптимизации и генерирования программного кода. Реальное выполнение оператора осуществляется подсистемой поддержки времени выполнения, представляющей собой интерпретатор внутреннего языка. Утилиты программируются с использованием интерфейса ядра СУБД или с проникновением внутрь ядра.

4 МОДЕЛИ ДАННЫХ

Традиционно СУБД делятся по типу моделей хранимых данных на ***иерархические***, ***сетевые*** и ***реляционные***. Такое деление моделей и СУБД основывается на характере связей между записями. При всей разнице в терминологии можно считать, что основными компонентами любой из этих моделей являются файлы, которые состоят из записей. Различают *внутризаписную* и *межзаписную* структуры.

В отличие от моделей внутризаписная структура может быть или линейной, или иерархической. При *линейной структуре* запись состоит из простых элементов, следующих один за другим. Такая структура считается нормализованной. *Иерархическая* структура включает простые и составные компо-

ненты, например векторы, повторяющиеся и неповторяющиеся группы. Иерархическая структура записи допускает многоуровневость (рис. 4.1).

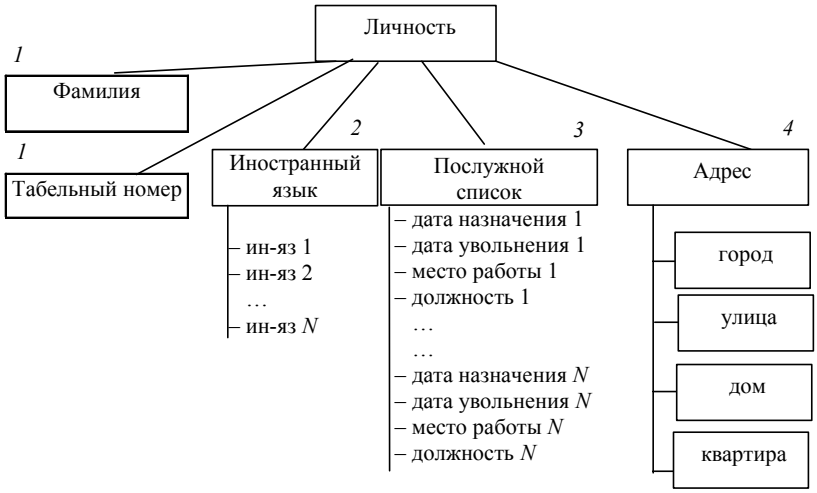


Рис. 4.1 Пример иерархической структуры записи:
1 – простой элемент; *2* – вектор (набор) однотипных элементов;
3 – повторяющаяся группа (набор разноплановых элементов);
4 – неповторяющаяся группа (набор разнотипных элементов)

Состав записей в структуре может быть постоянным или переменным. Например, если один из сотрудников окончил университет и имеет ученую степень и ученое звание с данными их присвоения, то другой сотрудник может их вообще не иметь. Это значит, что поля в соответствующих записях просто отсутствуют.

Основными характеристиками записи являются ее тип (символьный, числовой, дата, логический и т.д.) и длина (фиксированная, переменная и неопределенная).

Межзаписная структура или модель данных, как было уже отмечено, бывает иерархической, сетевой и реляционной. Рассмотрим их более подробно.

4.1 ИЕРАРХИЧЕСКАЯ И СЕТЕВАЯ МОДЕЛИ ДАННЫХ. ИХ ДОСТОИНСТВА И НЕДОСТАТКИ

В классических иерархических моделях имеется один файл, который является входом в структуру (корень дерева). Остальные файлы связаны друг с другом таким образом, что каждый из них за исключением корневой вершины имеет ровно одну исходную вершину ("предок") и любое число подчиненных вершин ("потомков"). Между записью файла – "предка" и записями порожденного файла имеется отношение "один ко многим" (как частный случай может быть отношение "один к одному"). Различают также тип связи "многие ко многим". Типичным представителем иерархических СУБД можно считать систему IMS (Information Management System).

Иерархическая БД состоит из упорядоченного набора деревьев, а точнее из упорядоченных наборов нескольких экземпляров одного типа дерева.

Пример схемы иерархической модели базы данных "Деталь" показан на рис. 4.2. Здесь "отдел" является предком ("родителем") для "начальник" и "сотрудники", а "начальник" и "сотрудники" – потомки "отдел".

Состав информации базы данных "Деталь":

1) для каждой детали: шифр детали (уникальный), название и краткое описание назначения детали, информация о технических характеристиках и наличии на складе;

2) характеристики детали включают необходимые технические данные (в примере – это вес детали, материал и ее габаритные размеры);

3) получение деталей на склад характеризуется датой получения, количеством деталей и уникальным номером накладной;

4) для каждой детали существует предприятие–изготовитель, имеющее почтовый адрес, название и шифр;

5) деталь может быть куплена через посредников, поэтому отдельно записывается название поставщика, его почтовый адрес, цена за одну деталь и шифр поставщика.

Таким образом, в базе данных есть пять типов сегментов: "Деталь", "Характеристики", "Приход", "Изготовитель", "Поставщик". Корневым сегментом является "Деталь". Остальные – подчиненные типы сегментов. Каждому подчиненному типу сегмента соответствует исходный тип сегмента. Каждому исходному типу сегмента будет соответствовать по крайней мере один порожденный тип сегмента.

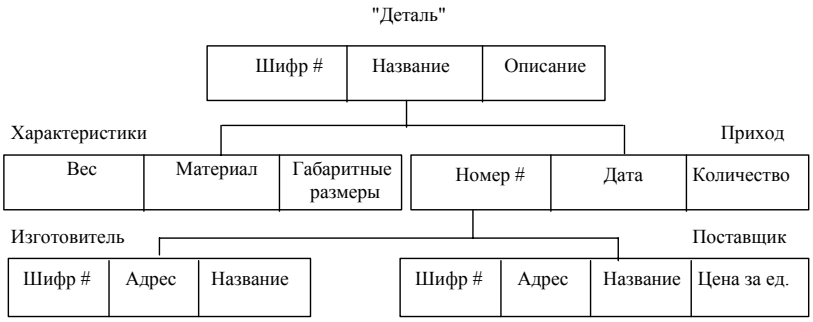


Рис. 4.2 Один экземпляр дерева иерархической БД

Для одного экземпляра любого заданного типа сегмента может существовать любое количество экземпляров, в том числе и нуль, каждого из его порожденных типов сегментов.

Например, имеется один экземпляр корневого типа сегмента "Деталь", один экземпляр "Характеристики", два экземпляра "Приход", один экземпляр "Изготовитель" и два экземпляра "Поставщик" (рис. 4.3). Первому экземпляру "Приход" подчинен один экземпляр "Изготовитель" и два экземпляра "Поставщик". Для второго экземпляра "Приход" пока неизвестны "Изготовитель" и "Поставщик" (деталь может быть собственного производства).

Иерархический порядок в БД считается очень важным, так как он определяет доступ к информации. Поиск в БД осуществляется по составному ключу, т.е. для выполнения запроса надо указывать значение ключа на каждом уровне иерархии. Значение ключа будет состоять из значения поля упорядочения данного сегмента с кодом типа сегмента в качестве префикса, которому предшествует значение ключа его исходного сегмента.

Например, для экземпляра сегмента "Поставщик", соответствующего адресу "Тамбов", значение ключа будет иметь вид: 1 D1 1 P500 2 M10.

Иерархический порядок определяется здесь возрастанием значений ключа иерархического упорядочения.

Примерами типичных операторов манипулирования иерархически организованными данными могут быть следующие:

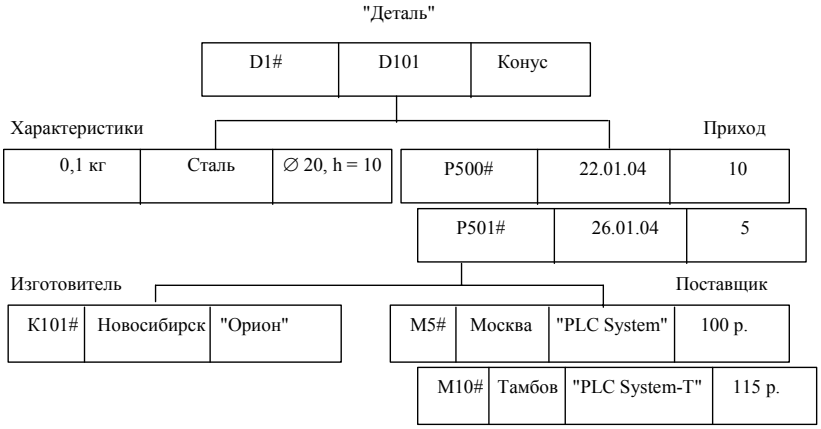


Рис. 4.3 Экземпляры дерева иерархической БД "Деталь"

1) найти указанное дерево БД (например, деталь D101);
2) перейти от одного дерева к другому;
3) перейти от одной записи к другой внутри дерева (например, от D101 к описанию "Конус");

4) перейти от одной записи к другой в порядке обхода иерархии;
5) вставить новую запись в указанную позицию;
6) удалить текущую запись.

Целостность ссылок между предками и потомками поддерживается автоматически. Основным правилом здесь является – *"никакой потомок не может существовать без своего родителя"*.

Следует заметить, что ссылки между записями различных деревьев не поддерживаются.

В сетевых моделях, если на нее не накладывается никаких ограничений, в принципе любой файл может быть точкой входа в систему. Каждый из файлов базы данных может быть связан с произвольными числами других файлов, и между записями связанных файлов могут быть любые отношения 1 : 1 ("один к одному"), 1 : M ("один ко многим"), M : M ("многие ко многим"). Однако, в реальных СУБД на модель могут накладываться различные ограничения.

Во многих сетевых СУБД не поддерживается отношение M : M. |В этих моделях каждая связь между парой файлов определяется отдельно, и для каждой из них один файл в этой паре объявляется "владельцем", а другой "членом" с отношением между записями 1 : M.

Связи между файлами в иерархических и сетевых моделях определяются при описании структуры БД и физически передаются при помощи различных указателей.

Типичным представителем является *Integrated Database Manegement System* (*IDMS*) компании *Cullinet software, inc*. Архитектура системы основана на предложениях *Data Base Task Group* (*DBTG*) комитета по языкам программирования *Counterence on Data Systems Languages* (*CODASYL*).

Сетевой подход к организации данных является расширением иерархического. В иерархических структурах запись-потомок должна иметь в точности одного предка; в сетевой структуре потомок может иметь любое число предков.

Сетевая БД состоит из набора записей и набора связей между ними, а точнее из набора экземпляров каждого типа из заданного в схеме базы данных набора типов записи и набора экземпляров каждого типа из заданного набора типов связи.

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка (рис. 4.4).

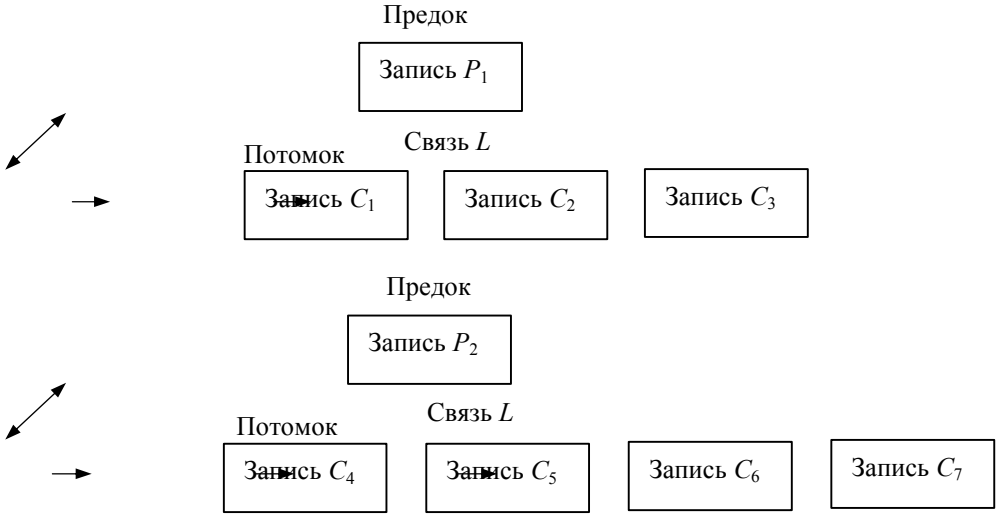


Рис. 4.4 Пример экземпляров набора

Для данного типа связи L с типом записи предка P и типом записи потомка C должны выполняться два условия: каждый экземпляр типа P является предком только в одном экземпляре L; каждый экземпляр C является потомком не более чем в одном экземпляре L.

На формирование типов связи не накладываются особые ограничения. Возможны, например следующие ситуации:

– тип записи потомка в одном типе связи L_1 может быть типом записи предка в другом типе связи L_2 (как в иерархии) (рис. 4.5, *а*);
– данный тип записи P может быть типом записи предка в любом числе типов связи (рис. 4.5, *б*);
– данный тип записи P может быть типом записи потомка в любом числе типов связи (рис. 4.5, *в*);

– может существовать любое число типов связи с одним и тем же типом записи предка и одним и тем же типом записи потомка; если L_1 и L_2 – два типа связи с одним и тем же типом записи предка P и одним и тем же типом записи потомка C, то правила, по которым образуется родство, в разных связях могут различаться (рис. 4.5, *г*);

– типы записи X и Y могут быть предком и потомком в одной связи и потомком и предком в другой (рис. 4.5, *д*);

– предок и потомок могут быть одного типа записи (рис. 4.5, *е*).

На рис. 4.6 показан простой пример сетевой схемы базы данных.

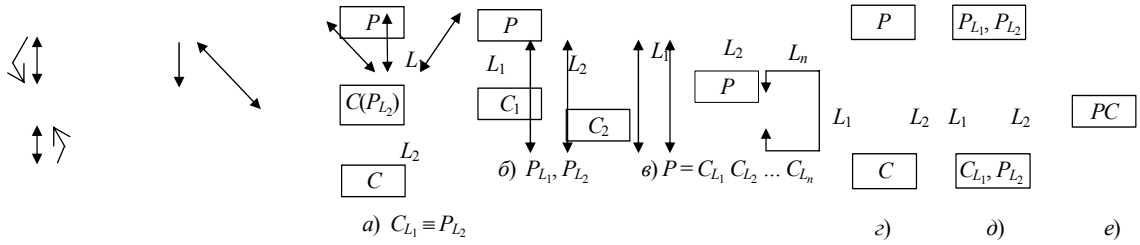


Рис. 4.5 Варианты типов связи

Рис. 4.6 Пример сетевой базы данных

Примерный набор выполняемых операций может быть таковым:

– найти конкретную запись в наборе однотипных записей, например, программиста Сидорова;
– перейти от предка к первому потомку (к первому сотруднику отдела САПР);
– перейти к следующему потомку в некоторой связи (от Сидорова к Иванову);
– перейти от потомка к предку по некоторой связи (найти отдел Сидорова);
– создать новую запись;
– удалить запись;
– модифицировать запись;
– включить связь;
– исключить из связи;
– переставить в другую связь и т.д.

К достоинствам сетевых и иерархических СУБД можно отнести следующее:

– развитые средства управления данными во внешней памяти на низком уровне;
– возможность построения эффективных прикладных систем вручную;
– возможность экономии памяти за счет разделения подобъектов (в сетевых системах).
– Недостатками являются:
– сложности в использовании;
– фактически необходимы знания о физической организации БД;
– прикладные системы зависят от организации БД;
– перегруженность логики деталями организации доступа.

4.2 РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

В реляционной модели данных используется своеобразная терминология, но это не меняет сущности модели. Так, на логическом уровне элемент чаще всего называют *атрибутом*; кроме того, для него используются термины *колонка*, *столбец*, *поле*. Совокупность атрибутов образует *кортеж* (ряд, запись, строку). Совокупность кортежей образует отношение (таблицу или файл БД).

Связи между файлами в реляционной модели в явном виде могут не описываться. Они устанавливаются динамически в момент обработки данных по равенству значений соответствующих полей. Структуры записей в реляционных базах данных – линейные.

Каждое отношение по определению имеет ключ, т.е. атрибут (простой ключ) или совокупность атрибутов (составной ключ), однозначно идентифицирующий кортеж.

Атрибут или группа атрибутов, которая в рассматриваемом отношении не является ключом, а в другом отношении ключом является, называется *внешним ключом*.

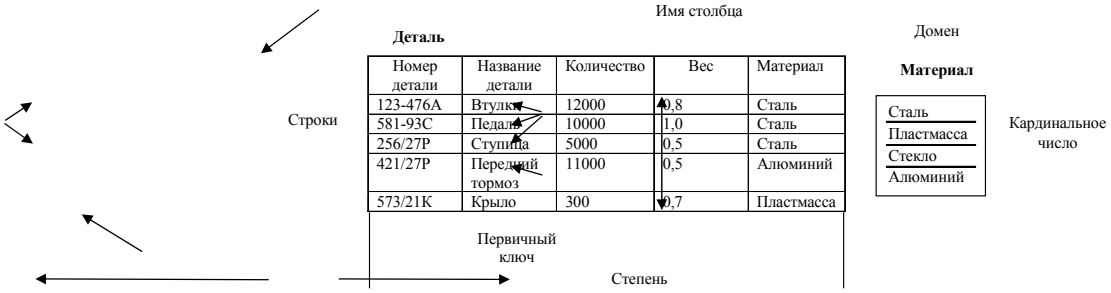
Если какая-то таблица содержит внешний ключ, то она:

а) логически связана с таблицей, содержащей соответствующий первичный ключ;

б) эта связь имеет характер один ко многим.

Реляционная модель была разработана Коддом в 1969 – 1970 гг. на основе математической теории отношений и опирается на систему понятий, важнейшими из которых являются таблица, отношение, строка, столбец, первичный ключ, внешний ключ.

Таким образом, реляционной считается такая модель данных, в которой все данные представлены для пользователя в виде прямоугольных таблиц значений данных, и все операции над базой данных сводятся к манипулированию таблицами. Таблица состоит из строк и столбцов и имеет имя, уникальное внутри базы данных. Таблица отражает тип объекта реального мира (сущность), а каждая ее строка – конкретный объект. Рассмотрим основные понятия реляционных моделей на примере таблицы "Деталь" (рис. 4.7).



Номер детали	Название детали	Количество	Вес	Материал
123-476А	Втулка	12000	0,8	Сталь
581-93С	Педаль	10000	1,0	Сталь
256/27Р	Ступица	5000	0,5	Сталь
421/27Р	Передний тормоз	11000	0,5	Алюминий
573/21К	Крыло	300	0,7	Пластмасса

Рис. 4.7 Основные понятия реляционной модели

Пусть в таблице "Деталь" содержатся сведения обо всех деталях, хранящихся на складе, а ее строки содержат набор значений атрибутов каждой конкретной детали.

Каждый столбец имеет имя, которое обычно записывается в верхней части таблицы. Оно должно быть уникальным в таблице, однако различные таблицы могут иметь столбцы с одинаковыми именами. Любая таблица должна иметь по крайней мере один столбец; столбцы расположены в таблице в соответствии с порядком следования их имен при ее создании. В отличие от столбцов (атрибутов), строки не имеют имен, порядок их следования не определен, а количество не ограничено.

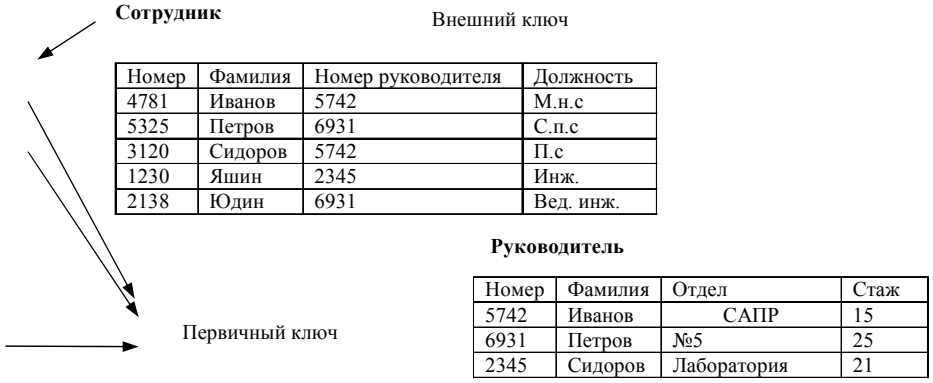
Любая таблица должна иметь один или несколько столбцов, значения которых однозначно идентифицируют каждую ее строку.

Первичный ключ в рассматриваемом примере (рис. 4.7) – это столбец "Номер детали".

Значения атрибутов выбираются из наименьшей информационной единицы – ***домена***. Другими словами, домен – это множество всех возможных значений атрибута объекта. Рассмотрим еще два понятия – ***степень*** и ***кардинальное число***. Под кардинальным числом отношения понимают количество кортежей, а степень отношения – это количество атрибутов данного отношения.

Взаимосвязь таблиц является важнейшим элементом реляционной модели данных. Она поддерживается внешними ключами. Рассмотрим пример, в котором база данных хранит информацию о сотрудниках (таблица "Сотрудник") и руководителях (таблица "Руководитель") в некоторой организации (рис. 4.8).

Первичный ключ таблицы "Руководитель" – столбец "Номер". Столбец "Фамилия" не является уникальным, поэтому не применяется в качестве первичного ключа. Столбец "Номер Руководителя" является внешним ключом в таблице "Сотрудник".



Номер	Фамилия	Номер руководителя	Должность
4781	Иванов	5742	М.н.с
5325	Петров	6931	С.п.с
3120	Сидоров	5742	П.с
1230	Яшин	2345	Инж.
2138	Юдин	6931	Вед. инж.

Номер	Фамилия	Отдел	Стаж
5742	Иванов	САПР	15
6931	Петров	№5	25
2345	Сидоров	Лаборатория	21

Рис. 4.8 Взаимосвязь таблиц

В базе данных дополнительно к самим данным должен храниться словарь данных и другие объекты, например, экранные формы, отчеты, просмотры (*views*) и прикладные программы.

Ограничения целостности в реляционных БД требуют, чтобы, например, значения атрибутов выбирались только из соответствующего домена, или внешний ключ не может быть указателем на несуществующую строку в таблице (целостность по ссылке).

Рассмотрим более подробно понятие *отношение*. В реляционных моделях следует различать *переменные отношений* и *значения отношений*. Переменная отношения – это обычная переменная, такая же, как и в языках программирования, т.е. именованный объект, значение которого может изменяться со временем, а значение этой переменной в любой момент времени и будет значением отношения. Если говорить о таблице, то точнее можно сказать *переменная базового отношения* (базовая таблица). Переменная отношения в разное время может представлять разные таблицы. В них будут разные строки, а столбцы будут одинаковыми.

Введем понятие *значение отношения*. Отношение R на множестве доменов $D_1, D_2, \ldots, D_n$ (не обязательно различных) содержит две части: *заголовок* и *тело*. В терминах таблиц, заголовок – это строка заголовков столбцов, а тело – это множество строк данных.

Заголовок содержит фиксированное множество атрибутов, а точнее пар <имя_атрибута, имя_домена> $\{A_1 : D_1, A_2 : D_2, \ldots, A_n : D_n\}$, причем каждый атрибут $A_j \in D_j \; \forall \; j = 1, \ldots, n$. Имена A_j при этом должны быть разными.

Тело содержит множество кортежей. Каждый кортеж содержит множество пар <имя атрибута; значение атрибута> $\{A_1 : Vi_1, A_2 : Vi_2, \ldots, A_n : Vi_n\} \;\; \forall \, i = 1, \ldots, m$; m – количество кортежей. В каждом кортеже есть пара $A_j : Vi_j \; \forall \; A_j$ в заголовке. Для любой пары A_j: Vi_j Vi_j является значением из уникального домена D_j, связанного с A_j.

Пусть R – переменная отношения. Переменная R будет иметь разные значения в разное время. Однако все возможные значения переменной R будут иметь одинаковые заголовки.

Можно выделить следующие свойства отношений:

- нет одинаковых кортежей;
- кортежи не упорядочены сверху вниз;
- атрибуты не упорядочены слева направо;
- все значения атрибутов атомарные.

Первое свойство следует из того факта, что тело отношения – это математическое множество кортежей, а множество не содержит одинаковых элементов. Это свойство показывает отличие отношения и таблицы. Следует заметить, что стандарт языка *SQL* допускает, чтобы таблицы содержали одинаковые строки.

Второе свойство следует из того, что тело отношения – это математическое множество, а простые множества не упорядочены. В каком бы порядке не были расположены кортежи, отношение останется тем же самым. Здесь тоже проявляется отличие от таблицы.

Третье свойство основано на понятии множества (атрибутов) и, следовательно, упорядочение не обязательно. С точки зрения модели данных невозможно различить первый, следующий или последний атрибуты.

Последнее свойство можно сформулировать еще и так: отношение не содержит групп повторения. В терминах таблицы это значит, что в каждой позиции пересечения строки и столбца располо-

жено только одно значение, а не набор значений. Такие отношения считают нормализованными, или представленными в первой нормальной форме (1НФ).

Например, в различных системах встречаются некоторые из видов отношений, которые перечислены ниже.

1 Именованное отношение – это переменная отношения, определенная в СУБД посредством операторов открытия или создания отношения.

2 Базовое отношение – это наиболее важное, автономное именованное отношение, являющейся частью базы данных.

3 Произвольное отношение определяется через другие именованные отношения, и, в конечном счете, через базовые отношения.

4 Выражаемое отношение получается из набора именованных отношений через некоторые реляционные выражения (результат отчетов).

5 Представлением (просмотром) называется именованное производное отношение. Представления виртуальны и представлены в системе через определения в терминах других именованных отношений.

6 Снимки (*shnapshot*) – это именованные производные отношения, как и представления, но реальны в отличие от последних. Создание снимка похоже на выполнение запроса, результат которого сохраняется в базе данных.

7 Результат запроса – именованное производное отношение, полученное в результате некоторого определенного запроса.

8 Промежуточным результатом называется именованное производное отношение, являющееся результатом некоторого реляционного выражения, вложенного в другое, большее выражение.

9 Хранимым называется отношение, которое поддерживается в физической памяти. Хранимое отношение не всегда совпадает с базовым.

Каждое отношение имеет некоторую интерпретацию, причем пользователи должны знать ее для эффективного использования БД. Например, интерпретация отношения "Деталь" может быть следующей: деталь с определенным номером (Номер_детали) имеет определенное имя (Название_детали), имеется на складе в количестве (Кол-во_детали) весом (Вес) килограмм каждая и выполненная из (Материал); кроме того нет двух деталей с одинаковыми номерами.

Это утверждение называется ***предикатом***, или функцией значения истинности, в нашем примере – функцией пяти аргументов. Подстановка значений аргументов приводит к получению выражения, имеющего истинное либо ложное утверждение. Операции вставки новых кортежей и обновления существующих выполняются в случае истинности предиката для данного кортежа, т.е. при соблюдении правил целостности.

К реляционным базам данных применяются два общих правила целостности. Относятся они к потенциальным (первичным) и к внешним ключам. Если говорить нестрого, то первичный ключ – это уникальный идентификатор для некоторого отношения. Однако первичный ключ является частным случаем общего понятия – ***потенциального ключа.*** Рассмотрим это понятие. Пусть R – некоторое отношение. Тогда потенциальный ключ K для R – это подмножество множества атрибутов R, обладающих следующими свойствами:

– свойством уникальности, нет двух различных кортежей в отношении R с одинаковым значением K;

– свойством неизбыточности, никакое из подмножеств K не обладает свойством уникальности.

Данное определение относится к значениям отношения, а не к переменным отношения. Для переменных отношения определение потенциального ключа дополняется следующим образом. Пусть R – некоторая переменная отношения, тогда потенциальный ключ K для R – это подмножество множества атрибутов R, всегда обладающее свойствами уникальности и неизбыточности. Свойство уникальности рассматривается для различных кортежей в текущем значении переменной R.

На практике отношения чаще всего имеют только один потенциальный ключ, хотя их может быть и несколько. Например, в периодической системе элементов химические элементы имеют уникальное имя, обозначение (Cu, Pb, Au, …) и атомное число. Это уже три различных потенциальных ключа, или составной потенциальный ключ, состоящий более чем из одного атрибута. Потенциальные ключи не должны включать лишних атрибутов для идентификации уникальности. Это и есть свойство неизбыточности.

Если в отношении "Деталь" определить потенциальный ключ как комбинацию {Номер_детали, Материал} вместо ключа "Номер_детали", то система не сможет соблюдать ограничение, обеспечивающее уникальность в "локальном смысле", т.е. для одного типа материала.

На практике физическое понятие индекса часто играет роль потенциального ключа.

Причина важности потенциальных ключей состоит в том, что они обеспечивают *основной механизм адресации на уровне кортежей.* Другими словами, единственный гарантируемый системой способ точно указать кортеж – это указать значение некоторого потенциального ключа.

Таким образом, базовое отношение может иметь больше одного потенциального ключа. В реляционной модели один из потенциальных ключей выбирают в качестве первичного ключа. Если есть еще потенциальные ключи в этом базовом отношении, то их считают альтернативными (обозначение элемента и название элемента, атомное число для примера периодической системы элементов).

Реляционная модель традиционно требует, чтобы внешние ключи в точности соответствовали первичным ключам, а не просто потенциальным ключам.

Уточним определение внешнего ключа. Пусть R_2 – базовое отношение. Тогда внешний ключ *FK* в отношении R_2 – это подмножество множества атрибутов R_2, такое, что:

– существует базовое отношение R_1 с потенциальным ключом *CK*;

– каждое значение *FK* в текущем значении R_2 всегда совпадает со значением *CK* некоторого кортежа в текущем значении R_1.

Внешний ключ может быть составным тогда и только тогда, когда соответствующий потенциальный ключ также составной. Аналогично определяется соответствие для простого ключа. Значение внешнего ключа представлено ссылкой к кортежу с соответствующим потенциальным ключом. Для баз данных иногда строят ссылочные (целевые) диаграммы. Например, если объединить отношения "Сотрудник" (С) и "Руководитель" (Р) в базу. "Предприятие" (П), то можно записать диаграмму С $\leftarrow$ П $\rightarrow$ Р, где стрелка обозначает внешний ключ. Иногда над ссылкой указывают имя атрибута $\text{С} \xleftarrow{\text{ном. сотр.}} \text{П} \xrightarrow{\text{ном. рук.}} \text{Р}$.

Отношение может быть одновременно ссылочным и ссылающимся, например для R_2: $R_3 \rightarrow R_2 \rightarrow R_1$.

Пусть в отношении R_n, R_{n-1}, …, R_2, R_1 имеется ссылочное ограничение из R_n в R_{n-1} и R_{n-1} в R_{n-1} и … и R_2 в R_1 или $R_n \rightarrow R_{n-1} \rightarrow R_{n-2} \rightarrow \ldots \rightarrow R_2 \rightarrow R_1$. Тогда цепочки стрелок из R_n в R_1 представляют *ссылочный путь* из R_n в R_1.

Отношения R_1 и R_2 в определении внешних ключей не обязательно различны. Такие самоссылающиеся отношения представляют собой отдельный случай. Отношения R_n, R_{n-1}, …, R_2, R_1 образуют *ссылочный цикл* ($R_n \rightarrow \ldots R_1 \rightarrow R_n$).

Вместе с понятием внешнего ключа реляционная модель включает правило *ссылочной целостности*, т.е. база данных не должна иметь несогласованных ключей. Если $R_i \rightarrow R_j$ (ссылается), то R_j должно существовать.

Для корректного применения внешних ключей существуют некоторые правила. Что будет происходить при удалении или обновлении объекта ссылки внешнего ключа? Если использовать правила *ограничения* и *каскадирования,* то целостность базы данных не нарушается.

Свойство ограничения заключается в том, что операция (удаления или добавления) выполняется до момента, пока не будет существовать соответствующих ссылок кортежей. Свойство каскадирования состоит в том, что операция выполняется столько раз, сколько кортежей будет обнаружено.

Пусть R_1 и R_2 имеют ссылочное отношение $R_2 \rightarrow R_1$. Тогда удаление кортежа из R_1 влечет удаление определенных кортежей в R_2.

Пусть имеем ссылочное отношение $R_3 \rightarrow R_2 \rightarrow R_1$. Тогда, если операция удаления из R_2 выполняется только для R_2, то нарушается ссылка $R_3 \rightarrow ? \rightarrow R_1$. Следовательно, такая операция не должна быть выполнена.

В реляционных моделях существует еще один фактор, связанный с потенциальными ключами – это нуль значения (*NULL*). Когда говорят о *null*-значении, то в основном подразумевают базис, используемый при решении проблемы отсутствующей информации. Эта проблема почти не имеет математической проработки.

5 РЕЛЯЦИОННАЯ АЛГЕБРА

5.1 ОСНОВЫ РЕЛЯЦИОННОЙ АЛГЕБРЫ

Реляционная алгебра, определенная Коддом, состоит из восьми операторов, составляющих две группы.

В первую группу входят традиционные операции над множествами: объединение ($\cup$), пересечение ($\cap$), вычитание (–) и декартово произведение (*). Все операции модифицированы с учетом того, что их операндами являются отношения, а не произвольные множества.

Вторую группу образуют специальные реляционные операции: выборка, проекция, соединение и деление.

Рассмотрим подробные результаты этих операций над отношениями.

Объединение ∪. Возвращает отношение, содержащее все кортежи, которые принадлежат одному из двух определенных отношений, или обоим (рис. 5.1, *а*).

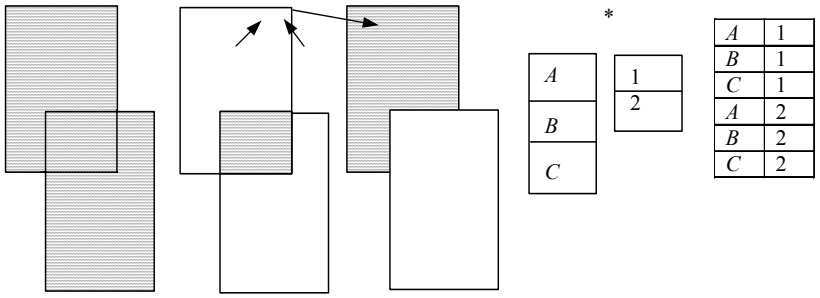
Пересечение ∩. Возвращает отношение, содержащее все кортежи, которые принадлежат одновременно двум определенным отношениям (рис. 5.1, *б*).

Вычитание –. Возвращает отношение, содержащее все кортежи, которые принадлежат первому из двух определенных отношений и не принадлежат второму (рис. 5.1, *в*).

Декартово произведение **. Возвращает отношение, содержащее всевозможные кортежи, которые являются сочетанием двух кортежей, принадлежащих соответственно двум определенным отношениям
(рис. 5.1, *г*).

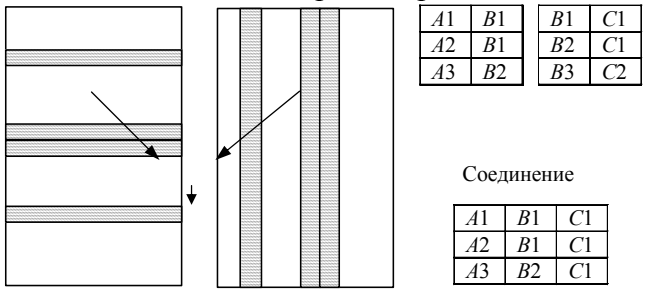
Выборка – возвращает отношение, содержащее все кортежи из определенного отношения, которое удовлетворяет определенным условиям. С точки зрения алгебраических операций это ограничение (рис. 5.2, *а*).

Проекция – возвращает отношение, содержащее все кортежи (подкортежи) определенного отношения после исключения из него некоторых атрибутов (рис. 5.2, *б*).



а) *б*) *в*) *г*)

Рис. 5.1 Операции над множествами:
а – объединение; *б* – пересечение;
в – вычитание; *г* – декартово произведение



а) *б*) *в*)

Рис. 5.2 Специальные реляционные отношения:
а – выборка; *б* – проекция; *в* – соединение

Соединение – возвращает отношение, кортежи которого – это сочетания двух кортежей (принадлежащих соответственно двум определенным), имеющих общее значение для одного или нескольких общих атрибутов этих двух отношений. Общие значения в результирующем кортеже появляются только один раз. Такое соединение называют естественным соединением (рис. 5.2, *в*).

Деление – для двух отношений бинарного и унарного, возвращает отношение, содержащее все значения одного атрибута бинарного отношения, которые соответствуют всем значениям в унарном отношении (рис. 5.3).

Результат каждой операции над отношением также является отношением. Это реляционное свойство называется свойством *замкнутости.*

Результат одной операции может использоваться в качестве исходных данных для другой. Следовательно, существует возможность, например, взять проекцию от объединения или соединение от двух выборок и т.д. Такие выражения считаются вложенными.

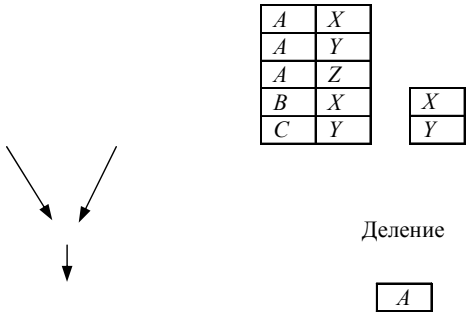


Рис. 5.3 Деление

Каждое отношение включает заголовок, тело, множество потенциальных ключей. При выполнении реляционных операций необходимо предусмотреть набор правил наследования имен атрибутов и потенциальных ключей.

Для операций объединения (*union*), пересечения (*intersect*) и вычитания (*minus*) должны выполняться два свойства:

- операнды должны иметь одну и ту же степень;
- соответствующие атрибуты должны быть определены на одном и том же домене.

Операция умножения не требует выполнения этих условий.

Рассмотрим выполнение операций над отношениями подробнее.

Объединением двух совместимых по типу отношений *A* и *B* (*A union B*) называется отношение *C* с тем же заголовком и телом, состоящим из множества кортежей *t*, принадлежащих *A* или *B* или обоим отношениям.

$$C = (A \text{ union } B) \mid t_i \in C \; \forall \; t_j \in A \; \& \; t_i \in C \; \forall \; t_j \in B.$$

Пример. Пусть отношения *A* и *B* будут такими, как они отражены ниже: *A* – детали изготовленные из стали; *B* – детали весом больше 0,5 кг. Тогда *A union B* представляет детали, которые *или* изготовлены из стали, *или* имеют вес больше 0,5 кг.

A

	Название детали		Материал
	*D*1		Сталь
	*D*2		Сталь
	*D*3		Сталь

B

	Название детали		Материал
	*D*1		Сталь
	*D*2		Сталь
	*D*4		Алюми-

							ний

В результате получим четыре кортежа, а не шесть, т.к. повторяющиеся значения удаляются.

C

K	Название детали	Вес	Материал
*K*1	*D*1	0,8	Сталь
*K*2	*D*2	1,0	Сталь
*K*3	*D*3	0,5	Сталь
*K*4	*D*4	0,7	Алюминий

Пересечением двух совместимых по типу отношений *A* и *B* (*A intersect B*) называется отношение с тем же заголовком и телом, состоящим из множества кортежей *t*, принадлежащих одновременно обоим отношениям *A* и *B*.

$$C = (A \; intersect \; B) \mid \forall \; t_i \in C \mid t_i \in A \; \& \; t_i \in B.$$

Пример. *A intersect B* для нашего примера представляет детали, изготовленные из стали и весом более 0,5 кг.

C

K	Название детали	Вес	Материал
*K*1	*D*1	0,8	Сталь
*K*2	*D*2	1,0	Сталь

Вычитанием двух совместимых по типу отношений *A* и *B* (*A minus B*) называется отношение с тем же заголовком и телом, состоящим из множества кортежей *t*, принадлежащих отношению *A* и не принадлежащих отношению *B*.

$$C = (A \; minus \; B) \mid \forall \; t_i \in C \mid t_i \in A \; \& \; t_i \notin B.$$

Пример. Выражение (*A minus B*) представляет детали, которые изготовлены из стали и весят не более 0,5 кг.

C = (*A minus B*)

K	Название детали	Вес	Материал
*K*3	*D*3	0,5	Сталь

C = (*B minus A*)

K	Название детали	Вес	Материал
*K*4	*D*4	0,7	Алюминий

Выражение (*B minus A*) представляет детали, которые не изготовлены из стали и весят больше 0,5 кг.

Следует заметить, что операция вычитания учитывает порядок основания операндов: *C* = (*A minus B*) не одно и то же, что *C* = = (*B minus A*).

Декартово произведение двух отношений есть множество упорядоченных пар кортежей, сохраняющих свойство замкнутости. Декартово произведение двух отношений *A* и *B* (*A times B*), где *A* и *B* не имеют общих имен атрибутов, определяется как отношение с заголовком, который представляет собой сцепление двух заголовков исходных отношений *A* и *B* и телом, состоящим из множества всех кортежей *t*, таких, что *t* представляет собой сцепление кортежа *a*, принадлежащего отношению *A* и кортежа *b*, принадлежащего отношению *B*.

$$C = (A \; times \; B) \mid \forall \; a_i \in A \; \& \; \forall \; b_i \in B \; \& \; a_i \neq b_i \; \exists \; t \mid t = a \cup b.$$

Кардинальное число результата равняется произведению кардинальных чисел исходных отношений *A* и *B*, а степень – сумме их степеней.

Пример. Пусть отношение *A* представляет детали, а отношение *B* – предприятия изготовители.

B

Изготовитель	Город	Телефон	Адрес
*P*1	..	…	..
*P*2	..	…	..
*P*3	..	…	..

Сокращенно представим таблицу *A* как столбец *K* = *K*1, *K*2, *K*3, *K*4, *K*5, а таблицу *B* как *P* = *P*1, *P*2, *P*3. Тогда *A times B* – это все пары деталь-изготовитель, изготовитель-деталь. Таблица *C* будет иметь пары *C* = {*K*1, *P*1; *K*1, *P*2; *K*1, *P*3; *K*2, *P*1; *K*2, *P*2; *K*2, *P*3; *K*3, *P*1; *K*3, *P*2; *K*3, *P*3; *K*4, *P*1; *K*4, *P*2; *K*4, *P*3; *K*5, *P*1; *K*5, *P*2; *K*5, *P*3}. Результат не говорит нам ничего нового, он просто подтверждает, что существуют номера деталей и предприятий-изготовителей.

Операция декартова произведения не очень важна на практике, за исключением операции Θ – соединения.

Операции объединения, пересечения и декартова произведения обладают свойствами:

– ассоциативности (*A union B*) *union C* ~ *A union* (*B union C*) ⇒ *A union B union C*;

– коммуникативности *A union B* ~ *B union A*, *A intersect B* ~ *B intersect A*, *A times B* ~ *B times A* (но не в теории множеств и 1 и 2 свойства не выполняются).

Указанные свойства не выполняются для операции вычитания.

5.2 СПЕЦИАЛЬНЫЕ РЕЛЯЦИОННЫЕ ОПЕРАЦИИ

Операция выборки (ограничение) представляет собой сокращенное название так называемой Θ-выборки, где Θ обозначают любой скалярный оператор сравнения (=, ≠, ≥ и т.д.). Θ-выборкой из отношения *A* по атрибутам *X* и *Y*: *A where X* Θ *Y* называется отношение, имеющее тот же заголовок, что и отношение *A* и тело, содержащее множество всех кортежей *t* отношения *A*, для которых условие *X* Θ *Y* имеет значение "истина". Атрибуты *X* и *Y* должны быть определены на одном и том же домене, а оператор должен иметь смысл для этого домена.

Пример. Получить список деталей весом от 1 кг и выше для отношения *A*.

A where Вес >= 1,0.

A

K	Название детали	Вес	Материал
*K*2	*D*2	1,0	Сталь

Операция сравнения может проводиться для символьных и строковых переменных (=, ≠). В качестве действия над атрибутами используют и логические операции *AND*, *OR*, *NOT*.

Пусть есть отношение *P* (таблица поставщиков деталей).

P

N_пост	Название завода	Город	Улица	Номер дома	Телефон

*P*1	Протон	Москва	–	–	–
*P*2	–	СПб.	–	–	–
*P*3	–	Зелено-град	–	–	–
*P*4	–	Москва	–	–	–
*P*5	–	Тамбов	–	–	–

Тогда очередную выборку можно провести для поиска, например, всех поставщиков из города "Москва":

P where Город = "Москва" или

P where Город = "Тамбов" *AND* улица = "Ленинградская".

На основании свойства замкнутости можно расширить условие в выражении *where* до произвольного числа логических сочетаний или простых сравнений, применяя следующие тождества:

1) *A where X and Y* ≡ (*A where X*) *intersect* (*A where Y*);
2) *A where X or Y* ≡ (*A where X*) *union* (*A where Y*);
3) *A where not X* ≡ *A minus* (*A where X*).

Проекцией отношения *A* по атрибутам *X*, *Y*, ..., *Z* где каждый из атрибутов принадлежит отношению *A* (*A*[*X*, *Y*, …, *Z*]), называется отношение с заголовками {*X*, *Y*, ..., *Z*} и телом, содержащим множество всех кортежей {*X* : *x*, *Y* : *y*, …, *Z* : *z*} таких, что в отношении *A* значение атрибута *X* равно *x*, атрибута *Y* равно *y*, атрибута *Z* равно *z*. Результат операции проекции – подмножество указанных в списке атрибутов из множества имеющихся атрибутов с последующим исключением дублирующих кортежей.

Операция проекции допускает тождественную *R* и нулевую *R*[ ] проекцию. В первом случае результат – то же отношение, во втором нет ни одного кортежа.

Например, необходимо найти проекцию отношения *P* по атрибуту "Город":

Город
Москва
СПб.
Зелено-град
Москва
Тамбов

или (*P where* Город = "Москва")

N_пост
*P*1
*P*4

Операция соединения имеет несколько вариантов: это наиболее важное *естественное* соединение и Θ-соединение. Для обозначения естественного соединения применим термин *join*. Пусть отношения *A* и *B* имеют заголовки {X_1, X_2, …, X_m, Y_1, Y_2, …, Y_n} и {Y_1, Y_2, …, Y_n, Z_1, Z_2, …, Z_p} соответственно. Предположим, что соответствующие атрибуты (с одинаковыми именами) определены на одном и том же домене. Рассмотрим выражения {X_1, X_2, …, X_m}, {Y_1, Y_2, …, Y_n} и {Z_1, Z_2, …, Z_p} как три составных атрибута *X*, *Y*, *Z*. Тогда естественным соединением отношений *A* и *B* (*A join B*) называется отношение с заголовком {*X*, *Y*, *Z*}и телом, содержащим множество всех кортежей {*X* : *x*, *Y* : *y*, *Z* : *z*}, таких, что в отношении *A* значение атрибута *X* равно *x*, а атрибута *Y* равно *y*, и в отношении *B* значение атрибута *Y* равно *y*, а атрибута *Z* равно *z*.

Пример. Пусть имеем таблицу деталей *C* и таблицу поставщиков *P* (см. выше).

C

K	Название детали	Вес	Материал	Город
*K*1	*D*1	0,8	Сталь	Москва
*K*2	*D*2	1,0	Сталь	Москва
*K*3	*D*3	0,5	Сталь	Пенза
*K*4	*D*4	0,7	Алюминий	Липецк

C JOIN P

	Название детали		Материал	Город	N – пост	Название завода	Улица	Дом	Телефон
			Сталь	Москва	*P*1				
			Сталь	Москва	*P*4				
			Сталь	Москва	*P*1				
			Сталь	Москва	*P*4				

Естественное соединение обладает свойствами ассоциативности и коммутативности:

$$(A \text{ join } B) \text{ join } C \sim A \text{ join } (B \text{ join } C) \sim A \text{ join } B \text{ join } C \text{ и } A \text{ join } B \sim B \text{ join } A.$$

Если *A* и *B* не имеют общих имен атрибутов, то $A \text{ join } B \equiv A \text{ times } B$ (соед. $\xrightarrow{\text{в}}$ произвед.).

Θ-соединение. Эта операция используется в более редких случаях, когда надо соединить два отношения на основе некоторых условий, отличных от эквивалентности. Θ-соединение эквивалентно двум операциям: нахождению декартова произведения от двух отношений и последующему выполнению указанной выборки из полученного результата. Тогда Θ-соединением отношения *A* по атрибуту *X* с отношением *B* по атрибуту *Y* называется результат вычисления выражения

(*A times B*) *where X*Θ*Y*.

Результатом является отношение с тем же заголовком, что при декартовом произведении отношений *A* и *B* и с телом, содержащим множество кортежей *t* таких, что *t* принадлежит этому декартову произведению и выполнение условия $X\ \Theta\ Y$ дает значение "истина" для этого кортежа.

Деление. Пусть отношения *A* и *B* имеют заголовки $\{X_1, X_2, \ldots, X_m, Y_1, Y_2, \ldots, Y_n\}$ и $\{Y_1, Y_2, \ldots, Y_n\}$ соответственно. Пусть соответствующие атрибуты определены на одном и том же домене. Пусть *X* и *Y* – два составных атрибута, где $X = \{X_1, X_2, \ldots, X_m\}$, $Y = \{Y_1, Y_2, \ldots, Y_n\}$. Тогда делением отношений *A* и *B* (*A divide by*) называется отношение с заголовком *X* и телом, содержащим множество всех кортежей $\{X : x\}$ таких, что существует кортеж $\{X : x,\ Y : y\}$, который принадлежит отношению *A* для всех кортежей $\{Y : y\}$, принадлежащих отношению *B*. Нестрого это можно сформулировать так: результат содержит такие *X*-значения из отношения *A*, для которых соответствующие *Y*-значения из *A* включают все *Y*-значения из отношения *B*. Пользоваться этой операцией следует осторожно, так как не исключено возникновение пустых отношений.

Пример операции деления. Пусть есть отношение исходное *AP* и делители B_i *для* $i = 1, 2, 3$.

AP

K	*P*	*K*	*P*
*K*1	*P*1	K2	*P*1
*K*1	*P*2	K2	*P*2
*K*1	*P*3	K3	*P*2
*K*1	*P*4	K4	*P*3
*K*1	*P*5	K4	*P*4
*K*1	*P*6	K4	*P*5

B_1

P
*P*1

B_2

P
*P*2
*P*4

B_3

P
*P*1
*P*2
*P*3
*P*4
*P*5
*P*6

*AP divide by B*1

K
*K*1
*K*2

*AP divide by B*2

K
*K*1
*K*4

Получить поставщиков, поставляющих "все" детали *AP divide by B*3

K
*K*1

6 ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

6.1 НОРМАЛЬНЫЕ ФОРМЫ

Отношение находится в некоторой ***нормальной форме***, если удовлетворяет некоторому заданному условию. Отношение находится в первой нормальной форме (1НФ), тогда и только тогда, когда оно содержит только скалярные значения.

Коддом были определены три нормальных формы (1НФ, 2НФ, 3НФ). Все нормализованные отношения входят в 1НФ, во 2НФ, в 3НФ и т.д. Вторая НФ более желательна, чем первая, третья – чем вторая и т.д. При проектировании баз данных следует использовать отношения не только в 1НФ, но и во 2НФ, и в 3НФ. При помощи определенной последовательности действий отношения преобразуются из одной нормальной формы в другую.

Бойсом и Коддом переработана 3НФ и в более строгом смысле она называется нормальной формой Бойса-Кодда (НФБК). В ней устранены некоторые неадекватности, возможные в 3НФ.

Фейгином одновременно с введением НФБК была определена 4НФ, они практически одинаковы. Далее Фейгин предложил 5НФ или проективно-соединительную НФ (рис. 6.1).

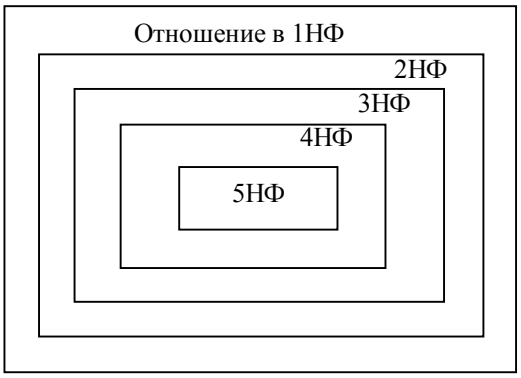


Рис. 6.1 Нормальные формы

6.2 ДЕКОМПОЗИЦИЯ БЕЗ ПОТЕРЬ ФУНКЦИОНАЛЬНОЙ ЗАВИСИМОСТИ

Процедура нормализации включает декомпозицию данного отношения на другие отношения. Декомпозиция должна быть обратимой.

Пример декомпозиции отношения типа {код, имя_детали, город} приведен ниже (рис. 6.2).

Декомпозиция проводилась с использованием теоремы Хеза, которая гласит: пусть $R\{A, B, C\}$ есть отношение, где A, B, C – атрибуты этого отношения. Если R удовлетворяет зависимости $A \rightarrow B$, то R равно соединению его проекций $\{A, B\}$ и $\{A, C\}$.

Рис. 6.2 Пример декомпозиции

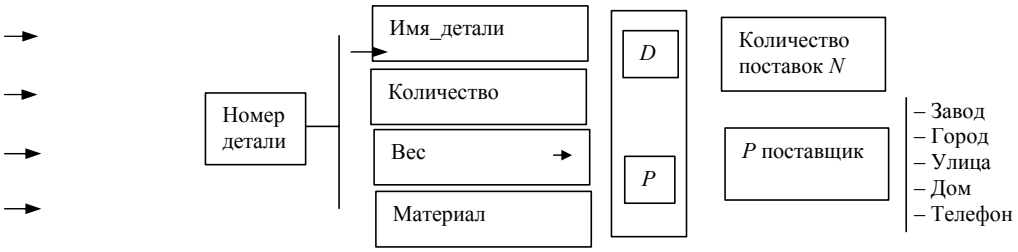


Рис. 6.3 Схема функциональной зависимости

Стрелкой ($\rightarrow$) в данном случае показана некоторая функциональная зависимость. Важную роль в понятии функциональных зависимостей (ФЗ) играет неприводимая слева ФЗ, т.е. левая часть зависимости не должна быть избыточной. Например ФЗ {код_детали, код_города, город} может быть записана без атрибута код_города, {код_детали} $\rightarrow$ город. Последняя ФЗ является неприводимой слева. ФЗ могут изображаться в виде диаграмм (схем).

Диаграмма функциональной зависимости строится для отношения и некоторого неприводимого множества зависимостей для этого отношения. Например, отношение "Деталь" имеет схему, представленную на рис. 6.3.

ФЗ – это особый вид ограничений целостности и, несомненно, семантическое понятие. Распознавание ФЗ является частью процесса выяснения смысла тех или иных данных. Например, Ном_дет $\rightarrow$ {название детали, количество, вес, материал} означает, что каждая деталь имеет свой код, который точно определяет ее название, количество, вес и материал.

6.3 НОРМАЛИЗАЦИЯ ОТНОШЕНИЙ

Одна из целей проектирования баз данных состоит в получении НФБК и форм более высокого порядка. Формы 1НФ, 2НФ и 3НФ представляют собой промежуточные результаты.

На примере вышеприведенной схемы введено добавочное отношение количества поставок N детали D_i поставщиком P_i; первичный ключ – комбинация $\{D, P\}$.

Рассмотрим формы отношений.

- Отношение находится в первой нормальной форме (1НФ) тогда и только тогда, когда все используемые домены содержат только скалярные значения.

Пусть имеем объединенное отношение деталей, поставщиков и количества поставок DP (D, имя_детали, количество, вес, материал, P, количество_поставок, завод, город, улица, дом, телефон). Дополнительное ограничение имя_детали $\xrightarrow{\text{ФЗ}}$ кол-во. Первичный ключ для DP это (D, P). Диаграмма ФЗ имеет вид, представленный на рис. 6.4.

Отношение избыточно. Например, в нем не может быть детали или поставщика, который был ранее известен, а в последнее время не сделал ни одной поставки и т.д., или количество деталей равно нулю.

- Отношение находится во второй нормальной форме (2НФ) тогда и только тогда, когда оно находится в 1НФ и каждый неключевой атрибут неприводимо зависит от первичного ключа.

В примере имеем ранее известную диаграмму ФЗ (рис. 6.5).

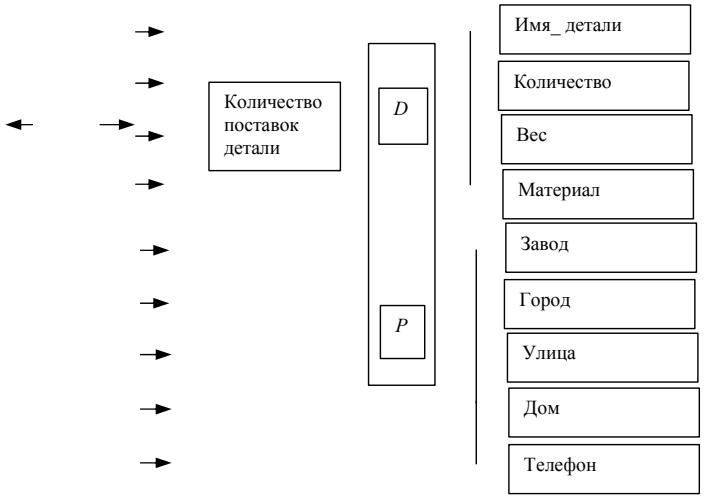


Рис. 6.4 Диаграмма функциональной зависимости в 1НФ

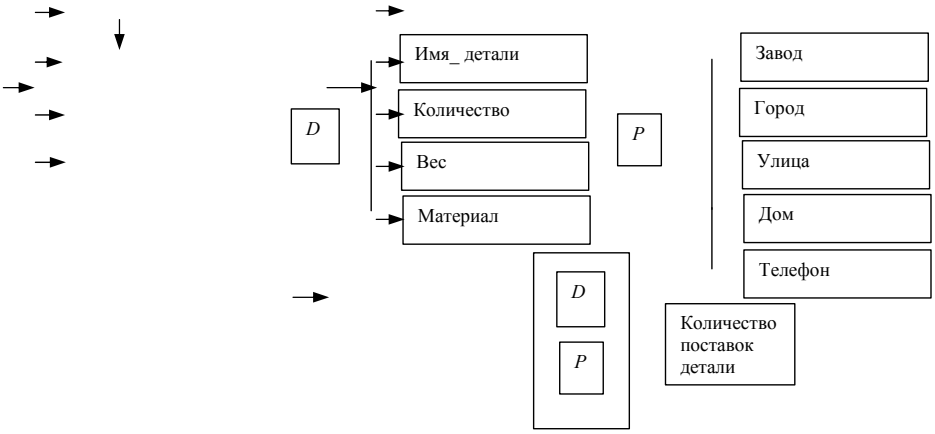


Рис. 6.5 Диаграмма функциональной зависимости в 2НФ

Первичные ключи – D, P, $\{D, P\}$ и три отношения.

• Отношение находится в третьей нормальной форме (3НФ) тогда и только тогда, когда оно находится во второй нормальной форме и каждый неключевой атрибут нетранзитивно (отсутствие какой либо зависимости) зависит от первичного ключа.

ФЗ для отношения "Детали" представлена на рис. 6.6.

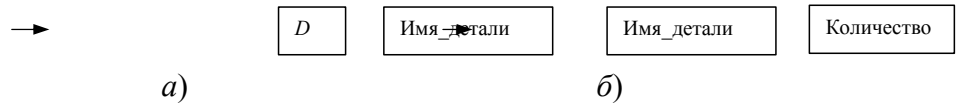


Рис. 6.6 Диаграмма функциональной зависимости в 3НФ

Отношения *А* и *Б* – в 3НФ, первичные ключи *D* и "Имя_детали".

Следовательно, мы имеем следующие этапы нормализации:

– создание проекций для исключения "приводимых" ФЗ;
– создание проекций для исключения транзитивных ФЗ.

• Отношение находится в нормальной форме Бойса-Кодда тогда и только тогда, когда каждая нетривиальная и неприводимая слева ФЗ обладает потенциальным ключом в качестве детерминанта (левая часть ФЗ).

На диаграмме стрелки ФЗ начинаются только с потенциальных ключей. Если убрать связь имя_детали → количество и ввести в рассмотрение дополнительный независимый атрибут, например *DD*, в качестве потенциального ключа, то схема, показанная на диаграмме, будет находиться в НФБК (рис. 6.7).

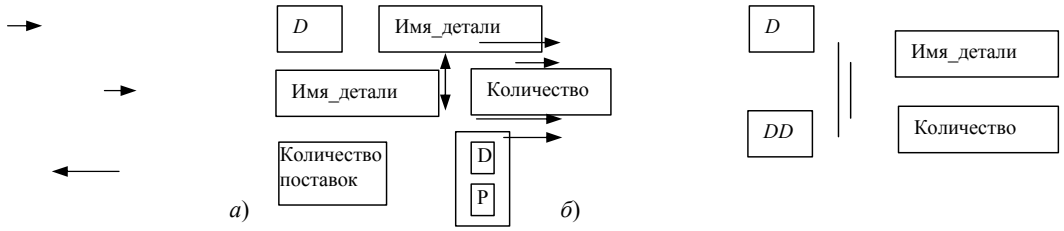


Рис. 6.7 Диаграмма функциональной зависимости в НФБК

Для "Веса" и "Материала" можно также ввести потенциальные ключи, ФЗ – усложнится.

6.4 ЭТАПЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

Число этапов проектирования баз данных напрямую зависит от количества уровней представления данных, или моделей данных. Известно четыре основных модели данных: даталогическая (ДЛМ), физи-

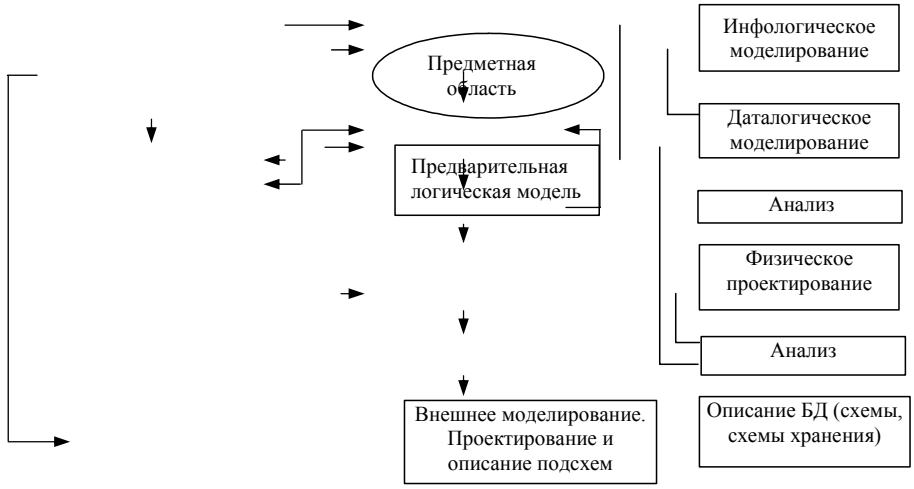


Рис. 6.8 Взаимосвязь этапов проектирования

ческая (ФМ), внешняя (ВМ) и инфологическая (ИЛМ). Таким образом, можно говорить о четырех этапах проектирования баз данных (рис. 6.8).

1. Даталогическое проектирование основано на модели логического уровня и представляет собой описание и построение схем связей между элементами данных безотносительно к их содержанию и среде хранения.

2. Физическое проектирование состоит в описании и построении схем хранения данных для определенной среды хранения. На этом этапе осуществляется выбор типа носителя, способ организации данных, методов доступа, определение параметров физического блока, управление работой памяти, считывание данных и т.д.

3. Внешнее моделирование состоит в описании и построении схем или логических структур с точки зрения конкретного пользователя. На этом этапе формализуются допустимые режимы обработки данных в рамках данной схемы или подсхемы. Для реляционных моделей это описание процедуры *View* конкретного приложения.

4. Инфологическое проектирование состоит в описании и построении схем отражений предметной области, выполненное без ориентации на используемые в дальнейшем программные и технические средства.

6.5 ИНФОЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

Инфологическая модель выполняется с использованием специальных искусственно формализованных языковых средств. Основное требование к инфологическому моделированию (ИЛМ) – это адекватное отражение предметной области. Дополнительные требования связаны с обеспечением возможности композиции и декомпозиции модели.

ИЛМ включает ряд компонентов (рис. 6.9). Центральной компонентой ИЛМ является ER-модель, описывающая объекты предметной области и связи между ними.

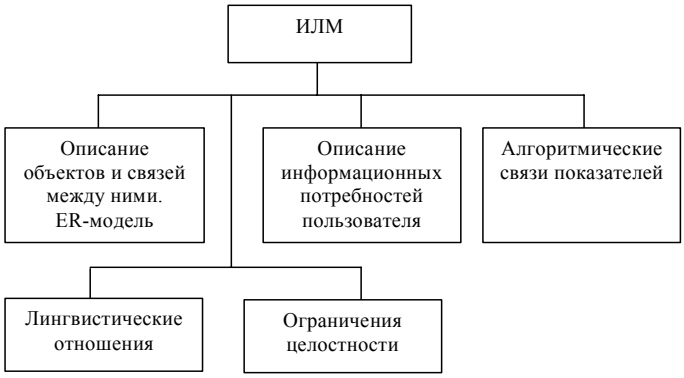


Рис. 6.9 Компоненты инфологической модели

Для описания ER-модели (объект–свойство–отношение) используют как языковые, так и графические средства (последние наиболее часто) (рис. 6.10).

Объекты, имеющие одинаковый набор свойств группируются в классы объектов со своими идентификаторами.

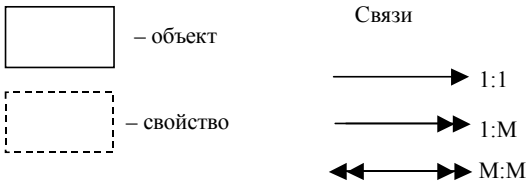


Рис. 6.10 Элементы ER-диаграмм

Свойства, не изменяющиеся во времени – статические (*S*), изменяющиеся – динамические (*D*).

Класс принадлежности показывает, может ли отсутствовать связь объекта одного класса с объектом другого класса или она не обязательна. В последнем случае в обозначение объекта добавляется разделитель с точкой (рис. 6.11).

Рис. 6.11 Обозначение класса принадлежности

В приведенных ниже примерах показаны диаграммы ER-экземпляров и ER-типов для базы данных имеющей два объекта – "Изделие" и "Деталь".

1 Изделие имеет в своем составе детали, но ни одно изделие не имеет более одной детали.

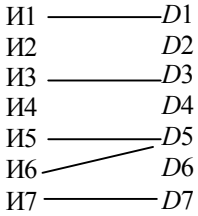


Диаграмма ER-экземпляров

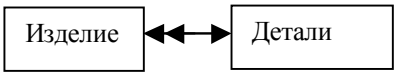


Диаграмма ER-типов M:1

2 Изделие имеет в своем составе детали. Каждое изделие должно иметь хотя бы одну деталь, но не более чем одну.

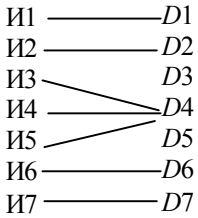


Диаграмма ER-экземпляров

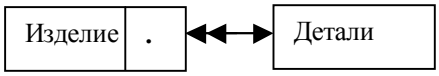


Диаграмма ER-типов M:1

3 Изделие имеет в своем составе детали. Но некоторые изделия состоят из нескольких деталей.

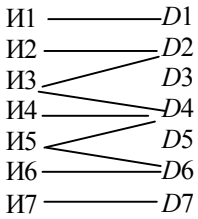


Диаграмма ER-экземпляров

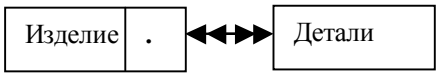


Диаграмма ER-типов M:M

4 Изделие имеет в своем составе детали. Каждое изделие обязательно состоит из нескольких деталей. Каждая деталь обязательно применяется в изделии.

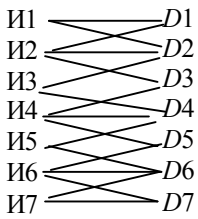


Диаграмма ER-экземпляров

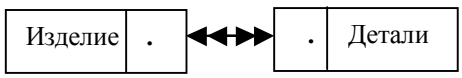


Диаграмма ER-типов M:M

Объекты могут быть простыми и сложными. Простой объект – это объект, который не делится на составляющие. Сложные – это составные, обобщенные и агрегированные объекты.

Составные объекты соответствуют отображению отношения "целое–часть". Например, изделие–детали, или группа–студенты и т.д. Специальных условных обозначений для них на схемах нет.

Обобщенный объект отражает наличие связи "род–вид". Например, объекты "студент", "аспирант", "школьник" образуют обобщенный объект "учащиеся" с наследованием некоторых свойств. В инфологической модели подклассы выделяются в явном или неявном виде и обозначаются треугольником.

Примером обобщенного объекта (рис. 6.12) может служить объект "личность", который имеет несколько категорий – "сотрудник", "студент", "аспирант". Объект "личность" разбит на два подкласса "сотрудник" и "учащийся". Объект "сотрудник" может быть классифицирован далее, например "преподаватели" и "администрация".

Агрегированные объекты соответствуют обычно какому-либо процессу, в который вовлечены другие объекты. Например, рассмотрим процесс поставки деталей заводом-изготовителем заказчику (рис. 6.13).

Агрегированный объект в инфологической модели обозначается ромбом. Объект и свойства обозначаются как и прежде, но они могут иметь и другие схематические изображения в различных стандартах (рис. 6.14).

В последнее время широко используются способы изображения в виде нотации Чена, Мартина и IDEF1X.

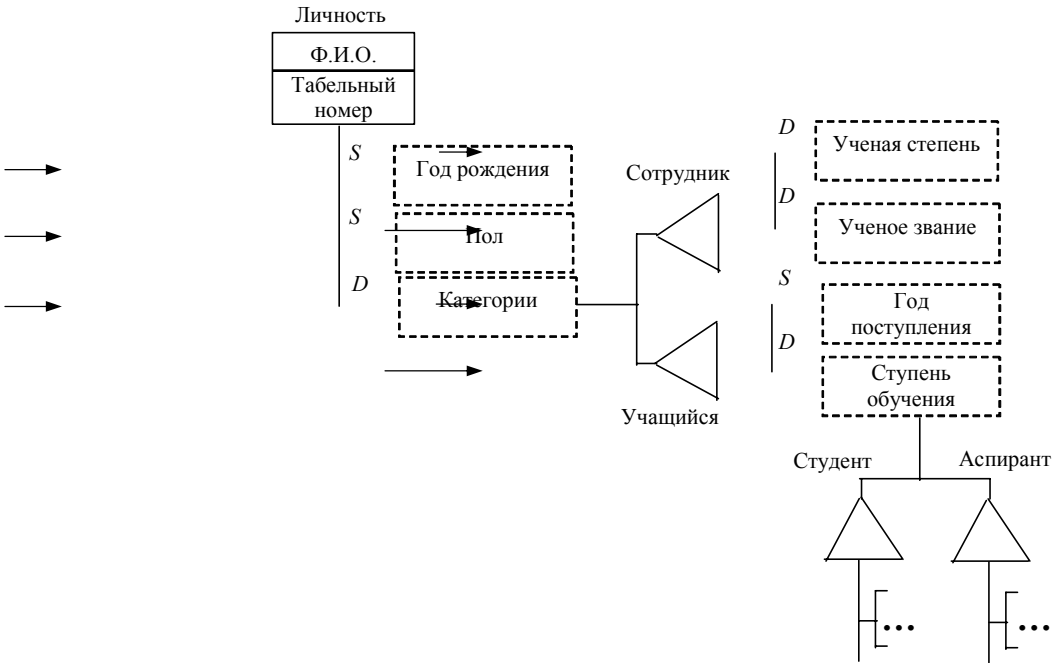


Рис. 6.12 Пример обобщенного объекта

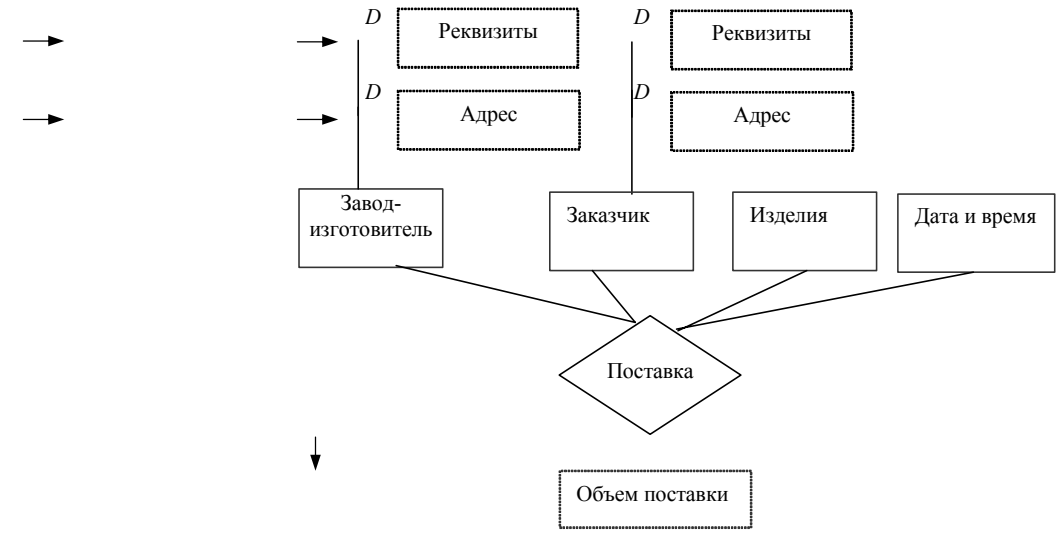
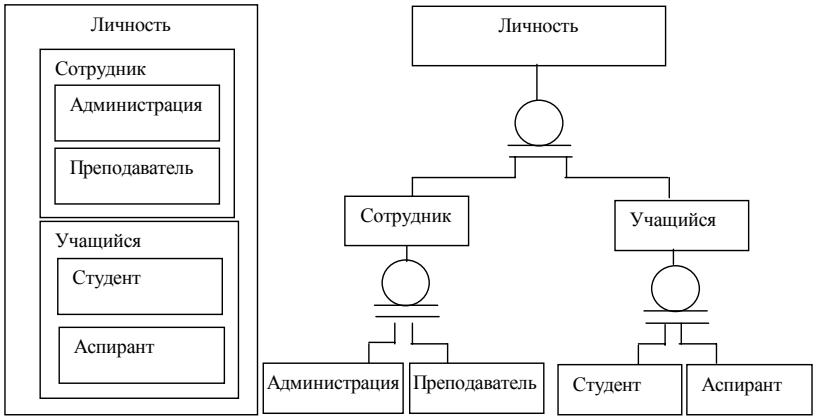


Рис. 6.13 Пример агрегированного объекта



а) *б*)

Рис. 6.14 Элементы ER-диаграммы в различных стандартах:
а – CASE ORACLE; *б – IDEF*

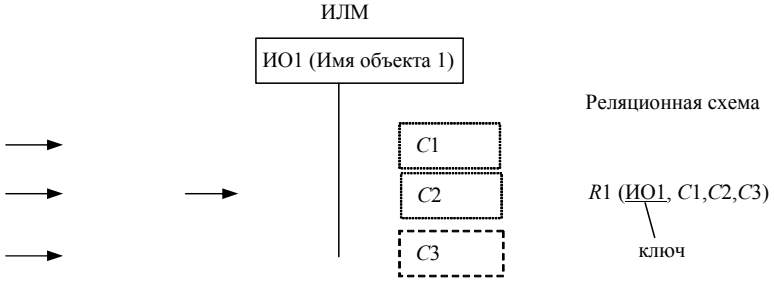
6.6 ДАТАЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

Конечным результатом даталогического проектирования является описание логической структуры базы данных на языке описания. В логической структуре базы определяются все информационные единицы и связи между ними, типы данных и количественные характеристики. Однако не все виды связей могут отображаться в даталогической модели (ДЛМ), например те, которые не поддерживает конкретная СУБД. На этапе разработки ДЛМ определяется состав базы данных, например, хранить только исходные данные, а все производные могут быть получены расчетным путем в результате запроса. При отображении объекта в файл исключаются одинаковые идентификаторы различных объектов, даются новые имена, определяются количество и структура файлов.

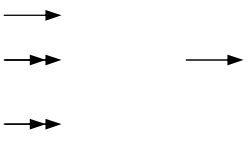
Важную роль на этом этапе играет внутризаписная структура данных (векторы, группы и проч.) и межзаписная структура (реляционная, иерархическая, сетевая).

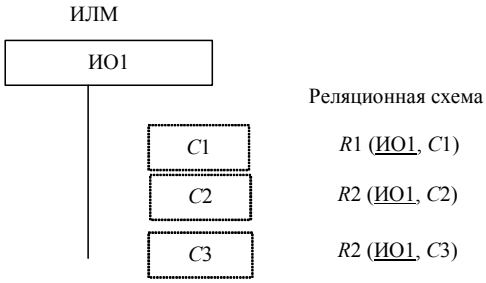
Для перехода от ИЛМ к реляционной ДЛМ надо выполнить следующие операции по замене ER-типа на описание атрибутов отношений:

1 Простой объект с единичными свойствами.

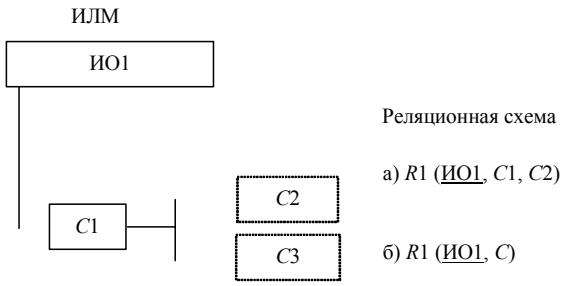


2 Множественные свойства объекта. Им в соответствие ставится отдельное отношение.

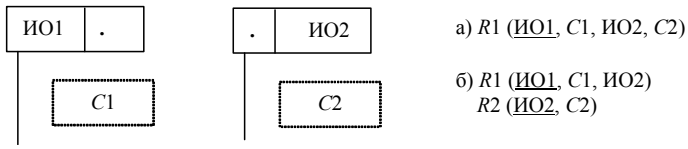




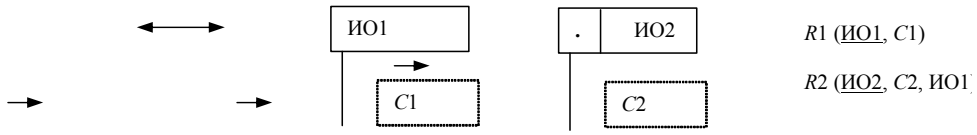
3 Объект с составным свойством. Если многие объекты обладают свойством, то их можно считать единичными (а). В противном случае – отдельное отношение с обобщенным свойством (б).



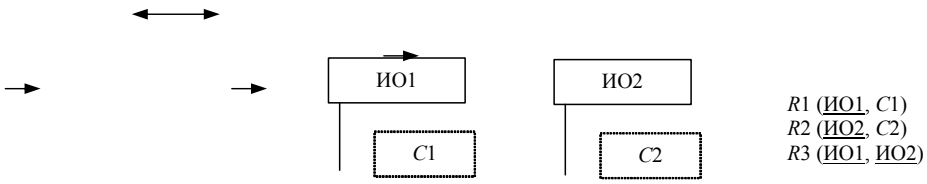
4 Связь 1 : 1 и обязательный класс принадлежностей сущностей.



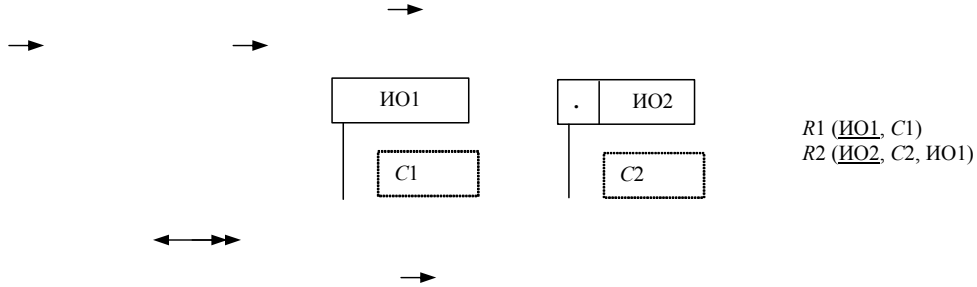
5 Связь 1 : 1 и обязательный класс принадлежностей одной из сущностей.



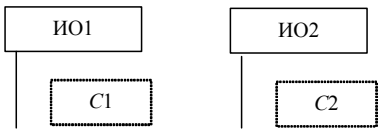
6 Связь 1 : 1 и необязательный класс принадлежности сущностей.



7 Связь 1 : M и обязательный класс принадлежностей *n*-связной сущности.

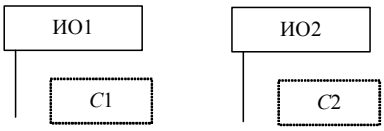


8 Связь 1 : M и необязательный класс принадлежности *n*-связной сущности.



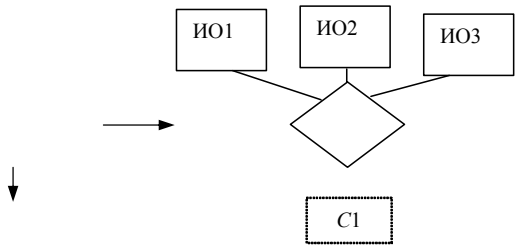
*R*1 (ИО1, *C*1)
*R*2 (ИО2, *C*2)
*R*3 (ИО1, ИО2)

9 Связь M : M и необязательный класс принадлежности *n*-связной сущности.



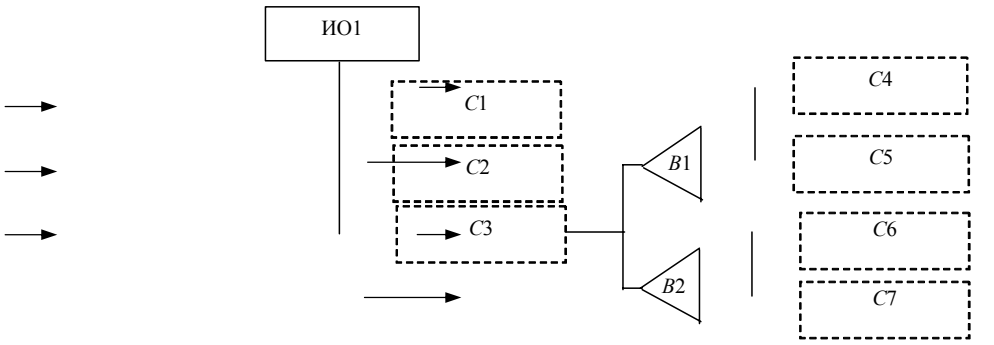
*R*1 (ИО1, *C*1)
*R*2 (ИО2, *C*2)
*R*3 (ИО1, ИО2)

10 Агрегированный объект.



*R*1 (ИО1, ИО2, ИО3, *C*1)

11 Обобщенный объект.



а) *R*1 (ИО1, *C*1, *C*2, *C*3, *C*4, *C*5, *C*6, *C*7)

б) *R*1 (ИО1, *C*1, *C*2, *C*4, *C*5)
*R*2 (ИО1, *C*1, *C*2, *C*6, *C*7)

Полученные реляционные отношения будут находиться в четвертой нормальной форме. Преобразования в пунктах 5, 7 и в пунктах 6, 8, 9 имеют одинаковые реляционные схемы.

На приведенных выше рисунках показан только вид заголовков таблиц. Полная даталогическая модель базы данных будет включать набор таких таблиц с указанием типов данных, длины переменной, принадлежности к первичному или внешнему ключу и т.д. и, соответственно, будет иметь более сложный вид. Одновременно надо указать связи между ключевыми атрибутами.

Ч а с т ь II

ПРАКТИЧЕСКИЕ ЭЛЕМЕНТЫ РАБОТЫ С БАЗАМИ ДАННЫХ

7 ЯЗЫК СТРУКТУРИРОВАННЫХ ЗАПРОСОВ SQL

7.1 ОСНОВЫ ЯЗЫКА SQL

Язык SQL представляет собой *структурированный язык запросов*. Это язык, который предоставляет пользователю возможность работать с реляционными базами данных, являющимися наборами связанной информации, сохраняемой в таблицах. Прежде, чем перейти к описанию самого языка SQL, еще раз вспомним, что такое реляционные базы данных.

Реляционная база данных представляет собой связанную информацию, сохраняемую в двумерных таблицах. Такой способ хранения информации может напоминать адресную или телефонную книгу.

В ней имеется большое количество входов, каждый из которых соответствует определенной особенности. Для каждой такой особенности может быть несколько независимых фрагментов данных, например имя, телефонный номер и адрес. Предположим, что необходимо сформировать эту адресную книгу в виде таблицы со строками и столбцами. Каждая строка (называемая также *записью*) будет соответствовать определенной сущности, каждый столбец будет содержать значение для каждого элемента записи (имеющего свой тип данных) – имени, телефонного номера и адреса, представляемых в строках. Проектируемая адресная книга могла бы выглядеть следующим образом:

Имя	**Телефон**	**Адрес**
Иванов Иван	333-100	Тамбов, ул. Советская 5, 15
Петров Петр	476-438	Тамбов, ул. Мичуринская 10, 3
Сидоров Олег	(203) 233-70	Котовск, ул. Интернациональная 23

То, что получилось, является основой реляционной базы данных, как и было определено в начале этого обсуждения – а именно, двумерной (с измерениями строка и столбец) таблицей информации. Однако, реляционные базы данных редко состоят из одной таблицы. Такая таблица меньше, чем файловая система. Создав несколько таблиц взаимосвязанной информации, можно выполнить более сложные и мощные операции с имеющимися данными. Мощность базы данных зависит от связей, которые можно создать между фрагментами информации, а не от самого фрагмента информации.

Язык SQL и предназначен для создания связей между различными данными таблиц базы данных и предоставления пользователю результатов выполнения операторов в процессе манипулирования над данными.

Язык SQL является языком интерпретируемым, т.е. для того, чтобы получить возможность выполнять написанные на этом языке команды для получения информации из базы данных, необходимо наличие специального программного обеспечения, которое будет их исполнять (обычно такая программа называется SQL-сервер). В зависимости от вида программного обеспечения различают интерактивный и вложенный SQL. Основное отличие этих версий языка проявляется при их использовании.

Интерактивный SQL используется для функционирования непосредственно в базе данных. При этом производится отбор из базы данных и вывод на печать информации сразу после ввода соответствующей команды.

Вложенный SQL состоит из команд языка SQL, вставляемых в текст программ, написанных на других языках программирования (например Паскаль или С). В этом случае прикладные программы становятся более мощными и эффективными. Однако, используя эти языки, приходится иметь дело со структурой SQL и стилем управления данными, который требует некоторых расширений к интерактивному SQL. Передача SQL команд во вложенный SQL является выдаваемой ("*passed off*") операцией для переменных или параметров, используемых программой, в которую они были встроены.

7.2 ТИПЫ ЗАПИСЕЙ

Не все типы значений, которые могут занимать поля таблиц, являются логически одинаковыми. Наиболее очевидное различие наблюдается между числами и текстом. Так как системы с реляционной базой данных основываются на связях между фрагментами информации, различные типы дан-

ных должны отличаться друга от друга так, чтобы соответствующие процессы и сравнения могли быть для них выполнены.

В SQL это достигается при помощи назначения каждому полю соответствующего типа данных, который указывает на тип значения, которое это поле может содержать. Все значения в данном поле должны иметь одинаковый тип.

Для описания чисел используются типы *INTEGER* (целое число) и *DECIMAL* (десятичное число), которые можно сокращать как *INT* и *DEC*. Эти типы адекватны соответствующим типам в большинстве прикладных программ, написанных на других языках программирования.

Для описания текста используется тип *CHAR* (символ), который относится к строке текста. Поле типа *CHAR* имеет определенную длину, которая определяется максимальным числом символов, которые могут быть введены в это поле. Большинство реализаций языка SQL также поддерживают нестандартный тип, называемый *VARCHAR* (переменное число символов), который является текстовой строкой, которая может иметь любую длину до определенного реализацией максимума (обычно 254 символа). *CHAR* и *VARCHAR* значения включаются в одиночные кавычки. Различие между *CHAR* и *VARCHAR* состоит в том, что *CHAR* должен резервировать достаточное количество памяти для строки максимальной длины, а *VARCHAR* распределяет память так, как это необходимо.

Также поддерживаются фактически стандартные типы данных *DATE* (дата) и *TIME* (время), хотя точный формат их представления может меняться в разных реализациях. Некоторые реализации SQL также поддерживают такие типы данных, как *MONEY* (деньги) и *BINARY* (двоичные).

7.3 СОЗДАНИЕ БАЗЫ ДАННЫХ

Здесь и далее используются следующие обозначения – [] для необязательных элементов команд и {} для возможных параметров команды.

Для создания баз данных используется оператор SQL, имеющий следующий формат:

CREATE {*DATABASE* | *SCHEMA*} <имя файла>
[*USER* <имя пользователя> [*PASSWORD* <пароль>]]
[*PAGE_SIZE* [=] <число>]
[*LENGTH* [=] <число> [*PAGE* [*s*]]]
[*DEFAULT CHARACTER SET* <набор символов>]
[<вторичный файл>];

где <имя файла> – спецификация файла, в котором будет храниться база данных;
USER <имя пользователя> – используется для проверки соединения пользователя с сервером;
PASSWORD <пароль> – проверяется совместно с именем пользователя;
PAGE_SIZE [=] <число> – размер страницы базы данных пользователя в байтах (по умолчанию 1024, 2048, 4096 или 8192);
LENGTH [=] <число> [*PAGE* [*s*]] – длина файла в страницах. По умолчанию 75 страниц, минимум 50, а максимум ограничен дисковым пространством;
DEFAULT CHARACTER SET – определяет набор символов, применяемых в базе данных, по умолчанию *NONE*.

Элемент <вторичный файл> в команде представляется конструкцией *FILE* <имя файла> [<файловая информация>], которая задает имя одного или нескольких файлов, в которых будет располагаться база данных. <файловая информация> определяется строкой *LENGTH* [=] <число> [*PAGE* [*s*]]] | *STARTING* [*AT* [*PAGE*]]. *STARTING* [*AT* [*PAGE*]] используется, если база данных занимает несколько файлов, этот параметр позволяет определить, с какой страницы располагается база в указанном файле.

В многофайловой базе данных самый первый файл называется *первичным*, а все остальные файлы – *вторичными*.

Например:

CREATE DATABASE "*D*:*BD**base.gdb*"
FILE "*D*:*BD**base.gd1*" *STARTING AT PAGE* 1001
LENGTH 500
FILE "*D*:*BD**base.gd2*";

В команде определяется база данных *d*:*bd**base.gdb*, состоящая из 3-х файлов: первичного *base.gdb* длиной 1000 страниц, *base.gd1* длиной 500 страниц и *base.gd2* неопределенной длины.

Если для вторичного файла не указана длина, следует указать, с какой страницы он должен начинаться.

Размер страницы указывается в байтах, например:

CREATE DATABASE "*base.gdb*" *PAGE_SIZE* 4096;

Увеличение размера страницы может привести к ускорению работы с базой данных за счет уменьшения глубины индексов, приводящих к уменьшению операций считывания длинных записей. Не является оправданным, когда запросы возвращают небольшое количество записей, так как происходит считывание страницы целиком, а в ней будет много лишних записей.

7.4 СОЗДАНИЕ ТАБЛИЦ

Перед созданием таблиц базы данных необходимо продумать определения всех столбцов таблицы и характеристики каждого столбца. При определении таблицы применяются домены. База данных, в которую добавляется таблица, должна быть открыта.

Создание таблицы базы данных осуществляется с помощью оператора

CREATE TABLE <имя таблицы> [*EXTERNAL* [*FILE*] <имя файла>]
(<оп_ст>[, <опр_ст> | <ограничение>…]);

где [*EXTERNAL* [*FILE*] <имя файла> – относится к внешним таблицам базы;
<оп_ст> – определение столбца базы данных.

Определение столбца базы данных имеет формат:

<имя столбца> {тип_данных | *COMPUTED* [*BY*] (<выражение>) | домен}
[*DEFAULT* {литерал | *NULL* | *USER*}]
[*NOT_NULL*] [<огранич_столбца>]
[*COLLATE COLLATION*]

где *COMPUTED* [*BY*] (<выражение>) – определение столбца вычисляемых значений;
DEFAULT – определяет значение столбца по умолчанию, ассоциировано с доменом;
<огранич_столбца> – ограничение на значение столбца;
COLLATE COLLATION – порядок сортировки символов.

Для столбцов вычисляемых значений соответствующее значение не вводится, а вычисляется согласно указанному выражению. Например, в таблице *A* есть столбцы номера квартала *n_q*, количество продаж в данном квартале в прошлом году *kol_s* и текущем году *kol_n* и прирост продаж за квартал *prirost*. В этом случае оператор создания такой таблицы выглядит следующим образом:

```
CREATE TABLE A (
    n_q INTEGER NOT NULL,
    kol_s INTEGER,
    kol_n INTEGER,
    prirost COMPUTED BY (kol_n – kol_s),
    PRIMARY KEY (n_q));
```

Ограничения целостности при создании таблиц бывают двух уровней – на уровне столбца или на уровне всей таблицы. Наложение ограничения целостности на отдельный столбец следует за его именем и типом:

```
tovar VARCHAR (20) NOT NULL PRIMARY KEY,
```

Во втором случае ограничения указываются после описаний всех столбцов:

CREATE TABLE… (
tovar VARCHAR (20) *NOT NULL*,
…,
PRIMARY KEY (*tovar*));

PRIMARY KEY – это первичный ключ построений по столбцу или столбцам. Столбцы должны иметь значение *NOT NULL*. Первичный ключ служит для установления связи с внешним ключом (*foreign key*) дочерней таблицы и определяет ссылочную целостность между родительской и дочерней таблицами.

Уникальный ключ строится по столбцу (столбцам), когда столбец не входит в первичный ключ и имеет уникальное значение (нет одинаковых значений).

CREATE TABLE klient (
Imja_klienta VARCHAR (20) *NOT NULL PRIMARY KEY*,
Nom_scheta VARCHAR (50) *NOT NULL*,
UNIQUE (*nom_scheta*));

Допускается также

nom_scheta VARCHAR (50) *NOT NULL UNIQUE*

Внешний ключ строится в дочерней таблице для соединения с родительской таблицей. Формат его описания имеет вид

FOREIGN KEY (<список_ст_внешнего_ключа>)
REFERENCES <имя_род_табл>
[<список_ст_род_табл>]
[*ON DELETE* {*NO ACTION* | *CASCADE* | *SET DEFAULT* | *SET NULL*}]
[*ON UPDATE* {*NO ACTION* | *CASCADE* | *SET DEFAULT* | *SET NULL*}]

где <список_ст_внешнего_ключа> – столбцы дочерней таблицы;
<имя_род_табл> – таблица, в которой описан первичный ключ (или столбец с атрибутом *UNIQUE*);
список_ст_род_табл – не обязателен при ссылке на первичный ключ родительской таблицы, в других случаях необходим способ изменения записей дочерней таблицы при удалении или изменении поля связи в родительской таблице (*ON DELETE*, *ON UPDATE*) со значениями *NO ACTION* – запрет, *CASCADE* – название, *SET DEFAULT* – в поле дочерней таблицы заносится значение, определенное ранее по умолчанию, *SET NULL* – заносится значение *NULL*.

Пример. Определим две таблицы:

– родительская *detal* с полями *name_det* (имя детали) и *cena_ed* (цена за единицу), первичный ключ по полю *name_det*;

– дочерняя *prihod* (приход со склада) с полями *n_prihod* (номер прихода), *date_prihod* (дата прихода), *name_det* (имя детали), *kolvo* (количество деталей в приходе). Первичный ключ по *n_prihod*, внешний – *name_det*.

Detal	
Name_det	*Zena_ed*
…	…
…	…

Prihod			
N_prihod	*Date_prihod*	*Name_det*	*kolvo*
…	…	…	…
…	…	…	…

Команды создания таких таблиц будут выглядеть следующим образом

CREATE TABLE detal (
Name_det VARCHAR (20) *NOT NULL*[1],
Cena_ed INTEGER NOT NULL,

[1] Должны быть описаны одинаково.

PRIMARY KEY (*name_det*));

CREATE TABLE prihod (
N_prihod INTEGER NOT NULL PRIMARY KEY,
Date_prihod DATE NOT NULL,
Name_det VARCHAR (20) *NOT NULL*[1],
Kolvo INTEGER NOT NULL,
FOREIGN KEY (*name_det*) *REFERENCES detal*);

Ссылочная целостность может быть определена следующим образом:

[*CONSTRAINT* <имя ссылочной целостности>]
FOREIGN KEY (<список столбцов внешнего ключа>)
REFERENCES <имя родительской таблицы>
[<список столбцов родительской таблицы>]

Необязательное имя ссылочной целостности присутствует в смешанных сообщениях и может использоваться при анализе базы данных. Для конкретного примера в таблицу *prihod* можно добавить:

Kolvo INTEGER NOT NULL,
CONSTRAINT po_detaly
FOREIGN KEY (*name_det*) *REFERENCES detal*

Требования к значениям столбцов могут быть определены как на уровне отдельного столбца, так и на уровне всей таблицы. Например, для таблицы параметров человека (*parametr*) рост (*rost*) должен быть больше веса (*ves*). Ограничения можно записать:

CREATE TABLE parametr (
Id INTEGER NOT NULL PRIMARY KEY,
Rost INTEGER NOT NULL,
Ves INTEGER NOT NULL CHECK (*rost* > *ves*));

или

CREATE TABLE parametr (
Id INTEGER NOT NULL,
Rost INTEGER NOT NULL,
Ves INTEGER NOT NULL,
PRIMARY KEY (*id*),
CHECK (*rost* > *ves*));

Ограничения, накладываемые на столбец, определяются предложением *CHECK*, имеющим форму

CHECK (<условия_поиска>)

Здесь <условия_поиска> задают конструкцию

{<значение><оператор>{<значение1> | (<выбор_одного>)} |
<значение> [*NOT*] *BETWEEN* <значение1> *AND* <значение2> |
<значение> [*NOT*] *LIKE* <значение> [*ESCAPE* <значение>] |
<значение> [*NOT*] *IN* <значение1>[, <значение2>…] | <выбор многих>)
| <значение> *IS* [*NOT*] *NULL*
| <значение>{[*NOT*]{=| < | > } >= | <= } {*ALL* | *SOME* | *ANY*}(<выбор многих>)

| *EXISTS* (<выражение_выбора>)
| *SINGULAR* (<выражение_выбора>)
| <значение> [*NOT*] *CONTAINING* <значение1>
| <значение> [*NOT*] *STARTING* [*WITH*] <значение1>
| (<условие_поиска>)
| *NOT* (<условие_поиска>)
| <условие_поиска> *OR* <условие_поиска>
| <условие_поиска> *AND* <условие_поиска>

где <значение>={столбец | <константы> | <выражение> | <функция> | *NULL* | *USER* | *RDB*$*DB_key*;
константа = число | 'строка';
функция = {
COUNT (* | [*ALL*] <значение> | *DISTINCT* <значение>)
| *SUM* ([*ALL*] <значение> | *DISTINCT* <значение>)
| *AVG* ([*ALL*] <значение> | *DISTINCT* <значение>)
MAX
MIN
CAST (<значение> *AS* <тип_данных>)
UPPER (<значение>)
| *BEN_ID* (генератор, <значение>)}
выбор_одного – *SELECT* возвращает одно значение;
выбор_многих – *SELECT* возвращает список или ни одного выражения выбора *SELECT*;
SINGULAR – TRUE для списка из одной строки;
EXISTS – TRUE если список не пустой.

7.5 СОЗДАНИЕ ДОМЕНА

При создании таблиц могут быть задействованы домены. Например, домен *Pol_type* используется при создании таблицы *Sotr*.

```
CREATE DOMAIN Pol_type AS CHAR(1) CHECK(VALUE IN ("м","ж"));

CREATE TABLE Sotr (
    Fio CHAR (20) NOT NULL,
    Pol Pol_type,
    otdel CHAR (10),
    otdel CHAR (10),
    PRIMARY KEY (Fio));
```

В этой конструкции *NOT NULL* определяет, что столбцы, ассоциированные с доменом, обязательно должны содержать значение.

Можно задавать ограничения на значения домена.

{*VALUE* <оператор> <значение>
| *VALUE* [*NOT*] *BETWEEN* <значение1> *AND* <значение2>
| *VALUE* [*NOT*] *LIKE* <значение> [*ESCAPE* <значение2>]
| *VALUE* [*NOT*] *IN* (<значение1> [, <значение2>…])
| *VALUE* [*NOT*] *NULL*
| *VALUE* [*NOT*] *CONTAINING* <значение>
| (<огр_домена>)
| *NOT* <огр_домена>
| <огр_домена> *OR* <огр_домена>
| <огр_домена> *AND* <огр_домена>}

где <оператор> = {= |<|>|<=|=>|!<|>!|<>|!=} (!< – не меньше, !> – не больше, != – не равно);
VALUE означает, что элементы считаются правильными;

BETWEEN <значение1> *AND* <значение2> – значение домена в интервале от значения 1 до значения 2;
LIKE <значение1> [<значение2>] – значение домена должно "походить" на значение 1;
LIKE "%*USD*" – вводимое значение должно оканчиваться на *USD*, все предыдущие – значения не имеют;
LIKE "__94" ("_"– единичный символ) вводится значение четырьмя символами, два последних – 94;
IN (<значение1>[,<значение2>…]) – значение домена должно совпадать с одним из приведенных параметров списка;
CONTAINING <значение> – значение домена должно иметь вхождения параметра <значение> в любом месте;
STARTING [*WITH*] <значение> – значение домена должно начинаться со <значение>, например "041".

Если '%' и '_' нужны в *LIKE* как символы, то указываются значения в *ESCAPE* и заменяются другими символами.

CREATE DOMAIN summa AS CHAR (10) *CHECK* (*LIKE* "%!%" *ESCAPE* "!");

После символа '!' служебные символы ('%') теряют свою силу и становятся обычными символами.

Большинство условий могут комбинироваться с помощью операций *AND* или *OR*, или указывать *NOT*.

CHECK (*VALUE NOT BETWEEN* 1 *AND* 100);

Изменения домена производятся с помощью команды

ALTER DOMAIN <имя> {
[*SET DEFAULT* {литерал | *NULL* | *USER*}]
| *DROP DEFAULT*]
| [*ADD* [*CONSTRAINT*] *CHECK* (<огр_домена>)]
| [*DROP CONSTRAINT*]};

Если использовано значение *NOT NULL*, то определение и тип данных нельзя изменить.

SET DEFAULT – значение по умолчанию; *DROP DEFAULT* – отменяет значение по умолчанию; *ADD* [*CONSTRAINT*] *CHECK* (<огр_домена…>) добавляет условие на значения столбцов; *DROP CONSTRAINT* – удаляет условия.

Пусть в таблице *A* определен домен:

CREATE DOMAIN id_type AS INTEGER CHECK (*VALUE* >=100);

CREATE TABLE A (
id id_type NOT NULL,
fio VARCHAR (20),
PRIMARY KEY (*id*));

Изменить ограничение *id* на ($100 \le id \le 500$) возможно двумя способами:
1) удалить старое условие *ALTER DOMAIN id_type DROP CONSTRAINT*;
2) добавить новое условие *ALTER DOMAIN id_type CHECK* (*VALUE* >= 100 *AND VALUE* <=500).

7.6 ВВОД ИНФОРМАЦИИ

Вся информация в SQL вводится с помощью команды модификации *INSERT*. В самой простой форме, эта команда использует следующий синтаксис:

INSERT INTO <имя таблицы> *VALUES* (<значение1>, <значение2> . . .);

Так, например, чтобы ввести строку в таблицу *Students*, необходимо использовать следующую команду:

INSERT INTO Students VALUES (*11*, *'Иванов'*, *'В-31'*);

Команда не производит никакого вывода данных в процессе работы. Имя таблицы (в нашем случае – *Students*) должно быть предварительно определено в команде *CREATE TABLE*, а каждое значение, присутствующее в списке параметров, должно совпадать с типом данных столбца, в который оно вставляется. Значения вводятся в таблицу в поименном порядке, поэтому первое значение в списке автоматически попадает в первый столбец таблицы, второе – во второй и т.д.

Если нужно ввести *пустое значение* (*NULL*), то его можно задавать точно так же, как и обычные параметры. Значение *NULL* – это специальный маркер, а не просто символьное обозначение, поэтому оно не включается в одиночные кавычки.

Можно также указывать столбцы, в которые нужно вставить соответствующее значение. Это позволяет пользователю вставлять имена в любом порядке. Предположим, что берутся значения для таблицы студентов из отчета выводимого на принтер, который помещает их в таком порядке: "группа", "фамилия". Для упрощения пользователь вводит эти значения в том же порядке.

INSERT INTO Students (*группа*, *фио*) *VALUES* (*'В-31'*, *'Петров'*);

Следует обратить внимание, что столбец *номер_в_журнале* отсутствует. Это значит, что строка автоматически установлена как значение по умолчанию. По умолчанию может быть введено или значение *NULL* или другая величина, определенная как "значение по умолчанию". Если ограничение запрещает использование значения *NULL* в данном столбце, и этот столбец не установлен как по умолчанию, то он должен быть обеспечен значением для любой команды *INSERT*, которая относится к таблице.

7.7 ИЗМЕНЕНИЕ СУЩЕСТВУЮЩИХ ТАБЛИЦ

Команда *ALTER TABLE* не является частью *ANSI* стандарта языка SQL, однако она является широко доступной и распространенной, хотя ее возможности несколько ограничены. Она используется для внесения изменений в определение существующей таблицы. Обычно она применяется для добавления столбцов к таблице. Иногда она может удалять столбцы или изменять их размеры, а также в некоторых программах добавлять или удалять ограничения. Типичный синтаксис команды при добавлении столбца к таблице выглядит следующим образом:

ALTER TABLE <имя таблицы> *ADD* <имя столбца> <тип> <размер>;

Столбец будет добавлен в таблицу со значением *NULL* для всех ее строк. Новый столбец станет последним по порядку столбцом таблицы. Можно добавить сразу несколько новых столбцов, отделив их запятыми, в одной команде. Имеется возможность удалять или изменять столбцы. Наиболее часто изменением столбца может быть просто увеличение его размера или добавление (удаление) ограничения. Система должна убедиться, что любые изменения не противоречат существующим данным – например, при попытке добавить ограничение к столбцу, который уже имел значение, при нарушении которого ограничение будет отклонено.

Из-за нестандартного характера команды *ALTER TABLE* необходимо посмотреть тот раздел документации используемой системы, в которой говорится об особых случаях.

ALTER TABLE не действует в тех случаях, когда таблица должна быть переопределена. Изменение структуры таблицы, когда она уже используется, опасно для базы. Просмотрите внимательно таблицы, которые, являясь вторичными таблицами с извлеченными данными из другой таблицы, не долго правильно работают, а программы, использующие вложенный SQL, выполняются неправильно или не всегда правильно. Кроме того, изменение может отменить всех пользователей, имеющих разрешение обращаться к таблице. По этим причинам необходимо разрабатывать таблицы базы данных так, чтобы использовать *ALTER TABLE* только в крайнем случае.

Таким образом, с помощью оператора *ALTER TABLE* можно:

- добавить определение нового столбца;
- удалить столбец из таблицы;
- удалить атрибуты целостности таблицы или отдельного столбца;
- добавить новые атрибуты целостности.

Допустим, была создана таблица

CREATE TABLE sotr (
Id_sotr INTEGER NOT NULL PRIMARY KEY,
Fio CHAR (10),
Otdel VARCHAR (10),
Doljnost CHAR (10));

Изменим характеристики столбца *fio* с *CHAR* (10) на *VARCHAR* (25):

1) добавим в таблицу новый временный столбец *fio_tmp*:

ALTER TABLE sotr ADD fio_tmp CHAR (10);

2) копируем данные из *fio* в *fio_tmp*;

UPDATE sort SET fio_tmp = fio;

3) удаляем столбец *fio*;

ALTER TABLE sotr DROP fio;

4) создаем новый стобец *fio*;

ALTER TABLE sotr ADD fio VARCHAR (25);

5) переписываем данные;

UPDATE sort SET fio = fio_tmp;

6) удаляем временный столбец.

ALTER TABLE sort DROP fio_tmp;

Добавление нового столбца в таблицу базы данных может быть осуществлено командой:

ALTER TABLE <имя таблицы> *ADD* <определение столбца>;

Добавление новых ограничений целостности:

ALTER TABLE <имя таблицы> *ADD* [*CONSTRAINT* <имя ограничения>] <определение целостности>;

Удаление столбца (столбцов) из таблицы:

ALTER TABLE <имя таблицы> *DROP* <имя столбца> [, <имя стодбца2>…];

Удаление ограничений целостности (уровень таблицы):

ALTER TABLE <имя таблицы> *DROP* <ограничения целостности>;

7.8 УДАЛЕНИЕ ТАБЛИЦ

Удаление таблицы целиком может быть выполнено с помощью команды

DROP TABLE <имя таблицы>;

Удаление может быть блокировано для родительских таблиц в том случае, если есть дочерние. В этом случае удаление таблицы разрушит ссылочную целостность базы данных.

7.9 РАБОТА С ИНДЕКСАМИ В ЯЗЫКЕ SQL

Первичный и внешний ключи строятся для обеспечения ссылочной целостности реляционно-связанных таблиц базы данных. Помимо этого, первичный ключ выполняет функции уникальности своих значений. Для этих же целей используется и просто уникальный ключ.

Индексы, в отличие от ключей, создаваемые оператором *CREATE INDEX*, служат для сортировки данных и оптимизации доступа к ним. В конечном итоге ключи и индексы преобразуются в *физические индексы* – специальный механизм быстрого доступа к данным.

Индексы имеет смысл создавать в следующих случаях:

- часто производится поиск в базе данных;
- часто строится объединение таблиц;
- часто производится сортировка.

Не рекомендуется строить индексы по столбцам или группами столбцов, которые:

- редко используются для поиска;
- часто меняют значение (надо часто обновлять индекс);
- содержат небольшое число вариантов значения.

Синтаксис оператора создания индексов выглядит следующим образом:

CREATE [*UNIQUE*] [*ASC*[*ENDING*] | *DESC* [*ENDING*]]
INDEX <имя индекса> *ON* <имя таблицы> (<столбец1>, …);

где *UNIQUE* – уникальный индекс, не допускает одинаковых значений;
ASC[*ENDING*] – сортировка полей индекса по возрастанию (по умолчанию);
DESC[*ENDING*] – сортировка полей индекса по убыванию.

Пример. Для таблицы *prihod* создадим индекс в порядке убывания значений *date_prihod* и *name_det*.

```
CREATE TABLE prihod (
    N_prihod INTEGER NOT NULL PRIMARY KEY,
    Date_prihod DATE NOT NULL,
    Name_det VARCHAR (20) NOT NULL,
    Kolvo INTEGER NOT NULL);

CREATE DESC INDEX D_P ON prihod (date_prihod, name_det);
```

После многократного внесения изменений в таблицу индексы могут быть разбалансированы, так как "глубина" индекса возрастает. Это приводит к увеличению времени поиска. Поэтому необходимо время от времени: выполнять балансировку индекса оператором *ALTER INDEX*; переписывать выбираемость индекса оператором *SET STATISTICS*; уничтожать и вновь создавать индекс операторами *DROP INDEX* и *CREATE INDEX*.

Балансировка индекса производится при помощи команд

ALTER INDEX <имя индекса> *DEACTIVATE*;
ALTER INDEX <имя индекса> *ACTIVATE*;

При этом нельзя перестроить индекс, участвующий в запросах, а также индекс, построенный по первичному или внешнему ключу.

Удалить индекс можно при помощи команды

DROP INDEX <имя индекса>;

7.10 ВЫБОРКА ИНФОРМАЦИИ В ЯЗЫКЕ SQL

Один из мощнейших операторов языка SQL – оператор *SELECT* – позволяет производить выборки данных, преобразовывать полученные результаты, реализует сложные условия выбора. Его синтаксис выглядит следующим образом:

SELECT [*DISTINCT* | *ALL* {* | <значение1> [, <значение2> …]}
FROM <таблица1> [, <таблица2> …]
[*WHERE* <условие_поиска>]
[*GROUP BY* <столбец> [*COLLATE COLLATION*]
[, <столбец1> [*COLLATE COLLATION*] …]
[*HAVING* <условия_поиска>]
[*UNION* <оператор_*SELECT*>]
[*PLAN* <план_выполнения_запроса>]
[*ORDER BY* <список_столбцов>];

В простейшем случае, когда требуется просмотреть все записи одной или нескольких таблиц, оператор имеет вид:

SELECT {* | <значение1> [, <значение2> …]}
FROM <таблица1> [, <таблица2> …];

В этой команде <значение1> и <значение2> обозначают имена столбцов, значения которых будут выбраны оператором, а <таблица1>, <таблица2> – имена таблиц, из которых происходит выборка данных.

Предложение *WHERE* используется для включения в базу данных лишь нужных записей, удовлетворяющих условию:

SELECT {* | <значене1> [, <значение2> …]}
FROM <таблица1> [, <таблица2> …] *WHERE* <условия поиска>;

При сравнении значения столбца с константой условие поиска будет состоять из константы, операции отношения и константы. В качестве операций отношения можно использовать =, <, >, <=, >=, <>.

При сравнении значения столбца одной таблицы со значением столбца другой таблицы условие поиска будет включать имена столбцов этих таблиц, разделенные операцией отношения. Для доступа к столбцам разных таблиц используются их уточненные имена, состоящие из имени таблицы и имени столбца, разделенные точкой.

Идентификация столбцов через имя таблицы неудобна из-за громоздкости обозначений. Существует возможность присвоить каждой таблице краткое имя. Такие имена называются *псевдонимами* таблиц. Они отделяются пробелом от фактического имени таблицы в списке *FROM*.

Результат выборки информации можно упорядочить с помощью предложения *ORDER BY* <список столбцов>. Если в списке столбцов указано больше одного столбца, то первый будет использоваться для глобальной сортировки, а каждый следующий – для сортировки внутри группы, определенной единым значением предшествующего столбца.

Ключевое слово *DISTINCT* позволяет избавиться в результирующем списке от повторяющихся записей. Повторяющимися считаются записи, содержащие идентичные значения во всех столбцах. Если в результирующем списке нужно показать все записи, то указывается ключевое слово *ALL*.

Для расчета вычисляемых столбцов результирующего списка используются арифметические выражения:

SELECT [*DISTINCT* | *ALL*] {* | <столбец1> [, <выражение1> …]}
FROM <таблица1> [, <таблица2> …];

Если столбцу надо присвоить нестандартное имя, то оно может быть указано за выражением при помощи ключевого слова *AS*.

В выражениях можно использовать следующие агрегатные функции:

– *COUNT* (<выражение>) – подсчитывает число вхождений значения выражения во все записи результирующего списка;
– *SUM* (<выражение>) – суммирует значение выражения;
– *AVG* (<выражение>) – находит среднее значение;
– *MAX* (<выражение>) – определяет максимальное значение;
– *MIN* (<выражение>) – определяет минимальное значение.

Например, необходимо вычислить общую стоимость оприходованных деталей за дату "4.10.00":

SELECT sum (*p.kolvo * d.zena_ed*) *AS itogo*
FROM prihod p, *detal d*
WHERE (*p.name_det=t.name_det*)*AND*(*p.date_prihod*= "04.10.2000");

Для группы записей столбца, характеризующих одинаковые значения, можно получить агрегированные значения (*MIN*, *MAX*, *AVG*). При этом один из столбцов представляется агрегирующей функцией и предложение *GROUP BY* <столбец1> [, <столбец2> …] ставится вместе с предложением *WHERE*.

На группировку записей можно наложить ограничения. Агрегация выдается только по группам, удовлетворяющим условию. Синтаксис такого выражения в команде *SELECT* выглядит следующим образом:

GROUP BY <имя столбца> *HAVING* <агрегирующая функция> <отношение> <константа>;

Часто возникает ситуация, связанная с тем, что невозможно решить поставленную задачу путем использования единственного запроса. Например, в тех случаях, когда при использовании условия поиска в предложении *WHERE* значение с которым надо сравнить, далее не определено, а вычисляется оператором *SELECT*. В таких случаях применяют вложенные запросы или подзапросы. Оператор *SELECT* имеет вид:

SELECT …
FROM …
WHERE <сравниваемое значение> <оператор *SELECT* >;

Оператор *SELECT* возвращает не одно значение, а список, поэтому может возникнуть ошибка. Чтобы ее избежать, надо заменить операцию отношения "=" на оператор выбора из нескольких возможных значений *IN*. Синтаксис вложенного запроса ничем не отличается от синтаксиса основного запроса. Это значит, что в подзапрос может быть вложен другой подзапрос и т.д.

Внешние соединения определяются в предложении *FROM* согласно спецификации:

SELECT {* | <значение1> [, <значение2>…]}
FROM <таблица1> <вид соединения > *JOIN* < таблица2>
ON <условие поиска>;

Внешнее соединение отличается от внутреннего тем, что в результирующий список включаются записи ведущей таблицы соединения, которые объединяются с пустым множеством записей другой таблицы. Какая из таблиц будет ведущей, определяет вид соединения (*LEFT* – левое внешнее соединение, ведущая <таблица1>; *RIGHT* – правое внешнее соединение, ведущая <таблица2>; *FULL* – полное внешнее соединение). В случае полного внешнего соединения ведущими являются обе таблицы. В результирующий список включаются все записи обеих таблиц согласно алгоритму:

1) если для записи <таблицы1> имеются записи в <таблицы2>, удовлетворяющие условию соединения, то в результирующий список включаются все комбинации записей обеих таблиц;

2) в противном случае в результат включается запись <таблицы1>, соединенная с пустой записью <таблицы2>;

3) пункты 1 и 2 повторяются для таблицы 2 и таблицы 1.

В запросах к базе данных эта операция может быть использована, например, когда требуется найти поставщика, соответствующего каждой поставке детали или найти все поставки по каждому поставщику и т.п.

Объединение результатов нескольких операторов *SELECT* производится оператором *UNION*. Одинаковые записи при этом не дублируются.

7.11 ДОБАВЛЕНИЕ, ИЗМЕНЕНИЕ И УДАЛЕНИЕ ЗАПИСЕЙ В ТАБЛИЦАХ

Добавление новых записей в таблицу может осуществляться с помощью команды, имеющей следующий синтаксис:

INSERT INTO <таблица> [(<столбец1> [, <столбец2> …])]
{*VALUES* (<значение1> [, <значение2>…]) | <оператор *SELECT*>};

Если значения присваиваются всем столбцам по порядку, то имена столбцов можно не писать, в противоположном случае имена столбцов надо указывать. Применение оператора *SELECT* не нарушает порядка присваивания значений столбцам.

Пусть определена таблица *prihod_data* аналогичная *prihod*:

```
CREATE TABLE prihod_data (
    N_prihod INTEGER NOT NULL,
    Date_prihod DATE NOT NULL,
    Name_det VARCHAR (20) NOT NULL,
    Kolvo INTREGER NOT NULL,
    Post VARCHAR (20),
    PRIMARY KEY (n_prihod);
```

Пусть в эту таблицу надо ежедневно копировать все записи о полученных деталях. Тогда выгрузка записей из *prihod* в таблицу *prihod_data* будет реализована оператором:

```
INSERT INTO prihod_data
    SELECT * FROM prihod WHERE date_prihod = "13-oct-2000";
```

Оператор *UPDATE* применяется для изменения значения в группе записей или одной записи. Он имеет следующий формат:

UPDATE <таблица>
SET <столбец1> = <значение1> [, <столбец2> = <значение2>…]
[*WHERE* <условие поиска>];

Если опустить *WHERE* <условие поиска>, то будут изменены все записи, в противном случае – только записи, удовлетворяющие условию.

Оператор *DELETE* применяется для удаления групп записей из таблиц или *VIEW* (см. далее). Он имеет формат:

DELETE FROM <таблица>
[*WHERE* <условие поиска>];

Если конструкция *WHERE* <условие поиска> отсутствует, то удаляются все записи, иначе – только записи, удовлетворяющие условию.

7.12 ПРОСМОТР *VIEW*

В базе данных может быть определен просмотр, являющийся виртуальной таблицей, в которой представлены все записи из одной или нескольких таблиц. Создать просмотр можно с помощью команды:

CREATE VIEW <имя просмотра> [(<столбец_*view*1> [, <ст_*view*>…])]
AS <оператор *SELECT*> [*WITH CHECK OPTION*];

Удалить просмотр можно с помощью команды
DROP VIEW <имя просмотра>;

Использование просмотров имеет несколько преимуществ:
- однажды определив просмотр, не нужно всякий раз формировать оператор *SELECT* для запроса;
- представляет подмножество столбцов из таблиц, что усиливает безопасность данных.

Пример. Создать просмотр, содержащий дату прихода, имя детали, их количество из таблицы *prihod* и цену из таблицы *detal*:

```
CREATE VIEW full
    AS SELECT p.date_prihod, p.name_det, p.kolvo, d.zena_ed
    FROM prihod p, detal d
    WHERE p.name_det = d.name_det;
```

Теперь обращение с запросом к базе данных упрощается:

```
SELECT * FROM full;
```

Действие над просмотром не отразится на исходной таблице.

Существуют следующие способы формирования просмотров:
- вертикальный срез таблицы (включается подмножество столбцов);

```
CREATE VIEW p_vert
    AS SELECT name_det, kolvo FROM prihod;
```

- горизонтальный срез таблицы (все столбцы, но не все записи);

```
CREATE VIEW p_vh
    AS SELECT name_det, kolvo WHERE name_det = "D1";
```

- вертикально-горизонтальный срез таблицы (не все столбцы и не все строки);

```
CREATE VIEW full_p
    AS SELECT p.name_det, p.kolvo, d.zena_ed
    FROM prihod p, detal d WHERE p.name_det = d.name_det;
```

- подмножество строк и столбцов разных таблиц;

```
CREATE VIEW full_p
    AS SELECT p.name_det, p.kolvo, d.zena_ed
    FROM prihod p, detal d WHERE p.name_det = d.name_det;
```

Если столбец вычисляемый, то его имя должно указываться в имени просмотра.

Чтобы просмотр можно было обновлять, необходимо:
- формировать *VIEW* из записей одной таблицы;
- включить все столбцы с атрибутами *NOT NULL* в исходной таблице;
- оператор *SELECT* просмотра не должен использовать агрегирующих функций, а также *DISTINCT*, *HAVING*, *JOIN*, хранимых процедур и функций.

Если *VIEW* удовлетворяет этим требованиям, то к нему применимы операторы *INSERT*, *UPDATE* и *DELETE*.

7.13 РАБОТА С ТРИГГЕРАМИ

Триггер – это процедура базы данных, автоматически вызываемая SQL-сервером при обновлении, удалении или добавлении новой записи в таблицы базы. Непосредственно из программы к триггерам обратиться нельзя. Нельзя и передавать им входные параметры и получать от них значе-

ния выходных параметров. Триггеры всегда используют действия. При откате *транзакции* отменяются все изменения, внесенные в базу данных триггерами. События изменения таблиц базы включают добавление новой записи, изменение существующей записи, удаление записи. Вызов триггеров может выполняться до наступления события или после его наступления. Преимущество использования триггеров состоит в следующем:

1) автоматическое обеспечение каскадных воздействий в дочерних таблицах, удаление записи в родительской таблице выполняется на сервере. Пользователь не заботится о каскадной реализации и одновременно не пересылает изменения на сервер;

2) изменения в триггерах не влекут необходимости изменения программного кода в клиентских приложениях.

Формат оператора создания триггера выглядит следующим образом:

CREATE TRIGGER <имя триггера> *FOR* <имя таблицы>
[*ACTIVE* | *INACTIVE*]
{*BEFORE* | *AFTER*}
{*DELETE* | *INSERT* | *UPDATE*}
{*POSITION* <номер>}
AS <тело триггера>;

Структура тела триггера имеет вид
[<объявление локальных переменных>]
BEGIN
<оператор>
END;

где *ACTIVE* | *INACTIVE* – указывает, активен триггер или нет. По умолчанию действует *ACTIVE*. Триггер можно создать заранее и установить *INACTIVE*. В дальнейшем этот параметр можно переопределить на *ACTIVE*;

BEFORE | *AFTER* – выполнение триггера до или после запоминания изменений в базе данных;

DELETE | *INSERT* | *UPDATE* – операция над таблицей при выполнении которой будет срабатывать триггер;

POSITION <номер> – указывает порядок выполнения, если применяется группа триггеров. Значение номера должно находиться в диапазоне от 0 до 32767. Триггеры с меньшими номерами выполняются раньше. В группе допускаются триггеры с одинаковыми характеристиками операции и времени вызова.

К значениям столбцов в триггерах допускается применение двух операций: *OLD* и *NEW*. Значение *OLD.имя_столбца* позволяет обратиться к состоянию столбца до внесения изменений, а значение *NEW.имя_столбца* – к состоянию после изменения. Если значение столбца не изменилось, то *OLD.имя_столбца* равно *NEW.имя_столбца*. Например, нужно внести изменения в таблицу *prihod*, если в записи таблицы *detal* изменилось значение столбца *name_det*:

```
CREATE TRIGGER ba_detal FOR detal
ACTIVE
BEFORE UPDATE
AS
BEGIN
        IF (OLD.name_det <> NEW.name_det) THEN
            UPDATE prihod
            SET name_det = NEW.name_det
            WHERE name_det = OLD.name_det
END
```

Ограничение ссылочной целостности таблиц по внешнему ключу приводит к блокировке изменения и удаления записи в родительской таблице, если для нее есть дочерние записи в дочерней

таблице. Для реализации автоматического выполнения каскадных обновлений и изменений необходимо:

1) в определении базы данных удалить ссылочные целостности, блокирующие такие изменения;
2) определить триггеры для родительской таблицы.

Триггер, реализующий каскадное обновление в дочерней таблице, будет содержать оператор

```
IF (OLD.поле_связи_родителя <> NEW.поле_связи_родителя) then
UPDATE <дочерняя таблица>
SET <поле связи дочерней таблицы> = NEW.поле_связи_родителя
WHERE поле_связи_дочерней_таблицы=OLD.поле_связи_родителя;
```

Триггер, реализующий каскадное удаление в дочерней таблице, будет содержать оператор

```
DELETE FROM <дочерняя таблица>
WHERE <поле связи дочерней таблицы> = <поле связи родителя>;
```

Например, напишем триггеры, выполняющие каскадное обновление и удаление в дочерней таблице *prihod* после соответствующего изменения записи в таблице *detal*:

```
CREATE TRIGGER ba_detal FOR detal
ACTIVE
BEFORE UPDATE
AS
BEGIN
        IF (OLD.name_det <> NEW.name_det) THEN
            UPDATE prihod
            SET name_det = NEW.name_det
            WHERE name_det = OLD.name_det;
END

CREATE TRIGGER ad_detal FOR detal
ACTIVE
AFTER DELETE
AS
BEGIN
        DELETE FROM prihod
        WHERE prihod.name_det = detal.name_det;
END
```

7.14 ВЕДЕНИЕ ЖУРНАЛА ИЗМЕНЕНИЙ

Журнал изменений в базе данных представляет собой таблицу базы, в которой фиксируются действия над всей базой данных или отдельными ее таблицами. Журнал позволяет определить источник недостоверных или искаженных данных.

Определим в базе данных таблицу *gurnal_det*:

```
CREATE TABLE gurnal_det (
Dat_izm DATE, /*дата изменения*/
Old_name VARCHAR (20), /*старое имя*/
New_name VARCHAR (20)); /*новое имя*/
```

Действия *ins*, *upd* и *del*, дата изменения, старое и новое значения столбца *name_det* будут фиксироваться в таблице *gurnal_det* с помощью триггеров

```
CREATE TRIGGER gurnal_det_add FOR detal
```

```
ACTIVE
AFTER INSERT
AS
BEGIN
        INSERT INTO gurnal_det (dat_izm, deistv, old_name, new_name)
        VALUES ("now", "ins", " ", NEW.name_det);
END

CREATE TRIGGER gurnal_det_upd FOR detal
ACTIVE
AFTER UPDATE
AS
BEGIN
        INSERT INTO gurnal_det (dat_izm, deistv, old_name, new_name)
        VALUES ("now", "upd", OLD.name_det, NEW.name_det);
END

CREATE TRIGGER gurnal_det_del FOR detal
ACTIVE
AFTER UPDATE
AS
BEGIN
        INSERT INTO gurnal_det (dat_izm, deistv, old_name, new_name)
        VALUES ("now", "del", OLD.name_det, " ");
END
```

Теперь все изменения столбца *detal.name_det* будут фиксироваться в журнале на сервере.

Следует заметить, что для операции удаления новое значение столбца *name_det* будет пустым, а для операции добавления пустым будет старое значение этого столбца. Просмотр журнала выполняется так же, как и просмотр обычной таблицы:

```
SELECT * FROM gurnal_det;
```

Операции изменения и удаления допускаются и над триггером. Существующий триггер можно изменить оператором:

```
ALTER TRIGGER <имя триггера> FOR <имя таблицы>
[ACTIVE | INACTIVE]
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE}
{POSITION <номер>]
AS <тело триггера>
```

После выполнения этого оператора все старые определения триггера заменяются на указанные в его составе.

Удалить триггер можно оператором:

```
DROP TRIGGER <имя триггера>;
```

7.15 РАБОТА С ХРАНИМЫМИ ПРОЦЕДУРАМИ (НА ПРИМЕРЕ СУБД INTERBASE)

Хранимая процедура – это модуль, написанный на процедурном языке *InterBase* и хранящийся в базе данных как метаданные. Хранимую процедуру можно вызвать из программы.

Создание выполняется оператором:

CREATE PROCEDURE <имя процедуры>
[(<входной параметр> <тип данных>
[, <входной параметр> <тип данных>…])]
[*RETURNS*
(<выходной параметр> <тип данных>
[, <выходной параметр> <тип данных>…])]
AS <тело процедуры>;

Входные и выходные параметры могут быть пропущены, если в них нет необходимости.

Тело процедуры имеет следующий формат:

[<объявление локальных переменных>]
BEGIN
<оператор>
[<оператор>]
END

Формат объявления локальных переменных:

DECLARE VARIABLE <имя переменной> <тип данных>

Пример:

CREATE PROCEDURE full_post (*det VARCHAR* (20))
RETURNS (*adr_post VARCHAR* (40)) *AS*
DECLARE VARIABLE naiden_post VARCHAR (20);
DECLARE VARIABLE max_kolvo;
BEGIN
…
END

Тело процедуры ограничивается операторными скобками *BEGIN…END*. Значения переменным присваивается оператором "=" (*out_detal = full_post* (*name_det*);). Оператор *IF… THEN… ELSE* имеет такой же формат, что и в среде программирования Object Pascal:

IF (<условие>) *THEN* <оператор 1> [*ELSE* <оператор 2>];

Оператор *SELECT* используется в хранимой процедуре для выдачи единичной строки. Синтаксис отличается добавлением предложения:

INTO :<переменная> [, <переменная>…]

Значение, возвращаемое *SELECT*, записывается в переменные.

Например:

SELECT AVG (*kolvo*), *SUM* (*kolvo*)
FROM prihod
WHERE name_det = :*in_name*
INTO :*avg_kolvo*, :*sum_kolvo*;

где *in_name* – входная переменная, содержащая значение.

Оператор *FOR SELECT… DO* <оператор> работает по алгоритму: выполняется оператор *SELECT* и для каждой строки полученного результата выполняется оператор, следующий за словом *DO*.

Оператор *WHILE IN DO* имеет формат:

WHILE (<условие>) *DO* <оператор>

В цикле проверяется условие. Если оно истинно, то выполняется оператор. Цикл продолжается, пока условие не перестанет выполняться.

Оператор *EXIT* инициирует прекращение выполнения процедуры и выход в вызывающее приложение.

7.16 ТРАНЗАКЦИИ

Транзакция – это единичное или групповое изменение базы данных, которое выполняется полностью или не выполняется вообще.

Управление транзакциями на SQL-сервере осуществляется операторами *SET TRANSACTION* (начать транзакцию), *COMMIT* (подтвердить) и *ROLLBACK* (откатить).

Оператор *SET TRANSACTION* имеет формат:

SET TRANSACTION [*READ WRITE* | *READ ONLY*]
[*WAIT* | *NO WAIT*]
[[*ISOLATION LEVEL*] {*SNAPSHOT* [*TABLE STABILITY*]
| *REAL COMMITTED* [[*NO*] *RECORD_VERSION*]}]
[*RESERVING* <список таблиц> [*FOR* [*SHARED* | *PROTECTED*]
[*READ* | *WRITE*]], [<список таблиц>…];

где *READ WRITE* | *READ ONLY* – устанавливает уровень доступа к данным (по умолчанию *READ WRITE*);

WAIT | *NO WAIT* – определяет поведение при возникновении конфликта между транзакциями. По умолчанию *WAIT*, т.е. ожидание выполняется контролирующей транзакцией. *NO WAIT* – аварийное завершение;

ISOLATION LEVEL – уровень изоляции транзакции на сервере по умолчанию *SNAPSHOT*, т.е. уровень *REPETABLE READ*;

REAL COMMITTED – разрешает читать только подтвержденные данные (*NO RECORD_VERSION* – читается последняя версия записи, даже не подтвержденная другой транзакцией; *RECORD_VERSION* – подтвержденная);

RESERVING – блокирует конкурирующие транзакции;

PROTECTED READ – конкурирующие транзакции могут читать данные, но не изменить;

PROTECTED WRITE – читать данные могут только транзакции с уровнем *SNAPSHOT* или *READ COMMITED* и никто не может изменить этот порядок.

Например, запускаем транзакцию

SET TRANSACTION WAIT ISOLATION LEVEL READ COMMITTED;

Следующий оператор запускает транзакцию *T1* и блокирует таблицы *table1* и *table2* для чтения, а *table3* – для записи.

SET TRANSACTION NAME T1 WAIT
ISOLATION LEVEL READ COMMITTED
RESERVED table1, *table2 FOR*
PROTECTED READ,
Table3 FOR PROTECTED WRITE;

8 ПРИМЕНЕНИЕ СРЕДСТВ ОПЕРАЦИОННЫХ СИСТЕМ ДЛЯ ДОСТУПА К БАЗАМ ДАННЫХ

В состав многих операционных систем входят средства, облегчающие прикладным программам работу с информацией, хранящейся в базах данных. Такие средства позволяют создавать максимально независимые друг от друга прикладные программы и базы данных, что повышает мобильность и эффективность прикладного программного обеспечения. В состав операционной системы MS Windows для решения этой задачи включен драйвер ODBC.

8.1 РАБОТА С ДРАЙВЕРОМ ODBC

Open Database Connectivity (*ODBC*) – это широко распространенный программный интерфейс, предоставляющий прикладным программам доступ к базам данных. Он основывается на *Call-level interface* (*CLI*) для интерфейсов СУБД, описанном в *X*/*Open* и *ISO*/*IEC* и использует структурированный язык запросов SQL как язык доступа к базам данных.

ODBC построен таким образом, чтобы приложение могло получать доступ к различным СУБД, используя одни и те же функции. Приложение вызывает функции интерфейса, который описан в одном из специфических для каждой СУБД модулей, называемых драйверами. Использование драйверов освобождает программиста от необходимости знать специфику конкретной СУБД. Так как драйверы загружаются динамически, достаточно загрузить соответствующий драйвер; нет необходимости перекомпилировать приложение. ODBC-драйверы поставляются вместе с любой СУБД, доступ к которой возможен через ODBC. Эти драйверы специфичны для конкретных СУБД. Благодаря стандарту ODBC клиентским приложениям становится возможным выполнять запросы SQL, определенные независимо от производителя СУБД.

Концепция ODBC оперирует одним менеджером драйверов и несколькими ODBC-драйверами или разделяемыми библиотеками для доступа к конкретным базам данных.

ODBC Driver Manager является центральным компонентом системы "клиент – сервер базы данных". Он связывает ODBC-приложение с используемыми этим приложением ODBC-драйверами. ODBC-драйверы в Windows представляют собой динамически загружаемую библиотеку и выполняют специфические действия по обработке SQL запросов при обращении к конкретной базе данных. Такой подход является универсальным и обеспечивает возможность его использования в других операционных системах.

ODBC для Windows состоит из трех основных компонентов:

- файл настроек *odbc.ini*;
- библиотека, поддерживающая механизм ODBC на уровне операционной системы *odbc.dll*;
- ODBC драйверы для каждой базы данных.

Файл настроек *odbc.ini* содержит информацию о базах данных и зарегистрированных в системе ODBC драйверах.

В первой секции файла настроек *odbc.ini* "[*ODBC 32 Bit Data Source*]" указывается список зарегистрированных в системе баз данных. Этот список можно увидеть также при добавлении нового драйвера в ODBC Driver Manager. Ниже приведен пример организации первой секции этого файла.

[*ODBC 32 Bit Data Source*]
MS Access 97 Database=Microsoft Access Driver (**.mdb*) (*32 bit*)
Excel Files=Microsoft Excel Driver (**.xls*) (*32 bit*)
FoxPro Files=Microsoft FoxPro Driver (**.dbf*) (*32 bit*)
Paradox Files=Microsoft Paradox Driver (**.db*) (*32 bit*)
Text Files=Microsoft text Driver (**.txt*;**.cvs*) (*32 bit*)
Reestr_Sybase=Sybase SQL Anywhere 5.0 (*32 bit*)
dBASEFile=INTERSOLV 3.01 32-BIT dBASEFile (**.dbf*) (*32 bit*)
Interbase=Interbase 4.x Driver by Visigenic (**.gdb*) (*32 bit*)

Каждый источник данных ниже описывается более подробно в своей секции файла *odbc.ini*. Название секции, где расшифровывается источник, формируется из левой части вышеописанных равенств.

[*MS Access 97 Database*]
Driver32=c:*win**system**odbcjt32.dll*
[*Excel Files*]
Driver32=c:*win**system**odbcjt32.dll*
[*FoxPro Files*]
Driver32=c:*win**system**odbcjt32.dll*
[*Text Files*]
Driver32=c:*win**system**odbcjt32.dll*
[*ODBC Data Sources*]

SQL Anywhere 5.0 Sample Client=Sybase SQL Anywhere 5.0
[*SQL Anywhere 5.0 Sample Client*]
driver=d:\sqlany50\winwod50w.dll
description=Sybase SQL Anywhere Client/Server
EngineName=place_server_name_here
Start=d:\sqlany50\win\dbclienw
[*Reestr_Sybase*]
Driver3232=d:\sqlany\win\wod50t.dll
[*dBASEFile*]
Driver32=c:\win\system\ivdfb12.dll
[*Interbase*]
Driver32=c:\win\system\iscdrv32.dll

Файл настроек можно также сформировать с помощью интерактивной программы *odbcad32.exe* (ODBC-администратор), поставляемой вместе с операционной системой. Информация о местоположении базы данных, используемая машина базы данных, имя пользователя по умолчанию и его пароль и прочее для каждого драйвера сохраняются в реестре по ключу *HKCU\SOFTWARE\ODBC\ODBC.INI*.

Добавления и настройки источника данных осуществляются с помощью "Администратора источника данных ODBC", внешний вид которого представлен на рис. 8.1.

Пользовательский DSN – источник данных пользователя. Эти источники данных используются на локальном компьютере текущим пользователем и другим пользователям этого компьютера недоступны.

Системные DSN определяются для компьютера, а не для конкретного пользователя. Такие источники данных могут использоваться членами системной группы или пользователями, имеющими соответствующие привилегии.

Вкладки "Пользовательский DSN" и "Системный DSN" полностью идентичны, их внешний вид приведен на рис. 8.2.

Файловые DSN доступны одновременно всем пользователям, у которых установлены такие же драйверы для доступа к базе данных. Назначение таких источников данных конкретному пользователю или привязка к локальному компьютеру не производится.

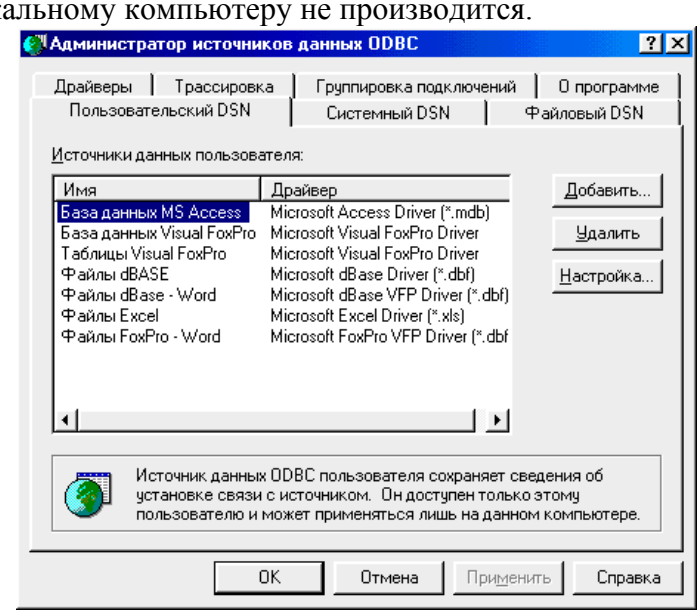


Рис. 8.1 Диалоговое окно ODBC-администратора

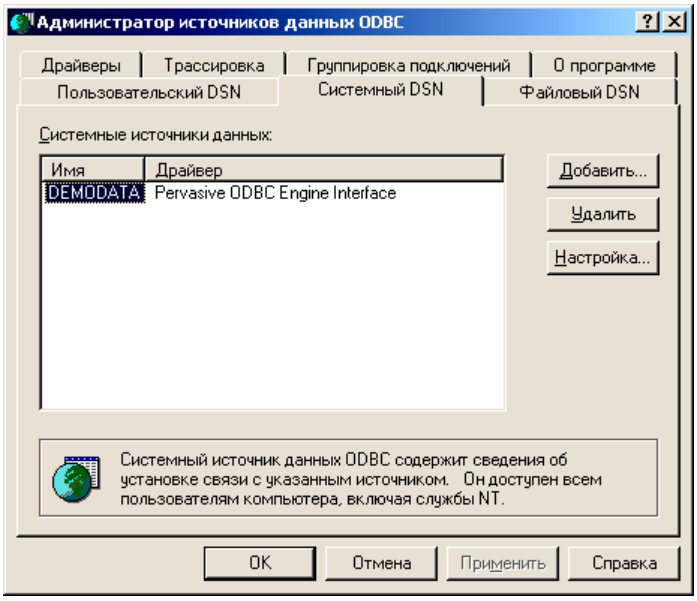


Рис. 8.2 Внешний вид вкладок "Пользовательский DSN" и "Системный DSN"

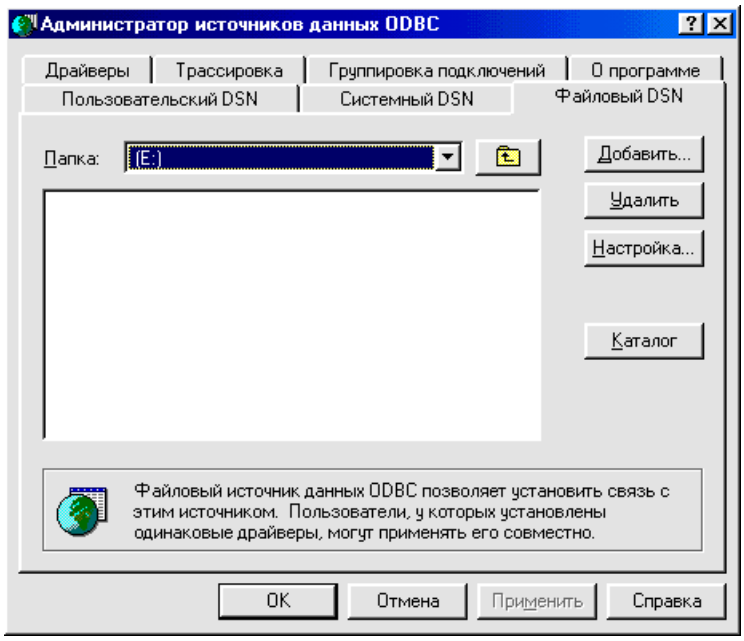


Рис. 8.3 Внешний вид вкладки "Файловый DSN"

Внешний вид вкладки "Файловый DSN" представлен на рис. 8.3. По функциональности он повторяет пользовательский и системный источники данных, однако здесь необходимо указать еще и имя файла, для которого создается источник. Например, для базы данных *Interbase* это будет файл с расширением *gdb* (**.gdb*).

Вкладка "Драйверы ODBC" содержит список доступных и установленных в системе драйверов ODBC. Каждый элемент списка включает в себя имя, версию драйвера, компанию-производитель, имя файла, дату создания файла каждого драйвера ODBC, установленного в системе. Добавление или удаление драйверов производится вместе с установкой или удалением соответствующего сервера баз данных.

Назначение кнопок "Добавить", "Удалить" и "Настроить" для всех вкладок одинаково. Ниже приведены особенности их функционального назначения. Кнопка "Настроить" открывает диалоговое окно настройки существующего источника данных. Для каждого драйвера параметры настройки могут быть разные. Вид диалогового окна и набор параметров определяет драйвер, а не ODBC-администратор. Кнопка "Удалить" удаляет источник данных из списка. Для двух вышеописанных случаев источник данных необходимо сначала выбрать из списка. Кнопка "Добавить" добавляет новый источник в выбранный список. Добавление нового источника можно условно разделить на два этапа. Первым является выбор необходимого для данного источника ODBC-драйвера. Окно выбора

показано на рис. 8.4. Список драйверов в этом окне повторяет список на вкладке "Драйверы ODBC" диалогового окна ODBC-администратора. После выбора нужного драйвера и нажатия кнопки "Готово" запускается менеджер настройки выбранного источника данных. Для каждого драйвера, как было сказано выше, менеджер настройки может быть организован по-разному и ответственность за список необходимых параметров и последовательность их заполнения полностью возлагаются на драйвер.

Для всех ODBC существует ряд общих свойств, некоторые из которых доступны на вкладке "Группировка подключений" (рис. 8.5). Это окно предназначено для изменения времени ожидания восстановления соединения и допустимого времени отсутствия соединения для выбранного драйвера в случае использования объединенных соединений. Предусмотрена возможность включения наблюдения за быстродействием, в ходе которого ведется статистика соединения.

Каждый элемент списка драйверов может содержать имя, версию драйвера, компанию-производитель, имя файла, дату создания файла каждого драйвера ODBC, установленного в системе. Время ожидания

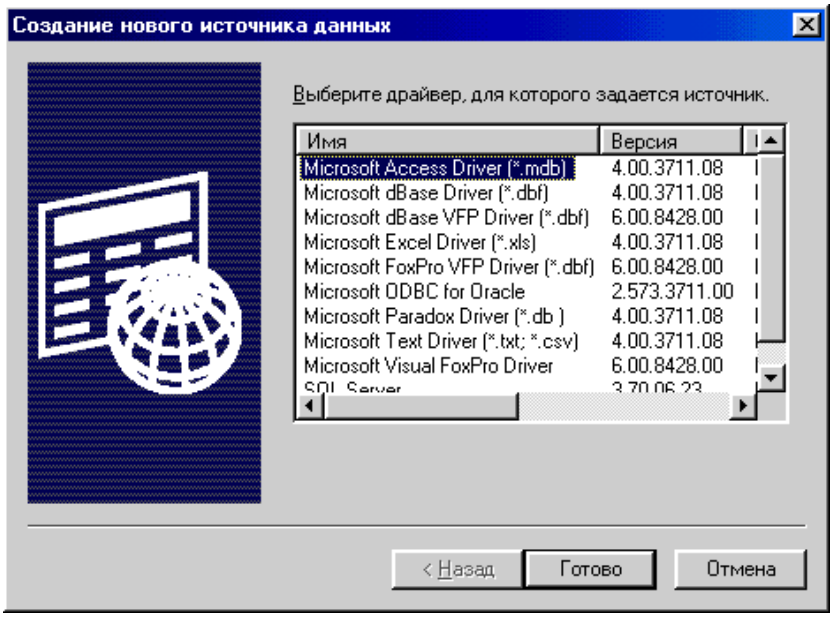


Рис. 8.4 Окно выбора драйвера для создания нового источника данных

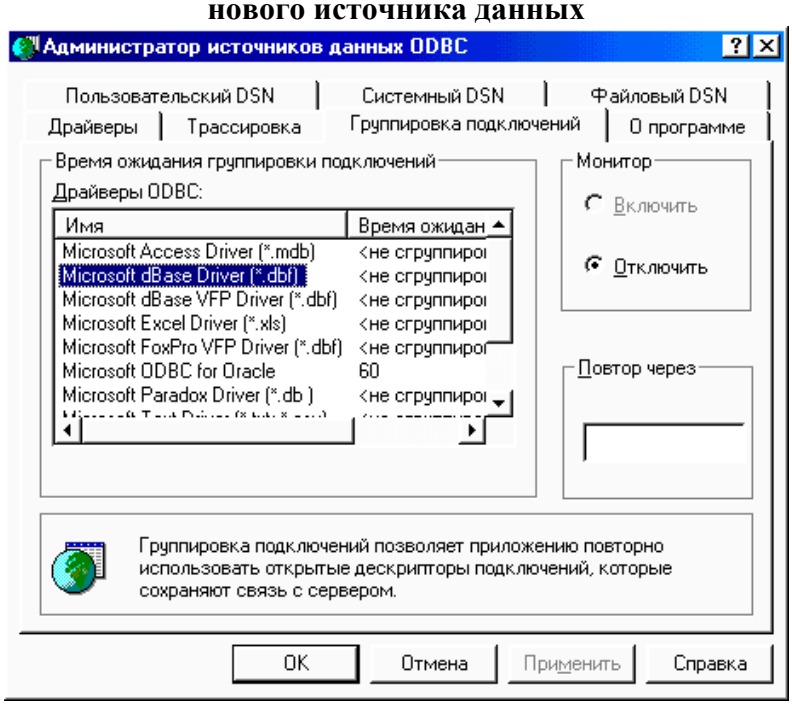


Рис. 8.5 Окно настройки ODBC-драйвер средствами ODBC-администратора

группировки – установка допустимого времени отсутствия соединения в секундах для выбранного драйвера. Щелкнув дважды на имени соответствующего драйвера, можно задать атрибуты объединенного соединения.

Переключатели "Включить" и "Отключить" управляют режимом наблюдения за быстродействием. В строке ввода "Повтор через" задается перерыв между попытками подключения в том случае, если диспетчером драйверов ODBC было установлено, что сервер баз данных недоступен. Значения могут состоять не более чем из пяти цифр.

Вкладка "Трассировка" (рис. 8.6) позволяет определить способ отслеживания вызовов функций ODBC диспетчером драйверов. Диспетчер драйверов может отслеживать вызовы постоянно или только для одного подключения; отслеживание может вестись динамически и с помощью какой-нибудь другой библиотеки *.dll*.

Кнопка "Начать трассировку" включает динамическое отслеживание на время, пока открыто диалоговое окно администратора ODBC. Динамическое отслеживание можно включить независимо от того, установлено ли в данный момент соединение или нет.

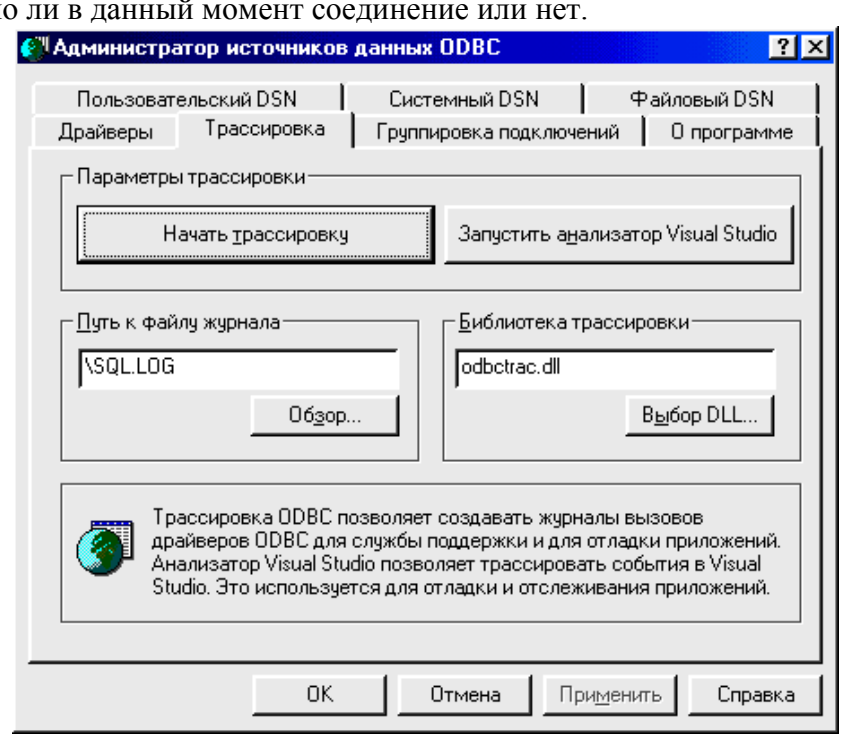


Рис. 8.6 Окно трассировки драйвера ODBC

После нажатия кнопки "Начать трассировку" надпись на ней заменяется надписью "Закончить трассировку". Отслеживание ведется до нажатия кнопки "Закончить трассировку". ODBC-обращения к функциям из прикладных программ, запущенных ранее, не регистрируются. При включенной трассировке журнал постоянно растет, а это воздействует на эффективность всех функционирующих прикладных программ ODBC.

Кнопка "Запустить анализатор Visual Studio" разрешает генерацию событий для анализатора Visual Studio. После нажатия кнопки "Запустить анализатор Visual Studio" надпись на ней заменяется надписью "Остановить анализатор Visual Studio", прекращающей генерацию событий для анализатора Visual Studio. Отслеживание проводится до ее нажатия.

Окно ввода "Путь к файлу журнала" позволяет задать путь и имя файла сведений об отслеживании. По умолчанию используется файл, указанный в сведениях о системе (*sql.log*), но его можно изменить, введя вручную новый путь и имя файла либо выбрав файл из списка с помощью кнопки "Обзор".

Окно ввода "Библиотека трассировки" позволяет заменить используемую для отслеживания библиотеку *Odbctrac.dll* на другую. Поставляемый в комплекте *Data Access SDK* файл *Odbctrac.dll* можно заменить другим файлом *.dll* по выбору. Для этого следует ввести путь и имя файла *.dll* или выбрать его из списка с помощью кнопки "Выбор DLL".

8.2 ПРОГРАММНЫЙ ИНТЕРФЕЙС ODBC

Ниже приведено описание основных функций программного интерфейса драйвера WinODBC:

1) SQLRETURN ***SQLAllocHandle***(SQLSMALLINT HandleType, SQLHANDLE InputHandle, SQLHANDLE* OutputHandlePtr);

Используется для соединения с базой данных. Эта функция заменяет функции SQLAllocConnect, SQLAllocEnv и SQLAllocStmt стандарта ODBC 2.0. При вызове данная функция заменяется менеджером драйверов на соответствующую из вышеперечисленных.

Параметры:

HandleType – тип указателя, который необходимо получить. Должен иметь одно из следующих значений: SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT, SQL_HANDLE_DESC;

InputHandle – задает указатель контекста, в котором его необходимо получить. Если *HandleType* задан как SQL_HANDLE_ENV, то *InputHandle* должен иметь значение SQL_NULL_HANDLE. Если *HandleType* – это SQL_HANDLE_DBC, то *InputHandle* должен быть environment handle. Если SQL_HANDLE_STMT или SQL_HANDLE_DESC, то connection handle.

OutputHandlePtr – указатель на буфер, в который будет помещен результат.

Возвращаемые значения:

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE или SQL_ERROR.

2) SQLRETURN ***SQLCloseCursor***(SQLHSTMT StatementHandle);

Закрывает курсор, связанный с указанным SQL-запросом. Результаты запроса удаляются.

Параметры:

StatementHandle – дескриптор оператора.

Возвращаемые значения:

SQL_SUCCESS или SQL_SUCCESS_WITH_INFO – в случае нормального завершения; SQL_ERROR или SQL_INVALID_HANDLE – в случае неудачи.

3) SQLRETURN ***SQLConnect***(SQLHDBC ConnectionHandle, SQLCHAR* ServerName, SQLSMALLINT NameLength1, SQLCHAR* UserName, SQLSMALLINT NameLength2, SQLCHAR* Authentication, SQLSMALLINT NameLength3);

Устанавливает соединение с драйвером и базой данных. После установки соединения с помощью *ConnectionHandle* можно получить всю информацию о соединении, включая статус, состояние транзакции и коды ошибок.

Параметры:

ConnectionHandle – указатель соединения;

ServerName – имя источника данных;

UserName – имя пользователя;

Authentication – строка аутентификации (как правило – пароль);

NameLength1, *NameLength2*, *NameLength3* – длины соответствующих строк параметров.

Возвращаемые значения:

SQL_SUCCESS или SQL_SUCCESS_WITH_INFO – в случае успешного завершения; SQL_ERROR или SQL_INVALID_HANDLE – в противном случае.

4) SQLRETURN ***SQLDisconnect***(SQLHDBC ConnectionHandle);

Завершает указанное соединение.

Параметры:

ConnectionHandle – дескриптор соединения, которое необходимо завершить.

Возвращаемые значения:

SQL_SUCCESS или SQL_SUCCESS_WITH_INFO – в случае нормального завершения; SQL_ERROR или SQL_INVALID_HANDLE – в случае неудачи.

5) SQLRETURN ***SQLEndTran***(SQLSMALLINT HandleType, SQLHANDLE Handle, SQLSMALLINT CompletionType);

Завершает или откатывает транзакцию для всех активных операций всех SQL-запросов, связанных с одним соединением или всеми соединениями указанного SQL-окружения.

Параметры:

HandleType – тип дескриптора. Аргумент должен иметь одно из следующих значений: SQL_HANDLE_ENV – для всех соединений, SQL_HANDLE_DBC – для одного соединения;

Handle – дескриптор, транзакцию которого необходимо очистить;

CompletionType – тип операции, которую необходимо провести. Этот аргумент должен иметь одно из следующих значений: SQL_COMMIT – в данном случае транзакция будет завершена, SQL_ROLLBACK – в данном случае будет совершена попытка отката транзакции.

Возвращаемые значения:

SQL_SUCCESS или SQL_SUCCESS_WITH_INFO – в случае нормального завершения; SQL_ERROR или SQL_INVALID_HANDLE – в случае неудачи.

6) SQLRETURN ***SQLExecDirect***(SQLHSTMT StatementHandle, SQLCHAR* StatementText, SQLINTEGER TextLength);

Исполняет непосредственно строку SQL-запроса, используя текущие значения параметров.

Параметры:

StatementHandle – дескриптор оператора;

StatementText – указатель на буфер, содержащий строку SQL-запроса;

TextLength – длина строки в *StatementText*.

Возвращаемые значения:

SQL_SUCCESS или SQL_SUCCESS_WITH_INFO – в случае нормального завершения; SQL_NEED_DATA – недостаточно данных для выполнения запроса; SQL_STILL_EXECUTING – запрос исполняется; SQL_ERROR или SQL_INVALID_HANDLE – в случае неудачи.

7) SQLRETURN ***SQLExecute***(SQLHSTMT StatementHandle);

Исполняет ранее подготовленный SQL-запрос.

Параметр: *StatementHandle* – дескриптор оператора.

Возвращаемые значения:

SQL_SUCCESS или SQL_SUCCESS_WITH_INFO – в случае нормального завершения; SQL_NEED_DATA – недостаточно данных для выполнения запроса; SQL_STILL_EXECUTING – запрос исполняется; SQL_NO_DATA – запрос неподготовлен; SQL_ERROR или SQL_INVALID_HANDLE – в случае неудачи.

8) SQLRETURN ***SQLFetch***(SQLHSTMT StatementHandle);

Используется для получения следующей порции данных результата SQL-запроса.

Параметр: *StatementHandle* – дескриптор оператора.

Возвращаемые значения:

SQL_SUCCESS или SQL_SUCCESS_WITH_INFO – в случае нормального завершения; SQL_NO_DATA – нет данных; SQL_STILL_EXECUTING – выполняется; SQL_ERROR или SQL_INVALID_HANDLE – в случае неудачи.

9) SQLRETURN ***SQLFetchScroll***(SQLHSTMT StatementHandle, SQLSMALLINT FetchOrientation, SQLINTEGER FetchOffset);

Используется для получения порции данных результата SQL-запроса.

Параметры:

StatementHandle – дескриптор оператора;

FetchOrientation – направление получения. Может иметь следующие значения:

SQL_FETCH_NEXT – следующая порция данных;

SQL_FETCH_PRIOR – предыдущая порция данных;

SQL_FETCH_FIRST – первая порция данных;

SQL_FETCH_LAST – последняя порция данных;

SQL_FETCH_ABSOLUTE – начиная с записи *FetchOffset*;

SQL_FETCH_RELATIVE – начиная с текущей записи + *FetchOffset*;

SQL_FETCH_BOOKMARK – начиная с закладки + *FetchOffset*.

FetchOffset – смещение.

Возвращаемые значения:

SQL_SUCCESS или SQL_SUCCESS_WITH_INFO – в случае нормального завершения; SQL_NO_DATA – нет данных; SQL_STILL_EXECUTING – выполняется; SQL_ERROR или SQL_INVALID_HANDLE – в случае неудачи.

10) SQLRETURN ***SQLFreeHandle***(SQLSMALLINT HandleType, SQLHANDLE Handle);

Очищает указанный дескриптор.

Параметры:

HandleType – тип дескриптора. Аргумент должен иметь одно из следующих значений: SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT, SQL_HANDLE_DESC. В противном случае будет возвращено значение SQL_INVALID_HANDLE;

Handle – дескриптор, который необходимо очистить.

Возвращаемые значения:

SQL_SUCCESS – в случае нормального завершения; SQL_ERROR или SQL_INVALID_HANDLE – в случае неудачи.

11) SQLRETURN ***SQLGetData***(SQLHSTMT StatementHandle, SQLUSMALLINT ColumnNumber, SQLSMALLINT TargetType, SQLPOINTER TargetValuePtr, SQLINTEGER BufferLength, SQLINTEGER* StrLen_or_Ind);

Используется для получения содержимого одного столбца из текущей записи результата SQL-запроса. Очищает указанный дескриптор.

Параметры:

StatementHandle – дескриптор оператора;

ColumnNumber – номер столбца, начиная с 0;

TargetType – тип данных, для которого предназначен буфер *TargetValuePtr*;

TargetValuePtr – указатель на буфер, в который будет помещено содержимое столбца;

BufferLength – длина буфера *TargetValuePtr* в байтах;

StrLen_or_Ind – указатель на буфер, в который будет помещено количество байт, записанных в *TargetValuePtr*. Также могут быть записаны значения: SQL_NO_TOTAL (если данные были урезаны из-за недостаточного размера буфера) и SQL_NULL_DATA (если столбец пуст).

Возвращаемые значения:

SQL_SUCCESS или SQL_SUCCESS_WITH_INFO – в случае нормального завершения; SQL_NO_DATA – нет данных; SQL_STILL_EXECUTING – выполняется; SQL_ERROR или SQL_INVALID_HANDLE – в случае неудачи.

12) SQLRETURN ***SQLPrepare***(SQLHSTMT StatementHandle, SQLCHAR* StatementText, SQLINTEGER TextLength);

Подготавливает строку с SQL-запросом к последующему исполнению.

Параметры:

StatementHandle – описатель дескриптора;

StatementText – текстовая строка, содержащая SQL-запрос;

TextLength – длина строки в *StatementText*.

Возвращаемые значения:

SQL_SUCCESS или SQL_SUCCESS_WITH_INFO – в случае нормального завершения; SQL_STILL_EXECUTING – если функция продолжает выполняться; SQL_ERROR или SQL_INVALID_HANDLE –в случае неудачи.

8.3 ОСНОВЫ ТВОРЧЕСКОГО ПОДХОДА ПРИ РАЗРАБОТКЕ БАЗ ДАННЫХ

Творчество, нахождение новаторских, прогрессивных выходов из создавшейся ситуации всегда было основным условием развития общества. Творчество – необходимое условие развития материи, образования ее новых форм.

"В общепринятом смысле творчество – условный термин для обозначения психического акта, выражающегося в воплощении, воспроизведении или комбинации данных нашего сознания, в (относительно) новой форме, в области отвлеченной мысли, художественной и практической деятельности" (Ф. Батюшков). Творчество заключено не в той деятельности, каждое звено которой полностью регламентировано заранее данными правилами, а в той, предварительная регламентация которой содержит в себе известную степень неопределенности, в деятельности, приносящей новую информацию, предполагающей самоорганизацию. "Сущность творческого процесса заключается в реорганизации имеющегося опыта и формировании на его основе новых комбинаций" (А. Матейко).

"Творчество – деятельность человека, созидающего новые материальные и духовные ценности, обладающие общественной значимостью" (С.Л. Рубинштейн).

Большая роль в интенсивном развитии экономики принадлежит творческому труду инженерно-технических работников на предприятиях и в научно-исследовательских организациях. Результатами этого труда являются новые конструкторские или технологические решения, научные закономерности, физические явления, которые позволяют более полно удовлетворять насущные и, что особенно важно, будущие потребности потребителей конечного продукта.

Уровень развития производства и нарастание информационных процессов, когда специалист не в состоянии, используя традиционные методы, "переварить" такое количество информации, определяет актуальность освоения нового, более творческого подхода к организации информационно-профессиональной деятельности. При этом конкурентоспособный специалист должен обладать способностью к ранжированию информации, интуитивным чутьем на ее актуальность, умением в окружающей действительности уяснить наиболее злободневную проблему и сформулировать профессиональную задачу, определить основные информационные источники. Информация должна быть воспринята инженером, "пропущена через себя", из нее отобрано самое ценное.

Создание инновационного продукта в информационной сфере немыслимо без развития креативности специалиста. Креативность – это одна из своеобразных сторон человеческого ума, отличная от тех качеств сознательной деятельности человека, которые обозначены термином "интеллект".

Творческий процесс по созданию объектов интеллектуальной собственности – это не теоретизирование, не манипуляция понятиями и словесными формулировки, а процесс целенаправленной, практически полезной деятельности, дающей результаты сейчас, в конкретных условиях жизненных обстоятельств, которые меняются ежечасно и ежесекундно.

Творческая компетентность специалиста в области систем автоматизированного проектирования – это способность к прогрессивному преобразованию действительности на основе креативности мышления и совокупности знаний, умений, навыков по разработке программных продуктов, баз данных и средств информационных технологий, и психологической готовности к такому преобразованию в современных экстремальных внешних и внутренних условиях индивидуально и в трудовом коллективе. Показатель творческой компетентности специалиста – его важнейшее личностное качество, определяющее готовность выявлять и анализировать актуальные проблемы в научной и производственной сферах, находить способы и средства для творческого их решения.

Структура творческой деятельности по созданию объектов интеллектуальной собственности (креативный процесс) представляет собой сложное, многоуровневое, системное образование, в центре которого находится креативность как общая универсальная способность к профессиональной творческой деятельности (творческая компетентность). Определяющим компонентом креативности является соответствующий уровень интеллектуальной активности, основанный на творчестве как свойстве личности и на владении технологией творчества.

Формирование творческой компетентности, необходимой для создания подлинно творческих объектов интеллектуальной собственности, возможно через решение творческих задач в области информационных технологий. "Мышление всегда начинается с проблемы или вопроса, с удивления или недоумения, с противоречия. Этой проблемной ситуацией определяется вовлечение личности в мыслительный процесс". Следствием установки у специалиста на преодоление препятствий и решение проблемной ситуации является возникновение активной мыслительной деятельности.

Задачи, с которыми встречается инженер-системотехник как в профессиональной деятельности, так и в учебной практике весьма разнообразны по содержанию и форме, но все они включают в себя:

– предметную область – совокупность фиксированных и предполагаемых объектов разного характера, о которых явно или неявно идет речь в задаче;

– отношения, которыми связаны объекты предметной области;

– требование или вопрос – указание о цели задачи;

– оператор задачи – совокупность тех действий, которые надо произвести над условиями задачи, чтобы выполнить ее требование. Решение задачи и состоит в том, чтобы найти оператор.

В современных условиях решение творческих задач в области информационных технологий выступает не только как средство активизации и укрепления свойств и способностей, необходимых в профессиональной деятельности инженера-системотехника, но и становится специфической фор-

мой познания действительности. Человек, воспитанный в условиях творческого отношения к действительности, способен на самые неожиданные открытия и свершения, которые будут двигать общество вперед по пути прогресса.

СПИСОК ЛИТЕРАТУРЫ

1. Боуман Дж., Эмерсон С., Дарновски М. Практическое руководство по SQL. Киев: Диалектика, 1997. 320 с.
2. Вейнеров О.М., Самохвалов Э.Н. Проектирование баз данных САПР. М.: Высшая школа, 1990. Кн. 4. 144 с.
3. Грабер М. Введение в SQL. М.: Лори, 1996. 382 с.
4. Дейт К. Дж. Введение в системы баз данных. Киев: Диалектика, 1998. 784 с.
5. Диго С.М. Проектирование и использование баз данных. М.: Финансы и статистика, 1995. 208 с.
6. Дьяков И.А. Базы данных. Язык SQL. Тамбов: Изд-во Тамб. гос. техн. ун-та, 2004. 80 с.
7. Карпова Т.С. Базы данных: модели, разработка, реализация. СПб.: Питер, 2001. 304 с.
8. Масленникова Н.П., Желтенков А.В. Менеджмент в инновационной сфере: Учебное пособие. М., 2005.
9. Попов А.И. Решение творческих профессиональных задач: Учебное пособие. Тамбов: Изд-во Тамб. гос. техн. ун-та, 2004.
10. Тиори Т., Фрай Дж. Проектирование структур баз данных. М.: Мир, 1985. 287 с.
11. Ульман Дж. Основы систем баз данных. М., 1983.
12. Федорук В.Г., Черненький В.М. Системы автоматизированного проектирования. Кн. 3: Информационное и прикладное программное обеспечение. Минск: Высшая школа, 1988. 157 с.
13. Хорафас Д., Легг С. Конструкторские базы данных. М.: Машиностроение, 1990. 224 с.