МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный технический университет имени К.И.Сатпаева

Е.Ж. Айтхожаева

СИСТЕМЫ БАЗ ДАННЫХ

Рекомендовано Министерством образования и науки Республики Казахстан в качестве учебника

Алматы

2002

ББК 32.973
УДК 681.3.06
Айтхожаева Е.Ж. Системы баз данных: Учебник.
- Алматы: КазНТУ, 2002. 226 с.
ISBN 9965 - 585 - 43 - 1

В учебнике рассматриваются теоретические аспекты систем баз данных (СБД) и практические вопросы проектирования баз данных (БД) и информационных систем – приложений баз данных.

Излагаются основополагающие сведения о СБД, рассматриваются основные модели данных в теории баз данных, языковые средства СБД, классификация СБД, теория современных реляционных систем управления базами данных (СУБД), вопросы проектирования информационных систем на основе технологии баз данных.

Обсуждаются вопросы защиты и безопасности информации в базах данных и информационных системах, стандартный язык управления данными SQL и его реализация, организация и принцип работы сервера баз данных - MS SQL Server, публикация баз данных в сети Интернет.

Особое внимание уделено xBase - языку автономного программирования dBase-подобных СУБД, применению в СБД объектно-ориентированного программирования и визуального проектирования баз данных и их приложений (на примере визуального инструментария СУБД Visual FoxPro).

Рассматривается развитие баз данных в базы знаний - системы, основанные на знаниях, в том числе экспертные системы.

По каждому разделу учебника имеются контрольные вопросы и упражнения.

Учебник предназначен для студентов вузов всех специальностей направления подготовки "Вычислительная техника и программное обеспечение", "Информационные системы" при изучении дисциплин, посвященных теоретическим и практическим вопросам проектирования баз данных и информационных систем - приложений баз данных. Он может быть использован студентами других специальностей.

Ил. 21. Табл..7. Библиогр. - 21 назв.

Рецензенты: *Д.Н.Шукаев,* проф., д-р техн. наук
И.Т.Утепбергенов, проф., д-р техн. наук
Б.Д.Хисаров, проф., канд.техн.наук

Печатается по плану издания Министерства образования и науки Республики Казахстан на 2002 г.

ISBN 9965 - 585 - 43 - 1 © КазНТУ, 2002

ВВЕДЕНИЕ

Во всех областях человеческой деятельности используется такое понятие как **информация**, так как любая взаимосвязь и координация работ возможны только благодаря информации. Человек создал естественные информационные системы, поскольку существовала насущная потребность снабжать производство информацией, необходимой при контроле и принятии решений, научился собирать эту информацию, обрабатывать и передавать её по назначению. Тем более, что в условиях рыночной экономики информация выступает как один из важнейших товаров.

Под **информацией** понимают любые сведения о каком-либо событии, сущности, процессе, являющиеся объектом таких операций, как: восприятие, передача, преобразование, хранение и использование.

Проблеме создания различных средств и методов обработки информации всегда уделялось большое внимание. Но качественный скачок в обработке информации произошел в конце 40-х годов в результате изобретения **электронных вычислительных машин** (ЭВМ) - **компьютеров.** Сейчас в некоторых развитых странах в сфере компьютерной обработки информации занято около половины трудоспособного населения, а вложенные сюда средства дают самую высокую прибыль. Компьютерная обработка информации лежит в основе **новых информационных технологий** (НИТ), массовое и эффективное применение которых определяет прогресс любого общества во всех областях деятельности человека.

Специальные программные системы, ориентированные на выполнение определенного типа обработки информации относятся к прикладным программам. С их помощью реализуются информационные технологии, базирующиеся на компьютерной обработке информации (компьютерные технологии).

В 60-е годы появился новый подход к организации процессов переработки информации, который основывается на понятии систем баз данных (СБД). Базой данных (БД) называют специальным образом организованные данные, хранимые в вычислительной системе (ВС). База данных создается для определенной предметной области (банк, биржа, магазин, склад, библиотека и т.д.). Сегодня базы данных можно встретить практически везде. Их используют в медицине, на транспорте, в правоохранительных органах, в городских справочных службах, на производстве и в учебных заведениях. Базы данных могут содержать в себе различную информацию, получить которую можно в считанные секунды, нажав для этого всего лишь несколько клавиш на клавиатуре компьютера.

Для создания и использования БД служат системы управления базами данных (СУБД), которые занимают особую позицию в мире программного обеспечения и нашей повседневной жизни. Системы управления базами данных обеспечивают реализацию новых концепций в организации

информационных служб через создание информационных систем на основе технологии баз данных. В настоящее время широко применяются муниципальные, банковские, биржевые информационные системы, информационные системы оптовой и розничной торговли, торговых домов, служб управления трудом и занятостью, базы данных рынка товаров и услуг, справочной и аналитико-прогнозной котировочной информации и др. Как правило, работа этих систем осуществляется в локальных вычислительных сетях различной архитектуры или их объединениях, получивших название корпоративных сетей, дальнейшая интеграция которых возможна с помощью глобальной сети Интернет.

Сетевая технология позволяет отдельным пользователям совместно работать с документами и проектами, дорогими сетевыми ресурсами (принтерами, жесткими и оптическими дисками), базами данных, архивами, книгами, быть в гуще мировых событий. Современные сети позволяют объединять пользователей различных типов компьютеров как с одинаковыми , так и различными операционными системами (**однородные** и **разнородные сети**). Сети становятся все более важным глобальным средством передачи информации, обмена данными и идеями.

Международная вычислительная сеть **Internet,** охватывающая большинство стран, обеспечивает реализацию единого **мирового информационного пространства.** Служба Web, являющаяся одной из основных служб, с которыми работают пользователи Internet, благодаря привлекательному графическому интерфейсу и мультимедийным средствам, позволяет **публиковать** на своих **Web-страницах** (и посылать) информацию о любом предмете, в том числе и данные из баз данных.

Грамотная организация информационно-вычислительной системы, в том числе и с использованием Интернет, позволяет не только экономить труд и время специалистов, но вместе с тем увеличивать творческую долю труда работников различных категорий, использующих данную систему в своей работе. Другими словами, использование такой системы, в первую очередь, увеличивает качество труда, а в некоторых случаях позволяет добиться прямого экономического эффекта. Технология систем баз данных позволяет с наименьшими затратами и наибольшей эффективностью организовать информационную систему для решения пользовательских задач и является одной из перспективных технологий обработки данных.

Глава 1. ОСНОВНЫЕ СВЕДЕНИЯ О СИСТЕМАХ БАЗ ДАННЫХ

1.1. ОСОБЕННОСТИ ТЕХНОЛОГИИ СИСТЕМ БАЗ ДАННЫХ

Данные – это информация, зафиксированная в определённой форме, пригодной для последующей обработки, хранения и передачи. Данные соответствуют зарегистрированным фактам об объектах или явлениях реального мира.

В традиционной технологии обработки данных каждая программа работает со своими файлами данных, что приводит к дублированию данных. Попытки избежать дублирования данных за счет использования одних и тех же файлов данных в различных программах приводят к зависимости программ друг от друга. Дело в том, что в системах традиционной обработки данных форматы хранимых данных и структура файла данных определяются программой его создающей, и должны учитываться в программах, использующих этот файл. Изменения в структуре файла данных приводят к изменениям в использующих его программах.

Чтобы избежать дублирования данных и зависимости между данными и программами необходимо иметь файлы данных, создание и изменение структуры которых не определяется программой какого-либо конкретного пользователя. Совокупность таких файлов стали называть **базой данных.** Программные системы, которые были созданы для разработки и управления базами данных получили название **систем управления базами данных.**

База данных (БД) - это специальным образом организованные массивы данных, хранимые в вычислительной системе, из определенной предметной области и не зависимые от программ их использующих. Под предметной областью (ПО) понимается часть реального мира, интересующая пользователя, – это область применения конкретной БД. Различают БД, применяемые в сфере управления предприятиями и организациями, транспортом, в медицине, научных исследованиях и т.д.

А совокупность языковых, программных, технических и организационно-методических средств, предназначенных для централизованного накопления и использования данных, называется **системой баз данных** - СБД (**database system**). В СБД выделяют два основных компонента: базы данных и системы управления базой данных (СУБД).

СУБД представляет собой совокупность языковых и программных средств, предназначенных для создания и ведения БД и программ.

На рис.1.1 представлена технология обработки данных на основе концепции БД. Прикладные программы (ПП) пользователей обращаются к СУБД на логическом уровне без учета того, как данные хранятся в БД. СУБД по запросу пользователя определяет необходимые физические файлы, посредством обращения к **описанию отображения данных,** и порядок

доступа к ним. После чтения данных информация приводится к необходимому прикладной программе виду.

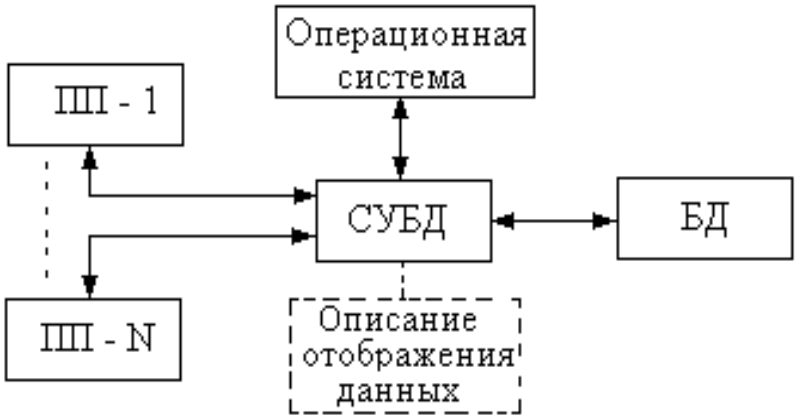


Рис.1.1. Технология обработки данных в СБД

От других способов организации данных СБД отличает ряд существенных преимуществ, таких как:

- значительное сокращение избыточности информации;
- независимость данных от программ и программ от данных;
- качественное управление данными;
- обеспечение контроля на целостность и непротиворечивость данных;
- уменьшение затрат на хранение и обработку данных.

Технология СБД имеет не только достоинства. Каждое обращение к БД осуществляется при помощи описания отображения данных, что определяет достоинства и недостатки этой технологии:

- увеличение доли служебной информации в общем объеме хранимых данных;
- повышенные требования к техническим и программным средствам системы, так как часть ресурсов расходуется на нужды самой системы;
- потеря эффективности отдельных приложений;
- последствия сбоев труднее исправлять по сравнению с традиционной технологией обработки данных.

Организация хранения и обработки данных - важнейший вопрос, стоящий перед разработчиками информационных систем. При разработке и эксплуатации СБД необходимо учитывать ряд требований, которым они должны удовлетворять:

- адекватность отображения предметной области;
- организация взаимодействия с системой пользователей разного уровня и в различных режимах;
- обеспечение секретности, надёжности и достоверности данных;
- независимость данных от программ их использующих и программ от данных;
-обеспечение приемлемых характеристик функционирования (время реализации запроса, требуемый объём памяти, сервис, стоимость системы и

т.д.).

СБД возникли в связи с потребностью в интеграции данных. Дальнейшее продвижение в этом направлении потребовало решения проблемы интеграции данных и процессов обработки, что привело к появлению **систем баз знаний.**

Знания – сложноорганизованные данные, содержащие одновременно как фактографическую (т.е. регистрация некоторого факта), так и семантическую (т.е. смысловое описание зарегистрированного факта) информацию, которая может потребоваться пользователю при работе с данными, а также знания о том, как использовать эту информацию.

В системах баз знаний решаемые задачи интегрируются как по данным, так и по их обработке, возрастает интеллектуализация этих систем, цель которой – максимальное удовлетворение потребностей пользователей и решение их проблем.

1.2. АРХИТЕКТУРА СИСТЕМ БАЗ ДАННЫХ

База данных создается для определенной предметной области и является ее моделью, так как при решении конкретных задач реальная действительность воспроизводится с существенными ограничениями, зависящими от области деятельности, поставленных целей и мощности вычислительных средств. Различают информационную модель и логическую (инфологическую, даталогическую) модель предметной области.

Информационные модели (ИМ) являются средством формирования представления о данных, их составе и использовании в конкретных условиях. Это семантические (смысловые) модели БД, так как они отражают информационное содержание конкретной предметной области. Например, информационное содержание базы данных, содержащей сведения о студентах и сданных ими зачетах.

Инфологическая модель – это детализированное описание предметной области без ориентации на используемые в дальнейшем технические и программные средства. Даталогическая модель – это реализация ИМ в конкретной СУБД с учетом всех ее особенностей.

Основу любой модели составляют объекты и взаимосвязи между ними. Объектом может быть любой предмет реального мира (человек, событие, место, документ или понятие), о котором необходимо иметь данные. У объектов, интересующих пользователя в реальном мире, есть свойства, которые и характеризуют отдельный экземпляр объекта. Сведения об объекте, которые имеют значение для данной ИМ, называются атрибутами объекта. Атрибут есть свойство объекта. Например, атрибутами, которые могут быть свойствами объекта Студент, являются: Фамилия, Имя, Номер зачетной книжки, Шифр специальности, атрибутами, которые могут быть свойствами

объекта Зачет, - Название дисциплины, Результат. Значения данных - это действительные данные, содержащиеся в каждом атрибуте. Примеры объектов, атрибутов и их значений приведены в табл. 1.1.

Таблица 1.1.

Примеры объектов, атрибутов и их значений

Объекты	**Атрибуты**	**Значения**
Студент	Фамилия	Сидоров
	Имя	Иван
	Номер зачётной книжки	99001
	Шифр специальности	3704
Зачёт	Название дисциплины	Физика
	Результат	Зачтено

Совокупность связанных по смыслу значений данных образует **запись.** Если в табл.1.1 приведёны значения данных по одному и тому же студенту, то столбец значений образует запись. В зависимости от того, какие атрибуты описывают объект, значения данных могут быть количественными, качественными или описательными.

Перечень типов объектов, относящихся к БД, их атрибутам, связям между ними и способов, с помощью которых объекты и связи отображаются на более низком уровне, называются схемой (или структурой) БД.

При работе с БД разные люди и разные программы, запускаемые этими людьми, видят БД по-разному. Конечный пользователь работает в каком-то приложении и скорее всего ничего не знает о внутренней структуре. Программа, которую использует этот пользователь и, соответственно, программист, написавший эту программу, видят и используют ту часть базы данных, которую им разрешено видеть. Администратор, позволяющий тем или иным пользователям видеть те или иные данные, видит всю схему базы данных. Группа программистов, разработавшая ядро СУБД, и предоставившая администратору возможность создавать схему базы данных, реализовала отображение базы данных на внешние устройства и в память.

Поэтому для описания информационной задачи используются три вида представления моделей данных:

- **Концептуальное** - охватывающее всю задачу с точки зрения администратора информационной системы, т.е. лица, ответственного в целом за работу с БД;
- **Внешнее** - отражает представление о задаче с точки зрения конкретного пользователя, т.е. лица, решающего узкую задачу работы системы на конкретном рабочем месте (каждая информационная система будет иметь несколько внешних представлений модели данных - A, B, C и т.д.);
- **Внутреннее** - отражает представление об информационной

задаче разработчика (программиста) с учётом особенностей и возможностей конкретной СУБД и компьютеров, на которых будет реализовываться решаемая задача.

В соответствии с этим выделяют несколько уровней представления данных, которые в совокупности образуют архитектуру системы баз данных.

Стандартная архитектура системы баз данных (стандарт ANSI - American National Standart Institut, затем ISO - International Standart Organization) имеет три уровня архитектуры: **внешний, концептуальный, внутренний.** На каждом уровне используется своя модель (представление), состоящая из записей (каждая запись состоит из данных об одном экземпляре объекта). На рис.1.2 представлена стандартная трехуровневая архитектура системы баз данных.

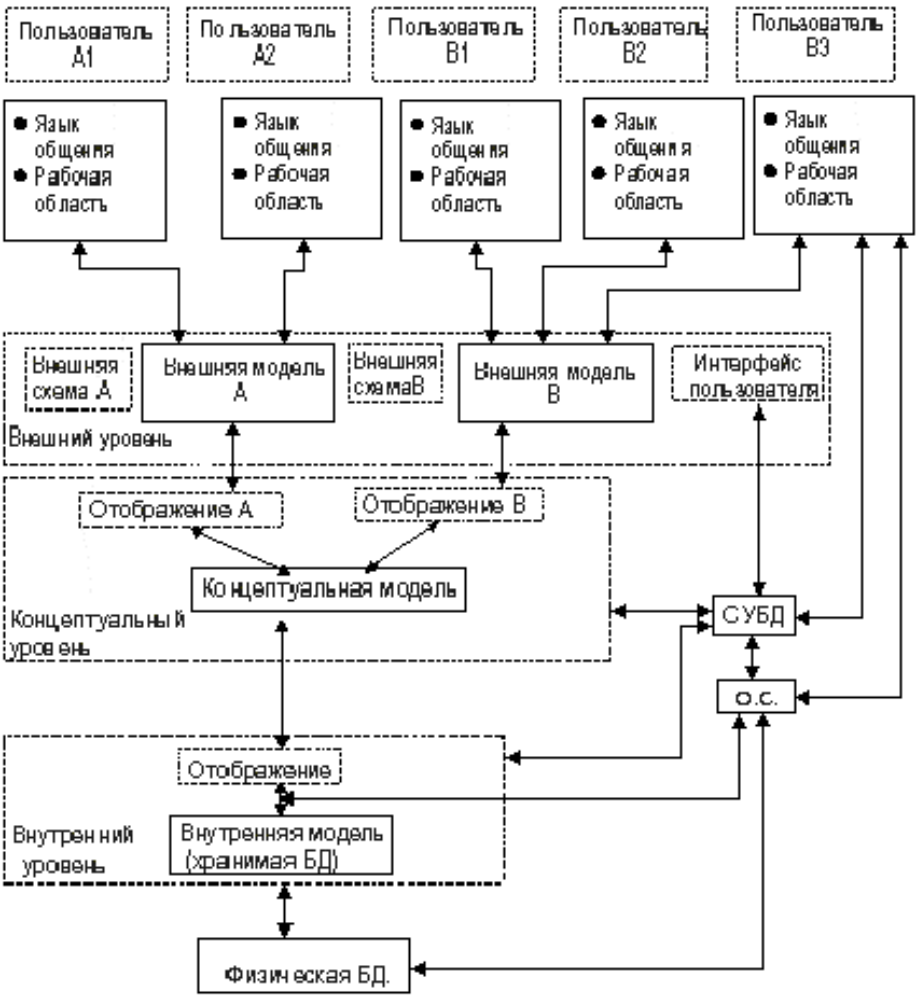


Рис.1.2. Стандартная архитектура СБД

Уровень конечного пользователя – **внешний**, прикладной (пользовательский) **уровень**. Именно на этом уровне работает пользователь, то есть человек, для которого и пишется программа. Пользователь видит базу данных как набор меню и команд, выполняя которые он решает его

собственную задачу, например, резервирует два билета до Москвы на 12 мая этого года. На этом уровне используется **внешняя модель.**

Внешний уровень наиболее близок к пользователям и связан с представлением пользователя о данных. Каждый пользователь обеспечивается рабочей областью, которая используется для приёма или передачи данных между пользователем и базой данных. Для прикладного программиста рабочей областью является область ввода-вывода. Для пользователя за терминалом эта область может состоять из рабочей памяти, выделенной терминалу. Пользователь имеет язык общения для работы с данными через рабочую область. Для каждого пользователя может существовать своя внешняя модель, в зависимости от того, что его интересует. Внешняя модель является информационным содержанием базы данных в том виде, в каком его представляет конкретный пользователь, т.е. для пользователя - внешняя модель есть база данных.

Каждая внешняя модель определяется посредством **внешней схемы,** которая в основном состоит из описаний всех типов внешних записей этой внешней модели. Внешняя схема определяет типы **внешних записей**, из которых состоит внешняя модель.

Уровень программиста и администратора – **концептуальный уровень.** На этом уровне работает программист, создающий прикладные программы и администратор, разрабатывающий структуру (схему) базы данных. Администратор видит всю схему, ему доступна вся информация. Программист может видеть только часть схемы. В принципе, его видение является некоторым подмножеством того, что видит администратор. На концептуальном уровне используется **концептуальная модель.**

Концептуальная модель – объединение всех внешних моделей. Концептуальная модель определяется посредством **концептуальной схемы,** которая включает определения каждого типа **концептуальных записей**. Из концептуальной схемы можно получить несколько подсхем, ориентированных на разного пользователя (например, одна подсхема ориентирована на бухгалтера, другая - на кладовщика).

Концептуальная модель есть представление полного информационного содержания базы данных в несколько абстрактной форме по сравнению со способом физического хранения данных. Концептуальная модель состоит из множества экземпляров различных типов концептуальных записей. Концептуальная запись не обязана быть такой же, как внешняя запись или как хранимая запись.

Внутренний уровень наиболее близок к физической памяти, т.е. связан со способом фактического хранения данных. **Внутренняя модель** состоит из различных экземпляров типов **внутренних записей**. Внутренняя модель описывается посредством **внутренней схемы**, которая не только определяет различные типы хранимых записей, но и то, какие индексы существуют, как представлены хранимые поля, какова физическая

последовательность хранимых записей и т.д.

Внутренняя схема определяет типы внутренних (хранимых) записей (например, наименование товара – символьная запись, остаток – числовая и т.д.), индексов, а также их последовательность. Поэтому внутреннюю схему называют также **структурой хранения данных.** Внутренние записи - это записи на языке программирования, который использовался для создания СУБД (например, язык программирования С). Как правило, внутреннее представление не только скрыто от посторонних глаз, но и является частной, закрытой информацией фирмы - производителя СУБД.

Уровень реализации - **физический уровень**. На физическом уровне определяется, как хранятся данные и как осуществляется доступ к ним. **Физическая база данных** хранит **физические записи.**

При переходе от одного уровня представления данных к другому могут потребоваться **отображения данных**. Отображения используются в архитектуре базы данных в том случае, если данные хранятся не в том виде, в котором они требуются. Например: данные хранятся в двоично-десятичной системе счисления, а внешней модели (пользователю) они необходимы в десятичной системе счисления.

Система управления базами данных (СУБД) - это совокупность языковых и программных средств, предназначенных для создания и ведения баз данных. Современные СУБД снабжаются также специальными инструментальными средствами для разработки прикладных программ. СУБД управляет всем доступом к базам данных и осуществляет контроль полномочий каждого пользователя. СУБД при обслуживании поступившего запроса выполняет следующие действия:

-принимает запрос;

-обследует по очереди внешнюю схему, отображение внешнее-концептуальное, концептуальную схему, отображение концептуальное-внутреннее, внутреннюю схему, в которой определяется месторасположение данных;

-передает управление операционной системе (ОС), ОС находит физические данные в физической базе данных (ФБД) и передает их на внутренний уровень;

-СУБД выполняет необходимые операции над данными.

Более подробно реализация запроса описана ниже в 1.3.3.

1.3. ВЗАИМОДЕЙСТВИЕ КОМПОНЕНТОВ СИСТЕМ БАЗ ДАННЫХ

1.3.1. Состав СБД

Система баз данных (СБД) представляет собой сложную человеко-машинную систему, включающую в свой состав различные взаимозависимые и взаимосвязанные компоненты. Можно сказать, что это компьютеризированная система хранения записей, включающая в свой состав комплекс специальных методов и средств (математических, информационных, программных, языковых, организационных и технических) для поддержания динамической информационной модели предметной области с целью обеспечения информационных запросов пользователей.

СБД является организационно-технической системой, состоящей из баз данных, технических (аппаратное обеспечение) и программных (программное обеспечение) средств формирования и ведения этих баз данных, а также пользователей баз данных. Среди пользователей различают **администраторов БД (АБД), прикладных программистов** и **конечных пользователей**. АБД – это специалист или коллектив специалистов, обеспечивающий функционирование системы.

Ядром СБД является база данных. **База данных** представляет собой поименованную структурированную совокупность связанных данных, относящихся к конкретной предметной области и находящихся под централизованным программным управлением.

Под **техническими средствами** (ТС) понимается аппаратное обеспечение СБД: вычислительная машина (процессоры, каналы связи, устройства ввода-вывода, внешние запоминающие устройства и др.), технические средства сети, аппаратура передачи данных и т.д. В СБД могут использоваться вычислительные машины любого класса, но наибольшее количество СБД функционирует на персональных компьютерах.

Программные средства (ПС) включают в себя систему управления базами данных (СУБД), трансляторы с языков СБД, различные вспомогательные программы (утилиты), средства разработки приложений, программные средства для взаимодействия с техническими средствами и т.д.

К компонентам СБД относится также **словарь СБД**, который содержит информацию о всей системе (метаданные): исходные и объектные формы всех существующих схем (внешних, концептуальной и т.д.) и отображений, а также установленные ограничения защиты и целостности данных. Метаданные необходимы для обеспечения правильной работы системы. Словарь данных иногда называют директорием, каталогом, репозиторием, энциклопедией данных. Функцию ведения словаря данных поддерживает СУБД.

В состав СБД входят организационно-методические и языковые средства. **Организационно-методические средства** СБД состоят из

нормативно-технологических и инструктивно-методологических материалов по организации и использованию СБД. **Языковые средства** предназначены для общения пользователей с СБД. Они служат для описания различных компонент СБД, для обращения к нужным частям СБД и т.д. В СБД в соответствии с функциональными требованиями систем баз данных, операторы описания и процедурные операторы можно разделить и представить в виде таких языков, как языки описания данных и языки общения с базой данных.

Языки описания данных (ЯОД) используются для логического описания данных и должны выполнять следующие функции: идентифицировать тип данных, присваивать уникальное имя каждому типу элемента данных, определять ключи, специфицировать порядок вхождения одних данных в другие, устанавливать отношения между данными. Среди них можно выделить языки описания схем, языки описания подсхем, языки описания структуры хранения, языки описания внешних данных.

Языки описания схем позволяют описывать логическую структуру данных со всеми взаимосвязями.

Языки описания подсхем устанавливают соответствия между логическим описанием данных в схеме и данными, требуемыми пользователем.

Языки описания структуры хранения данных описывают методы адресации и индексирования, способы размещения данных, способы доступа к ним, способы управления памятью и т. д.

Языки описания внешних данных используются крайне редко, существуют лишь в некоторых неалгоритмических языках высокого уровня.

Очень большое значение имеют языки общения с базой данных, которые описаны ниже.

Базовый язык - это язык программирования, использованный для создания СУБД.

Язык манипулирования данными (ЯМД) обеспечивает интерфейс между пользователем и СУБД. Если манипулирование данными встраивается в базовый язык, тогда базовый язык является языком программирования СУБД. В настоящее время используются так называемые автономные языки программирования, в которые встраиваются ЯМД, что облегчает процесс программирования.

Языки запросов - терминальные языки, которые обеспечивают взаимодействие с СУБД специалистов - непрограммистов (например, язык запросов SQL).

В современных СУБД это деление является чисто функциональным. Обычно при реализации операторы ЯОД и ЯМД объединяются в **единую языковую среду.**

1.3.2. Взаимодействие компонентов СБД

На рис.1.3 представлена схема последовательности шагов взаимодействия компонентов СБД, описание которых приведено ниже.

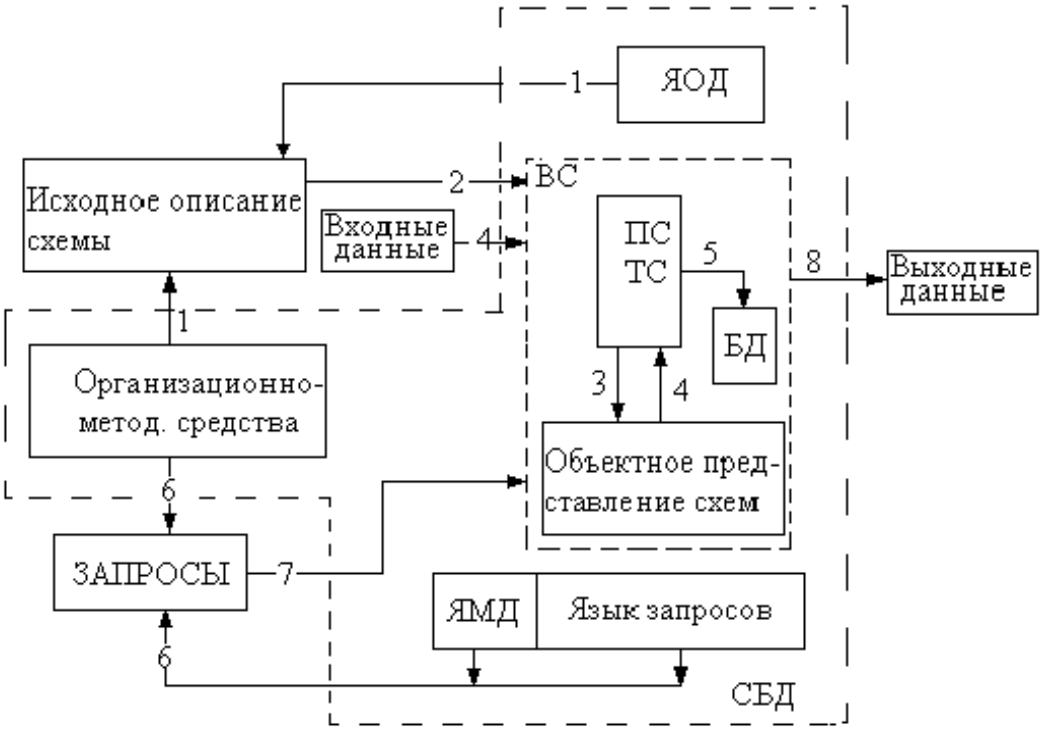


Рис. 1.3. Взаимодействие компонентов СБД

1. С использованием языка описания данных (ЯОД) и в соответствии с организационно – методическими средствами составляется исходное описание схем моделей.

2. Описанные схемы переносятся на машинные носители и вводятся в вычислительную систему (ВС).

3. Оттранслированные описания, переведенные в объектное или загрузочное представления, записываются в библиотеки СУБД.

4. Подготавливаются и вводятся в ВС входные данные в соответствии с объектным представлением схем.

5. Осуществляется загрузка БД данными.

6. Формируется запрос к БД на языке общения с БД, руководствуясь организационно- методическими средствами.

7.Запрос вводится в систему и реализуется.

8. Результаты реализации запроса выдаются в виде выходных данных.

Непосредственная реализация запроса к БД (см.п.7) осуществляется под управлением СУБД и описана ниже.

1.3.3. Реализация запроса в СБД

СУБД - система управления базой данных, под управлением и контролем которой осуществляется не только создание и ведение базы данных, но и реализация запроса. На рис.1.4 представлена схема последовательности шагов реализации запроса в СБД, описание которых

приведено ниже.

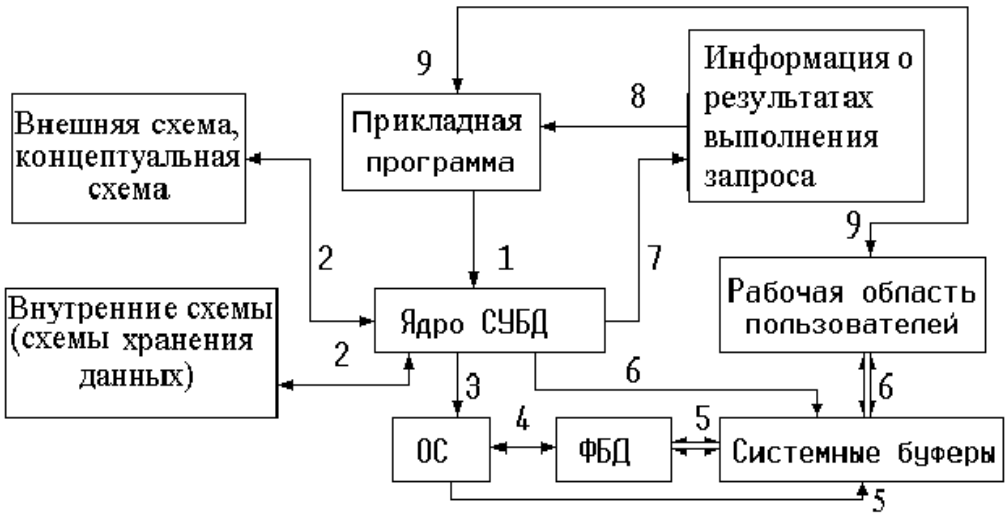


Рис.1.4. Реализация запроса в СБД

1. Прикладная программа - прикладной процесс (ПП) обращается к СУБД посредством операторов ЯМД.

2. СУБД принимает запрос, анализирует и дополняет его данными из схемы и подсхемы, на которые ссылается ПП. Для этого СУБД обследует по очереди внешнюю схему (и выполняет отображение внешняя модель - концептуальная модель), концептуальную схему (и выполняет отображение концептуальная модель - внутренняя модель), внутреннюю схему, по которой определяет, какие данные запрошены пользователем.

3. СУБД обращается к операционной системе (ОС) для выполнения физических операций ввода-вывода необходимых данных.

4. Под управлением ОС происходит поиск необходимых данных в физической памяти (физической БД).

5. Найденные данные передаются в системные буферы.

6. Из системных буферов под управлением СУБД данные передаются в рабочую область пользователя, при этом СУБД осуществляет следующие отображения: внутренняя модель - концептуальная модель, концептуальная модель - внешняя модель.

7. СУБД выдаёт информацию о результатах выполнения запроса.

8. Информация воспринимается и анализируется ПП, принимается решение о путях продолжения процесса.

9. Полученные данные, находящиеся в рабочей области пользователя, обрабатываются прикладной программой и передаются пользователю.

Следует отметить, что системные буферы используются совместно несколькими программами; управляет системными буферами СУБД.

1.4. КЛАССИФИКАЦИЯ СИСТЕМ БАЗ ДАННЫХ

Классификацию СБД можно проводить по многочисленным критериям, ниже приведена классификация по некоторым из них.

По числу уровней в архитектуре выделяют системы БД одноуровневые, двухуровневые, трёхуровневые и многоуровневые. Трехуровневая архитектура СБД является стандартной, но необязательной.

В зависимости **от типа схемы** данных реализуемой СУБД, системы БД делятся на форматированные (структурированные) и неструктурированные.

В структурированных СБД используются детерминированные схемы данных: типы объектов, их свойства, их взаимосвязи заранее определены. В свою очередь, структурированные СБД **по типу используемой модели** данных делятся на **реляционные, сетевые, иерархические.**

В неструктурированных СБД совокупность объектов, их свойств и взаимосвязей определяется в момент появления объектов в поле зрения СУБД. В неструктурированных системах условно можно выделить модели дескрипторные, дескрипторные с грамматикой, фреймовые модели, модели на семантических сетях.

По используемому языку общения различают СБД замкнутые и открытые. В открытых системах для программирования используется базовый включающий язык, дополненный операторами языка манипулирования данными. В замкнутых системах используется неалгоритмический язык высокого уровня, близкий к естественному языку.

В зависимости **от сферы применения** системы БД могут быть разделены на универсальные, проблемно-ориентированные и с широкой областью применения, которые наиболее распространены. Эта классификация связана с типом объектов, хранимых в базе данных. В универсальных СБД могут интегрироваться как традиционные, так и нетрадиционные данные (текст, звук, рисунок, видео, страницы HTML и др.). Проблемная ориентация проблемно-ориентированных СБД обуславливается разными причинами. Например, особенностями типов объектов только из определенной предметной области; особенностями используемых языковых средств; включением в СУБД специальных процедур обработки данных и т.д.

В зависимости **от типов объектов** (типа данных), хранимых в БД, СБД подразделяются на: административные (хранятся символьные, числовые, логические данные, даты); графические (хранятся графические данные); библиографические или полнотекстовые (хранятся тексты в целом: статьи, журналы и т.д.); комбинированные (все типы данных).

По **характеру организации хранения и обработки данных** СБД делятся на **локальные** (на одной ЭВМ) и **распределённые** (на нескольких ЭВМ, работающих в локальной или глобальной сети).В первом случае говорят о доступе к локальным данным, во втором - о доступе к удаленным данным.

Локальные данные, как правило, располагаются на жестком диске компьютера, на котором работает пользователь, и находятся в монопольном ведении этого пользователя. Пользователь при этом работает автономно, не зависит от других пользователей и никоим образом не влияет на их работу. Удаленные данные располагаются вне компьютера пользователя (пользователей) на специально выделенном для этих целей компьютере.

Распределённые СБД в свою очередь могут быть однородными или разнородными. В однородных РСБД используются СУБД и ЭВМ одного типа, а в разнородных - различного типа. Для распределённых СБД в сети по её узлам распределяются либо БД, либо БД и СУБД одновременно.

В распределённых СБД в настоящее время используются две технологии (архитектуры) обработки удаленных данных: файл-серверная и клиент– серверная.

В стандартной файл-серверной архитектуре данные располагаются на файл-сервере, который является, по сути, пассивным источником данных для рабочих станций. Файл-сервером называется компьютер, используемый как централизованное хранилище коллективно используемых файлов. Рабочие станции - это компьютеры любого класса, совместно использующие ресурсы серверов при решении своих прикладных задач. На рис.1.5 изображена схема получения и обработки данных пользователем (рабочей станцией) с файл-сервера.

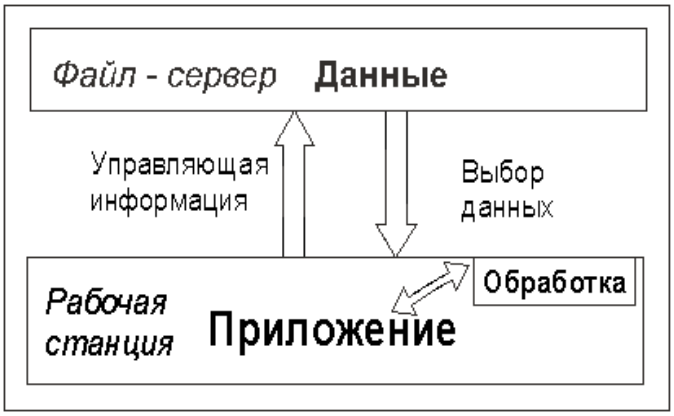


Рис.1.5. Архитектура файл-сервер

Вся ответственность за получение и обработку данных, а также за поддержание целостности базы данных лежит на приложении, запущенном с рабочей станции. При этом, поскольку обработка данных осуществляется на рабочей станции, по сети передается вся необходимая для этой обработки информация, хотя интересующий пользователя объем данных может быть намного меньше пересылаемого. Например, если пользователя интересуют сотрудники какого-либо предприятия, участвующие в конкретном проекте, его приложение получит данные о всех сотрудниках и всех проектах из базы данных, и только после этого произведет требуемую выборку.

Исторически на персональных компьютерах использовался именно этот подход как более простой в освоении. Однако большой объем передаваемых по сети данных существенно ограничивает число пользователей в сети. Этот основной и самый существенный недостаток заставил искать способы уменьшения нагрузки на сеть.

В архитектуре клиент-сервер для обработки данных выделяется специальное ядро – так называемый сервер баз данных, который принимает на себя функции обработки запросов пользователей, именуемых теперь клиентами.

Сервер баз данных представляет собой программу, выполняющуюся, как правило, на мощном компьютере. Приложения-клиенты посылают с рабочих станций запросы на выборку (вставку, обновление, удаление) данных. При этом сервер выполняет всю работу по отбору данных, отправляя клиенту только требуемые данные. Если приведенный выше пример реализовать на клиент-серверной архитектуре, то приложение - клиент получит от сервера в качестве результата данные только о тех сотрудниках, которые участвуют в указанном проекте. На рис.1.6 изображена схема получения обработанных данных пользователем (клиентом) с сервера БД.

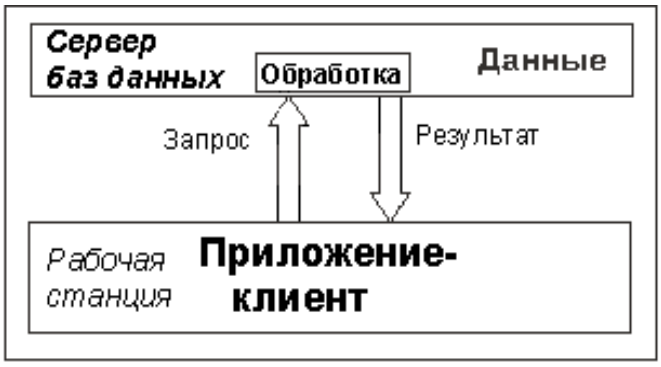


Рис.1.6. Архитектура клиент – сервер

Сервер БД (в случае реляционных баз данных обычно SQL-сервер) возвращает клиентскому приложению только требуемые данные, которые, в общем случае, составляют малую часть от общего объема БД. Поэтому в сети не наблюдается резкого увеличения нагрузки при увеличении числа клиентов. Клиентские же приложения могут выполняться на менее мощных (по сравнению с сервером) компьютерах.

Но наиболее важным результатом перехода в архитектуру клиент-сервер является гарантированное сохранение логической целостности базы данных, т.е. система становится более устойчивой и более защищенной. Достигается это благодаря возможности переложить заботу о сохранении целостности базы данных на сервер. Для этого серверы обладают большим набором встроенных механизмов, защищающих систему от неверных действий клиентов. Среди этих механизмов можно назвать такие, как

ограничение целостности, декларативная ссылочная целостность, триггеры, виртуальные таблицы (представления), авторизация пользователей и др.

Таким образом, клиент-серверная архитектура обеспечивает решение трёх важных задач: уменьшение нагрузки на сеть, уменьшение требований к компьютерам клиента, повышение надёжности и сохранение логической целостности базы данных.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что собой представляет предметная область?
2. Что собой представляют базы данных и системы баз данных?
3. Какие основные преимущества и недостатки у обработки данных на основе технологии баз данных?
4. Какой компонент систем баз данных предназначен для создания и ведения баз данных?
5. Какой компонент систем баз данных устанавливает права доступа к базе данных?
6. Что собой представляет информационная модель предметной области?
7. Сколько уровней имеет стандартная архитектура баз данных?
8. Что собой представляет внешняя модель данных?
9. Что собой представляет концептуальная модель данных?
10. Что собой представляет внутренняя модель данных?
11. Как называется уровень архитектуры баз данных, доступный пользователям любой квалификации?
12. Как называется уровень архитектуры баз данных, в схеме и модели которого содержится вся информация о данных без учета структуры хранения и стратегии доступа?
13. Какие языковые средства можно выделить в составе СУБД?
14. К какому классу языковых средств СУБД относится язык SQL?
15. К какому классу языковых средств СУБД относится язык Си?
16. Как реализуется запрос в СБД?
17. По каким критериям возможна классификация СБД?
18. В чем отличие файл-серверной технологии обработки данных от клиент-серверной технологии?
19. В чем заключается основное преимущество клиент-серверной технологии обработки данных?

Глава 2. МОДЕЛИ ДАННЫХ

2.1. СТРУКТУРЫ И СХЕМЫ ДАННЫХ

Развитие теории и практики проектирования и эксплуатации баз данных сопровождается интенсивным развитием моделей данных. Каждая конкретная СУБД поддерживает определенную модель данных. Основными компонентами модели данных являются **структуры данных, операции над данными, ограничения целостности** данных. Они тесно взаимосвязаны между собой и в различных конкретных моделях могут быть реализованы разными способами. Ниже рассматриваются общие принципы структуризации данных, правила построения структур, основные операции и понятие ограничения целостности, используемые при создании и ведении БД, независимо от типа модели данных.

Несмотря на большое разнообразие терминов, используемых для обозначения определённых типов структур данных, в настоящее время наиболее распространена терминология, предложенная КОДАСИЛ (CODASYL – the Conference on Data Systems Languages) – Ассоциацией по языкам систем обработки данных: элемент данных, группа (агрегат) данных, запись, набор, файл, база данных. Схема, представленная на рис.2.1, определяет порядок композиции этих структур модели данных.

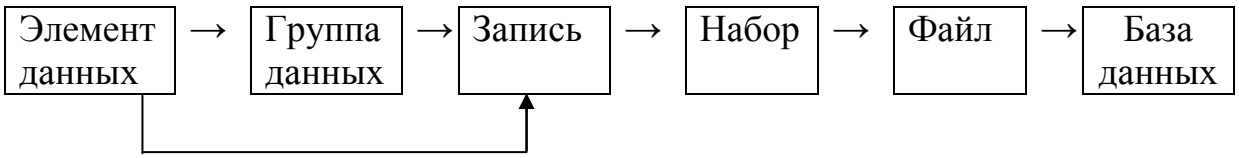


Рис.2.1. Соотношение структур модели данных

Элемент данных - наименьшая поименованная единица данных (аналог поля в файловых системах). Элемент данных - минимальная единица данных, к которой может непосредственно адресоваться СУБД и с помощью которой осуществляется построение всех остальных структур. Обычно элемент данных используется для обозначения свойства (атрибута) объекта. Значение элемента данных может быть любого типа: символьным, числовым, типа дата/время, логического типа и т.д. Например, объект - студент, свойство - дата рождения, значение элемента данных - 20.07.1982.

Группа (агрегат) **данных** - поименованная совокупность элементов данных и (или) других групп внутри записи, которую можно рассматривать как единое целое.

Простая группа состоит только из элементов данных (рис.2.2). Составная группа может включать в свой состав другие группы (рис.2.3).

Дата рождения			Имя группы
Число	Месяц	Год	Имена элементов
20	07	1982	Значения элементов

Рис.2.2. Простая группа данных

Предприятия г. Алматы				Имя состав-ной группы
Наз-вание		Адрес		Имя простой группы
	Район	Улица	Дом	Имена элементов

Рис.2.3. Составная группа данных

Существуют группы типа **вектор** (повторяющаяся компонента - простой элемент данных) и типа **повторяющаяся группа** (повторяющаяся компонента представлена совокупностью данных) (рис.2.4). В повторяющуюся группу могут входить отдельные элементы данных, векторы, группы или другие повторяющиеся группы.

Заработная плата		
За январь	.....	За декабрь

Рис.2.4. Группа типа вектор

Запись – поименованная совокупность элементов или элементов и групп, которые не входят ни в какую другую группу. Обычно запись отображает данные о свойствах экземпляра объекта (рис.2.5).

Набор - поименованная совокупность записей двух типов, образующих двухуровневую иерархическую структуру. Каждый тип набора представляет собой отношение между двумя или несколькими типами записей. Основное назначение набора - представление связей между записями.

Файл данных – упорядоченная совокупность записей. **База данных** – поименованная совокупность экземпляров записей различного типа, содержащая ссылки между записями, представленные экземплярами наборов. База данных может состоять из одного или нескольких файлов данных.

Студент					
Номер зачетной книжки	Ф.И.О.	Номер группы	Дата рождения		
			год	месяц	число
99001	Исин А.С.	ЭВМ-98	1971	11	29

Рис.2.5. Организация записи

В БД используются **групповые отношения**, которые служат для задания связей между группами в базе данных. Можно сказать, что БД – поименованная совокупность экземпляров групп и групповых отношений. В групповом отношении один тип группы - владелец отношения, а другой - член отношения. Тип группового отношения задаётся между двумя типами групп. По характеру бинарных связей между экземплярами рассматриваемых групп различают групповые отношения вида: 1:1 (один-к-одному), **1:M** (один-к-многим), **M:1** (многие-к-одному), **M:M** (многие-к-многим).

Отношение "один-к-одному" допускает в данный момент времени связь только между двумя экземплярами разных объектов. Например, объект СТУДЕНТ и объект ЭКЗАМЕНАЦИОННЫЙ БИЛЕТ - на экзамене каждому студенту выдается один конкретный билет с вопросами (рис. 2.6).

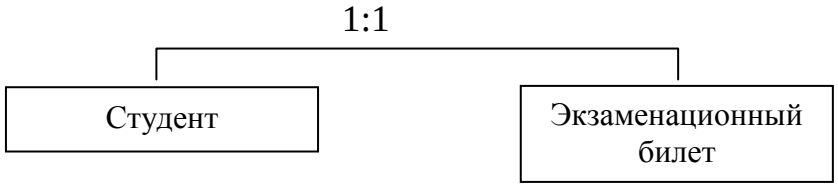


Рис.2.6. Связь один-к-одному

Отношение "один-к-многим" допускает связь одного экземпляра объекта с несколькими экземплярами другого объекта. Например, объект УЧЕБНАЯ ДИСЦИПЛИНА и объект СТУДЕНТ - одну учебную дисциплину изучает целая группа студентов (рис. 2.7).

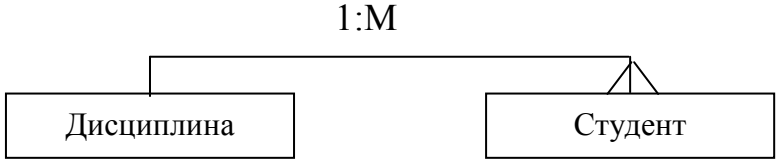


Рис.2.7. Связь один-к-многим

Отношение "многие-к-одному" допускает связь с одним экземпляром объекта нескольких экземпляров другого объекта. Например, объект СТУДЕНТ и объект ПРЕПОДАВАТЕЛЬ - много студентов обучается у одного преподавателя (рис. 2.8).

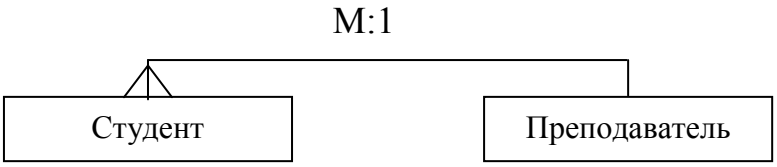


Рис.2.8. Связь многие-к-одному

Отношение "многие-к-многим" допускает связь нескольких экземпляров объекта с неограниченным количеством экземпляров другого объекта. Например, объект СТУДЕНТ и объект БИБЛИОТЕКА - разные студенты могут посещать различные библиотеки (рис. 2.9).

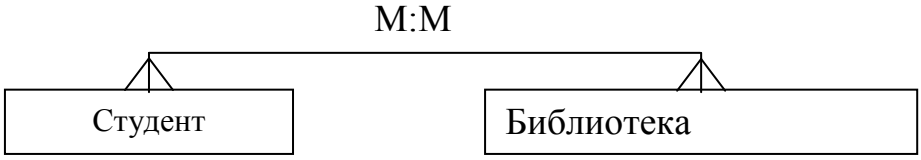


Рис.2.9. Связь многие-к-многим

Совокупность свойств, присущих данному типу структуры данных, задаёт **схема структуры данных**. Построение структур данных в каждой конкретной модели данных выполняется по определённым правилам. Схемы структур данных используются для описания типов групп и групповых отношений, т.е. описание структуры БД задаётся её схемой.

Для представления схемы данных используются чаще всего две формы представления данных**: графовая** форма, **табличная** форма.

При графовой форме представления схемы типы групп изображаются вершинами графа, а групповые отношения - соответствующими дугами.

Табличная форма представления схемы обычно представляет двумерную таблицу. С точки зрения понимания и интерпретации она более удобная для пользователя. В качестве аналогов таблиц и их элементов при машинной обработке могут выступать файлы, записи, поля.

Одним из основополагающих понятий в БД является понятие ключа. **Ключ** – уникальный элемент (атрибут) или совокупность элементов данных, которые служат для идентификации записей. Ключевой атрибут обеспечивает доступ к необходимым значениям данных, так как его значение однозначно определяет значения других атрибутов этого же объекта. Ключевых атрибутов может быть несколько, в зависимости от конкретных условий, но значение ключевого атрибута не может быть равным нулю. Например, для объекта СТУДЕНТ ключевым атрибутом является НОМЕР ЗАЧЕТНОЙ КНИЖКИ, а атрибут ФАМИЛИЯ в качестве ключа использовать нельзя, так как могут быть одинаковые фамилии.

Различия в способе представления взаимосвязей между объектами определяют тип модели данных. Для описания информационных процессов в системах со структурированными данными используют три основных типа

моделей: **иерархическую, сетевую** и **реляционную.**

2.2. ОПЕРАЦИИ НАД ДАННЫМИ И ОГРАНИЧЕНИЯ ЦЕЛОСТНОСТИ

2.2.1. Операции над данными

Динамические свойства модели данных выражаются множеством операций, которые определяют допустимые действия над некоторой реализацией базы данных для её перевода из одного состояния в другое. Это множество операций соотносят с языком манипулирования данными (ЯМД).

Любая операция над данными включает в себя: **селекцию (**выделение искомых данных) и **операцию над выделенными данными.**

Условие селекции специфицируется в виде некоторого критерия отбора данных, над которыми должно быть выполнено требуемое действие.

Селекция выполняется с использованием одного из следующих значений: с использованием логической позиции данного, с использованием значения данного, с использованием связей между данными.

Селекция с логической позицией данных основана на том факте, что все данные в БД упорядочены по позициям, номера которых содержатся в специальных объектах БД. Для обозначения логической позиции данного используются специальные указатели записей БД - идентификаторы. Эти указатели получили ещё одно название – индикаторы текущего состояния. Для реализации этого вида селекции необходимо внутренний указатель переместить на нужную запись либо указать номер логической позиции необходимой записи.

При селекции с использованием значения данного задаются условия отбора данных. Ими могут быть как простые, так и булевы условия отбора. Простые условия задаются на одном атрибуте и одном значении. К операторам условия относятся арифметические операции сравнения: <, >, <=, >=, =, <>. В булевом или логическом условии используют несколько простых условий, соединённых логическими операторами: И - AND, ИЛИ - OR, НЕ - NOT. Условие отбора данных имеет следующую структуру: <Имя_атрибута> <Условие> <Значение_атрибута>. Для определения порядка выполнения в логических условиях предусмотрена возможность использования скобок.

Например, если необходимо выделить все записи о студентах 1984 года рождения и имеющих специальность 3701 или 3706, то в этом случае условие выделения примет следующий вид:

ГОД_РОЖДЕНИЯ = 1982 AND (СПЕЦИАЛЬНОСТЬ = 3701 OR СПЕЦИАЛЬНОСТЬ = 3705).

Селекция по связям между данными основывается на использовании атрибута одного объекта для селекции данных другого объекта, если эти данные связаны между собой. Например, по атрибуту ИНОСТРАНЕЦ объекта СТУДЕНТ можно выбрать все ФАКУЛЬТЕТЫ, на которых учатся иностранные студенты.

Все операции над выделенными данными могут быть разделены на основные и вспомогательные.

К основным операциям относятся:

- **идентификация** данных и нахождение их позиции в БД;
- **выборка** (чтение) данных из БД;
- **включение** (запись) данных в БД;
- **уничтожение** (удаление) данных в БД;
- **изменение**, модификация данных в БД.

К вспомогательным операциям относятся установление и разрыв связей между структурами данных.

По способу получения результата все операции делятся на **навигационные** и **спецификационные.**

В навигационных операциях задаётся способ получения данного, то есть указывается как манипулировать значениями индикаторов текущего состояния БД. Результат получают путём прохождения по связям, реализованным в БД. Результат операции – это единичный объект БД (экземпляр записи).

В спецификационных операциях определяются требования, предъявляемые к результату операции, но не задаётся способ его получения. Результат выполнения этих операций может иметь множество значений. Для определения требований могут использоваться исчисления предикатов.

Обобщённые операции БД (процедуры БД) могут запускаться пользователем или автоматически самой СУБД. Это последовательность операций, позволяющих реализовывать определённый алгоритм обработки данных.

Свойства операций и обобщённых операций – неделимость их действия. Такие операции и процедуры называются транзакциями. Транзакция (transaction) – логический блок операций, содержащих одну или несколько инструкций, представляющих собой единое целое, который не допускает прерывания во время исполнения. Транзакция всегда должна быть выполнена до конца или не выполнена вообще.

2.2.2. Ограничения целостности

Существенным компонентом модели данных является ограничение целостности. Под ограничениями целостности понимаются логические ограничения, накладываемые на данные, с целью сохранения непротиворечивости данных и соответствия структурным спецификациям конкретных СУБД.

Проблема **целостности** – это проблема обеспечения точности и корректности данных, хранящихся в базе данных. При обеспечении целостности система должна содержать сведения о тех правилах – ограничениях целостности, которые пользователю не следует нарушать. Кроме того, СУБД каким-то образом должна следить за соблюдением

заданных правил при выполнении пользователем некоторых операций.

Обычно ограничения целостности применяются для описания базовых отношений, поскольку базовые отношения отражают реальность и, следовательно, должны регулироваться таким образом, чтобы содержать корректные или, по крайней мере, правдоподобные значения. Производные отношения также могут регулироваться ограничениями целостности, которые наследуются автоматически или могут быть заданы.

Ограничения целостности бывают:

- **внутренние** (представляются в модели данных правилами композиции структуры данных в конкретной схеме БД и описываются в структурных спецификациях);
- **явные** или внешние (ограничения задаются в явном виде администратором БД или программистом).

Ограничения целостности можно специфицировать для элементов данных, групп, записей, групповых отношений. В современных СУБД для задания явных ограничений целостности используются разнообразные правила проверки вводимых значений полей, записей, специальные триггеры вставки, удаления, модификации и другие средства.

2.3. ИЕРАРХИЧЕСКАЯ И СЕТЕВАЯ МОДЕЛИ ДАННЫХ

2.3.1. Иерархическая модель данных

Иерархическая модель данных (ИМД) наиболее органично отражает естественную иерархию данных в предметной области. Иерархическая модель организует данные в виде древовидной структуры и является реализацией логических связей по типу "целое-часть". Хорошим примером иерархической модели является любая административная структура. В этой модели выделяется главный объект, через который обеспечивается доступ ко всем остальным объектам системы. Иерархическая модель данных использует такие понятия, как: поле (элемент данных), сегмент (группа), запись.

Для представления схемы в ИМД используется графовая форма. В ИМД объекты и связи между ними представляются в виде графа типа **дерева.** В узлах дерева размещены **объекты**, а его рёбра отображают **связи** между объектами. Верхняя вершина является корнем дерева, конечные вершины – листьями. Дерево - ориентированный граф без циклов, ориентация от корня, т.е. дуги, соответствующие функциональным связям, направлены от корня к листьям дерева. В дереве любые два узла, один из которых исходный, а другой зависимый, связаны иерархической связью. Длина пути - количество дуг, которые необходимо пройти для достижения определённой вершины. Иерархический путь включает все связанные между собой узлы, начиная с корневого узла и кончая заданным. В древовидной структуре иерархические пути - линейные. Любой узел, находящийся на иерархическом пути выше

рассматриваемого узла, является для него исходным узлом (родительским). Любой узел, находящийся на иерархическом пути ниже рассматриваемого узла, является для него порождённым узлом (наследником). У каждого узла - объекта может быть несколько “наследников“ и существует один “родитель“. Узлы и функциональные связи между ними в ИМД - именованные, то есть имеют имя. Для каждого уровня связи вводится интерпретация, зависящая от предметной области и описывающая взаимоотношение между “родителями” и “наследниками”. Каждый экземпляр объекта представляется с помощью записи. Структура данных, используемая для представления этой записи, обычно содержит некоторые атрибуты - характеристики каждого объекта.

Операции в ИМД имеют нелогичный позаписный характер. Аппарат перемещения по структуре в графовых моделях служит для установки тех объектов данных, к которым будет применяться очередная операция манипулирования данными. Такие объекты называются текущими. Механизмы доступа к данным и перемещения по структуре данных в таких моделях достаточно сложны.

При использовании ИМД выделяются уровни иерархии данных. Например, автомобиль состоит из двигателя, корпуса и ходовой части; двигатель состоит из клапанов, цилиндров, свеч и т.д. На рис.2.10. представлена иерархическая модель данных для хранения данных о составных частях и деталях автомобиля. В этой модели каждая запись БД представляет конкретную деталь. Между записями существуют отношения “родитель-наследник”, связывающие каждую часть с деталями, входящими в неё.

Иерархическая модель очень хорошо подходит для устоявшихся предметных областей с четкими зависимостями “родитель-наследник”. В случае, когда эти условия выполнены, проявляются достоинства иерархической модели - очень быстрая скорость поиска. Основные достоинства ИМД: простота построения и использования, обеспечение определенного уровня независимости данных, простота оценки операционных характеристик. К достоинствам относится также возможность реализации быстрого поиска нужных значений, когда условия запроса соответствуют иерархии в схеме базы данных.

Недостатки иерархической модели проистекают оттого, что данная модель слишком жесткая. В ИМД возможны представления только двух типов отношений: “один-к-одному” и “один-к-многим”. Отношение "многие-к-многим" реализуется очень сложно, дает громоздкую структуру и требует хранения избыточных данных, что особенно нежелательно на физическом уровне, иерархическая упорядоченность усложняет операции удаления и включения, доступ к любой вершине возможен только через корневую, что увеличивает время доступа. Если запрос не соответствует имеющейся иерархии, то и его программирование, и его исполнение потребуют значительных усилий. Недостатком иерархической модели

является также сложность внесения в нее изменений. Если изменились условия задачи, и модель предметной области перестала быть иерархической, то приведение схемы базы данных в соответствие предметной области становится нетривиальной задачей.

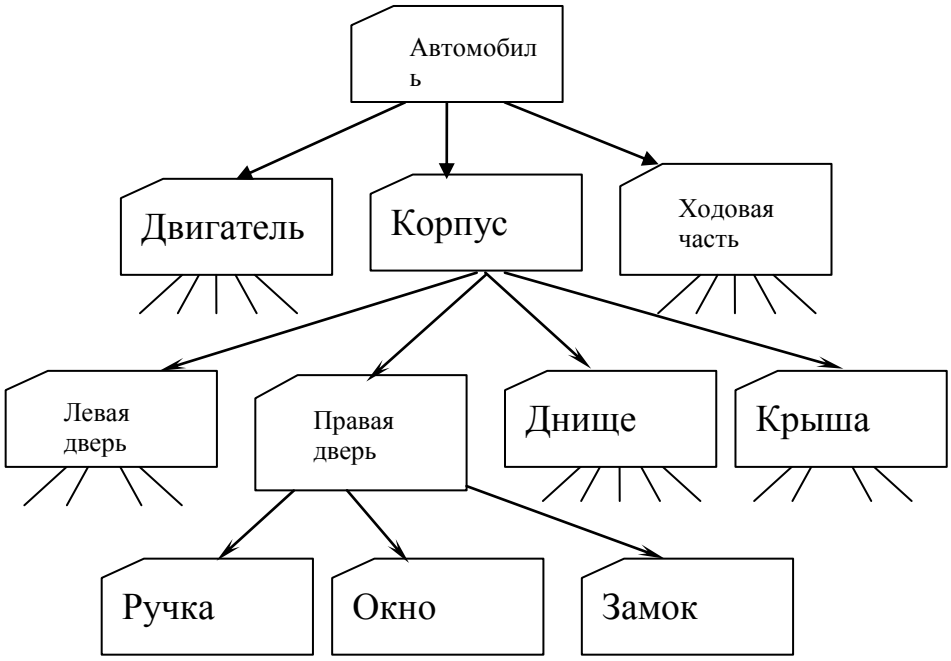


Рис.2.10. Графическое представление ИМД для описания состава изделия

Что касается способов описания конкретных схем, базирующихся на иерархической модели, или языков манипулирования данными, работающими с такой моделью, то они зависят от конкретной реализации. Классическим примером СУБД, использующей иерархическую модель, является давно разработанная система IMS фирмы IBM. К числу СУБД иерархического типа можно отнести СУБД PC/Focus, Team-Up, Data Edge.

2.3.2. Сетевая модель данных

Если структура данных оказывалась сложнее, чем обычная иерархия, простота структуры иерархической базы данных становилась её недостатком. В связи с этим возникла необходимость в разработке **сетевой модели данных** (СМД). Сетевая модель позволяет любому объекту быть связанным с любым другим объектом. Сетевые модели образуют сложные структуры, что создаёт определённые трудности при необходимости модернизации или развития модели данных. Сетевая модель данных замышлялась как инструмент для пользователей баз данных - программистов. В связи с этим в СМД больше внимания уделяется структуризации данных, чем развитию ее операционных возможностей.

Сетевую модель данных (СМД) можно рассматривать, как более общий

случай ИМД, в которой разрешены циклы и всевозможные связи. По своим базовым концепциям ИМД и СМД очень похожи. В сетевой модели, так же как и в иерархической модели, есть понятие узла и связи, которая может быть именована. Главное отличие сетевой модели от иерархической заключается в том, что к каждому узлу может идти связь не от одного родителя, а от нескольких.

Доминирующее влияние на развитие сетевой модели данных и соответствующих СБД, построенных на основе этой модели, оказали предложения рабочей группы по базам данных (DBTG) Ассоциации по языкам систем обработки данных (КОДАСИЛ). **Модель КОДАСИЛ** постоянно развивается и совершенствуется. Эта модель считается наиболее развитой сетевой моделью данных и лежит в основе построения всех СБД сетевого типа. Ниже использована терминология КОДАСИЛ. Основные типы структур данных, используемых при построении СБД сетевого типа: элемент данных, агрегат (группа), запись, набор, база данных.

В сетевых моделях данных объекты и связи между ними представляются в виде **сети**. Сеть – ориентированный циклический граф с помеченными вершинами и дугами. Каждая вершина данных описывает некоторое понятие из предметной области (объект), представляет собой запись и характеризуется некоторыми атрибутами.

В узлах сети расположены **объекты**, а рёбра отображают **связи** между объектами. Таким образом задается логика взаимоотношениями между данными в реальной предметной области. Сетевые структуры могут содержать **циклы**, когда связи между объектами образуют замкнутый контур, начинаясь и оканчиваясь у одного и того же объекта на сети. Связи от каждой вершины могут быть направлены на произвольное количество других вершин. На каждую вершину могут быть направлены связи от произвольного числа других вершин (объектов).

Записи в графовой форме схемы БД изображаются вершинами сети и используются для представления типов объектов. Элементы данных используются для представления атрибутов объектов. Для каждой записи (запись - это часть схемы) в реальной базе данных может существовать несколько экземпляров. Соответственно, с каждым конкретным экземпляром по конкретной связи может быть связано разное число экземпляров другой записи, но число видов связи одинаково для всех экземпляров одной записи.

Наборы представляют типы связей между типами объектов, которые представлены соответствующими записями. На схеме БД они изображаются поименованными дугами между соответствующими записями. Дуга исходит из записи - владельца набора и заходит в запись - член набора. Набор - основной структурный элемент, с помощью которого выполняется построение структуры БД сетевого типа.

При построении структуры БД сетевого типа исходят из следующих основных композиционных правил: база данных может содержать любое

количество записей и наборов; между двумя типами записи может быть определено любое количество типов наборов; тип записи может быть владельцем и одновременно членом нескольких типов наборов.

В СМД реализованы все возможные типы групповых отношений: 1:1, 1:M, M:1, M:M, поэтому они используется при наличии самых различных связей между объектами. Например, в базе данных для хранения заказов один заказ мог участвовать в трёх различных отношениях “родитель-наследник”, связывающих заказ с клиентом (разместившим его), со служащим (принявшим его) и с заказанным товаром, что иллюстрирует рис.2.11. Такие структуры данных не представимы иерархическими моделями.

Существует большое число удачных реализаций сетевой модели данных. В каждой из реализации, как правило, используется свой собственный язык для описания схемы базы данных и доступа к данным. Язык CODASYL является классическим примером работы с сетевыми моделями данных и является попыткой стандартизировать язык для работы с сетевой моделью данных.

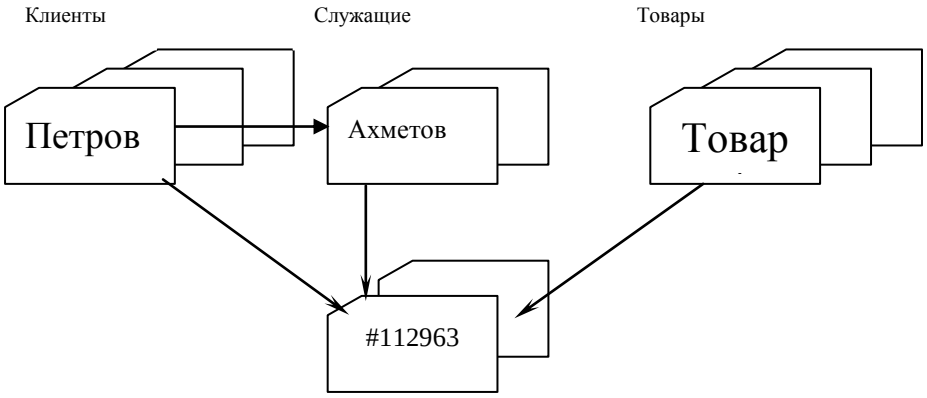


Рис.2.11. Графическое представление СМД для описания заказов

Сетевые базы данных обладают рядом преимуществ:

Гибкость. Множественные отношения “родитель-наследник” позволяют сетевой базе данных хранить данные, структура которых сложнее простой иерархии.

Стандартизация. Появление стандарта CODASYL, популярность сетевой модели обусловили тот факт, что такие поставщики мини-компьютеров, как Digital Equipment Corporation и Data General, реализовали сетевые СУБД.

Быстродействие. Вопреки своей большой сложности, сетевые базы данных достигают быстродействия, сравнимого с быстродействием иерархических баз данных. Множества представлены указателями на физические записи данных, и в некоторых системах администратор может задать кластеризацию данных на основе множества отношений.

Конечно, у сетевых баз данных есть недостатки. Как и иерархические базы данных, сетевые базы данных имеют жесткую структуру. Наборы отношений и структуру записей необходимо задавать заранее. Изменение структуры базы данных означает перестройку всей базы данных.

В 70-х годах независимые производители программного обеспечения реализовали сетевую модель в таких продуктах, как IDMS компании Cullinet, Total компании Cincom и СУБД Adabas, которые приобрели большую популярность. Наиболее известной из СУБД сетевого типа в настоящее время является система DB_VISTA III, поддерживающая модель данных CODASYL. Она реализована на языке С и может эксплуатироваться в среде операционных систем MS DOS, OS/2, UNIX, XENIX, ULTRIX и ряде других ОС на персональных компьютерах IBM PC, PC/2, SUN и VAX.

2.4. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

2.4.1. Основные понятия реляционной модели данных

В конце 60-х годов одновременно несколькими авторами были выдвинуты предложения об использовании теоретико-множественных операторов для организации доступа к данным, что и легло в основу реляционной модели данных. Реляционная модель представляет объекты и взаимосвязи между ними в виде таблиц. Впервые она была предложена доктором Э. Ф. Коддом в 1970 г. Реляционная модель данных (РМД) является в настоящее время самой распространенной. На этой модели данных базируются практически все современные СУБД для ПЭВМ. Существенными преимуществами реляционной модели является простота представления данных, естественный характер формирования запросов, так как они не привязаны жестко к заранее определённой структуре, и высокая степень независимости данных, так как интерфейс пользователя не связан с деталями физической структуры памяти и стратегией доступа.

В основе реляционной модели данных лежит математическое понятие **отношения,** представляющего собой подмножество декартова произведения **доменов**. Отношение определяется на множестве **атрибутов.** Домен – это множество значений, из которых извлекаются фактические значения атрибута.

По-английски отношение называется **relation,** отсюда и название реляционная. В реляционной модели данных объекты и связи между ними описываются в виде отношений, которые представляются двумерными таблицами. Каждому отношению соответствует своя таблица. Разница между таблицей в привычном смысле и понятием отношения заключается в том, что в отношении нет порядка – это, вообще говоря, неупорядоченное множество записей. Это, конечно, не значит, что там нет никакого порядка, просто он не подвластен ни программисту, ни администратору. Порядок определяется не отношением, а конкретной выборкой из отношения. Из одного и того же

отношения можно выбрать данные в порядке возрастания зарплаты, в алфавитном порядке фамилий и т.д.

Столбцы таблицы являются **атрибутами**. Каждая строка таблицы образует **кортеж** (или **запись**). **Домен** есть множество значений, принадлежащих одной области определения. Значения атрибута берутся из домена, т.е. атрибут является подмножеством значений домена. Элементами отношения являются кортежи. Длина кортежа - количество значений в кортеже, взятых из различных доменов (столбцов таблицы). В отношении не может быть двух одинаковых кортежей. Длина кортежа определяет **арность отношения**. Для выполнения операций над отношениями используются реляционная алгебра и реляционное исчисление.

Для эффективной работы РМД на отношения накладывается ряд ограничений, одним из которых является условие нормализации. Существует несколько уровней нормализации, обязательным из которых является первый - **первая нормальная форма** (1НФ). Остальные уровни являются желательными, но необязательными. Отношение, находящееся в 1НФ, называется нормализованным. Отношение является нормализованным, если значение атрибута в каждом кортеже является неделимым, т.е. значение атрибута не может быть множеством. Нормализация отношений выполняется декомпозицией их схем.

Достоинством реляционной модели является сравнительная простота инструментальных средств ее поддержки, недостатком – жесткость структуры данных (невозможность, например, задания строк таблицы произвольной длины) и зависимость скорости работы от размера базы данных. Для многих операций, определенных в такой модели, может оказаться необходимым просмотр всей базы данных.

2.4.2. Операции над отношениями в реляционной модели

В реляционной модели для получения информации из отношений используются языки манипулирования данными на основе реляционной алгебры и реляционного исчисления.

Реляционная алгебра (РА) включает пять основных операций над отношениями, которые рассмотрены ниже.

Объединение отношений - эта операция применяется только к отношениям одной и той же арности. Результат объединения - отношение той же арности. Запись операции объединения отношений R1 и R2 на языке РА имеет следующий вид: R=R1 U R2. На рис.2.12 представлены исходные отношения R1 и R2 и результат их объединения.

R1

a	b	c
s	d	f

R2

s	d	f
q	w	r

R=R1 U R2

a	b	c
s	d	f
q	w	r

R=R1-R2

a	b	c

Рис.2.12. Объединение R=R1 U R2 и разность R=R1 - R2 отношений

Разность отношений - эта операция также относится к основным операциям над отношениями. Разностью отношений R1 и R2 называется множество кортежей, принадлежащих R1, но не принадлежащих R2. Отношения R1 и R2 должны быть одинаковой арности. Запись этой операции на языке РА: R=R1 - R2. На рис.2.12 представлены исходные отношения R1 и R2 и результат их разности.

Декартово произведение отношений. Если отношение R1 имеет арность k1, а отношение R2 - арность k2, то декартовым произведением отношений R1 и R2 является множество кортежей арности (k1+k2). Причём первые k1 элементов образуют кортеж из отношения R1, а последние k2 элементов - из отношения R2. Запись на языке РА имеет вид R=R1xR2. На рис.2.13 представлены исходные отношения R1 и R2 и их декартово произведение R.

R1

a	b	c
s	d	f

R2

s	d	f
q	w	r

R=R1 x R2

a	b	c	s	d	f
a	b	c	q	w	r
s	d	f	s	d	f
s	d	f	q	w	r

Рис.2.13. Декартово произведение отношений R=R1x R2

Операция **проекция отношений** заключается в том, что из исходного отношения выбираются указанные столбцы и компонуются в указанном порядке. Рассмотрим проекцию отношения R1 на компоненты i1, i2, ..., ir. На языке РА запись этой операции имеет вид: R= $\Pi_{i1,i2,...ir}$ (R1), где i1,i2,...,ir -номера (имена) столбцов отношения R1. На рис.2.14 представлено исходное отношение R1 и его проекция R=$\Pi_{1,2}$ (R1).

Селекция отношения по формуле F сводится к выбору кортежей отношения, удовлетворяющих условию, заданному в формуле F, которая может быть образована:

- операндами, являющимися номерами (именами) столбцов;
- логическими операторами И (^), ИЛИ(v), НЕ(¬);
- арифметическими операторами сравнения <, >, >=, <=, <>,=.

На языке РА эта операция в общем виде записывается как R=σF(R1). На рис.2.14 представлено исходное отношение R1 и селекция отношения

R1 по формуле F: 1=a v 3=c.

R1

a	**b**	**c**
s	**d**	**f**
q	**w**	**r**

$R = \Pi_{1,2}$ (R1)

a	**b**
s	**d**
q	**w**

$R = \sigma_{1=a \text{ v } 3=r}$ (R1)

a	**b**	**c**
q	**w**	**r**

Рис.2.14. Проекция $R=\Pi_{1,2}(R1)$ и селекция $R=\sigma_{1=a \text{ v } 3=r}$ (R1) отношения R1

Следующие операции РА являются вспомогательными, т.к. их можно получить с помощью основных, но они имеют самостоятельное значение: пересечение отношений, частное отношение, соединение и естественное соединение отношений.

Реляционное исчисление позволяет описать отношения и операции над ними в виде аналитических выражений путем использования ограниченного исчисления предикатов, кванторов и переменных выборки. При этом просто указывается желаемое отношение, которое необходимо получить, и не указываются операции (в отличие от реляционной алгебры), приводящие к поставленной цели.

2.4.4. Сравнение моделей данных

Все модели данных можно сравнить по ряду признаков, характеризующих их основные свойства, а также преимущества и недостатки.

По многим параметрам реляционная модель является предпочтительней, чем иерархическая и сетевая модели. РМД является наиболее удобной в общении и лёгкой в использовании, так как использует привычную табличную форму представления данных. В то время как СМД и ИМД используют не всем понятную графовую форму представления данных. ИМД и СМД являются и более сложными в реализации по сравнению с РМД. И по независимости данных на первом месте стоит РМД, затем идет СМД, потом ИМД. РМД является наиболее приемлемой также по теоретическому обоснованию, т.к. она использует хорошо отработанную теорию множеств и теорию реляционных отношений. Теоретические основы ИМД и СМД начали развиваться лишь в последнее время.

Но у реляционной модели есть существенный недостаток - она неэффективна в реализации (затраты памяти и машинного времени). В РМД в различных таблицах происходит дублирование данных и для поиска данных необходим просмотр всех отношений. Поэтому СБД на основе РМД использовали вначале только для небольших БД (тысячи записей). Но в середине 80-х годов реляционная модель данных стала преобладающей при разработке СБД. Это стало возможным благодаря разработке эффективных, быстрых и надежных методов хранения и доступа к реляционным данным.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какая структура данных лежит в основе построения любых структур данных в БД?
2. Какие операции относятся к основным операциям над данными в БД?
3. Что собой представляет схема данных?
4. Какие формы представления схем данных используются в СБД?
5. Как называется ключ в отношении, который однозначно идентифицирует запись?
6. Какая модель данных использует табличное представление данных?
7. Какая модель данных использует для представления данных граф типа “дерево”?
8. Какая модель данных использует для представления данных граф типа “сеть”?
9. Для чего в моделях данных используются групповые отношения?
10. Какие основные операции над данными определяются в модели данных?
11. Что собой представляют ограничения целостности данных и для чего они используются?
12. Какая модель данных находит наибольшее применение в современных системах баз данных и почему?
13. Что собой представляет иерархическая модель данных, какие групповые отношения возможны в ИМД, достоинства и недостатки ИМД?
14. Что собой представляет сетевая модель данных, какие групповые отношения возможны в СМД, достоинства и недостатки СМД?
15. Что собой представляет реляционная модель данных, какие разделы математики используются для манипуляций над данными в РМД, достоинства и недостатки реляционной модели данных?
16. В чем отличие реляционной алгебры от реляционного исчисления?
17. Какие операции реляционной алгебры используются при выполнении операций над данными в РМД?
18. Какие операции реляционной алгебры являются унарными, бинарными?
19. Каким условиям должны удовлетворять исходные отношения при выполнении операций объединения и разности отношений в реляционной алгебре?
20. Какая операция реляционной алгебры позволяет получить отдельные столбцы из отношения?
21. Какая операция реляционной алгебры позволяет получить из отношения кортежи, удовлетворяющие указанным условиям?

Глава 3. СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

3.1. РЕЛЯЦИОННЫЕ СИСТЕМЫ БД

Подавляющее большинство современных СБД представляют собой системы реляционного типа, т.е. использующие реляционную модель данных. Реляционные СБД (РСБД) - это компьютеризованные системы хранения записей в табличном виде. Под БД в различных РСБД понимается табличное хранение данных, но название **база данных** может объединять не только таблицы, но и производные этих таблиц (в виде отчетов, форм, виртуальных таблиц - представлений), формы запросов, программные модули и т.д. СУБД, поддерживающие реляционную модель данных, называются реляционными СУБД (РСУБД).

История реляционных СБД ведет свое начало с конца 60-х годов. Первой была экспериментальная система управления базами данных System R и использованный в ней язык SEQUEL, который можно считать непосредственным предшественником языка запросов SQL**.** Доктор Е.Ф.Кодд (E.F.Codd) разработал модель реляционной базы данных в 1970 г. как более удобное средство хранения, выборки и манипулирования данными по сравнению с иерархическими и сетевыми базами данных, которые создавали определенные трудности при проектировании и разработке - иногда было очень сложно написать подходящий запрос для доступа к данным. В реляционной модели базы данных Кодда обращение к данным происходит таким образом, как будто они сохранены в двумерной таблице.

Модель двумерной таблицы позволяет обращаться к данным как к строкам и столбцам таблицы. По этой причине данные воспринимаются значительно легче. Это связано с тем, что в каждодневной жизни часто встречаются такие таблицы с хранящимися в них данными. Например, такой таблицей является расписание самолетов (см. табл.3.1).

Таблица 3.1

Расписание авиарейсов

Время отправления	Пункт назначения	Номер рейса	Номер стойки регистрации
6:00	Москва	4521	1
6:27	Владивосток	6710	2
12:10	Симферополь	2217	1
15:34	Москва	4527	1
16:15	Воронеж	3452	2

Таблицу можно представить как двумерный массив или как набор записей одинаковой (для данной таблицы) структуры. Записи еще называют строками, рядами. Другими словами, таблица состоит из строк и столбцов (полей). Число столбцов (полей) и записей для каждой таблицы теоретически неограниченно, хотя на практике ограничения, естественно, существуют.

Превысить ограничения хорошего сервера баз данных практически невозможно - например, сервер баз данных Infomix Online DSA позволяет иметь до 32 767 столбцов и до 8 миллиардов записей в каждой таблице.

Каждая строка в таблице описывает некий отдельный объект, поля содержат характеристики - значения неких свойств (атрибутов) этих объектов. Таблица является набором записей и содержит записи, объединенные по какому-то признаку. Таблица должна удовлетворять следующим требованиям:

- каждая строка представляет собой кортеж из k-значений, принадлежащих k-столбцам (атрибутам);
- порядок столбцов фиксирован, порядок строк - безразличен;
- любые две строки различаются хотя бы одним элементом;
- строки и столбцы таблицы могут обрабатываться в любой последовательности;
- значение каждого атрибута в каждом кортеже является неделимым (атомарным), т.е. значение элемента не может быть множеством (условие нормализации).

База данных в современных РСБД – это набор таблиц. Связь между таблицами существует на мысленном, логическом уровне и определяется предметной областью. Практически связь между таблицами устанавливается за счет логически связанных данных в разных таблицах, для чего используется, так называемый, **внешний ключ.**

В модели реляционной базы данных атрибут (столбец) или набор атрибутов (столбцов), значение которого однозначно определяет строки таблицы, называется **ключом** (key). Ключ, который уникально определяет строку в таблице, называется **первичным ключом** (primary key). Он используется для идентификации записи в таблице. Если добавляется столбец, который является первичным ключом одной таблицы, ко второй таблице, то там этот столбец будет называться **внешним ключом** (foreign key). Столбец называется внешним ключом потому, что он используется только для реализации операции поиска строк с одинаковыми значениями ключа в логически связанных между собой таблицах.

В реляционных БД используются таблицы, представляющие нормализованные отношения. Нормализовать их можно посредством разбиения определенным образом. Нормализация представляет собой формальную процедуру, в ходе которой одна таблица разбивается на две или несколько в соответствии со специальной процедурой разбиения. Задачами нормализации является:

- исключение из таблиц повторяющейся информации;
- создание структуры, в которой предусмотрена возможность ее будущих изменений;
- создание структуры, в которой влияние структурных изменений на приложения, использующие информацию этой базы данных, сведено к

минимуму.

При нормализации таблицы предпринимается попытка ограничить количество повторяющихся в ней данных. Существует много уровней и типов нормализации. Наиболее распространенной процедурой является приведение базы данных к третьей нормальной форме (Third Normal Form – 3NF), поскольку в большинстве случаев этот уровень нормализации является компромиссом между полной нормализацией и простотой реализации. Существуют уровни и выше, чем 3NF, но на практике они применяются достаточно редко, т.к. сильно затрудняют разработку структур данных и снижают их функциональность. Ниже кратко описаны эти три основные формы нормализации.

Первая нормальная форма (First Normal Form – 1NF) – это основа реляционной системы. При ней требуется, чтобы таблица была двумерной и не содержала ячеек, включающих несколько значений. В реляционных системах значение атрибута не может быть множеством. Примером таблицы, не приведенной к первой нормальной форме, является табл.3.2. Приведение таблицы к 1NF, являющееся обязательным для реляционных систем, выполняется достаточно легко – при помощи простого процесса разбиения одной записи на несколько записей, но при этом в таблицу добавляется большой объем избыточных данных. В таблице, приведенной к 1NF, каждая строка содержит элементарные значения столбцов.

Таблица 3.2

Ненормализованная таблица

Наименование книги	Автор	Изд-во	Год издания	Телефон автора
Эффективная работа с СУБД	Горев А. Ахаян Р. Макашарипов С.	Питер	1997	666-66-66 222-22-22 444-44-44
Современные микропроцессоры	Корнеев В. Киселев А.	Нолидж	1998	555-55-55 333-33-33
Стандарт SQL	Дейт К.	Наука	1995	999-99-99
Введение в системы баз данных	Дейт К.	Наука	1998	999-99-99

Табл.3.3 представляет собой экземпляр нормализованной (1НФ) таблицы, которую называют универсальной таблицей базы данных. В нее включаются все представляющие интерес атрибуты, и она содержит все данные, которые предполагается размещать в этой базе данных.

Универсальная таблица является основой проектирования базы данных.

Таблица 3.3

Универсальная нормализованная таблица

Наименование книги	Автор	Изд-во	Год издания	Телефон автора
Эффективная работа с СУБД	Горев А.	Питер	1997	666-66-66
Эффективная работа с СУБД	Ахаян Р.	Питер	1997	222-22-22
Эффективная работа с СУБД	Макашарипов С.	Питер	1997	444-44-44
Современные микропроцессоры	Киселев А.	Нолидж	1998	333-33-33
Современные микропроцессоры	Корнеев В.	Нолидж	1999	555-55-55
Стандарт SQL	Дейт К.	Наука	1998	999-99-99
Введение в системы баз данных	Дейт К.	Наука	1998	999-99-99

Вторая нормальная форма (Second Normal Form – 2NF) требует, чтобы данные во всех не ключевых столбцах полностью зависели от первичного ключа или каждого поля (столбца) первичного ключа, если он является составным. Под полной зависимостью понимается то, что значение в каждом не ключевом столбце однозначно определяется значением первичного ключа. Если же в таблице имеется хотя бы одно поле, которое не зависит от величины первичного ключа, то в этот ключ необходимо включить дополнительные столбцы. Перед проверкой на соответствие 2NF таблица должна быть приведена к первой нормальной форме. Процесс приведения таблицы ко второй нормальной форме позволяет избавиться от большей части повторяющихся данных, оставшихся от первого этапа нормализации. В рассматриваемом примере для приведения таблицы ко второй нормальной форме необходимо разбить ее на две, вынеся в одну информацию о книгах (табл.3.4), а во вторую - об авторах (табл.3.5). Как легко увидеть, в табл.3.4 поле **Номер книги** однозначно идентифицирует все остальные столбцы таблиц и поэтому может использоваться в качестве первичного ключа. Аналогично, в табл.3.5 поле **Автор** позволяет однозначно определить номер телефона автора. Однако при этом оказалась потерянной информация о том, какой автор какую книгу написал.

Для решения этой проблемы необходима еще одна таблица, которая

осуществляла бы связь между двумя таблицами. Табл.3.6 позволяет сопоставить книги и авторов. Легко убедиться, что все созданные таблицы имеют вторую нормальную форму.

Таблица 3.4

Книги

Номер книги	Наименование книги	Изд-во	Год издания
1	Эффективная работа с СУБД	Питер	1997
2	Современные микропроцессоры	Нолидж	1999
3	Стандарт SQL	Наука	1995
4	Введение в системы баз данных	Наука	1998

Таблица 3.5

Авторы

Автор	Телефон автора
Горев А.	666-66-66
Ахаян Р.	222-22-22
Макашарипов С.	444-44-44
Киселев А.	333-33-33
Корнеев В.	555-55-55
Дейт К.	999-99-99

Таблица 3.6

Авторы-книги

Автор	Номер книги
Горев А.	1
Ахаян Р.	1
Макашарипов С.	1
Киселев А.	2
Корнеев В.	2
Дейт К.	3
Дейт К.	4

Третья нормальная форма (Third Normal Form – 3NF) требует, чтобы все не ключевые столбцы таблицы зависели от первичного ключа, но были независимы друг от друга. Очевидно, что для этого таблицы должны быть приведены к первой и второй нормальной формам. В данном случае все рассматриваемые таблицы уже приведены в третьей нормальной форме, поскольку они не содержат повторяющихся данных, и данные не ключевых полей зависят от значения первичного ключа.

3.2. РЕЛЯЦИОННЫЕ СУБД

СУБД реляционного типа широко применяются на вычислительных системах любого класса: от персональных компьютеров до мейнфреймов и систем с массовым параллелизмом (MPP), включая и симметричные многопроцессорные системы (SMP). Например, классическая СУБД DB2 для больших вычислительных систем фирмы IBM; набор

серверов БД Informix для симметричных мультипроцессорных систем и систем с массовым параллелизмом; распределенная СУБД Teradata для SMP и MPP-систем; высокопроизводительная СУБД Red Brick Warehouse xPP4.0 фирмы Red Brick Systems для платформ с массово-параллельной обработкой. Две последние СУБД предназначены для организации очень крупных информационных хранилищ для множества предметных областей, позволяют обслуживать сотни и тысячи пользователей, обращающихся к **терабайтам (Тб) данных. Максимальное число пользователей ограничивается лишь** конфигурацией оборудования. СУБД Red Brick Warehouse xPP4.0 имеет полный набор средств, доступных на однопользовательской версии системы Red Brick Warehouse VPT 4.0 (версия Unix), а также дополнительные средства, необходимые для работы на MPP. Функционирует на платформе RS/6000 Scalable POWERParallel (SP), поддерживающей до 128 узлов, каждый из которых может иметь более 870 Гбайт пространства памяти на жестком диске.

Но так как из всех вычислительных систем наибольшее распространение получили персональные компьютеры, то и большинство систем БД создается для них. Практически все СУБД, используемые для организации баз данных на персональных компьютерах, в подавляющем большинстве случаев являются реляционными, т.е. они используют реляционную модель данных. В настоящее время реляционные системы лучше соответствуют техническим возможностям персональных компьютеров и вполне удовлетворяют большинство пользователей. Скоростные характеристики этих СУБД поддерживаются специальными средствами ускоренного доступа к информации - индексированием баз данных.

Все современные СУБД, как правило, имеют сходный функциональный состав, в который входят: пользовательские средства - диалоговые средства для работы с данными; средства разработчика, обеспечивающие возможность создания пользовательского приложения; дополнительные средства, от состава которых зависят функциональные возможности и мощность разрабатываемых приложений.

Современные СУБД любого класса (Oracle, Sybase, Informix, Ingres, Visual dBase, Visual FoxPro, Paradox, Access и другие) имеют объектно-ориентированные графические интерфейсы конечного пользователя с применением техники меню и окон и объектно-ориентированные инструментальные среды для разработки приложений.

Интерфейс СУБД позволяет выполнять подавляющее число функций, предоставляемых СУБД пользователю и разработчику, включая:

- ввод и редактирование записей;
- создание входных экранных форм;
- проектирование форматов выходных отчетов;

- поиск записей, удовлетворяющих задаваемому условию и выдачу указываемых полей найденных записей на экран, принтер, или в файл;

- индексирование и сортировку записей;

- интерактивную генерацию приложений, включая генерацию главного меню приложения и т.д.

В связи с необходимостью разработки информационных систем на основе технологии баз данных большинство пакетов СУБД максимально учитывает потребности программистов. Разработчикам приложений предоставляется мощный встроенный язык программирования высокого уровня, относимый к языкам четвертого поколения (язык 4GL) за реализованную возможность непосредственно адресоваться к полям таблицы базы данных на диске, скрывая тем самым операции ввода-вывода, присущие традиционным языкам программирования. Используя этот язык, приближенный к естественному языку, программист может реализовать нетривиальные приложения по обработке данных, ориентированные на конкретного пользователя.

Все современные СУБД могут работать в локальных вычислительных сетях (ЛВС). Для этого в версиях систем имеются дополнительные возможности для совместного использования данных, такие как команды блокировки и разблокировки файлов и записей; команды защиты файлов, записей, полей данных паролями пользователя или паролем группы пользователей. Многие СУБД существуют в нескольких версиях, работающих в различных операционных системах.

Реляционные СУБД условно можно разделить на три класса. Первый класс - это однопользовательские (локальные) СУБД, способные работать в качестве клиента в клиент-серверной архитектуре. Второй класс - SQL-серверы, специализированные для работы в многопользовательском режиме в качестве серверов баз данных, применяющие язык SQL в качестве стандартного интерфейса с фронтальными программами на рабочих станциях. При этом в качестве фронтальных программ могут выступать как программы в среде СУБД - клиентов, так и любые программы, поддерживающие SQL-интерфейс (например, программы на языке С, электронные таблицы, издательские системы и т.д.). Третий класс - это СУБД, включающие в свой состав SQL-серверы и способные работать как на рабочей станции в однопользовательском режиме, так и на сервере в многопользовательском режиме. В таких СУБД есть клиентская и серверная части, которые устанавливаются соответственно на компьютер - клиент и компьютер - сервер.

Использование серверов БД и соответствующих СУБД позволяет обеспечить коллективный доступ к совокупности взаимосвязанных данных, интеграцию и централизацию управления данными, устранение излишней избыточности данных при минимальных нагрузках на сеть.

Перечень современных средств разработки систем автоматизации

обработки данных на основе технологии баз данных весьма обширен. В настоящее время на рынке программного обеспечения СУБД доминируют несколько фирм производителей. Для разработки систем масштаба корпорации предназначены СУБД с SQL-серверами, такие как: Oracle корпорации Oracle; Sybase компании Sybase; Informix фирмы Informix; Ingres фирмы Computer Associates и другие СУБД.

СУБД этих производителей ориентированы на сетевые операционные системы UNIX, NetWare и служат для распределенной обработки больших массивов информации. Для управления данными в этих СУБД используется стандартный язык запросов SQL, что позволяет разрабатывать мощные сетевые приложения в архитектуре клиент-сервер. Также следует отметить, что данные СУБД обеспечивают высокую степень защиты информации и обеспечения целостности данных.

Например, СУБД Oracle, широко используемая в банковской деятельности, совместима с СУБД DB2 больших вычислительных систем фирмы IBM, работает во многих средах и поддерживает все основные ОС как для клиентов, так и для серверов, включая MS-DOS, Windows, NetWare, Novell UnixWare, OS/2, большинство вариантов UNIX и VMS, что дает возможность использовать эту систему на любых платформах. Сетевое програмное обеспечение Oracle SQL*Net поддерживает все основные сетевые комуникационные протоколы, включая TCP/IP, SPX/IPX, Named Pipes и DECNet. Аналогичная по возможностям система Informix работает под управлением многих операционных систем, включая Unix, Windows, Windows NT, OS/2, Macintosh.

Эти СУБД позволяют связать множество массивов данных и сетей, объединенных в неоднородную вычислительную систему (а такая неоднородность характерна для большинства предприятий и организаций). Они обеспечивают полную поддержку языка SQL, поддерживают обработку транзакций, имеют гибкие генераторы форм и отчетов, средства клиент-сервер, которые разработчики могут использовать для сокращения сетевого ввода-вывода между клиентами и серверами. Таким образом, приложения на базе этих систем могут обеспечить пользователям необходимую производительность для продуктивного выполнения работы.

В эти СУБД включены средства, облегчающие администрирование сложных систем клиент-сервер, что может обеспечить преимущества систем клиент-сервер при сохранении в разумных пределах издержек на администрирование больших сложных систем.

Недостатком указанных выше СУБД являются высокие требования к аппаратному и системному программному обеспечению, что увеличивает стоимость разрабатываемых проектов.

Поэтому для управления данными предприятий среднего и малого бизнеса целесообразней применение таких СУБД, как: Paradox, Visual dBase, InterBase от компании Borland International (впоследствии Inprise); Access,

Visual FoxPro, SQL Server корпорации Microsoft и т.д.

3.3. ЯЗЫК УПРАВЛЕНИЯ ДАННЫМИ SQL

3.3.1. Основные сведения

Структурированный язык запросов SQL (Structured Query Language) – это язык, разработанный корпорацией IBM в 1970 г. Он фактически стал стандартом в качестве языка реляционных баз данных. Служащий IBM доктор Е.Ф.Кодд предложил язык SQL (называвший тогда SEQUEL – Structured English Query Language, структурированный английский язык запросов) как средство извлечения информации из реляционной базы данных.

В настоящее время SQL представляет собой не просто язык запросов, а наиболее распространенный язык управления реляционными базами данных типа клиент-сервер. Основное достоинство SQL заключается в том, что он унифицирован: стандартный набор инструкций SQL можно использовать в любой системе управления базами данных, которая поддерживает SQL. Первый американский стандарт SQL был зарегистрирован в 1986 г. как ANSI X3.135-1986. ANSI (Американский национальный институт стандартизации) – это организация, которая занимается созданием и обновлением научных и инженерных стандартов. ANSI-стандарт SQL был принят в качестве всемирного стандарта отделом ООН Международной организацией стандартизации (ISO) в 1989 г. - SQL/89. Стандартом текущей версии является стандарт ANSI X3.135-1992, широко известный как SQL/92.

SQL является языком управления реляционными базами данных, а не языком программирования. Таким образом, ANSI SQL не включает ни средств управления выполнением программы (ветвлений и циклов), ни средств для создания форм или отчетов. А функции управления реализуются в языках программирования (например, xBase, C, C++ или COBOL). Однако в некоторые версии (диалекты) SQL, например, в Transact-SQL, используемый в серверах БД Sybase и Microsoft SQL Server, добавлены два оператора (IF ELSE и WHILE).

При работе с языком SQL в клиент-серверной архитектуре все строится на понятии транзакции. **Транзакция** (transaction) – логический блок операций, содержащих одну или несколько инструкций SQL и рассматриваемых как единое целое, которые необходимо выполнить на одном или нескольких серверах (в последнем случае это **распределенные транзакции**).

Основное назначение транзакций – это преобразование одного согласованного состояния базы данных в другое, причем в промежуточных точках база данных находится в несогласованном состоянии. Система, поддерживающая процесс транзакции, гарантирует, что если во время некоторых обновлений базы данных произошла ошибка (по любой причине),

то все эти обновления будут аннулированы.

Транзакции обладают четырьмя важными свойствами: атомарность, согласованность, изоляция и долговечность.

Атомарность. Транзакции атомарны - выполняется все из транзакции или ничего.

Согласованность. Транзакции защищают базу данных согласованно. Это означает, что транзакции переводят одно согласованное состояние базы данных в другое без обязательной поддержки согласованности во всех промежуточных точках.

Изоляция. Транзакции отдельны одна от другой. Это означает, что, если даже будет запущено множество конкурирующих друг с другом транзакций, любое обновление определенной транзакции будет скрыто от остальных до тех пор, пока эта транзакция выполняется.

Долговечность. Когда транзакция выполнена, ее обновления сохраняются, даже если в следующий момент произойдет сбой системы.

Отсюда вытекает необходимость управления транзакциями, что предусмотрено в стандарте языка SQL. ANSI SQL включает набор стандартных команд, сгруппированных по шести категориям: описание данных, выполнение запросов, манипулирование данными, управление курсором, управление транзакциями, а также администрирование или управление. В 1989 г. в исходный стандарт ANSI были добавлены инструкции для обеспечения целостности данных.

Существует три способа реализации SQL:

1. **Непосредственный вызов**. Инструкции SQL передаются в систему управления реляционными базами данных, которая их выполняет. Примером непосредственного вызова может служить предложение SQL системы dBASE или FoxPro.

2. **Язык модулей**. Позволяет записывать в текстовый файл инструкции SQL, которые позднее выполняются приложением. Инструкция SQL должна начинаться с ключевого слова MODULE и определять язык программирования (LANGUAGE).

3. **Встроенный SQL**. Это наиболее распространенный метод реализации: инструкции SQL генерируются системой управления или включаются как текст в команды языка этой системы.

В настоящее время именно язык SQL является стандартом для работы с реляционными СУБД. SQL стал единственным языком баз данных клиент-сервер. Сервер баз данных (нижний уровень) отвечает за хранение данных. Приложения-клиенты (верхний уровень) добавляют или обновляют данные. Кроме того, приложение генерирует инструкции SQL. При регулярной работе с базами данных знание SQL обязательно. Это также относится к разработчикам, которым требуется создавать приложения с определяемыми пользователем запросами.

Существует несколько диалектов (версий) SQL. Например, диалект

SQL, который используется в Sybase и MS SQL Server, называется Transact-SQL и базируется на грамматике SQL/92.

Некоторые диалекты SQL используются во всех реляционных базах данных. Если приходится работать с несколькими реляционными базами данных разных форматов или необходимо конвертировать файлы из одной базы данных в другую, использование синтаксиса SQL является самым общим и легким решением. Хранимые наборы инструкций SQL, если они имеют общий синтаксис, могут быть легко конвертированы или использованы различными реляционными базами данных.

3.3.2. Стандарт SQL

SQL - очень динамичный, быстро развивающийся язык. Поэтому за достаточно короткое время были приняты уже два стандарта (SQL/89 и SQL/92) и в настоящее время имеется проект третьего стандарта (SQL-3).

В SQL/89 были стандартизованы следующие типы данных: символьные (CHARACTER), числовые (NUMERIC), целые (INTEGER), укороченные целые (SMALLINT), вещественные (REAL), с удвоенной точностью (DOUBLE PRECISION), с плавающей запятой (FLOAT), десятичные (DECIMAL).

В настоящее время используется стандарт SQL/92, в котором появилась стандартная версия динамического SQL. В SQL/92 были добавлены следующие типы данных: даты (DATE), время (TIME), интервалы (INTERVAL).

Динамический SQL включает специальные динамические операторы, поддерживающие динамическую компиляцию базовых операторов SQL, то есть компиляцию во время выполнения прикладной программы.

В языке SQL часть операторов выражается как в реляционной алгебре, часть - в стиле реляционного исчисления. В командах SQL могут использоваться арифметические операторы сравнения (>,<,>=,<=,=,<>) и логические операторы (AND, OR, NOT). Для обозначения всех кортежей в отношении используется символ звездочка (*). В шаблонах используются символ подчеркивание (_) для обозначения любого одиночного символа и символ процент (%) для обозначения любой последовательности символов. Круглые скобки используются для определения порядка выполнения действий, в фигурных скобках указываются константы типа даты, символьные константы указываются в апострофах или двойных кавычках. Ниже рассматриваются основные синтаксические конструкции языка.

Так как SQL начинал создаваться как язык запросов, то и большинство команд предназначено для выборки (SELECT) результатов из таблиц баз данных, то есть отношений. Основная конструкция операции выборки: SELECT - FROM - WHERE (выбрать-из-где). Для исключения дублирующих результатов при необходимости используется опция DISTINCT (различные): SELECT DISTINCT (что) FROM (откуда) WHERE (условие). Например, для

выборки из отношения R1 значений атрибутов A1 и A3 со значением атрибута A1 равного S4 и значением атрибута A2 равного 1000 с исключением дублирующих значений используется команда: SELECT DISTINCT A1, A3 FROM R1 WHERE A1=”S4” AND A2=1000.

Для удаления кортежей (строк, записей) из отношений используется аналогичная конструкция команды DELETE (удалить): DELETE - FROM - WHERE. Например, для удаления из отношения (таблицы) R1 записей со значением атрибута A2, равного или больше 200, используется команда: DELETE FROM R1 WHERE A2>=200. Для удаления всех записей из отношения используется команда: DELETE <имя отношения>.

Для изменения значений атрибутов используется команда UPDATE (обновить): UPDATE - SET- WHERE. Обновление осуществляется в записях, удовлетворяющих условию, указанному после опции WHERE (или для всех записей, если отсутствует опция WHERE). Имя и значение обновляемого атрибута указывается после опции SET. Например, если в отношении R1 необходимо заменить значение атрибута A1 на S2 для записей со значением атрибута A2, равного 500 или 1000, то используется команда: UPDATE R2 SET A1=’S2” WHERE A2=500 OR A2=1000.

Для вставки строк в отношения используется команда: INSERT INTO (включить в). Можно вставлять одну строку (кортеж) или произвольное количество строк. Значения вставляемых строк определяются списком кортежей или получают в результате выполнения внутреннего запроса (подзапроса). Например, для вставки в отношение R1 записей из отношения R2 со значением атрибута A5, равного ”Алматы”, используется команда: INSERT INTO R1: SELECT * FROM R2 WHERE A5=”Алматы”. Для вставки в отношение R1 строки со значением атрибута A1, равного S5, и значением атрибута A3, равного 2000, используется команда: INSERT INTO R1 (A1, A3) VALUES (“S5”, 2000).

Стандарт SQL предусматривает также команды для работы с отношениями (таблицами) целиком:

CREATE TABLE - создание новой таблицы базы данных (структура таблицы и параметры полей задаются в команде);

ALTER TABLE - изменение структуры таблицы (изменение структуры таблицы и параметров полей задается в команде);

DROP TABLE - удаление (уничтожение) таблицы.

В командах SQL возможно использование следующих операторов:

MINUS (DIFFERENCE) - вычитание множеств, полученных в результате нескольких запросов;

UNION - объединение множеств, полученных в результате нескольких запросов;

INTERSECT - пересечение множеств, полученных в результате нескольких запросов;

IS IN - указание принадлежности множеству;

IS NOT IN - указание непринадлежности множеству;

EXISTS - квантор существования, используется как альтернатива оператора IS IN;

CONTAINS - указание содержания каких-либо значений;

DOES NOT CONTAINS - указание несодержания каких-либо значений;

BETWEEN - задание диапазона для чисел, букв и т.д.;

= - указание эквивалентности множеств;

┐ - указание неэквивалентности множеств;

LIKE - указание использования шаблона, содержащего символы _ и %;

GROUP BY - задание разделения на группы (группировка данных), группы указываются через запятую);

HAVING - задание условия отбора в группы (используется только с оператором GROUP BY);

ORDER BY DESCENDING - задание упорядочения по какому-либо столбцу (атрибуту) по возрастанию (по умолчанию) или убыванию (DESCENDING) ASCII-кодов.

При формировании запросов в языке SQL могут использоваться собственные функции работы с полями языка SQL (агрегатные функции), которые представлены ниже. Функции работы с полями не могут быть вложенными, работают по вертикали.

AVG (<выражение>) - среднее значение по колонке числовых данных.

COUNT (<выражение>) - подсчет числа кортежей (строк).

MIN (<выражение>) - определяет наименьшее значение искомого элемента в колонке.

MAX (<выражение>) - определяет наибольшее значение искомого элемента в колонке.

SUM (<выражение>) - определяет суммарный итог по колонке числовых данных.

В настоящее время во всех современных СУБД реляционного типа реализован язык SQL: Oracle, Informix, Sybase, InterBase, Ingres, DB2, MS SQL Server, dBASE-подобные СУБД (в том числе FoxPro), Access и др. Реализация SQL отличается в различных СУБД и рассматривается для широко распространенной СУБД клиентского типа FoxPro далее в главе 7.

3.4. СЕРВЕРЫ БАЗ ДАННЫХ

Серверы баз данных относятся к числу сложных существующих программных продуктов, предназначенных для работы в вычислительной сети. Рабочая станция пользователя представляет собой клиента (обслуживаемая сторона), а СУБД выполняется на сервере (обслуживающая сторона). В такой системе обработка данных распределяется между двумя вычислительными системами.

Программа - сервер баз данных является центральной с точки зрения доступа к данным. Именно поэтому большая часть функций, которая должна быть реализована в информационной системе, ложится на сервер баз данных. Основные функции, которые должна выполнять программа - сервер баз данных:

- выполнять клиентские запросы по извлечению и модификации данных;
- обеспечивать одновременный доступ к данным нескольких пользователей;
- обеспечивать идентификацию пользователей и разграничение прав доступа разных пользователей к разным данным;
- обеспечивать целостность и непротиворечивость данных в случае аппаратных и программных сбоев;
- защищать данные от несанкционированного доступа;
- предоставлять дополнительные средства администрирования системой.

Рассмотрим эти требования более подробно.

Выполнять клиентские запросы по извлечению и модификации данных. Это основная функция сервера баз данных. Механизм реализации этой функции может быть скрыт от пользователя, то есть пользователь (а, точнее, работающая с ним программа-клиент) просто формулирует, что ему нужно, а сервер баз данных исполняет этот запрос.

Предоставлять механизм одновременного доступа к данным нескольких пользователей. При многопользовательском доступе к данным, используемым в технологии клиент-сервер, возникают дополнительные задачи, которые должны быть решены. К таким задачам относится, например, блокировка данных. Блокировка означает, что часть данных в некоторые моменты времени должна быть закрыта для модификации или даже для чтения другим пользователем. Другой аспект многопользовательского доступа - это распараллеливание доступа. То есть сервер баз данных должен уметь обрабатывать несколько запросов одновременно. В этом аспекте сервер баз данных аналогичен многозадачной операционной системе.

Обеспечивать идентификацию и разграничение прав доступа разных пользователей к разным данным. В реальных системах баз данных должно существовать разграничение по правам доступа к данным. Какие-то пользователи могут и читать и модифицировать данные, какие-то пользователи могут только читать, а кто-то вообще может только вводить данные, а читать не имеет право. Таким образом, сервер баз данных должен, во-первых, уметь понимать команды, которые описывают такое разграничение прав, а во-вторых, в процессе обслуживания запросов пользователя контролировать соблюдение этих разграничений.

Обеспечивать целостность и непротиворечивость данных в случае

аппаратных и программных сбоев. В случае внезапного выключения и последующего включения компьютера, на котором работает сервер баз данных, информация не должна быть искажена или потеряна. Аналогично, если будет случайно выключен компьютер, на котором работает программа-клиент, сервер баз данных должен определить этот факт и снять те блокировки, которые данная программа-клиент установила, отменить незавершенные транзакции и, возможно, выполнить другие действия.

Что касается программных сбоев, то здесь надо различать умышленные попытки исказить информацию, или случайные ошибки в программах, которые могут привести к таким же последствиям. Например, списывание денег с банковского счета должно приводить или к зачислению этой суммы на другой счет, или к появлению расходного документа. То есть нельзя просто так списать или начислить деньги на счет. Кроме того, при программировании банковской системы в нее может закрасться ошибка, которая может привести к рассогласованию данных. Поэтому сервер баз данных должен уметь проверять корректность производимых манипуляций с данными.

Защищать данные от несанкционированного доступа. В современных системах баз данных все данные или, по крайней мере, их значительная часть является конфиденциальной. Помимо разграничения доступа для разных категорий пользователей, сервер баз данных должен обеспечивать защиту от попыток получить доступ к данным тем лицам, которые не являются пользователями информационной системы.

Предоставлять средства администрирования. В реальной системе баз данных должен быть предусмотрен механизм восстановления системы после стихийных бедствий. Чтобы не потерять накопленные данные, должна существовать процедура архивирования и восстановления данных. Кроме того, существует вероятность, что в процессе проектирования системы не были предусмотрены какие-то типы запросов, и при их реализации оказалось, что они работают слишком медленно. Следовательно, сервер баз данных должен уметь управлять ресурсами и производительностью. Вся деятельность данного типа относится к администрированию сервера баз данных. Качественный сервер баз данных должен предоставлять достаточный набор возможностей по администрированию, а именно, возможность настройки по производительности, средства анализа потока запросов и определения причин недостаточной скорости работы, средства созданий резервных копий (архивов) и восстановления с них и т.д.

В настоящее время все серверы реляционных БД являются SQL-серверами, их количество достаточно велико. SQL-серверы - это СУБД с архитектурой клиент-сервер на основе языка SQL. Для персональных компьютеров (ПК) SQL-серверы создаются для того, чтобы справляться с гигантскими базами данных и многопользовательскими нагрузками, которые были традиционной сферой деятельности больших ЭВМ, мини-ЭВМ и

микропроцессорных серверов на базе ОС Unix. Файл-серверная архитектура обработки данных, в которой все данные передаются по сети рабочей станции, просто недостаточно эффективна, чтобы справиться с задачами такого масштаба. Технология клиент-сервер позволяет внешним средствам работы с базами данных передавать функции SQL-обработки базовому серверу. Клиенту назад отправляется только результат, что существенно снижает нагрузку на сеть.

Низкие затраты на повышение производительности центрального процессора, увеличение емкости жестких дисков и памяти делают переход к серверам на базе ПК (ПК-серверам) почти неизбежным. 32-разрядные ОС, такие, как Microsoft Windows NT, NetWare и OS/2 обеспечивают высокую производительность и сохранность данных, относительную легкость использования на ПК-серверах, а применение графических средств на компьютере клиента существенно упрощает составление запроса на языке SQL. В настоящее время для персональных компьютеров наибольшее распространение получили SQL-серверы, использующие процессоры фирмы Intel: Oracle, Sybase, Informix, Ingres, Watcom (Anywhere), DB2/2 (версия DB2 для персональных компьютеров), InterBase, Microsoft SQL Server и др.

В большинстве случаев SQL-серверы входят в состав функционально полных серий продуктов, реализующих технологию баз данных. Например, серия Oracle включает серверы баз данных, эффективно управляющие своими ресурсами, информацией в базе данных, обслуживающие множество **клиентов, обращающихся к базе данных с** запросами и посылающих данные по сети. СУБД Oracle имеет множество важных средств, которые делают его не только исключительной СУБД, но и превосходным сервером баз данных, который можно с успехом применять в вычислениях баз данных клиент-сервер. Последняя версия обеспечивает пятикратное увеличение скорости при работе в сети и поддержку многосерверного режима. Система построена на основе самонастраивающейся адаптируемой многопотоковой архитектуры.

Серия продуктов Informix включает семейство серверов реляционных баз данных Informix OnLine, которые обеспечивают аналогичные возможности и имеют встроенный механизм параллелизма, что позволяет использовать преимущества мультипроцессорных систем для параллельной обработки данных.

Достаточно много серверов баз данных разрабатывается и в качестве отдельного программного продукта, например, Microsoft SQL Server.

3.4.1. Взаимодействие сервера и клиента БД

Программное обеспечение сервера баз данных обрабатывает запросы, инициализированные программным обеспечением клиента, посылая результат обратно клиенту БД, который посылал запрос. В общем случае взаимодействие сервера и клиента БД можно представить в виде следующих

шагов:

- пользователь запускает клиента и клиент посылает запрос на установление связи;

- сервер посылает клиенту подтверждение о том, что он готов к работе;

- клиент запрашивает регистрационное имя и пароль пользователя, после ввода этих данных имя и пароль посылаются на сервер;

- сервер проверяет, соответствует ли пароль имени пользователя и имеет ли право данный пользователь работать с указанной базой данных. Если все права соблюдены, то клиенту отсылается подтверждение;

-пользователь вводит данные (например, заполняет форму ввода) и введенная информация передается на сервер с запросом на ее внесение в базу; - сервер проверяет, имеет ли право данный пользователь вносить новую информацию. Если пользователь обладает этими правами, то сервер проверяет не будет ли нарушена целостность данных. Если все нормально, то производится вставка новых данных и клиенту пересылается уведомление о том, что все прошло успешно;

- клиент сообщает пользователю об успешном завершении операции.

Конечно же, в реальной ситуации взаимодействие между сервером баз данных и программой-клиентом может быть значительно сложнее. Да и действия, производимые сервером баз данных, также включают в себя дополнительные проверки и действия.

Ниже рассматривается взаимодействие программного обеспечения сервера с программным обеспечением клиента в наиболее распространенной для персональных компьютеров ОС Windows на примере MS SQL Server. Программное обеспечение клиента соединяется с программным обеспечением сервера баз данных, делает запрос, получает результаты и затем обрабатывает их (например, выводит результаты на экран). СУБД FoxPro, Paradox, Access - примеры программного обеспечения клиента баз данных, которое соединяется с базами данных SQL Server.

Компьютер-сервер запускает программное обеспечение сервера баз данных SQL Server. Программное обеспечение клиента баз данных SQL Server может также быть запущено с компьютера-сервера. Когда клиентское программное обеспечение запускается на компьютере-клиенте, соединение с SQL Server осуществляется через сеть.

На рис.3.1 показано взаимодействие компьютеров клиентов и компьютера-сервера. Компьютер-клиент, который работает под управлением Windows или MS-DOS, запускает клиентское приложение SQL Server. Приложение посылает запросы на SQL Server, загруженный на компьютер-сервер, который работает под управлением операционной системы Windows и запускает приложения SQL Server, включая клиентское и серверное приложения.

Программное обеспечение клиента баз данных состоит из нескольких уровней или компонентов. Пользователь взаимодействует с клиентским

приложением, которое определяет представление системы баз данных (например, форма входного заказа). Все клиентские приложения взаимодействуют с SQL Server, используя один или два собственных API (Application Programming Interfaces - интерфейсы прикладного программирования) для доступа к базе данных клиента: ODBC (открытое соединение баз данных), DB-Library (библиотека баз данных).

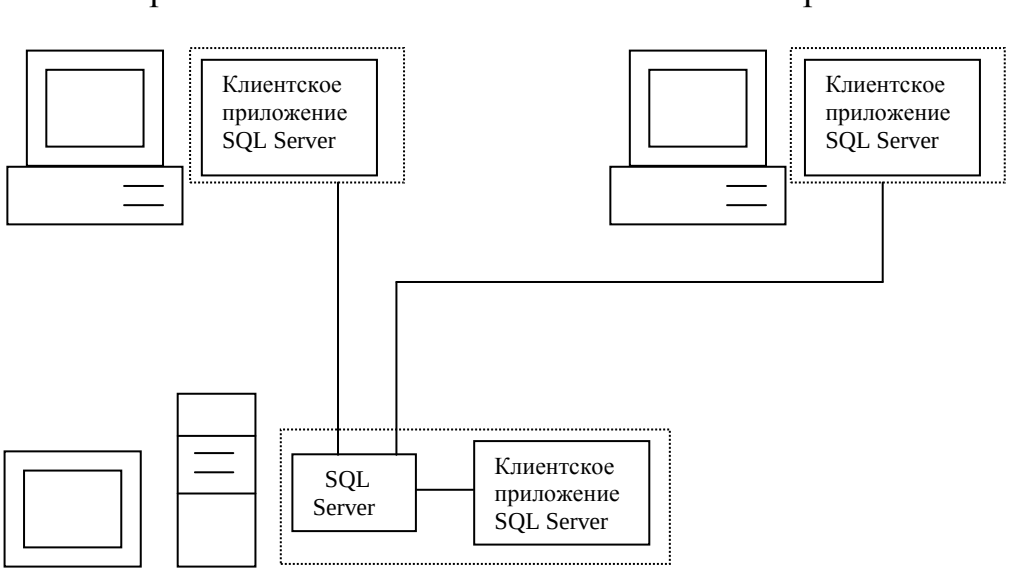


Рис.3.1. Взаимодействие сервера и клиента баз данных

На рис.3.2 показано сетевое соединение компьютера-клиента и клиентской сетевой библиотеки с компьютером-сервером и его сетевой библиотекой.

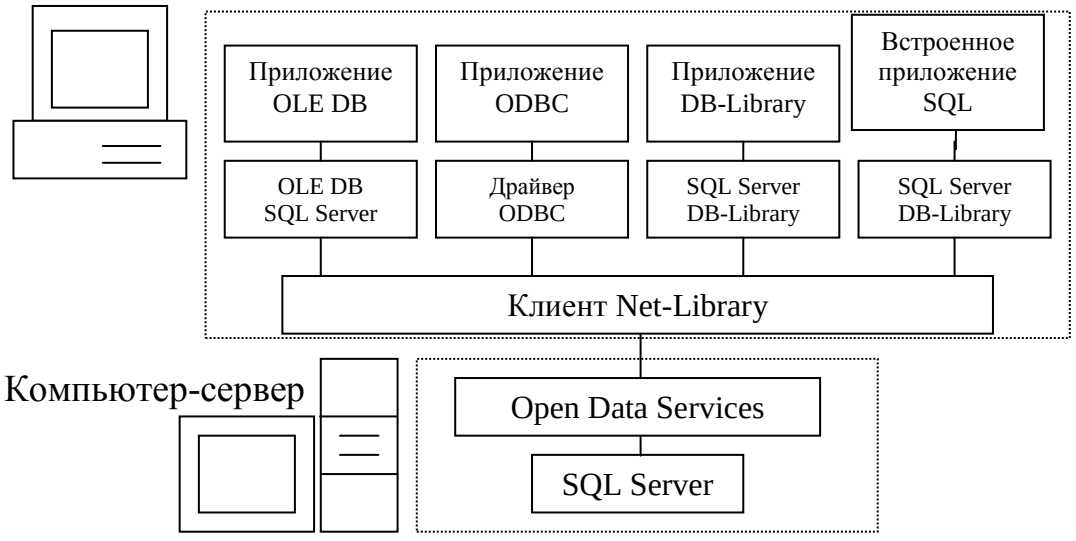


Рис.3.2. Взаимодействие компьютеров клиента и сервера

И ODBC и DB-Library используют один и тот же клиентский уровень сетевой библиотеки (Net-Library) для создания соединения с SQL Server. Программное обеспечение сервера также содержит несколько уровней. Серверный уровень сетевой библиотеки принимает соединение от клиента. Open Data Services (Сервисы открытых данных) – это API для написания программного обеспечения сервера баз данных. Ядро базы данных SQL Server – приложение Open Data Services. SQL Server связывается с клиентом через Open Data Services, принимая запросы и возвращая результаты.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какую модель данных используют реляционные системы баз данных?
2. Что собой представляет реляционная база данных?
3. Каким требованиям должны удовлетворять таблицы реляционных баз данных?
4. Что собой представляет первичный ключ и внешний ключ в таблице базы данных и для чего они используются?
5. Какому обязательному условию должны удовлетворять таблицы данных реляционных систем, находящиеся в первой нормальной форме?
6. Какому обязательному условию должны удовлетворять таблицы данных реляционных систем, находящиеся во второй нормальной форме?
7. Какому обязательному условию должны удовлетворять таблицы данных реляционных систем, находящиеся в третьей нормальной форме?
8. Что собой представляет универсальная таблица базы данных?
9. Какие инструментальные средства входят в состав реляционных СУБД?
10. Выполнение каких функций обеспечивает интерфейс СУБД?
11. Какие классы можно выделить среди СУБД, их отличительные признаки, их представители?
12. Для чего используется язык управления данными SQL, его основное назначение?
13. Какие категории операций выделяют в стандартном SQL?
14. Какие существуют способы реализации SQL?
15. Какие типы данных и операции над данными стандартизованы в SQL?
16. Какие собственные функции в операциях над данными стандартизованы в SQL?
17. В чем заключаются основные функции сервера баз данных?
18. Что собой представляет SQL-сервер и какие СУБД относятся к SQL-серверам?
19. Что собой представляет клиент сервера баз данных?
20. Как осуществляется взаимодействие сервера и клиента баз данных?

Глава 4. ЗАЩИТА И БЕЗОПАСНОСТЬ ИНФОРМАЦИИ В СИСТЕМАХ БАЗ ДАННЫХ

4.1. ОСНОВНЫЕ ПОНЯТИЯ ЗАЩИТЫ И БЕЗОПАСНОСТИ ИНФОРМАЦИИ

Защита информации представляет собой охрану данных от множества возможных угрожающих ситуаций (как преднамеренных, так и случайных), так как всегда существует риск потери данных. Потеря данных в системах баз данных может быть обусловлена разными причинами:

- система может быть разрушена во время выполнения некоторых программ, оставив при этом базу данных в непредсказуемом состоянии;
- при выполнении двух конкурирующих программ между ними может возникнуть конфликт, что приведет к получению неправильных результатов;
- данные в базе данных могут быть испорчены или изменены анонимным пользователем и т.д.

Термин **защита** относится к защищенности данных от несанкционированного доступа, изменения или умышленного разрушения. Под обеспечением **безопасности** данных подразумевается поддержание целостности данных, т.е. предотвращение их разрушения при доступе со стороны санкционированных пользователей. Среди многочисленных аспектов проблемы защиты и безопасности данных необходимо отметить следующие.

1. Правовые, общественные и этические аспекты (например, имеет ли некоторое лицо юридическое право получить запрашиваемую информацию).
2. Физические условия (например, закрыт ли данный компьютер или терминальная комната или они защищены каким-либо другим образом).
3. Организационные вопросы (например, как в рамках предприятия, обладающего некой системой, организован доступ к данным).
4. Вопросы реализации управления (например, если используется метод доступа по паролю, то как организована реализация управления и как часто меняются пароли).
5. Аппаратное обеспечение (обеспечиваются ли меры безопасности на аппаратном уровне, например, с помощью защитных ключей или привилегированного режима управления).
6. Безопасность операционной системы (например, уничтожает ли операционная система содержимое оперативной памяти и дисковых файлов после прекращении работы с ними).

Существует четыре основных направления по защите и безопасности информационных систем, в том числе и на основе технологии баз данных.

Предотвращение - только авторизованный персонал имеет доступ к информации и технологии.

Обнаружение - обеспечивается раннее обнаружение преступлений и

злоупотреблений, даже если механизмы защиты были обойдены.

Ограничение - уменьшается размер потерь, если преступление все-таки произошло, несмотря на меры по его предотвращению и обнаружению.

Восстановление - обеспечивается эффективное восстановление информации при наличии документированных и проверенных планов по восстановлению

На каждом из этих этапов необходимо проводить соответствующую работу по поддержанию защиты и безопасности информационной системы. Основной контроль за соблюдением правил работы с информацией и техническими средствами лежит на системных администраторах или людях, которые специально занимаются этой проблемой. Но в настоящее время обстоятельства складываются так, что невозможно эффективно использовать все методы и меры по защите и безопасности информации без участия в этом процессе конечных рядовых пользователей. Ведь в конечном счете любой рядовой пользователь может стать источником опасности для организации, в которой он работает.

Меры защиты - это меры, вводимые руководством, для обеспечения безопасности информации. Основной целью мер защиты является предотвращение преступлений и злоупотреблений. Меры защиты могут также выполнять функцию ограничения, уменьшая размер ущерба от преступления. К мерам защиты относятся административные руководящие документы (приказы, положения, инструкции), аппаратные устройства или дополнительные программы.

Информационной защитой и безопасностью называют меры по защите информации от неавторизованного доступа, разрушения, модификации, раскрытия и задержек в доступе.

Информационная безопасность включает в себя меры по защите процессов создания данных, их ввода, обработки и вывода. Цель информационной безопасности: обезопасить ценности системы, защитить и гарантировать точность и целостность информации, и минимизировать разрушения, которые могут иметь место, если информация будет модифицирована или разрушена. Информационная безопасность требует учета всех событий, в ходе которых информация создается, модифицируется, к ней обеспечивается доступ или она распространяется. При этом достигаются следующие цели:

- **конфиденциальность** критической информации;
- **целостность** информации и связанных с ней процессов(создания, ввода, обработки и вывода);
- **доступность** информации, когда она нужна;
-**учет** всех процессов, связанных с информацией.

Под критическими данными понимаются данные, которые требуют защиты из-за вероятности нанесения (риска) ущерба и его величины в том

случае, если произойдет случайное или умышленное раскрытие, изменение, или разрушение данных. Именно критические данные нужно защищать от различного вида несанкционированного воздействия.

Защита информации должна осуществляться комплексно, начиная с **административных мер**, включая **физическую безопасность**, заканчивая чисто **программными областями.**

Административно – каждый сотрудник несет определенную ответственность за разглашение конфиденциальной информации, существуют системы охраны, пожаротушения и т.п. Физический доступ к информации должен быть ограничен. Необходимо установить запирающиеся двери, лучше с электронными замками, которые защищают сервер. Рекомендуется жестко связать сервер с платформой, на которой он стоит. Современные сетевые ОС предоставляют широкие возможности удаленного администрирования, поэтому можно убрать мониторы и клавиатуру от сервера, который должен быть помещен в область с интенсивным доступом. Эти действия предотвращают возможность доступа к серверу случайно подошедшего пользователя. В качестве альтернативы можно применить защищенные серверные сейфы, которые могут быть использованы для обеспечения лучшей безопасности сервера.

На **аппаратном уровне** применяется защита и безопасность, которая осуществляется с помощью систем бесперебойного питания, фильтров напряжения, использования сетевого экрана, защитных ключей, резервирования баз данных на CD-диски и т.д. Основной принцип защиты программ с использованием аппаратуры защиты состоит в том, что при несанкционированном копировании программы вырабатывается сигнал на самоуничтожение программы.

На **программном уровне** система безопасности и защиты включает в себя систему защиты сетевой ОС и систему безопасности конкретного сервера. При входе в сеть действует система идентификации и определения прав для пользователей сетевой ОС, при входе в базы данных действует система проверки и идентификации пользователей сервера БД. В настоящее время практически все серверы реляционных БД являются SQL-серверами, поэтому в дальнейшем речь идет о системах защиты и безопасности SQL-серверов.

База данных содержит не только значения данных, но и стандартный набор атрибутов для этих значений. Другими словами, база данных представлена в ЭВМ как множество пар **атрибут – значение**. Стандартные атрибуты позволяют администратору системы и пользователям иметь одинаковое представление о типах и характеристиках элементов базы данных. Такое единство важно для сохранения целостности базы данных во время ее модификации и при параллельной работе с ней нескольких пользователей. Современные системы управления базами данных обеспечивают режим непосредственного доступа к базе данных, режим диалога со многими

пользователями и режим, допускающий динамические перераспределение периферийных устройств. Для организации гибкой защиты данных необходимо, чтобы система управления базой данных допускала как частично совпадающие, так и даже противоречащие друг другу требования по защите данных для разных пользователей. Если же отсутствует единое представление о базе данных, ее владелец (администратор системы и пользователь) не сможет указать спецификаций защиты, поскольку, дав их, он затруднит другим пользователям доступ к базе данных. Не имея спецификаций, система управления базой данных (СУБД) не сможет присвоить защитных атрибутов данным и обеспечить их защиту. СУБД должна идентифицировать данные по их типам и характеристикам так, чтобы значение, тип и характеристика данного могли быть защищены в полном соответствии со спецификациями защиты. Поскольку общим данным нескольких пользователей назначаются различные спецификации защиты, СУБД должна различать данные разных типов, характеристик. Без этого СУБД не узнает, являются ли спецификации защиты частично совпадающими или противоречивыми.

Таким образом, после того, как СУБД приняла решение о разрешении доступа, считается, что запрос в конце концов обеспечит обращение к требуемым данным. Желательно, чтобы СУБД обращалась только к тем данным, которые включены в запрос. Если это выполняется, увеличивается производительность системы и упрощается ее защита. С точки зрения защиты данных производительность СУБД определяется эффективностью запрета несанкционированного просмотра. Проблема несанкционированного просмотра возникает в том случае, когда СУБД для получения необходимых данных вынуждена обращаться к данным, которые имеют иные требования защиты. Если последние требования более строги, то ситуация становится критической. Таким образом, несанкционированный просмотр будет запрещен, если СУБД обеспечит доступ только к тем данным, которые имеют те же атрибуты защиты, что и данные в запросе. Целью каждого проектировщика и программиста при создании СУБД является разработка таких систем защиты, которые позволяют избежать несанкционированного просмотра.

Все решения относительно допуска пользователей к выполнению тех или иных операций принимаются на **стратегическом**, а **не техническом** уровне. Поэтому они находятся за пределами досягаемости самой системы управления базами данных, и все, что может в такой ситуации сделать система управления базами данных, - это только привести в действие уже принятые ранее решения. Исходя из этого, можно отметить следующее.

1. Результаты стратегических решений должны быть известны системе (т.е. выполнены на основе утверждений, заданных с помощью некоторого подходящего языка) и сохраняться в ней (путем сохранения их в каталоге в виде правил безопасности, которые также называются полномочиями).

2. Должны быть некоторые средства регулирования запросов доступа по отношению к соответствующим правилам безопасности (под «запросом доступа» подразумевается комбинация запрашиваемой операции, запрашиваемого объекта и запрашивающего пользователя). Такая операция выполняется подсистемой безопасности системы управления базами данных, которая также называется подсистемой полномочий.

3. В системе должны быть предусмотрены способы опознания источника этого запроса, т.е. опознания запрашивающего пользователя. Поэтому в момент входа в систему от пользователя обычно требуется ввести не только его **идентификатор** (например, имя или должность), но также и **пароль** (чтобы подтвердить свои права на заявленные ранее идентификационные данные).

Разные пользователи могут обладать одним и тем же идентификатором некоторой группы. Таким образом, в системе могут поддерживаться группы пользователей и обеспечиваться одинаковые права доступа для пользователей одной группы. Кроме того, операции добавления отдельных пользователей в группу или их удаления из нее могут выполняться независимо от операции задания привилегий для этой группы.

Пароли обеспечивают сохранение целостности программного обеспечения в составе вычислительной системы, но для поддержания системы паролей требуется высокая дисциплинированность. Обычно предполагается, что пароль известен только системе и некоторым лицам с особыми правами. Пароли должны быть просты для запоминания, но не должны быть столь очевидными для нарушителя. При формировании пароля часто прибегают к помощи генератора случайных чисел или букв в зависимости от данных, которые задает пользователь. Однако защита с помощью пароля может оказаться неэффективной, если требуется хранение копии пароля, которая может оказаться в руках нарушителя.

В современных системах управления базами данных поддерживается один из двух широко распространенных подходов к вопросу обеспечения безопасности данных, а именно **избирательный** подход или **мандатный** (обязательный) подход. В обоих подходах единицей данных или объектом данных, для которых должна быть создана система безопасности, может быть как вся база данных целиком или какой-либо набор отношений, так и некоторое значение данных для заданного атрибута внутри некоторого кортежа в определенном отношении. В системе управления базами данных поддерживается либо избирательное, либо мандатное, либо оба типа управления доступом.

В случае избирательного управления некий пользователь обладает различными правами (привилегиями или полномочиями) при работе с разными объектами. Более того, разные пользователи обычно обладают и разными правами доступа к одному и тому же объекту. Поэтому избирательные схемы характеризуются значительной гибкостью.

Избирательный подход включает также предотвращение возможности раскрытия данных посредством логического вывода. Поскольку в базе данных хранится семантика данных, то логический анализ содержимого базы может привести к нарушению секретности данных.

В случае мандатного управления каждому объекту данных присваивается некоторый классификационный уровень, а каждому пользователю присваивается некоторый уровень допуска. При таком подходе доступом к определенному объекту данных обладают только пользователи с соответствующим уровнем допуска. Поэтому мандатные схемы достаточно жестки и статичны и применяются к базам данных, в которых данные имеют достаточно статичную или жесткую структуру.

Как было отмечено выше, при использовании методов мандатного управления доступом, каждый объект данных имеет некоторый уровень классификации, например: секретно, совершенно секретно, для служебного пользования и т.д., а каждый пользователь имеет уровень допуска с такими же градациями, что и уровень классификации. Предполагается, что эти уровни образуют строгий иерархический порядок. Тогда на основе этих сведений можно сформулировать два очень простых правила безопасности:

- пользователь i имеет доступ к объекту j, только если его уровень допуска больше или равен уровню классификации объекта j;

- пользователь i может модифицировать объект j, только если его уровень допуска равен уровню классификации объекта j.

В последнее время методы мандатного (обязательного) управления доступом получили широкое распространение. Требования к такому управлению изложены в двух публикациях, называемых **оранжевой** книгой, в которой перечислен набор требований к безопасности для **надежной вычислительной базы**, и **розовой** книгой, где дается интерпретация этих требований для систем управления базами данных.

До сих пор подразумевалось, что предполагаемый нелегальный пользователь пытается незаконно проникнуть в базу данных с помощью обычных средств доступа, имеющихся в системе. Возможны случаи, когда такой пользователь пытается проникнуть в базу данных, минуя систему, т.е. физически перемещая часть базы данных или подключаясь к коммуникационному каналу. Наиболее эффективным методом борьбы с такими угрозами является шифрование данных, т.е. хранение и передача особо важных данных в зашифрованном виде с использованием **криптографических методов.**

Использование криптографических методов зависит от поставленных целей. Если цель в том, чтобы затруднить доступ к файлам обычным пользователям, следует принять во внимание, что достаточная степень защиты может быть обеспечена без использования сложных шифров. Защищенность шифра тем надежнее, чем короче зашифрованный текст, и использование даже простого шифра для маскировки идентификационных

меток существенно увеличивает время раскрытия кода защиты программы или данных. Исходные данные (незакодированные) называются открытым текстом. Открытый текст шифруется с помощью специального алгоритма шифрования. В качестве входных данных для такого алгоритма выступают открытый текст и ключ шифрования, а в качестве выходных – зашифрованная форма открытого текста. Если детали алгоритма шифрования могут быть опубликованы или, по крайней мере, могут не утаиваться, то ключ шифрования обязательно хранится в секрете. Именно зашифрованный текст хранится в базе данных и передается по коммуникационному каналу.

В данном контексте методы криптографии представляют собой способы кодирования информации с целью скрыть ее смысл. Контроль доступа к данным означает, что система должна идентифицировать пользователя, обеспечить доступ только к разрешенным данным и выполнять только допустимые операции над ними. До тех пор пока пользователь не будет идентифицирован, система должна запрещать пользователю любые операции над данными. Чтобы выделить данные, с которыми пользователю разрешено работать, система должна знать смысл данных. Без этого система будет не в состоянии выделить те данные, к которым разрешен доступ. Таким образом, смысловой контекст (семантика) данных играет важную роль при реализации контроля доступа. Только подход с позиции искусственного интеллекта может обеспечить правильную реализацию санкционированного доступа независимо от сложности смыслового контекста вызываемых данных.

Важно понимать, что не бывает неуязвимых систем безопасности, поскольку настойчивый потенциальный нарушитель всегда сможет найти способ преодоления всех систем контроля, особенно если за это будет предложено достаточно высокое вознаграждение. Поэтому при работе с очень важными данными или при выполнении критических операций возникает необходимость регистрации контрольного следа выполняемых операций. Если, например, противоречивость данных приводит к подозрению, что совершено несанкционированное вмешательство в базу данных, то контрольный след должен быть использован для прояснения ситуации и подтверждения того, что все процессы находятся под контролем. Если это не так, то контрольный след поможет, по крайней мере, обнаружить нарушителя.

Для сохранения контрольного следа обычно используется особый файл, в котором система автоматически записывает все выполненные пользователем операции при работе с обычной базой данных. Типичная запись в файле может содержать следующую информацию: запрос (исходный текст); терминал, с которого была вызвана операция; пользователь, задавший операцию; дата и время запуска операции; вовлеченные в процесс

исполнения операции базовые отношения, кортежи и атрибуты; старые значения; новые значения.

Констатация факта, что в данной системе поддерживается контрольное слежение, в некоторых случаях весьма существенна для предотвращения несанкционированного проникновения в систему.

4.2. ТРАНЗАКЦИИ И БЛОКИРОВКИ

При работе в локальной сети возможность одновременного доступа к общим данным является мощным средством совместной работы группы пользователей (совместно используемая база данных хранится на сервере). Однако при этом в процессе разработки приложения, требуются дополнительные действия по упорядочению доступа к базе данных. Основной проблемой является устранение противоречий между параллельным доступом к данным и сохранением целостности базы данных. Оптимизировать совместную работу группы пользователей позволяет использование транзакций и блокировок.

Транзакция в том смысле, в котором она используется в СУБД, представляет собой логическую единицу работы. В общем случае транзакция представляет собой последовательность нескольких операций, которая преобразует некоторое целостное состояние базы данных в другое целостное состояние, однако не гарантирует сохранения целостности во все промежуточные моменты времени. Преобразование выражается в обновлении данных в таблицах базы.

Система, которая поддерживает обработку транзакций, является лучшей с точки зрения обеспечения гарантий надежности. В частности, она гарантирует, что если в транзакции осуществлялись некоторые обновления, а затем по какой-либо причине произошла ошибка до того, как транзакция достигла ее нормального завершения, то эти обновления будут аннулироваться. Следовательно, все обновления, внесенные в базу данных в рамках транзакции, либо одновременно подтверждаются, либо не подтверждается ни одно из них. Принцип восстановления и процесс транзакции в целом не зависят от того, является ли базовая система реляционной или какой-либо еще.

Основное назначение транзакций - это **защита** групп обрабатывающих инструкций.

Существуют некоторые требования, которые предъявляются к транзакциям:

- **атомарность** - предполагается, что транзакция должна быть либо завершена, либо не начата, т.е. можно выполнить только всю транзакцию целиком, но никак не любую из ее составляющих;
- **постоянство** - транзакции защищают базу данных согласованно, т.е. транзакции переводят одно согласованное состояние базы данных в

другое без обязательной поддержки согласованности во всех промежуточных точках, после завершения транзакции система должна оставаться такой, какой она была до начала транзакции;

- **изолированность** - транзакции должны быть изолированы, то есть не влиять на другие транзакции и не зависеть от них. Это означает, что если даже будет запущено множество конкурирующих друг с другом транзакций, любое обновление определенной транзакции будет скрыто от остальных до тех пор, пока эта транзакция выполняется. Зависимость от других транзакций вызывает тупиковые ситуации, что ведет к **откату** (аннулированию) транзакций;
- **устойчивость** - если транзакция завершена и ее цель достигнута, не может быть никаких веских причин для отката транзакции. Другими словами, после окончания транзакции она становится завершенной, даже если с системой что-то случается. По этой причине любой набор важных операций с таблицами данных выполняется за одну транзакцию. Это гарантирует, что из заданных в наборе операций будут выполнены либо все, либо ни одной.

Для восстановления баз данных при возникновении ошибок, требующих отката или наоборот обновления информации, используется **журнал транзакций**. Журнал транзакций - это то средство, с помощью которого СУБД обеспечивает целостность баз данных, причем как в отношении правил, которыми управляет сервер, так и в отношении обработки, которую реализуют приложения.

Журналы транзакций (transaction logs) - это обычно рабочие области, используемые СУБД для управления информационной картиной системы в состоянии до и после транзакции. Благодаря этой информации имеется возможность управлять откатом транзакций в случае, если возникнет такая необходимость.

Откат (возврат к предыдущему стабильному состоянию) используется также для восстановления базы данных при неожиданном отказе или перезагрузке системы. Каждая база данных имеет соответствующий ей журнал транзакций. Можно сказать, что журнал транзакций - это место, куда СУБД записывает все транзакции, которые проводятся над базой данных перед тем, как записать их в базу данных.

Термин **параллелизм** означает возможность одновременной обработки в системе управления базами данных многих транзакций с доступом к одним и тем же данным, причем в одно и то же время. В такой системе для корректной обработки **параллельных транзакций** без возникновения конфликтных ситуаций необходимо использовать некоторый метод управления параллелизмом.

Каждый метод управления параллелизмом предназначен для решения некоторой конкретной задачи. Тем не менее, при обработке правильно составленных транзакций возникают ситуации, которые могут привести к

получению неправильного результата из-за взаимных помех среди некоторых транзакций. Основные проблемы, возникающие при параллельной обработке транзакций следующие:

- проблема потери результатов обновления;
- проблема незафиксированной зависимости;
- проблема несовместимого анализа.

Одним из методов управления параллелизмом является **блокировка.** Эффект блокировки состоит в том, чтобы **заблокировать доступ к объекту со стороны других транзакций.**

Блокировка - это методика управления параллельным выполнением процессов. Принцип использования блокировки заключается в следующем: в случае, когда для выполнения некоторой транзакции необходимо, чтобы некоторый объект (обычно это кортеж базы данных) не изменялся непредсказуемо и без ведома этой транзакции, то такой объект блокируется.

Таким образом, эффект блокировки состоит в том, чтобы заблокировать доступ к этому объекту со стороны других транзакций, а значит, предотвратить непредсказуемое изменение этого объекта. Следовательно, первая транзакция в состоянии выполнить всю необходимую обработку с учетом того, что обрабатываемый объект остается в стабильном состоянии настолько долго, насколько это нужно.

При написании многопользовательских приложений баз данных существует два подхода к управлению транзакциями: **оптимистическая** и **пессимистическая** блокировки.

При оптимистической блокировке предполагается, что разработчик в приложении концентрируется только на его логике и явно не управляет блокировкой записей, оставляя эту задачу системе управления базами данных.

При пессимистической блокировке предполагается, что приложение само пытается реализовать определенный тип механизма блокировки.

4.3. РЕЗЕРВИРОВАНИЕ БАЗ ДАННЫХ

Существует несколько способов защиты от потери данных или, по крайней мере, сведение их к минимуму. Эти способы включают физическое дублирование баз данных, а также резервирование баз данных и журналов транзакций через определенные интервалы времени работы с СУБД.

Основными понятиями резервирования являются: **резервирование базы данных, хранение резервной копии, восстановление базы данных из резервной копии.**

Один из первых вопросов, который нужно обдумать администратору баз данных, - это как часто следует выполнять резервирование данных. Ответ на этот вопрос зависит от того, как много данных можно позволить потерять.

Желательно – вообще не терять никакие данные, но нужно определить, какие затраты понадобятся для резервирования всех данных на случай катастрофического отказа системы. Обычно подобное резервирование подразумевает применение зеркального отображения баз данных, которое стоит довольно дорого и применяется значительно реже обычного резервирования.

Поэтому необходимо определить информацию, в действительности представляющую ценность в данный момент времени. Необходимо резервировать две различные компоненты системы: **базу данных** и **журнал транзакций**. Резервирование баз данных - это получение копий БД в определенный момент времени, а журнал транзакций содержит все изменения с тех пор, как сделаны эти копии. Наиболее оптимальным является резервирование базы данных каждую неделю, а журнала транзакций - каждый день. Наилучшим подходом в резервировании является система ротации резервирования (rotating backups). Такая система имеет самую высокую надежность и обеспечивает хорошую устойчивость к любым отказам системы, которые могут произойти.

Резервные копии используются **для восстановления**, которое означает в первую очередь восстановление самой базы данных, т.е. возвращение базы данных в правильное состояние, если какой-либо сбой сделал текущее состояние неправильным или подозрительным. Основной принцип, на котором строится такое восстановление, достаточно прост – это **избыточность**. Но эта избыточность организуется на физическом уровне. Такая избыточность будет скрыта от пользователя, а следовательно, не видна на логическом уровне. Другими словами, если любая часть информации, содержащаяся в базе данных, может быть реконструирована из другой хранимой в системе избыточной информации, значит база данных восстанавливаема.

Система должна быть готова к восстановлению не только после небольших локальных нарушений, таких как невыполнение операций в пределах определенной транзакции (см.4.2), но также после глобальных нарушений типа сбоев в питании центрального процессорного устройства. Местное нарушение по определению поражает только транзакцию, в которой оно собственно и произошло. Глобальное нарушение поражает сразу все транзакции и, следовательно, приводит к значительным для системы последствиям. Существует два вида глобальных нарушений, вызванных отказами.

1. Отказ системы, поражающий все выполняющиеся в данный момент транзакции, но физически не нарушающий базу данных в целом. Такие нарушения в системе называют аварийным отказом программного обеспечения.

2. Отказ носителей информации или аварийный отказ аппаратуры (например, поломка головок дискового накопителя).

Критической точкой в отказе системы является потеря содержимого основной (оперативной) памяти (в частности, рабочих буферов базы данных). Точное состояние какой-либо имеющейся в момент нарушения транзакции неизвестно, транзакция может не завершиться успешно и, таким образом, будет отменена («прокручена» назад) при перезагрузке системы.

Восстановление носителей после нарушений включает перезапись (или восстановление) базы данных с резервной копии и последующее использование файла регистрации – как его активной, так и архивной части. Такое восстановление – это повторное выполнение всех транзакций, вызванное применением резервной копии. При этом нет необходимости отмены транзакций, которые выполнялись в момент отказа носителей, поскольку по определению все обновления таких транзакций полностью утеряны. Поэтому для восстановления базы данных необходима информация журнала транзакций.

Кроме резервирования и восстановления, многие серверы баз данных поддерживают дополнительную опцию устойчивости к отказам - **зеркальное отображение** (mirroring). Стоимость систем зеркального отображения может быть весьма высокой, но поскольку жесткие диски являются наиболее слабым звеном систем и подвержены отказам, которые затрагивают данные, можно реализовать возможность установки зеркального отображения как средства, помогающего защитить базы данных. Зеркальное отображение происходит на уровне устройств и прозрачно для приложения. Если устройство должно быть зеркально отображено, при его создании следует указать серверу БД, где необходимо расположить **зеркало**. Нужно тщательно выбирать местоположение зеркала. Нельзя располагать его на том же самом физическом устройстве, что и источник. При такой конфигурации отказ устройства приведет к потере как источника, так и зеркала. Зеркальное отображение выполняется, когда СУБД пишет изменения в два места одновременно. Основным местоположением является стандартное устройство. Это место, где обычно происходит работа с базой данных, и которое включено в системное приложение. Резервное устройство может представлять собой какое-нибудь другое устройство или, в крайнем случае, другой файл на диске. Так как оба устройства идентичны с точки зрения информационного содержания, то переключение с основного устройства на дополнительное не вызывает никакой потери данных и пользователи данной системы не прекращают работу при сбоях в системе.

Следует отметить, что сетевые ОС также поддерживают несколько опций устойчивости к отказам, которые работают на уровне операционной системы.

4.4. АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ

Администрирование баз данных предусматривает выполнение функций, направленных на обеспечение надежного и эффективного

функционирования системы баз данных, адекватности содержания базы данных информационным потребностям пользователей, организации доступа к базам данных.

Когда говорят об управлении доступом, то различают два вида сущностей - активные и пассивные.

Активные сущности (субъекты) осуществляют доступ к пассивным сущностям - объектам. Субъекты - это пользователи или, точнее, приложения, выступающие от их имени.

Объектами, к которым регулируется доступ (или первичными объектами), являются базы данных, таблицы, представления и процедуры. Возможность для субъекта пользоваться другими объектами, например формами, отчетами, графиками и т.п., определяется доступом к первичным объектам.

Права доступа - это права, дающие возможность доступа к объекту базы данных. Право доступа предоставляется пользователю или группе пользователей для выполнения таких функций, как выборка данных, добавления новых строк или обновление данных.

При решении проблемы доступа к элементам базы данных необходимо рассматривать целый ряд факторов:

- доступ, управляемый значением (в этом случае принятие решения о доступе зависит от текущих значений данных);

- доступ, зависящий от состояния (в этом случае основную роль играет текущее состояние СУБД, например, пользователю может быть разрешено открывать данный файл только тогда, когда база данных, включающая этот файл, находится в незаблокированном состоянии);

- доступ, зависящий от полномочий (в этом случае принятие решения о доступе к данным основывается на полномочиях, предоставляемых пользователю для использования данных).

Можно выделить несколько видов прав доступа:

- **администратор системы -** владеет всеми базами данных, может выполнять все операции по управлению данными, дает санкции другим пользователям на использование баз данных;
- **владелец базы данных** - в любой системе тот, кто создает базы данных, автоматически становится их владельцем, который распоряжается личными базами данных, может выполнять все операции по управлению данными в своих базах данных и давать разрешение другим пользователям работать со своей базой;
- **пользователи -** те, которые не являются владельцами баз данных, могут выполнять лишь некоторые операции по управлению данными и не имеют права наделять полномочиями других пользователей.

4.5. ОГРАНИЧЕНИЯ ЦЕЛОСТНОСТИ

Термин **целостность** используется для описания точности и корректности данных, хранящихся в базе данных. При обеспечении целостности система должна содержать сведения о тех правилах - ограничениях целостности (сохранение непротиворечивости данных и соответствие структурным спецификациям конкретных СУБД), которые пользователю не следует нарушать. Кроме того, СУБД каким - то образом должна следить за соблюдением заданных правил при выполнении пользователем некоторых операций.

Целостность данных должна обеспечить гарантированную сохранность информации в базах данных при отказах аппаратуры или сбоях программ (защита от потери данных), а также обеспечить согласованность информации при попытках ее искажения (защита от искажения данных). Обычно ограничения целостности применяются для описания базовых отношений, поскольку базовые отношения отражают реальность и, следовательно, должны регулироваться таким образом, чтобы содержать корректные или, по крайней мере, правдоподобные значения. Производные отношения также могут регулироваться ограничениями целостности, которые наследуются автоматически или могут быть заданы. Аспекты целостности необходимо учитывать как при проектировании базы данных, так и во время ее использования.

Для обеспечения целостности данных при вводе можно использовать функцию проверки достоверности введенного значения. Альтернативным решением этой проблемы является запрет ввода данных непосредственно в поле таблицы БД и разрешение использовать для этого только выбор значения из возможного списка значений. Конечно, не всегда возможно предугадать все значения, которые может принять данная переменная, но при этом всегда имеется возможность ограничить эти значения, т.е. установить диапазон значений.

Ссылочные ограничения отвечают за целостность связей между главной и подчиненной таблицами, логически связанных между собой. Подобное ограничение требует, чтобы каждому значению в столбце или группе столбцов в подчиненной таблице соответствовало ровно одно значение в другой - главной таблице. Ограничения всех видов накладываются владельцем таблицы и влияют на исход последующих операций с данными. При обнаружении нарушений ограничений СУБД должна сигнализировать о ненормальном завершении и аннулировать внесенные оператором изменения.

Для обеспечения ссылочной целостности данных при вводе в подчиненную таблицу ключевых связующих значений можно использовать функцию проверки наличия введенного значения в главной таблице. Альтернативным решением этой проблемы является запрет ввода данных

непосредственно в поле таблицы БД и разрешение использовать для этого только выбор значения из главной таблицы. При вводе данных, имеющих отношение один-к-многим, при добавлении экземпляра записи - члена отношения в подчиненную таблицу, необходимо предусмотреть автоматическое добавление значений связующих полей (для связи с главной таблицей), иначе новые записи не будут даже отображаться в окне подчиненной таблицы.

При удалении данных чаще всего используются два режима обеспечения ссылочной целостности данных: запрет на удаление и каскадное удаление.

При использовании режима **запрет на удаление** невозможно удалить запись в главной таблице, если хотя бы в одной подчиненной таблице имеется на нее ссылка. Для поддержки этого режима необходимо перед удалением записи проверить все подчиненные таблицы и разрешить удаление записи только в том случае, если ни в одной из них не содержится ссылка на данную запись.

При использовании режима **каскадное удаление** при удалении записи главной таблицы удаляются все записи подчиненных таблиц. Для поддержки этого режима необходимо при удалении записи главной таблицы удалить и все связанные с ней записи подчиненных таблиц.

Проверка целостности данных может осуществляться программным способом. Например, при добавлении в таблицу какой-либо информации, в связанной с ней таблице можно проверить, правильно ли была произведена запись. Но более правильным является использование первичных и внешних ключей таблиц, которые автоматически отслеживают целостность данных с помощью так называемых триггеров (удаления, добавления и изменения) и хранимых процедур. Назначение триггеров (triggers) и хранимых процедур (stored procedure) может быть самое различное, но основная цель – обеспечение корректности работы пользователей с базой данных, безопасности и защиты баз данных, поддерживание целостности таблиц.

Триггер инициализируется на определенное действие, связанное либо с обновлением, любо с удалением, либо с добавлением данных в таблице. На каждое такое действие должен быть определен отдельный триггер. Выполнение триггера происходит непосредственно в месте хранения БД (в архитектуре клиент-сервер – на сервере).

Хранимые процедуры – процедуры, которые инициализируются приложением клиента и выполняются либо на локальной машине, либо на сервере приложений. Это своего рода подпрограммы, которые срабатывают в нужный момент, приняв переданные параметры от пользователя и выдав конечный результат пользователю. Хранимые процедуры позволяют применять правила и другую логику, помогающую контролировать вводимую в систему информацию. Они хранятся в самой базе данных и являются, можно сказать, интеллектуальным ядром базы данных.В хранимые

процедуры можно передавать значения и получать посредством них результаты работы, даже не имеющие отношение к рабочей таблице. Хранимая процедура может вычислить результаты в процессе работы. Хранимые процедуры могут возвращать значения и модифицировать их. Внутри процедур можно выполнять сравнение вводимых пользователем значений с заранее установленной в системе информацией.

Хранимые процедуры являются способом, с помощью которого можно создавать подпрограммы, работающие на сервере и управляемые его процессами. Подобные подпрограммы могут быть активизированы вызывающим их приложением. Кроме того, они могут быть вызваны правилами, поддерживающими целостность данных, или триггерами. Хранимые процедуры могут выступать в качестве инструмента улучшения безопасности. Можно создать хранимые процедуры, осуществляющие операции добавления, изменения, удаления и отображения списков. Таким образом можно получить контроль над каждым из аспектов доступа к информации. Использование для этого хранимых процедур является наилучшим решением, так как они обладают специфическими свойствами, определяющими их преимущества. Хранимые процедуры в архитектуре клиент-сервер располагаются на сервере. Так как SQL-серверы управляют базами данных, они являются лучшим местом для выполнения хранимых процедур, работающих с данными. Сервер — это, как правило, более мощная машина, поэтому время выполнения процедуры на сервере значительно меньше, чем на рабочей станции. Кроме того, информация из базы данных и хранимая процедура находятся в одной и той же системе, поэтому время, затрачиваемое на передачу записей по сети физически отсутствует. Хранимые процедуры имеют непосредственный доступ к базам данных, что делает работу с информацией очень быстрой. Хранимые процедуры применяют в работе мощные аппаратные решения SQL-серверов, они ориентированы на базы данных. Это позволяет получить высокую производительность при обработке данных.

Хранимые процедуры компилируются при первом выполнении и сохраняются в системной таблице текущей базы данных. При компилировании они оптимизируются. При этом выбирается самый лучший путь доступа к информации таблицы. Подобная оптимизация учитывает реальное положение данных в таблице, доступные индексы, загрузку таблицы и т.д. Откомпилированные хранимые процедуры могут значительно улучшить производительность системы.

Хранимые процедуры могут быть выполнены либо на локальной машине, либо на удаленном SQL-сервере БД. Это дает возможность активизировать процессы на других машинах и работать не только с локальными базами данных, но и с информацией на нескольких серверах.

Прикладные программы, написанные на таком языке высокого уровня, как C, могут также выполнять хранимые процедуры, что обеспечивает

оптимальное решение по распределению нагрузки между программным обеспечением и клиентской, и серверной частей. При разработке клиентского приложения программист сосредотачивает свои усилия на представлении и анализе данных, а управлением данными занимается сервер базы данных. Разработчику при создании нового приложения не нужно каждый раз проектировать БД и решать вопросы, связанные с созданием хранимых процедур на сервере БД.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что собой представляет защита информации?
2. В чем заключается безопасность данных?
3. Какие существуют аспекты проблемы безопасности данных?
4. Какие существуют основные направления защиты и безопасности информационных систем на основе технологии баз данных?
5. Какие меры включает система защиты и безопасности информации и что они собой представляют?
6. Какими возможностями по организации защиты и безопасности обладают серверы баз данных?
7. В чем состоит отличие между избирательным и мандатным (обязательным) управлением обеспечения безопасности данных?
8. Какую роль играют криптографические методы в защите и безопасности данных?
9. Что собой представляют транзакции, в чем состоит их основное назначение, какие требования предъявляются к транзакциям в технологии баз данных?
10. Что собой представляют журналы транзакций и для чего они используются в СУБД?
11. Что собой представляют блокировки и для чего они используются в СУБД?
12. Для чего применяется резервирование баз данных и журналов транзакций?
13. Чем определяется необходимость зеркального отображения устройств?
14. Какая роль отводится администрированию данных в системе защиты и безопасности данных?
15. Что собой представляет право доступа и для чего оно используется в СБД, какие существуют виды прав доступа?
16. Каким образом можно обеспечить целостность данных в СБД?
17. Что собой представляют триггеры и хранимые процедуры и для чего они используются в СБД?
18. Где располагаются хранимые процедуры в архитектуре клиент-сервер при использовании серверов баз данных?

Глава 5. ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

5.1. ОБЩИЕ ВОПРОСЫ ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Информационная система (ИС) - это система сбора, хранения, обновления, обработки и выдачи информации, необходимой пользователям для управления, для принятия решения, для проектирования и т.д. Информационную систему можно рассматривать как некий механизм для хранения, поиска, извлечения и модификации некоторой информации. Прообразом современных информационных систем могут служить архивы и библиотечное дело. Современные информационные системы базируются на компьютерных технологиях. Задачи информационных систем остались, тем не менее, прежними.

Выделяют различные классы автоматизированных информационных систем: информационно-справочные, информационно-поисковые, системы поддержки принятия решения, информационно-логические и др. Общим для них является то, что все они предназначены для регистрации, хранения и переработки информации с целью поиска и выдачи ответов на запросы пользователей. Автоматизированные информационные системы являются подклассом класса обобщённых динамических систем.

В настоящее время информационные системы в большинстве случаев разрабатываются на основе реляционных систем баз данных. Каждая информационная система имеет свою предметную область, представляющую интерес для пользователя, которая является информационным пространством, содержащим совокупность информационных объектов.

Классический пример информационной системы - это система резервирования авиабилетов. В ней хранится информация о рейсах, о том, сколько мест и какого класса есть на каждом рейсе, сколько билетов продано и сколько зарезервировано и т.д. Оператор, работая с такой системой, ничего не знает о том, как хранится и обновляется информация, как вообще сделана эта система. Оператор знает, как работать с этой системой - для бронирования авиабилета он выбирает кнопку с названием "Забронировать место". А нажатие этой кнопки приводит к изменениям в данных, отражающих факт резервирования билета на определенный рейс. Другой оператор, резервирующий билеты на этот же самый рейс, должен увидеть на одно свободное место меньше.

Информационная система - это комплексная система, включающая организационную часть, аппаратуру, программы и, собственно, информацию. В ИС можно выделить **техническое** обеспечение, **программное** обеспечение, **информационное** обеспечение.

Техническое обеспечение ИС - это совокупность технических средств, обеспечивающих сбор, обработку, хранение и передачу информации.

Программное обеспечение ИС включает обычно несколько прикладных программ (приложений), которые в совокупности позволяют достичь цели, поставленной перед разработчиком информационной системы. Предъявляются высокие требования к надёжности программного обеспечения, а также к свойствам мобильности (переносимость на различные типы ЭВМ) и адаптивности к изменяющейся операционно-технической среде.

При использовании технологии баз данных ядром программного обеспечения является СУБД, а ядром информационного обеспечения является база данных. Поэтому вопросы проектирования ИС находятся в тесном взаимодействии с вопросами проектирования баз данных. Проектирование любой БД проводится с учетом цели разработки всей ИС, так как это один из вопросов проектирования ИС. Информационное обеспечение обычно включает в себя различные справочники, классификаторы, кодификаторы, средства формализованного описания данных. Весь процесс проектирования ИС - итеративный, т.е. если не достигнут желаемый результат, то предыдущие этапы повторяются.

При проектировании ИС реальные информационные процессы отображаются через восприятие служебных функций отдельных пользователей, объединяются с точки зрения работы информационной системы в целом, и на этой основе разрабатывается информационная модель предметной области, которая физически реализуется в виде программы для компьютера и баз данных, размещённых на физическом носителе информации (например, на магнитном или оптическом диске).

Для разработчика прикладных программ представление об информационных моделях важно, в первую очередь, с точки зрения правильной постановки задачи. Путь до реальной базы данных достаточно сложен и требует многочисленных обсуждений с заказчиком и конечными пользователями, представление которых о будущей задаче должно быть обобщено в концептуальной модели. Чем ближе эта модель отображает реальные потребности в обработке данных, тем быстрее программист сможет в виде блок-схем и алгоритмов описать внутреннюю модель системы и перейти к её физической реализации.

Ниже рассмотрены те задачи, которые должны быть решены при разработке информационной системы, то есть те функции, которые должна позволять реализовывать СБД:

- обеспечивать работу пользователя (оператора) по извлечению и модификации данных;
- обеспечивать одновременный доступ нескольких пользователей к базе данных;
- предоставлять возможность выполнения административных действий по поддержанию работоспособности информационной системы;

- обеспечивать идентификацию и разграничение прав доступа разных пользователей к разным данным;

- обеспечивать целостность и непротиворечивость данных в случае аппаратных и программных сбоев;

- защищать данные от несанкционированного доступа.

Ниже эти требования рассмотрены более подробно.

Обеспечивать работу пользователя (оператора) по извлечению и модификации данных. Вообще говоря, это основная, видимая сторона любой информационной системы. Механизм реализации этих возможностей должен быть скрыт от пользователя, то есть пользователь просто нажимает некоторую кнопку (например, “Зарезервировать билет”), а уж как это реализовано внутри - это должно быть скрыто.

Обеспечивать одновременный доступ нескольких пользователей к базе данных. При одновременной работе нескольких операторов по резервированию авиабилетов не должны возникать конфликты типа того, что оба видят на экране своих терминалов свободное место под номером “10А” и оба выписывают на него билеты. С другой стороны, недопустима ситуация, когда один оператор, работая с тем или иным рейсом, блокирует работу других операторов с этим же рейсом.

Предоставлять возможность выполнения административных действий по поддержанию работоспособности информационной системы. В реальной информационной системе очень важно обеспечить технологичность, т.е. информационная система должна позволять добавлять или удалять пользователей, настраиваться на новые аппаратные ресурсы и т.д.

Обеспечивать идентификацию и разграничение прав доступа разных пользователей к разным данным. В реальных информационных системах должно существовать разграничение для разных категорий пользователей по правам доступа к данным. Какие-то пользователи могут и читать и модифицировать данные, какие-то пользователи могут только читать, а кто-то вообще может только вводить данные, а читать не имеет право. Например, любой посетитель в аэропорту может иметь возможность просмотреть список обычных пассажирских авиарейсов и количество свободных мест на них, но изменять эти данные он не имеет право.

Обеспечивать целостность и непротиворечивость данных в случае аппаратных и программных сбоев. Несмотря на то, что за последнее время надежность аппаратной части существенно выросла, абсолютно надежной техники не бывает. В случае внезапного выключения и последующего включения компьютера, на котором работает система, информация не должна быть искажена. Также информационная система должна быть максимально устойчива к программным ошибкам (или к злому умыслу). Например, следует исключить возможность выдачи билета на несуществующий рейс. В случае выхода аппаратуры из строя должна

существовать возможность восстановить работоспособность системы с минимальными потерями информации.

Защищать данные от несанкционированного доступа. В большинстве информационных систем значительная часть данных является конфиденциальной. Информационная система должна обеспечивать защиту от попыток получить доступ к данным тем лицам, которые не являются пользователями информационной системы.

Создание ИС начинается с анализа предметной области и формулировки и анализа требований пользователя. Определяются функции программного обеспечения, подлежащие реализации в системе. Выделяются параметры предметной области, необходимые для выполнения запросов пользователей. На основе проведенного анализа осуществляется постановка задачи, разработка схемы базы данных, алгоритмов решения задач ИС, их реализация.

Вопросы, рассматриваемые при проектировании ИС**,** являются общими независимо от типа используемой модели данных (РМД, СМД, ИМД). Но так как РСБД практически вытеснили системы другого типа, то все, что касается детализации проектирования ИС, излагается с ориентацией на БД реляционного типа.

В ИС отображение предметной области представлено моделями нескольких уровней (см.1.2). Информационное моделирование в СБД имеет ряд особенностей, вызванных с одной стороны идеологией СБД, а с другой - особенностями используемой СУБД. Процесс разработки ИС - приложений баз данных можно условно разделить на определенные этапы (любой разработчик так или иначе выполняет все эти этапы, какое бы приложение ни разрабатывалось):

- определение назначения и круга задач приложения;
- изучение и анализ предметной области, определение форм входных и выходных документов;
- определение объектов, их свойств и взаимосвязей, разработка информационной модели предметной области;
- представление выбранных объектов, свойств, взаимосвязей в терминах реляционной модели, разработка инфологической модели;
- выбор СУБД и разработка даталогической модели предметной области, создание базы данных в выбранной СУБД;
- разработка структуры приложения, обеспечивающего работу с базой данных;
- разработка алгоритмов решения задач информационной системы;
- реализация приложения в среде выбранной СУБД;
- тестирование приложения на предмет соответствия его назначению и правильного его функционирования;
- настройка и установка приложения;

- разработка инструкций пользователя (администратора БД, конечного пользователя, программиста).

При проектировании систем с архитектурой клиент-сервер имеются свои особенности**.** Одним из преимуществ системы клиент-сервер является то, что каждый функциональный компонент системы можно специализировать для наилучшего выполнения тех или иных операций. Это приводит и к преимуществу разработки системы в архитектуре клиент-сервер, которое заключается в возможности раздельного создания программного обеспечения клиента и сервера, что позволяет уменьшить время, необходимое для разработки системы. При разработке клиентского приложения программист сосредотачивает свои усилия на представлении и анализе данных. Тем временем управлением данными занимается сервер базы данных. Разработчику при создании нового приложения не нужно каждый раз проектировать БД.

5.2. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

БД является наиболее "долгоживущим" компонентом ИС. Жизненный цикл системы БД делится на две фазы: анализ и проектирование, реализация и функционирование.

В **фазе анализа и проектирования** анализируется предметная область и требования, которым должна удовлетворять БД. Исходя из результатов анализа определяются атрибуты (свойства) объектов, интересующие пользователя, и разрабатывается информационная модель решаемой задачи в виде объектов и взаимосвязей. Тип взаимосвязи определяется связями между объектами, существующими в реальном мире. Для каждого объекта определяется структура таблицы, с учетом необходимых свойств. Необходимо предусмотреть поля для первичных ключей и внешних ключей для установления взаимосвязей между таблицами. При наличии взаимосвязи "многие-к-многим" необходимо ввести промежуточную таблицу для промежуточного объекта, которая будет содержать идентификаторы связываемых объектов. Присутствие промежуточного объекта необходимо по причине невозможности установления в информационной модели между двумя объектами связи "многие-к-многим" в большинстве реляционных СУБД. Выбирается СУБД и в соответствии с требованиями выбранной СУБД проектируются таблицы и определяются параметры полей (столбцов) таблицы.

В **фазе реализации и функционирования** осуществляются процедуры генерации схемы БД, подготовки среды хранения, ввода и контроля данных, загрузки и корректировки БД в среде выбранной СУБД. Загрузка БД выполняется путем внесения в таблицы записей, между таблицами устанавливаются взаимосвязи в соответствии с информационной моделью.

Функционирование БД характеризуется выполнением следующих трёх основных процедур:

- **реорганизация** БД (реструктуризация и реформатизация БД);
- **организация доступа** к данным (поиск и обновление данных, разграничение доступа, инициализация и завершение работы СУБД);
- **контроль состояния** БД (сбор и анализ статистики работы СУБД, контроль целостности и непротиворечивости данных, копирование и восстановление данных в БД).

Проектирование БД представляет собой сложный процесс отображения описания предметной области в схему внутренней модели БД. Этот процесс разбивается на ряд более простых, обычно итеративных процессов проектирования моделей: внешней, концептуальной и внутренней. Этапы проектирования базы данных с учетом рассмотренных выше аспектов представлены на рис.5.1.

В основу проектирования БД должны быть положены представления конечных пользователей конкретной организации - концептуальные требования к системе. Именно конечный пользователь в своей работе принимает решения с учетом получаемой в результате доступа к базе данных информации. От оперативности и качества этой информации будет зависеть эффективность работы организации. Данные, помещаемые в базу данных, также предоставляет конечный пользователь. При рассмотрении требований конечных пользователей необходимо принимать во внимание следующее:

- база данных должна удовлетворять актуальным информационным потребностям организации, а получаемая информация должна по структуре и содержанию соответствовать решаемым задачам;
- база данных должна обеспечивать получение требуемых данных за приемлемое время, то есть отвечать заданным требованиям производительности;
- база данных должна удовлетворять выявленным и вновь возникающим требованиям конечных пользователей;
- база данных должна легко расширяться при реорганизации и расширении предметной области;
- база данных должна легко изменяться при изменении программной и аппаратной среды;
- данные до включения в БД должны проверяться на достоверность;
- доступ к данным должны иметь только лица с соответствующими полномочиями.

Информационная модель предметной области является исходной по отношению к **логической модели** БД. Она служит связующим звеном между специалистами предметной области и администрацией БД в процессе проектирования БД и должна отвечать следующим требованиям:



Рис.5.1. Этапы проектирования базы данных

- обеспечить адекватность отображения предметной области и давать возможность получить интегрированное представление о предметной области;

- содержать информацию о предметной области, достаточную для дальнейшего проектирования;

- гарантировать однозначность трактовки модели. Для информационной модели используются как аналитические, так и графические способы ее представления.

В системе баз данных (см. архитектуру СБД в 1.2) основной является концептуальная модель, которая включает описание всех объектов и их взаимосвязей, представляющих интерес в рассматриваемой предметной области и выявляемых в результате анализа данных. Здесь имеются в виду данные, используемые как в уже разработанных прикладных программах, так и в тех, которые только будут реализованы. Проектирование концептуальной модели основано на анализе решаемых в конкретной организации задач по обработке данных без указания способов их физического хранения. Таким образом, концептуальная модель является, по существу, детализированной моделью предметной области.

Логическая модель (инфологическая, даталогическая) может быть реляционной, иерархической или сетевой. При проектировании логической модели базы данных используется понятие атрибуты объекта. В реляционной модели значения атрибутов объекта объединены в кортеж (запись). Реляционная модель - это таблица, где строка отражает экземпляр объекта, так как в столбцах (полях) содержатся значения атрибутов конкретных объектов. Также логическая модель отражает логические связи между атрибутами объектов вне зависимости от их содержания и среды хранения. Связь представляет собой соединение между двумя или более объектами.

Решение каждой задачи проектирования зависит от результатов предыдущих, и чем тщательнее оно будет продумано, тем меньше итераций понадобится в дальнейшем.

Основные задачи проектирования базы данных:

- определение объектов (источников данных), которые должны быть включены в базу данных;

- выявление связей между объектами;

- определение основных свойств объектов;

- выявление связей между свойствами объектов;

- определение отношений между таблицами базы данных на основе связей между объектами данных, содержащихся в них;

- определение операций, выполняемых при создании и изменении информации в таблицах, включая обеспечение целостности данных;

- выявление индексов, необходимых для ускорения выполнения запросов;

- учет вопросов безопасности (какие полномочия и каким

пользователям предоставлять);

- разработка процедур создания резервных копий и восстановления исходных файлов.

В качестве примера рассмотрим вопросы, связанные с разработкой базы данных "Книжный магазин", что позволяет автоматизировать операции учета товарно-материальных ценностей в различных книготорговых организациях.

При проектировании системы учета данных в книготорговой организации можно выделить следующие основные объекты: Книга, Специализация, Поступление, Продажа, Поставщик, Покупатель.

Объект **Книга** является основной единицей учета при проведении операций купли-продажи. Объект **Специализация** определяет тематический раздел книги. Объект **Поступление** фиксирует операции поступления и движения книги на складе. Объект **Продажа** фиксирует операции сбыта книги со склада. Объекты **Поставщик** и **Покупатель** отражают данные о конкретном поставщике книги и ее покупателе. На рис.5.2 показана максимально упрощенная информационная модель системы учета и обработки данных в книготорговой организации.



Рис.5.2. Концептуальная модель базы данных «Книжный магазин»

Как изображено на рис.5.2, двойные стрелки от объекта **Книга** в направлении объектов **Продажа** и **Поступление** указывают на то, что одна книга может фигурировать во многих документах учета поступления или продажи данной книги. Аналогично стрелки от объектов **Поставщик** и **Покупатель** указывают, что один поставщик или один покупатель могут участвовать во многих операциях поступления и продажи. Одинарная стрелка между объектами **Книга** и **Специализация** говорит о том, что одна книга может принадлежать только одному тематическому разделу.

Информационная модель детализируется в инфологической концептуальной модели, которая транслируется в модель данных, совместимую с выбранной СУБД. Возможно, что отраженные в концептуальной модели взаимосвязи между объектами окажутся впоследствии нереализуемы средствами выбранной СУБД, что потребует изменения концептуальной модели. Конечная версия концептуальной модели,

которая может быть обеспечена конкретной СУБД, является даталогической моделью.

При проектировании БД руководствуются определенными целями, среди которых наиболее важными представляются следующие: возможность хранения в базе данных всех необходимых данных; исключение избыточности данных; сведение к минимуму числа хранимых в базе данных таблиц; нормализация таблиц для упрощения решения проблем, связанных с обновлением и удалением данных.

Возможность хранения в базе данных всех необходимых данных является обязательным условием, поскольку отсутствие в ней необходимой информации сводит на нет ценность всей базы данных. Поэтому важным шагом в процессе проектирования является определение всех объектов и их атрибутов, которые должны быть помещены в базу данных. После определения всех атрибутов нужно решить, сколько требуется таблиц и какие атрибуты в какие таблицы надо включать. При проектировании БД сначала определяется для каждого объекта универсальная таблица, находящаяся в первой нормальной форме - 1NF (см. 3.1) и включающая все необходимые атрибуты объекта. Универсальная таблица используется в качестве отправной точки при проектировании базы данных.

При этом необходимо следить за исключением избыточности данных. В табл.5.1 представлен объект **Книга**. Он включает атрибуты: **Наименование книги**, **Автор**, **Телефон**. Предполагается, что у каждого автора имеется только один телефонный номер. В этой таблице номер телефона автора нескольких книг будет появляться несколько раз и эта дублированная информация о телефонных номерах является избыточной.

Таблица 5.1

Наименование книги	Автор	Телефон
Эффективная работа с СУБД	Горев А.	666-66-66
Стандарт SQL	Дейт К.	999-99-99
Современные микропроцессоры	Корнеев В.	555-55-55
Visual FoxPro 5.0. Книга для программистов	Горев А.	666-66-66
Введение в системы баз данных	Дейт К.	999-99-99
Параллельные вычислительные системы	Корнеев В.	555-55-55

Способ исключения избыточности телефонных номеров заключается в разбиении исходной табл.5.1 на две таблицы, одна из которых содержит информацию о книгах и их авторах (табл.5.2), а вторая - информацию об авторах и их телефонах (табл.5.3). Табл.5.2 и 5.3 находятся во второй нормальной форме - 2NF.

Разбиение одной таблицы на две или больше таблиц меньшего размера желательно с точки зрения исключения избыточности, но не совсем удобно для пользователя. Поэтому нельзя допускать неограниченного роста числа таблиц и необходимо сводить к минимуму число таблиц, хранимых в базе данных.

Таблица 5.2

Наименование книги	Автор
Эффективная работа с СУБД	Горев А.
Стандарт SQL	Дейт К.
Современные микропроцессоры	Корнеев В.
Visual FoxPro 5.0. Книга для программистов	Горев А.
Введение в системы баз данных	Дейт К.
Параллельные вычислительные системы	Корнеев В.

Таблица 5.3

Автор	Телефон
Горев А.	666-66-66
Дейт К.	999-99-99
Корнеев В.	555-55-55

Практически для всех таблиц очень важны проблемы удаления и обновления данных. Решить эти проблемы позволяет использование нормализованных отношений - нормализованных таблиц. Поэтому необходимо нормализовать таблицы и привести их к 3NF - третьей нормальной форме.

Нормализация - это один из методов проектирования баз данных, к которому тесно примыкает метод функциональных зависимостей и приведение каждой таблицы к нормальной форме Бойса-Кодда (НФБК). Оба эти метода достаточно просто использовать при небольшом числе атрибутов - до 15-20. Нормализация отношений - таблиц в реляционных системах баз данных описана в 3.1. Если же необходимо хранить в базе данных значительно больший объем информации, то лучше воспользоваться другими методами. Одним из них является ER-метод, в котором используется схема “сущность-связь” (Entity-Relationship), разработанный в 1976 г. Питером Ченом. Ниже в 5.3 излагаются краткие основы ER-метода.

5.3. СУЩНОСТИ И СВЯЗИ

В ER-методе используется терминология, в которой основными терминами являются сущность, связь, атрибут.

Сущность определяется как некоторый объект, представляющий интерес. Этот объект должен иметь экземпляры, отличающиеся друг от друга и допускающие однозначную идентификацию. Единственный определяющий признак, который может помочь в нахождении сущностей, состоит в том, что сущность – это, как правило, существительное. Примерами сущностей могут

служить машины, банковские счета, колледжи, служащие и контракты. На рис. 5.3 и 5.4 сущностями являются КНИГА и АВТОР, в то время как отдельные экземпляры каждой сущности идентифицируются с помощью номера-книги и номера-автора, соответственно. Класс принадлежности сущности характеризует обязательность включения каждого экземпляра сущности в связь.

Связь представляет собой соединение между двумя или более экземплярами сущностей. При поиске связей в основном следует полагаться на то обстоятельство, что связь обычно выражается глаголом. Типичные примеры связей между двумя сущностями:: авторы ПИШУТ книги, студенты ИЗУЧАЮТ дисциплины, рабочие ОБСЛУЖИВАЮТ механизмы.

Тесно связанно с предыдущими третье важное понятие, а именно – атрибут. Атрибут есть свойство сущности. Например, атрибутами, которые могут быть свойствами сущности КНИГА, являются номер книги, количество страниц, тираж, цена и т.д.

Предположим, что проектируется база данных, предназначенная для хранения информации об авторах и о тех книгах, которые они написали. Двумя главными объектами, или сущностями, представляющими в данном случае интерес, являются АВТОР и КНИГА. Эти две сущности соотносятся с помощью связи ПИШЕТ, что позволяет сказать: АВТОР ПИШЕТ КНИГУ.

Связь ПИШЕТ, существующая между двумя сущностями АВТОР и КНИГА может быть графически представлена несколькими способами. Ниже рассмотрены два из них: диаграммы ER-экземпляров и диаграммы ER-типа. На рис.5.3 иллюстрируется использование диаграммы ER-экземпляров, показывающей, какую книгу какой автор пишет. В этом примере каждый автор идентифицируется номером-автора, а каждая книга – номером-книги. Рис.5.4 отображает диаграмму ER-типа (или ER-диаграмму) и содержит более общую информацию, чем рис.5.3. В то же время, рис. 5.4 содержит всю информацию, необходимую для генерации таблиц в начале проектирования для БД.

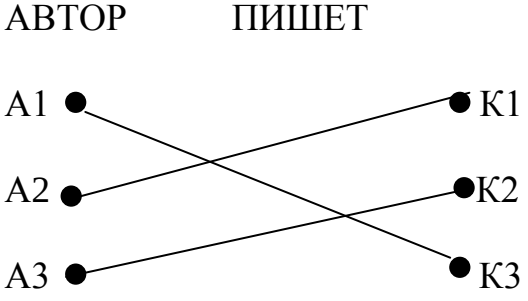


Рис.5.3. Пример диаграммы ER-экземпляров

На диаграмме ER-экземпляров (рис.5.3) названия всех сущностей помещены над экземплярами этих сущностей, и в них использованы

прописные буквы, в то время как каждый экземпляр сущности идентифицируется значением атрибута. Так, КНИГА является сущностью, а К3 – конкретным экземпляром сущности. Связь также является именованной, и ее название (ПИШЕТ), составленное из прописных букв, размещается над экземплярами этой связи. При этом экземпляр каждой отдельной связи задается линией между теми двумя экземплярами сущностей, которые эта связь соединяет. Например, экземпляр связи между А3 и К2 означает, что автор с номером автора, равным А3, пишет книгу с номером книги, равным К2.



Рис.5.4. Пример диаграммы ER-типа

В некоторых случаях для идентификации каждого экземпляра сущности может понадобиться набор атрибутов. Атрибут или набор атрибутов, используемый для идентификации экземпляра сущности, называется ключом сущности. Каждый экземпляр связи однозначно определяется набором ключей сущностей, соединяемых этой связью. Таким образом, набор А4, К3 является ключом связи. На данном этапе процесса проектирования необходимыми единственными атрибутами являются те, которые требуются для формирования ключей сущностей. Другие атрибуты вместе с определенными над ними функциональными зависимостями добавляются на более поздних этапах проектирования.

На диаграммах ER-типа, подобных показанной на рис.5.4, сущности представляются в виде прямоугольника, а связи – в виде ромбов. Ниже каждой сущности размещается атрибут, или набор атрибутов, являющийся ключом для данной сущности (Na и Nk).

Важной характеристикой связи между двумя (или несколькими) сущностями является степень связи. Степень связи может равняться 1:1 (один-к-одному), 1:M (один-к-многим), M:1 (многие-к-одному), M:M (многие-к-многим). На рис.5.4 представлена ER-диаграмма со степенью связи 1:1.

При проектировании с использованием ER-метода строится ER-диаграмма, включающая в себя все важные сущности и связи. По построенной ER-диаграмме определяется набор предварительных таблиц (каждой сущности - свою таблицу) и первичный ключ (ключ сущности) для каждой таблицы. Затем подготавливается список всех представляющих интерес атрибутов и каждый из них назначается одной из предварительных таблиц с тем условием, чтобы эти таблицы были нормализованы или приведены к НФБК. На этом шаге для каждой таблицы определяются межатрибутные функциональные зависимости, с помощью которых проверяется соответствие таблицы нормальной форме (3NF или нормальной форме Бойса-Кодда). Если полученные в итоге таблицы не приведены к

нормальной форме или если некоторым атрибутам не находится логически обоснованных мест в предварительных таблицах, то в этих случаях необходимо пересмотреть ER-диаграммы.

Особенность ER-метода заключается в том, что два проектировщика могут рассматривать одну и ту же проблему с разных точек зрения, получая в итоге различные наборы сущностей и связей. Определение лучшего из нескольких возможных наборов является вопросом личного предпочтения.

5.4. ПУБЛИКАЦИЯ БАЗ ДАННЫХ В СЕТИ ИНТЕРНЕТ

Современные информационные технологии позволяют экономить труд и время как программистов, так и специалистов по эксплуатации и поддержке программных продуктов. Всего за несколько лет пользователи перешли от эксплуатации программ на отдельных компьютерах - к сетевым системам, основанным на файл-сервере. Для удовлетворения возрастающих требований к обработке данных активно развиваются системы клиент-сервер, в том числе с использованием технологий Интернет.

Наиболее популярным сервисом в сети Интернет является сервис **WWW** - World Wide Web (всемирная паутина). В основу данной системы положено понятие гипертекста, то есть множества отдельных текстов, которые имеют ссылки друг на друга. Эти тексты называют также документами, статьями или страницами. Слова, находящиеся в одном документе, как бы "привязаны" к другим документам с помощью ссылок. Так как ссылки могут указывать на любой документ, находящийся в Интернет в любом месте земного шара, данная система и названа Всемирной паутиной. Для работы с WWW используется специальный протокол **HTTP** - Hyper Text Transfer Protocol (протокол передачи гипертекста).

Гипертекстовые документы создаются с помощью специального языка **HTML** - Hyper Text Markup Language (язык разметки гипертекста). Документ в WWW, составленный на языке HTML и доступный для просмотра пользователем, называется Web-страницей. Кроме собственно текста в документах WWW могут находиться графические изображения, звуки и даже видеоклипы. Данный сервис бурно развивается, и различная информация, ранее распространяемая другими средствами Интернет, теперь доступна из WWW. Эта служба, являющаяся одной из основных служб, с которыми работают пользователи Internet, благодаря привлекательному графическому интерфейсу и мультимедийным средствам, позволяет **публиковать** на своих **Web-страницах** (и посылать) информацию о любом предмете, в том числе информацию из баз данных.

Обеспечением пользователей всеми возможностями Интернет занимаются программы, работающие на компьютерах сети. При этом необходимы две программы. Одна программа (сервер) занимается хранением

и передачей информации по запросу других компьютеров, вторая (клиент) - устанавливается на компьютере пользователя и служит для посылки запросов на сервер, принятия и отображения полученной ответной информации. Например, WWW-сервер хранит Web-страницы и поддерживает протокол обмена HTTP. Клиентом для WWW-сервера является программа просмотра Web-страниц, называемая Web-браузером. Следует отметить, что сервером называют также и компьютер, на котором работают программы, обеспечивающие доступ к сетевым ресурсам. Web-браузер обеспечивает отображение HTML-документов.

Различают **статический** и **динамический** HTML. Статическим называют HTML-документ, однажды созданный и помещенный на Web-сервер. Когда браузер заказывает статическую HTML-страницу, Web-сервер всегда присылает ее в той форме, в какой она была создана. Динамические HTML-документы создаются на ходу в ответ на запрос от браузера. Генерирующая их программа называется расширением серверной части. Сам Web-сервер не принимает участия в процессе динамической генерации, а просто передает запрос на доставку страницы программе-расширению сервера и отсылает возвращаемый HTML-документ браузеру. Оба подхода обычно совмещаются на одном сайте WWW, содержащем базу данных. Статический HTML используется для создания форм ввода, а динамический - для отображения результатов запроса.

Сайтом называют совокупность программных средств для обеспечения доступа пользователей сети Интернет к различной информации (в том числе и к базам данных), размещенной на Web-сервере. Чтобы подчеркнуть принадлежность сайта к Интернет, часто употребляют термин Web-сайт – основная составляющая «Всемирной паутины» (World Wide Web). Наряду с предоставлением информации широкому кругу пользователей Web-сайт сам может выступать в роли пользователя для других сайтов. Но не нужно путать понятия Web-сервер и Web-сайт, так как Web-сервер - это совокупность заранее разработанного системного программного обеспечения, установленного на сервере-компьютере, а программная структура Web-сайта создается и поддерживается конечным пользователем-разработчиком. Минимальный информационный объем Web-сайта – одна HTML-страница, а максимальный зависит от технических характеристик сервера-компьютера, так как профессионально разработанный Web-сайт может состоять из множества интерактивных страниц и иметь ссылки на другие сайты в рамках всей сети Интернет.

В технологиях Интернет от СУБД будет все меньше требоваться мощных средств разработки интерфейса пользовательского приложения, так как эти функции возьмут на себя программы просмотра Интернет. Однако будет возрастать роль СУБД как серверов больших объемов данных. В этих условиях с учетом широкого распространения Web-сервиса сети Интернет и

корпоративных сетей Интранет, использующих Интернет-технологии, особенно актуальным является вопрос публикации баз данных в Интернет.

Публикация баз данных - это представление информации из баз данных. Нужно заметить, что ключевым словом здесь является "представление", то есть говорится о размещении информации из баз данных таким образом, чтобы пользователи могли получать к ней доступ при помощи специальных средств, характерных для публикации баз данных. Это не то же самое, что программирование баз данных. Системы публикации баз данных можно рассматривать как особый класс информационных систем со своими собственными программными средствами.

При работе с базой данных в Интернет может понадобиться:

- просмотреть информацию (**статическая публикация БД**);
- просмотреть, добавить и (или) отредактировать информацию (**динамический доступ к БД**).

Если при работе с СУБД эти действия не требовали существенного различного подхода в доступе к данным, то в Интернет это не так.

В случае просмотра информации будет достаточно считать данные и перевести их в формат гипертекста (HTML) для просмотра браузером. Как правило, все современные СУБД имеют для этого встроенные средства. Процедура трансляции данных в формат HTML достаточно автоматизирована и называется публикацией данных в Web. При этом создается Web-страница, на которой размещается результат запроса, и эта страница может быть размещена на Web-сервере для просмотра пользователями. Некоторые средства, например Web Assistant (СУБД Access), автоматически обновляют данные, опубликованные на Web-странице. В любом случае пользователь не знает о месте хранения данных. Для работы он нуждается только в программе просмотра Интернет и не имеет доступа к БД.

Публикация данных может быть эффективным средством доступа к информации, хранящейся в БД, в следующих случаях:

- выбор данных, которые должны быть опубликованы в Интернет, определяется не пользователем, а владельцем данных (администратором сервера Web), который периодически обновляет данные на основании текущих данных БД;
- пользователь нуждается только в просмотре данных и не должен иметь возможности их изменения;
- к результатам запроса необходимо организовать свободный доступ нескольких пользователей, поэтому отпадает необходимость индивидуального формирования запроса. Достигается экономия времени и снижается вероятность ошибок.

Если же необходимо обеспечить изменение данных или выполнение индивидуальных запросов со стороны пользователя, то требуется применение специальных программных средств.

Как правило, изменение данных или выполнение индивидуальных запросов при работе в Интернет требуется в следующих случаях:

- пользователи нуждаются в данных, требования к которым трудно формализовать и, таким образом, невозможно заранее сформировать запрос к данным;
- данные изменяются очень интенсивно, в режиме реального времени;
- пользователи должны изменять данные в соответствии с изменяемыми условиями работы.

Приложения, работающие с базами данных для Интернет, используют новые типы программных компонентов, организованных также по-новому. Процесс развития еще продолжается, но уже выделяют несколько категорий архитектур и программного обеспечения. Ниже рассматривается один из вариантов архитектур.

Инструментарий WWW и базы данных - это две отдельные технологии, разрабатывающиеся независимо друг от друга. Но и та и другая основываются на двухуровневой технологии клиент-сервер. Приложения, работающие с базами данных для WWW, объединяют свои двухуровневые технологии-предшественницы в новый тип системы, основанный на трехуровневой архитектуре клиент-сервер. Как показано на рис.5.5, клиентский уровень занимает браузер, на уровне сервера располагается сервер базы данных, а на промежуточном уровне находятся Web-сервер и программа-расширение сервера.

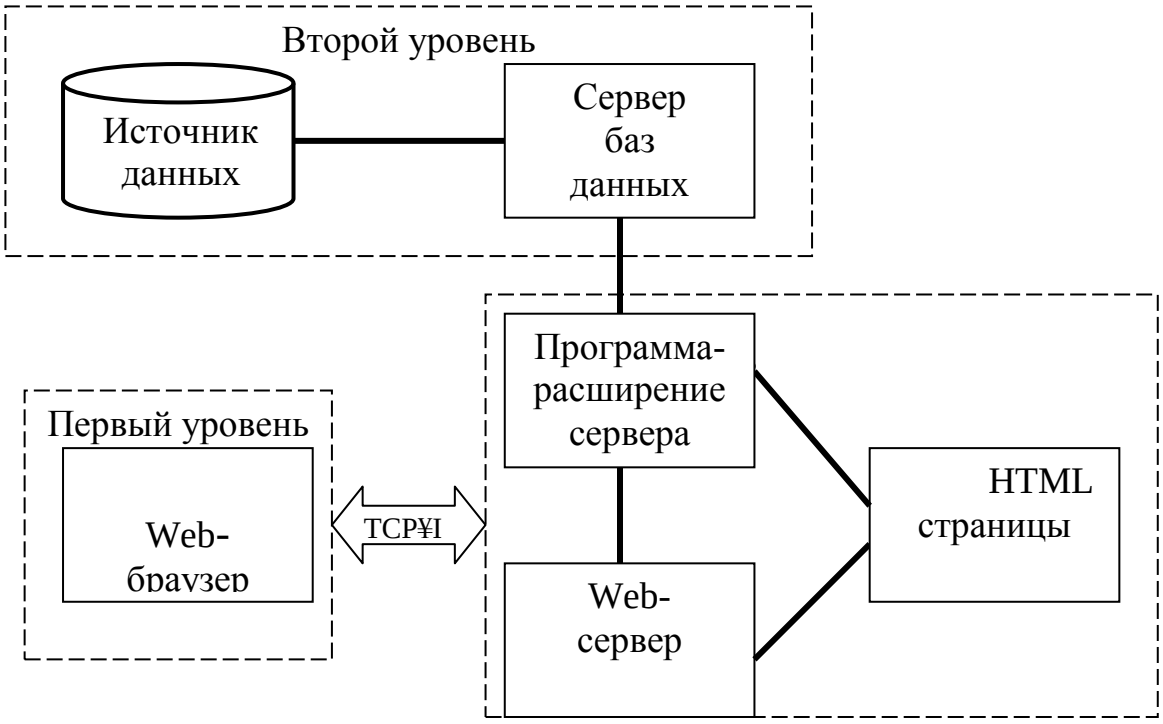


Рис.5.5. Трехуровневая архитектура приложений баз данных для Интернет

Такое архитектурное решение позволяет уменьшить сетевой трафик, делает компоненты взаимозаменяемыми и повышает уровень безопасности. Однако такая архитектура также затрудняет обработку транзакций базы данных ввиду природы протокола HTTP, не запоминающего состояния (этот протокол используется для передачи данных между браузером и сервером базы данных).

Браузер посылает Web-серверу запросы на доставку Web-страниц или данных. Web- сервер обслуживает заявки на Web-страницы, а запросы данных отправляет программе-расширению серверной части. Последняя принимает передаваемые ей запросы, преобразует их в форму, понятную серверу базы данных (например, в запросы ODBC SQL) и передает в сервер БД. Затем сервер баз данных выполняет работу по обслуживанию запроса (например, добавление записи) и возвращает результат программе-расширению серверной части, которая преобразует результат в формат HTML, приемлемый для браузера, и передает их Web-серверу, а тот в свою очередь - браузеру.

Все современные пакеты СУБД имеют встроенные программные средства, обеспечивающие либо статическую публикацию баз данных в Интернет, либо позволяющие организовывать динамический доступ через Интернет к базам данных (например, Visual FoxPro WWW Search Page Wizard, DbWeb в средствах корпорации Microsoft и др.).

Вопросы **защиты публикации** БД в Интернет стоят особенно остро, поскольку информация, прежде чем попасть в пункт назначения, проходит длинный путь через множество серверов. Путь (совокупность программных и аппаратных средств), по которому передается информация, большей частью не контролируется. Уровень защиты - это электронный фильтр, защищающий внутренние сети компаний от сетей, подключенных к Web-серверу. Системы защиты являются прозрачными для информации различного рода, будь то деловая информация, мультимедиа, электронная почта или видеоконференции и служат для защиты информационно-вычислительной системы в целом.

Простейший уровень защиты - это **запрос пароля** (ключевого слова). Ключевое слово, как правило, хранится в программе в виде таблицы, которую администратор сети может редактировать. В зависимости от того, какое ключевое слово будет введено, пользователь получает соответствующие полномочия доступа: от полного отказа подключиться к системе - до предоставления полного контроля над ней.

Другой тип контроля доступа позволяет **блокировать доступ** на уровне TCP/IP протокола, рассматривая, например, адрес получателя и отправителя, протокол обмена, порт подключения. Такие системы, как правило, реализованы в **системных маршрутизаторах.**

Более разумные системы контроля доступа используют так называемые **бастионные узлы** (шлюзы), которые представляют собой

отдельный компьютер, обеспечивающий соединение локальной сети пользователя с Интернет. Бастионные узлы обычно используют операционную систему UNIX. На такой узел устанавливают несколько программ, обеспечивающих обмен данными между внутренней сетью и внешней - Интернет. Бастионный узел работает как фильтр, отсеивающий все нежелательные запросы к данным. Все желающие подключиться к внутренней сети должны сначала войти в бастионный узел с помощью системы опроса паролей.

Методы защиты постоянно развиваются вместе с Интернет, но в настоящее время разработано и применяется два метода защиты.

Первый метод, который используется в программном обеспечении фирмы Netscape Communication, называется SSL (Secure Socket Layer protocol). Как клиентская, так и серверная часть Netscape содержит в себе реализацию **криптографической технологии**, которая и позволяет защищать передаваемые данные. Только браузер конкретного пользователя и программа на Web-сервере в состоянии расшифровать переданные или принятые данные. В процессе передачи данных между промежуточными серверами Интернет информационный блок представляет собой совокупность несвязных кодов. В принципе пользователи промежуточных серверов могут перехватить такие сообщения, но понять что-либо из них они не смогут, а значит информация останется защищенной.

Используя криптографические технологии, Netscape способен производить аутентификацию серверов Интернет, которая предотвращает возможность подмены сервера пункта назначения - промежуточным.

Единственными программами, поддерживающими SSL, являются Web-браузер Netscape Navigator и Web-сервер Netscape Commerce Server. Недостаток системы в том, что она несовместима ни с каким другим программным обеспечением.

Другой стандарт защиты называется SHTTP (Secure HTTP - Протокол Передачи Защищенного Гипертекста) и разработан консорциумом W3, в состав членов которого входят: Digital Equipment Corporation, AT&T, MCI, IBM, Novell Inc., Netscape Communication Corporation, the National Center for Supercomputing Applications (NCSA) и Microsoft Corporation.

Достоинство SHTTP заключается в гибкости системы. SHTTP может определить, с какого сервера взят тот или иной документ и тем самым проконтролировать нарушение защиты. Система поддерживается программами нескольких фирм, а значит, является более совершенным стандартом, чем SSL. Но вместе с тем SHTTP использует такой же механизм криптографии, как и SSL, поэтому SHTTP можно назвать SSL-совместимой системой.

Для правильной работы систем защиты их нужно корректно конфигурировать, так как малейшие ошибки могут открыть широкую дорогу для несанкционированного доступа к данным.

КОНТРОЛЬНЫЕ ВОПРОСЫ И УПРАЖНЕНИЯ

1. Что собой представляет информационная система?
2. Из каких компонентов состоит информационная система?
3. Что является ядром информационноного обеспечения ИС?
4. Что собой представляет программное обеспечение ИС?
5. Какие требования предъявляются к ИС?
6. Какие этапы выполняются при разработке ИС?
7. Каким требованиям должна удовлетворять база данных ИС?
8. Что собой представляет информационная модель предметной области ?
9. Какую роль играет информационная модель предметной области при проектировании базы данных информационной системы?
10. Что собой представляет логическая модель базы данных информационной системы?
11. Какие основные задачи решаются при проектировании базы данных информационной системы?
12. В чем заключается инфологическое проектирование базы данных ИС?
13. В чем заключается даталогическое проектирование базы данных ИС?
14. В чем заключается физическое проектирование базы данных ИС?
15. Создайте концептуальную модель (информационную и логическую) базы данных ИС обслуживания читателей в библиотеке (объекты Книга, Читатель, Формуляр выдачи книг), используя нормализованные отношения.
16. Создайте концептуальную модель (информационную и логическую) базы данных ИС учета товара в магазине (объекты Поставщик, Товар, Покупатель), используя нормализованные отношения.
17. Составьте диаграмму ER-типа и диаграммы ER-экземпляров для сущностей “студент”, “дисциплина”, используя связь “изучает”. Определите ключи сущностей, их атрибуты, степень связи. Сопоставьте каждой сущности и связи свою таблицу, каждой таблице – соответствующий набор атрибутов.
18. Какой сервис сети Интернет используется для публикации баз данных?
19. На чем основан принцип работы в WWW-системе?
20. В чем состоит различие между Web-сервером и Web-сайтом?
21. Какие разновидности публикации баз данных возможны в сети Интернет?
22. Какая технология используется для публикации баз данных в сети Интернет?
23. Какие методы защиты данных используются в Web-браузерах?
24. Что собой представляет бастионный узел?

Глава 6. ЯЗЫК ПРОГРАММИРОВАНИЯ XBASE

6.1. ОСНОВНЫЕ ПОЛОЖЕНИЯ

Язык программирования xBase относится к языкам четвертого поколения **4GL** (4 Generation Language) и используется в реляционных dBase-подобных СУБД. Родоначальником семейства dBase-подобных СУБД является СУБД dBase, предложенная фирмой Ashton-Tate в начале 80-х годов. В это семейство, кроме СУБД dBase, входят системы управления базами данных, разработанные другими фирмами. Их объединяет входной dBase-подобный язык процедурного типа с некоторыми отличиями, известный как язык программирования xBase. Далее рассматриваются основные положения и команды языка программирования xBase, так как он используется в любых реализациях dBase-подобных СУБД, в том числе и в объектно-ориентированных версиях с визуальными средствами программирования. В случае наличия нескольких версий одной и той же команды, используемых в различных СУБД, описывается версия команды, реализованная в СУБД FoxPro. СУБД FoxPro является одной из наиболее распространенных и высокоскоростных систем управления БД для персональных компьютеров и характеризуется умеренными требованиями к вычислительным ресурсам, что позволяет использовать ее на персональных компьютерах любого класса. В настоящее время существует несколько модификаций этой системы, ориентированных на работу в широком диапазоне, как операционных систем (от MS DOS до Unix), так и аппаратных платформ (от IBM PC и их аналогов до мини-ЭВМ типа VAX).

Команды манипулирования данными языка программирования xBase предназначены для работы с отношениями - таблицами данных, состоящими из полей (столбцов) и записей (строк). Таблицы хранятся в файлах данных, имена которых имеют расширение **.DBF**. Каждое поле таблицы имеет свое **имя, тип, длину** (количество символов в поле для одной записи) и **точность**- количество дробных знаков после запятой (для числовых полей). В различных dBase-подобных СУБД набор типов полей различается, но некоторые типы полей применяются во всех СУБД и не только dBase-подобных. К ним относятся следующие типы полей: **символьный** - Character (значение поля - любые символы, в том числе и буквы любого алфавита), **числовой** - Numeric или Float (значение поля - любые числа), **дата** - Date (значение поля - дата с указанием дня, месяца и года), **логический** - Logical (значение поля - .F.(False-ложь) или .T.(True-истина)), поле типа **примечание** - Memo.

Команды языка xBase можно использовать в двух режимах. Интерактивный режим работы - это работа в диалоговом режиме в интерактивной среде СУБД: команда - результат команды. Программный режим работы представляет собой выполнение в среде СУБД программ,

написанных на языке программирования xBase. Все СУБД, использующие xBase, поддерживают встроенный язык управления данными SQL. Поэтому в программы xBase можно включать команды SQL.

В командах языка программирования xBase, так же как и в алгоритмических языках, могут использоваться прописные и строчные буквы алфавита и знак подчеркивания; десятичные цифры от 0 до 9; специальные символы, такие как +, -, *, **, ?, ! и т. д.

Предусмотрено выполнение четырех типов операций над выражениями:

- математические операции - сложение (+), вычитание (-), умножение (*), деление (/), возведение в степень (** или ^);
- логические операции - НЕ (NOT), И (AND), ИЛИ (OR);
- символьные операции - сцепление двух и более строк символов (+), сцепление символьных строк с переносом замыкающих пробелов (-), сравнение символьных строк ($),сравнение на полное тождество символьных строк(==);
- операции отношения - меньше (<), больше (>), равно (=), не равно (<> или !=), меньше или равно (<=), больше или равно (>=).

Логические операции и операции отношения дают логический результат .F. или .T. Они используются в командах языка программирования xBase для организации различных циклов, сравнений и переходов по условию. Приоритет выполнения логических операций следующий: NOT, AND, OR.

В выражениях все операции выполняются слева направо с учетом их приоритетов. Для указания порядка выполнения операций могут использоваться круглые скобки. Символьные константы в языке программирования xBase выделяются апострофами, кавычками или квадратными скобками. Константы типа даты заключаются в фигурные скобки, логические константы выделяются точками.

xBase - это язык программирования реляционных систем, в которых данные хранятся в виде записей таблиц. Для перемещения в таблице данных используется так называемый внутренний указатель записи, в дальнейшем - **указатель записи.**

Указатель записи - это некоторое средство, позволяющее отмечать запись в таблице данных, с которой можно работать. Отмеченная запись является текущей записью.

Для **перемещения** указателя записи используются следующие команды:

<n> - установить указатель записи на n-ю запись БД;

GOTO <n> - установить указатель записи на n-ю запись БД;

GO TOP - установить указатель записи на первую запись БД;

GO BOTTOM - установить указатель записи на последнюю запись БД;

SKIP [k]/[-k] - переместить указатель записи на k (или -k) записей

относительно его последней позиции.

Большинство команд языка программирования xBase, обрабатывающих записи таблицы данных, в общем виде имеют следующую структуру:

ГЛАГОЛ
[<диапазон>][<список_выражений>][FOR/WHILE<условие>], где ГЛАГОЛ - команда; <диапазон> - область действия глагола; <список_выражений> - формулы, временные переменные памяти, имена полей таблицы данных, операторы, константы и т.д.; <условие> - условия команд FOR или WHILE. Допустимо сокращение составляющих команд до первых четырех символов. Квадратные скобки отмечают необязательные параметры. Символом / разделены взаимоисключающие параметры (см. FOR/WHILE). FOR <условие> указывает, что действие команды распространяется внутри диапазона только на записи, удовлетворяющие заданному условию. WHILE <условие> означает выполнение команды до тех пор, пока указанное условие истинно.

Для указания **диапазона** используются следующие служебные слова: ALL (область действия - вся БД); RECORD <n> (область действия - n-я запись в файле таблицы данных); NEXT <z> (область действия - следующие z записей, начиная с текущей записи); REST (область действия - все записи, начиная с текущей записи и до конца таблицы данных).

Работать с записями в таблице данных можно, если она открыта для пользователя, т.е. если она является активной. Активизация имеющейся таблицы данных осуществляется использованием команды:

USE <имя_файла_таблицы>

При этом открытая ранее свободная таблица данных закрывается. При открытии таблицы, входящей в состав БД, автоматически открывается и БД, ее содержащая. Закрытие активной таблицы данных можно выполнить командой USE (без параметров).

Завершение работы с СУБД в языке программирования xBase осуществляется командой:

QUIT

В результате выполнения этой команды происходит закрытие всех файлов. Выход из системы без команды QUIT может привести к повреждению открытых файлов и потере информации.

6.2. СОЗДАНИЕ И ИЗМЕНЕНИЕ СТРУКТУРЫ ТАБЛИЦЫ ДАННЫХ

При создании БД для определенной предметной области (ПО) прежде всего необходимо определить интересующие объекты и их свойства. Например, предметная область - **деканат**, объект - **студент**. Интересующие нас свойства студента: фамилия, дата рождения, номер зачетной книжки, семейное положение, стипендия. Каждому студенту в таблице данных будет

соответствовать своя запись (строка), а каждому свойству - поле (столбец). Для создания таблицы данных прежде всего необходимо определить ее структуру, т.е. определить количество полей (столбцов) в таблице и их обязательные параметры: имя, тип, длину, точность. Следует учесть, что имя поля может содержать не более 10 символов и должно начинаться обязательно с буквы. Имя поля может содержать буквы (в ранних версиях dBase-подобных СУБД - только латинские), цифры и знак подчеркивания (_). Если в имени поля используется пробел, то вместо пробела ставится знак подчеркивания, иначе система не воспримет буквы после пробела.

Исходя из вышеперечисленных свойств студентов, необходимо определить параметры пяти полей:

1) фамилия студента - имя поля FIO, тип поля - символьный, ширина поля 15 позиций;

2) дата рождения - имя поля DATA_R, тип поля - дата, для поля этого типа система автоматически сама отводит 8 позиций;

3) номер зачетной книжки - имя поля ZAH_KN, тип поля - числовой, ширина этого поля 7 позиций;

4) семейное положение - имя поля FAM, тип поля - логический, для поля этого типа система автоматически сама отводит 1 позицию;

5) стипендия - имя поля STIP, тип поля - числовой, достаточно 6 позиций под это поле, из них 2 позиции под дробную часть;

6) студенческая группа: имя поля GRUP, тип поля - символьный, достаточно 10 позиций под это поле.

Структура файла таблицы данных создается командой:

CREATE <имя_файла>

Введенное <имя_файла> - это имя таблицы данных (требования к именам файлов таблиц определяются операционной системой). Не следует использовать буквы от A до J для имен файлов таблиц данных, так как они обычно зарезервированы СУБД для псевдонимов имен. В то же время имя AA является корректным именем файла. Если файл таблицы данных создается на диске, отличном от активного, или не в текущем каталоге, то перед именем файла следует указать имя дисковода и имя каталога (например, A:¥FOX) либо предварительно ввести команду:

SET DEFAULT TO <имя_дисковода>/<имя_каталога>

Если имя таблицы данных не задано, то система выведет запрос на ввод имени создаваемой таблицы данных. Затем будет дана возможность задать параметры каждого поля таблицы данных: имя, тип, длина и количество дробных десятичных знаков.

Для создания таблицы с именем STUD необходимо выполнить команду:

CREATE STUD

Если при этом есть открытая БД, то таблица создается в открытой БД, в противном случае создается свободная таблица. На экране монитора

появится окно с обозначением параметров полей таблицы данных: имя (**Name**), тип (**Type**), длина (**Width**) и количество дробных десятичных знаков (**Dec**) (табл.6.1). Описываются параметры первого поля, затем второго и т.д. После ввода имени первого поля курсор переходит к описателю типа поля. Описатель типа поля первоначально установлен Character (символьный). Если необходим другой тип поля, то его выбирают из раскрывающегося списка типов полей. Затем вводится длина поля, а для цифровой информации еще и число позиций в дробной части. Аналогично вводятся описания всех остальных необходимых полей. После ввода параметров полей для прекращения процедуры формирования структуры таблицы необходимо нажать клавишу **OK.**

Таблица 6.1

Name	Type	Width	Dec
FIO	Character	15	0
DATA_R	Date	8	0
ZAH_KN	Numeric	7	0
FAM	Logical	1	0
STIP	Numeric	6	2
GRUP	Character	10	0

Система запрашивает, будет ли производиться ввод записей. Ответ:Y/N, где Y (YES) - да (после него предоставляется возможность ввода записей) или N (NO)-нет (запоминается созданная структура в файле STUD.DBF и происходит возврат в систему). Дальнейший ввод записей в таблице данных возможен путем использования команд APPEND или INSERT.

При наличии в структуре поля типа Memo создается дополнительный файл таблицы данных с тем же именем, но имеющий расширение **.FPT**, предназначенный для хранения примечаний. Этот файл работает только совместно с основным файлом таблицы данных (формата DBF).

При необходимости **изменения** структуры активной таблицы данных используется команда:

MODIFY STRUCTURE

При ее выполнении на экран выводится структура таблицы данных и предоставляется возможность ее изменения. Команда позволяет: добавлять новые поля, удалять поля, изменять имя поля, изменять тип поля, изменять размер поля. Чтобы вставить новое поле, курсор устанавливается на начало поля, перед которым делается вставка. При одновременном нажатии клавиш Ctrl и N (**Ctrl**+**N**) или кнопки **Insert** открывается новое поле, после чего производится определение его параметров. Для удаления поля курсор устанавливается на начало имени этого поля и нажимаются одновременно клавиши: Ctrl и U (**Ctrl**+**U**) или кнопка **Delete**. Не рекомендуется одновременно добавлять и удалять поля, при изменении имени поля не рекомендуется одновременно изменять остальные его параметры. Для их

изменения необходимо повторно использовать команду MODIFY STRUCTURE.

Следует учитывать, что СУБД обеспечивает преобразование числовых значений полей в символьные значения при изменении типа поля, но обратное преобразование возможно не всегда. Цифровые символы преобразовываются в число до тех пор, пока не встретится не цифровой символ. Если первый символ слова не является цифрой, то в результате преобразования будет получен нулевой результат. Возможно преобразование логических данных и дат в символьные данные и обратные преобразования. Изменения в ключевых полях при наличии индексированных файлов вызывают повторное индексирование (см. 6.6).

Запоминание новой структуры таблицы данных осуществляется при одновременном нажатии клавиш Ctrl и W (**Ctrl**+**W**) или нажатием кнопки **OK**. При этом активному файлу таблицы данных присваивается расширение **.BAK** и все записи из него копируются в новую структуру. Создается и становится активным файл таблицы данных с новой структурой с тем же именем, имеющий расширение **.DBF**. При этом дополнительному файлу таблицы данных с расширением **.FPT** присваивается расширение **.TBK** и создается новый файл таблицы данных с тем же именем и тем же содержимым, имеющий расширение **.FPT**.

Получение информации о структуре активной таблицы данных осуществляется командами:

DISPLAY STRUCTURE [TO PRINT]

или

LIST STRUCTURE [TO PRINT]

Этот формат команды DISPLAY (или LIST) выводит структуру таблицы данных, используемой в текущий момент. Этим пользуются для того, чтобы вспомнить имена полей, их типы, размеры, количество дробных десятичных знаков и т.д. при работе с таблицей данных в интерактивном режиме. Кроме того, выводится текущее количество записей в файле и дата его последней корректировки. Необязательный параметр TO PRINT позволяет выдать информацию о структуре активной таблицы данных на печать.

6.3. ВВОД И ПРОСМОТР ЗАПИСЕЙ В ТАБЛИЦЕ ДАННЫХ

Команда **ввода** записей в конец таблицы данных имеет формат:

APPEND [BLANK]

Эта команда позволяет вводить новые записи в конец файла для таблицы данных, находящейся в активном режиме. При использовании необязательного параметра BLANK автоматически добавляется одна пустая запись в конец таблицы данных.

После ввода команды APPEND на экране появятся имена полей. Информация, являющаяся значением того или иного поля, вводится напротив имени этого поля. Даты (по умолчанию) вводятся в американском формате **mm/dd/yy**, где mm-порядковый номер месяца; dd-порядковый номер дня в месяце; yy-две последние цифры года. Для заполнения поля типа Memo необходимо перейти из основного файла (*.DBF) в дополнительный файл (*.FPT). Для этого курсор устанавливается напротив имени поля типа Memo и одновременно нажимаются клавиши **Ctrl** и **Home** или выполняется двойной щелчок мыши. Происходит переход в дополнительный файл (*.FPT) и открывается свободное окно, где и набирается информация. Для возврата в основной файл (*.DBF) используется одновременное нажатие клавиш Ctrl и W (**Ctrl**+**W**) или клавиша **Escape.**

Ввод записей (загрузка таблицы данных) производится поочередно. Выход из режима загрузки БД осуществляется одновременным нажатием клавиш Ctrl и END (**Ctrl**+**End**) или Ctrl и W (**Ctrl**+**W**).

Представление структуры БД на экране при заполнении полей:

FIO _____
DATA_R __/__/__
ZAH_KN _____0
FAM F
STIP 0.00
GRUP _____

Можно вводить новые записи после текущей записи. Для этого используется команда **вставки** записей:

INSERT [BEFORE] [BLANK]

Работа с экраном при этом осуществляется аналогично работе с командой APPEND. Необязательный параметр BEFORE дает возможность вставить новую запись перед текущей записью. При использовании необязательного параметра BLANK в таблице данных автоматически добавляется одна пустая запись (после или перед текущей записью). Если БД проиндексирована и индексный файл подключен, то после завершения добавления (APPEND) или вставки (INSERT) новой записи она будет помещена на место, соответствующее выражению ключа индекса (см. 6.6).

Используя команды DISPLAY или LIST можно найти и **просмотреть** необходимые записи в таблице данных. В этом случае используются следующие форматы команд DISPLAY или LIST:

DISPLAY [<диапазон>] [FIELDS <список_полей>] [FOR/WHILE <условие>] [OFF] [TO PRINT] или

LIST [<диапазон>] [FIELDS <список_полей>] [FOR/WHILE <условие>] [OFF] [TO PRINT]

Этот формат команд используется для вывода значений данных во всех или некоторых полях записей активной в текущий момент таблицы данных. Для записей в пределах диапазона на экран выводятся поля, перечисленные в

списке <список_полей>. Если задан оператор FOR <условие>, выводятся только те записи, для которых выражение <условие> истинно. Если задан оператор WHILE <условие>, вывод записей производится до тех пор, пока выражение < условие > остается истинным. Диапазон по умолчанию соответствует ALL для команды LIST, а для команды DISPLAY диапазоном по умолчанию является текущая запись. Если конструкция FOR (WHILE) отсутствует, команда LIST (или DISPLAY ALL) относится ко всем записям в указанном диапазоне. Но команда DISPLAY ALL выводит записи порциями (поэкранно) с приостановкой, а не непрерывным потоком, как команда LIST. Если не указан параметр OFF, перед каждой выведенной строкой выводится номер записи. Оператор TO PRINT используется для вывода информации на принтер.

Например, возможна следующая последовательность команд xBase:

```
USE STUD
APPEND BLANK
GOTO 5
INSERT BEFORE
DISPLAY ALL TO PRINT
```

В этом случае активизируется таблица данных с именем STUD, при этом указатель записи устанавливается на первую запись. Командой APPEND BLANK добавляется пустая запись в конец таблицы БД. Затем указатель записи устанавливается на 5-ю запись и перед ней вставляется новая запись (командой INSERT). Командой DISPLAY на принтер выводятся записи активной таблицы данных. При этом введенная командой INSERT запись является 5-й записью, а номера следующих записей, соответственно, увеличиваются на единицу.

Следует отметить, что команды DISPLAY и LIST могут использоваться для вывода различной информации. Различные форматы этой команды позволяют выводить данные из таблиц (см. выше), информацию о файлах, содержимое буфера команд, переменные памяти (см. 6.7), статусную строку, структуру данных (6.2).

Найти необходимые записи в активной таблице данных можно также командой LOCATE:

LOCATE [<диапазон>] [FOR/WHILE <условие>]

Эта команда выполняет **поиск** в указанном диапазоне первой записи, удовлетворяющей указанному условию, но найденную запись на экран не выводит. Просмотр найденной записи осуществляется другими командами (например, командой DISPLAY). Диапазоном по умолчанию является ALL (все).

Для продолжения поиска по указанному условию и в указанном диапазоне используется команда без параметров: CONTINUE

Например, возможна следующая последовательность команд xBase:

```
USE STUD
```

```
LOCATE FOR GRUP=’ЭВМ-99-1’
DISPLAY
CONTINUE
```

В этом примере активизируется таблица данных с именем STUD, при этом внутренний указатель записи устанавливается на первую запись. Командой LOCATE ищется первая запись со значением поля GRUP, равным ’ЭВМ-99-1’. Найденная запись выводится командой DISPLAY на экран и поиск (по условию команды LOCATE) продолжается командой CONTINUE.

6.4. РЕДАКТИРОВАНИЕ ТАБЛИЦЫ ДАННЫХ

Для просмотра, редактирования, удаления имеющихся записей и добавления новых записей в активную таблицу данных используется команда BROWSE, имеющая многочисленные опции, определяющие режим работы команды в каждом конкретном случае ее использования. Диапазоном действия команды по умолчанию является ALL (все), если отсутствуют необязательные параметры: <диапазон>, FOR <условие>. В качестве диапазона не используется параметр NEXT <z>.

Сокращенный формат команды представлен ниже:

```
BROWSE <диапазон> [FIELDS <список_полей>]
[FONT <имя_шрифта> [, <размер_шрифта>]] [ FOR <условие>] [REST]
[FREEZE <имя_поля>] [LOCK <число_полей> ] [NOAPPEND]
[NODELETE][NOEDIT | NOMODIFY] [NOLGRID] [NORGRID]
[PARTITION <номер_столбца> [LEDIT] [REDIT]]
[TITLE <текст_заголовка>] [WHEN <условие>]
```

При работе этой команды на экран в специальное окно просмотра выводятся имена полей и записи активной таблицы данных в виде строк. В окне можно перемещаться по вертикали и по горизонтали, просматривая, редактируя, добавляя или удаляя записи. Можно пользоваться специальным меню команды BROWSE (в ранних версиях FoxPro).

Параметр FIELDS<список_полей> позволяет выводить только те поля таблицы данных, которые указаны в списке полей, и в том порядке, в котором они приведены в списке. В список можно включать поля из других связанных таблиц. Перед именем такого поля нужно ставить псевдоним соответствующей таблицы с точкой. Если предложение FIELDS опущено, будут отображены все поля таблицы в том порядке, в каком они расположены в рамках табличной структуры. В списке полей можно указать любую комбинацию полей или вычисляемых полей.

Вычисляемое поле содержит данные в режиме "только для чтения", созданные с помощью выражения. Это выражение может носить произвольную форму, но непременно должно быть допустимым выражением используемой СУБД. В этом случае в список полей входят операторы

создания вычисляемых полей. Формат оператора, используемого для создания вычисляемого поля: <вычисляемое поле> = <выражение>

В следующем примере при выводе в режиме BROWSE создается вычисляемое поле LOCATION из содержимого полей FIO и GROUP, выбранной в данное время таблицы данных STUD:

```
USE STUD
BROWSE FIELD LOCATION = FIO + ', ' + GROUP
```

Список полей предложения FIELDS включает опции, которые обеспечивают специальные виды обработки полей, отображаемых в окне просмотра. Сокращенный синтаксис указания опций поля следующий:

<имя_поля> [:R] [:<ширина_столбца >]
[:V = <выражение>]
[:B = <нижнее_значение>, <верхнее_значение>]
[:H = <заголовок_столбца>]
[:W = <выражение>]

Опция **:R** устанавливает, что поле доступно только для чтения. Содержащиеся в нем данные можно будет просматривать, но не редактировать.

Опция **:<ширина_столбца>** задает размер отображения поля в столбцах. Значение параметра не влияет на размер поля в таблице, он только определяет внешний вид поля в окне просмотра.

Опция проверки **:V** позволяет выполнять в окне просмотра проверку достоверности данных на уровне поля с использованием указанного выражения. Если при выведении курсора из поля значение выражения есть "истина" (.T.), то помещенные в него данные считаются корректными и курсор переходит в следующее поле.

Опция **:B** задает диапазон, в который должны попадать значения данных в поле. Граничные выражения **нижнее_значение** и **верхнее_значение** должны соответствовать типу данных поля и не могут быть функциями, определенными пользователем. Если введенные значения данных не попадают в интервал между указанными значениями, то выдается системное сообщение об ошибке, в котором приводится допустимый диапазон для данных.

Опция **:H** замещает принимаемые по умолчанию имена полей названиями, которые задаются с помощью аргумента **заголовок_столбца.** По умолчанию в качестве заголовков столбцов в окне просмотра используются имена полей.

Опция **:W** позволяет запретить перемещение курсора в поле в зависимости от значения указанного выражения. Если значение выражения есть "ложь" (.F.), перемещать курсор в поле запрещается. Если значение выражения - "истина" (.T.), курсор можно ввести в поле. В выражении допускаются функции, определенные пользователем.

Следующая команда BROWSE, используя рассмотренные выше опции задания полей, обеспечивает редактирование только поля STIP в диапазоне от 1500 до 1800 при выводе поля FIO (только для чтения и с заголовком Фамилии) и поля STIP (с заголовком Стипендия).

BROWSE FIELDS FIO :R :H = 'Фамилии' , STIP :H = ’Стипендия‘ :B = 1200, 1800

В команде BROWSE можно задать шрифт окна просмотра и размер шрифта. Имя шрифта определяется символьным выражением, а его размер - числовым выражением. Например, приведенная ниже часть команды BROWSE задает 16-пунктовый шрифт Courier для полей в окне просмотра: FONT 'Courier', 16 .

FOR <условие> задает условие отображения в окне просмотра: отображаются только те записи, для которых значение указанного условия - "истина".

Параметр REST предотвращает перемещение указателя записи из текущей позиции к началу таблицы, когда окно просмотра открывается с предложением FOR. В противном случае команда BROWSE по умолчанию устанавливает указатель записи в начало таблицы.

Параметр FREEZE <имя_поля> разрешает вносить изменения только в одно поле в окне просмотра. Это поле задается своим именем. Остальные поля только отображаются, но не могут редактироваться.

Параметр LOCK <число_полей> используется для указания количества полей в левой части экрана, которые не будут перемещаться при смещении экрана по полям таблицы. Эти поля останутся на экране для просмотра или редактирования.

Параметр NOAPPEND не позволяет вводить в таблицу новые записи. Параметр NOEDIT или NOMODIFY запрещает внесение изменений в записи таблицы, используется для предоставления пользователю возможности просмотра записей, предотвращая возможные повреждения БД.

Параметр NODELETE запрещает помечать записи на удаление в окне просмотра. По умолчанию, чтобы пометить запись на удаление, нужно нажать клавиши **Ctrl**+**T**, или выбрать команду Toggle Delete в меню Table в Visual FoxPro или в меню Browse в ранних версиях FoxPro или щелкнуть самый левый столбец удаляемой записи.

Параметр PARTITION разбивает окно просмотра на левый и правый разделы, линия разбиения проходит по указанному номеру столбца. Параметры NOLGRID и NORGRID, соответственно, удаляют линии сетки полей в левом разделе окна просмотра и в правом разделе окна просмотра. Параметры LEDIT и REDIT , соответственно, задают режим редактирования для левого и правого разделов окна просмотра. Чтобы открыть оба раздела в режиме редактирования, нужно указать оба ключевых слова.

В следующем примере окно просмотра открывается с линией разбиения в 20-м столбце и с правым разделом в режиме редактирования (для представления данных используется 12-пунктовый шрифт Arial):

BROWSE FONT ‘Arial’, 12 PARTITION 20 REDIT

Параметр TITLE заменяет принимаемые по умолчанию имя или псевдоним таблицы, появляющиеся в строке заголовка окна просмотра, на **текст_заголовка**. Иначе в строке заголовка появится имя или псевдоним просматриваемой таблицы. Если выдать команду BROWSE WINDOW, чтобы поместить окно просмотра в окно, определенное пользователем, заголовок пользовательского окна будет заменен заголовком окна просмотра.

Параметр WHEN <условие> проверяет указанное условие при переводе курсора в другую запись. Если условие истинно (.T.), то пользователь может модифицировать запись, в которую он перешел. Если значение условия ложно (.F.) или 0, запись, в которую произошел переход, становится доступной только для чтения и не может быть модифицирована.

Предложение WHEN не выполняется, когда активизируется еще одно окно.

Например, возможна следующая последовательность команд xBase:

```
USE STUD
BROWSE NOAPPEND NOMODIFY NODELETE
BROWSE FIELDS FIO :H='Фамилии', GRUP :H=’Группа‘ TITLE 'Просмотр'
```

В этом случае активизируется таблица с именем STUD. Затем командой BROWSE обеспечивается просмотр записей без дополнения, изменения и удаления (за счет использования параметров NOAPPEND, NOMODIFY, NODELETE). Следующей командой BROWSE обеспечивается редактирование поля FIO (с заголовком **Фамилии**) и поля GRUP (с заголовком **Группа**) в окне с заголовком **Просмотр.**

Команды редактирования CHANGE и EDIT имеют аналогичный формат. Отличие состоит в том, что записи представляются не в виде таблицы, а построчно - на экран поочередно выводится по одной записи. Ее представление на экране аналогично представлению записи при работе с командами APPEND и INSERT.

Например, возможна следующая последовательность команд xBase:

```
USE STUD
EDIT NEXT 3 FIELDS FIO, STIP
EDIT FOR  STIP<>0
```

В этом случае активизируется таблица данных с именем STUD. Диапазон NEXT 3 позволяет выполнить редактирование только 3-х первых записей, а список полей FIELDS позволяет выводить на экран и выполнять редактирование только двух перечисленных полей (FIO и STIP). Последней командой обеспечивается возможность редактирования всех полей для записей, имеющих ненулевое значение поля STIP.

6.5. УДАЛЕНИЕ ЗАПИСЕЙ В ТАБЛИЦЕ ДАННЫХ

Удаление записей из активной таблицы данных обычно осуществляется в два этапа. Сначала удаляемые записи должны быть **помечены** с помощью команды DELETE. Формат команды:

DELETE [<диапазон>] [FOR/WHILE <условие>]

Если параметр <диапазон> отсутствует, диапазоном по умолчанию является только **текущая** запись. Если отсутствует параметр FOR <условие>, то все записи в пределах указанного диапазона помечаются для удаления. В противном случае помечаются для удаления записи, для которых выполняется заданное условие. Если в составе команды имеется параметр WHILE <условие>, пометка записей на удаление производится до тех пор, пока выражение <условие> истинно.

Помеченные записи физически удаляются командой PACK. Формат команды:

PACK

Команда PACK окончательно **удаляет** все записи в активном файле таблицы данных, помеченные для удаления командой DELETE. Если активная таблица данных используется совместно с одним или несколькими индексными файлами, команда PACK переформатирует индексные файлы (см. 6.6).

Например, возможна следующая последовательность команд xBase:

```
USE STUD
DELETE NEXT 3
PACK
```

В этом примере активизируется таблица данных с именем STUD. Диапазон NEXT 3 позволяет выполнить пометку 3-х первых записей на удаление. Команда PACK физически удаляет помеченные записи.

Пометка записей для удаления может быть **отменена** при помощи команды RECALL. Формат команды:

RECALL [<диапазон>] [FOR/WHILE <условие>]

Параметры команды RECALL имеют аналогичное значение, что и для команды DELETE. Любая команда DELETE может быть отменена последующей командой RECALL. Команда RECALL может быть использована для отмены удаления, если по отношению к записям файла таблицы не были использованы команды PACK или ZAP.

Команда ZAP удаляет все записи из активной таблицы данных без предварительной пометки на удаление, оставляя структуру таблицы.

Формат команды:

ZAP

Команда ZAP аналогична использованию последовательно команд DELETE ALL и PACK. Записи, удаленные из активной таблицы данных командой ZAP, не могут быть восстановлены командой RECALL.

Например, возможна следующая последовательность команд xBase:

```
USE STUD
DELETE RECORD 4
RECALL
PACK
```

В этом примере активизируется таблица данных с именем STUD. Для пометки 4-й записи на удаление в команде DELETE используется диапазон - RECORD 4. Команда RECALL отменяет пометку текущей записи (4-й) для удаления. Выполнение команды PACK не приводит к удалению записей в активной таблице данных, так как ни одна запись не помечена для удаления.

6.6. СОРТИРОВКА И ИНДЕКСИРОВАНИЕ БАЗ ДАННЫХ

Сортировка и индексирование проводятся с целью упорядочения записей в таблицах данных в алфавитном, хронологическом или числовом порядке (в зависимости от типа поля, по которому выполняется сортировка или индексирование). При **сортировке** создается новый отсортированный файл, куда заносятся записи (или часть записей) из сортируемого файла в определенном порядке. Команда сортировки активной таблицы данных имеет формат:

SORT <диапазон> TO <файл> ON <поле1> [/A][/C][/D] [,<поле2> [/A][/C][/D]...] [FOR/WHILE <условие>] [FIELDS <список_полей>]

Сортировка идет по полю - ключу, указанному в параметре ON <поле1>. Отсортированная таблица записывается в файл с расширением .DBF. При наличии поля типа Memo при сортировке создается также файл с расширением .FPT. Сортировка может выполняться по возрастанию (/A), по убыванию (/D), с игнорированием верхнего и нижнего регистра (/C). При этом рассматриваются ASCII - коды символов поля <поле1>. Параметр /C может быть использован совместно с параметрами /A или /D. Если вводятся два параметра, используется только одна косая черта (например, /AC). При отсутствии параметров /A, /C, /D сортировка идет по возрастанию. Диапазон по умолчанию имеет значение ALL.

Параметры FOR/WHILE <условие> задают условия выбора записей из таблицы. Параметр FIELDS указывает поля, которые должны войти в создаваемую отсортированную таблицу данных. Имя отсортированного файла имеет расширение **.DBF**. Возможна сортировка по нескольким полям. В этом случае они указываются в команде через запятую: <поле1>, <поле2> и т.д.

Например, можно выполнить следующую последовательность команд:

```
USE STUD
SORT TO STUDS ON STIP/D
USE STUDS
BROWSE
```

В этом случае активизируется таблица данных STUD. Создается новый отсортированный файл STUDS.DBF, куда записываются записи по убыванию поля STIP. Затем активизируется файл STUDS (при этом файл STUD автоматически закрывается) и просматривается командой BROWSE.

В случае необходимости экономии памяти вместо команды SORT используется команда INDEX, создающая **индексный файл**. В индексном файле хранятся только указатели номеров записей (индексы) в упорядоченном виде, но записи там отсутствуют. Формат команды:

INDEX ON <поле1> [+<поле2>[+...]] TO <индексный_файл>

[FOR/WHILE <условие>]

Команда INDEX используется для формирования индексного файла для активного файла таблицы данных. Индексирование (упорядочивание) записей идет по полю - ключу, указанному в параметре ON <поле1>. Индексирование проводится по возрастанию ASCII-кодов символов поля <поле1>. Создаваемый индексный файл будет иметь расширение имени **.IDX**. Индексирование можно проводить по нескольким полям. В этом случае они должны быть однотипны (или приведены к одному типу) и в команде их имена надо объединить знаком «+». Упорядочение будет проводиться сначала по полю <поле1>, внутри каждой группы <поля1> - по полю <поле2> и т.д.

При работе с индексным файлом он должен быть подключен к активной таблице данных, для которой создавался. При индексации создаваемый индексный файл автоматически подключается к файлу таблицы данных. Отключение индексного файла осуществляется автоматически при закрытии активного файла таблицы данных или командой SET INDEX TO (без параметров). Подключение индексного файла к активной таблице данных осуществляется командой:

SET INDEX TO <имя_индексного_файла>

Подключение индексного файла можно также осуществить при активизации таблицы данных командой:

USE <файл_таблицы_БД> INDEX <имя_индексного_файла>

Например, можно выполнить следующую последовательность команд:

```
USE STUD
INDEX ON FIO TO STUDI FOR STIP <1200
LIST
SET INDEX TO
```

В этом случае активизируется таблица STUD, которая индексируется командой INDEX по содержимому поля FIO (по возрастанию). При этом создается файл STUDI.idx, где содержится информация о записях со значением поля STIP < 1200, упорядоченных по возрастанию данных поля FIO (в алфавитном порядке). Проиндексированный файл просматривается командой LIST и индексный файл отключается.

Главное назначение индексирования - это ускорение доступа к данным. Поэтому аппарат индексирования широко применяется в реляционных системах, одним из недостатков которых является медленный поиск данных. В проиндексированном файле можно осуществить быстрый поиск необходимых данных по полю, по которому проводилось индексирование (по ключу).

6.7. ВРЕМЕННЫЕ ПЕРЕМЕННЫЕ ПАМЯТИ

Данные могут храниться не только в таблице данных, но и в отдельных временных переменных памяти (ВПП). Максимально допустимое число ВПП в различных СУБД различно. Имя переменной может содержать любые цифры, знак подчеркивания, буквы (в ранних версиях - только латинские), но не может начинаться с цифры. Длина имени определяется версией конкретной СУБД. Тип ВПП явно не задается, а определяется ее значением. Возможные типы ВПП в xBase соотвествуют возможным типам полей в данной СУБД: символьные (C), числовые (N), типа даты (D), логические (L) и т.д. Присвоение переменной некоторого значения осуществляется двумя способами:

STORE < выражение> TO < список_ВПП >

или

<ВПП> = <выражение>

Например, для присвоения переменным некоторых значений можно использовать следующие команды:

STORE 5 TO A1, B1

STORE “ПЕТРОВ” TO A2

A3 = 5+2

B2 = “Петров”

B3 = A3+A1

В этом примере с использованием команд STORE создаются две переменные A1 и B1 с численным значением 5 и переменная A2 с символьным значением “ПЕТРОВ”. А с использованием знака равенства (=) создаются: переменная A3 с численным значением 7, переменная B2 с символьным значением “Петров”, переменная B3 с численным значением 12.

Из ВПП можно организовать массив. Но массив предварительно должен быть описан. Массивы могут быть одномерными и двумерными, для их описания используется служебное слово DIMENSION или DECLARE. Имена массивов и имена ВПП не должны совпадать. К элементу двумерного массива возможно обращение как к элементу одномерного массива. При этом индекс такого элемента определяется порядком следования этого элемента в массиве. Элементы одного и того же массива могут быть разных типов, индексация элементов массива начинается с 1. При описании массива всем его элементам автоматически присваивается логическое значение (.F.).

Например, для описания массивов и присвоения значений элементам массивов можно использовать следующую последовательность команд:

```
DIMENSION   C1 (3), C2 (2*3), C3 (2,3)
STORE .T. TO C1 (1)
STORE {^1995/11/23}   TO   C1 (2)
STORE O TO   C2
STORE 7   TO   C3 (2,2)
C3 (2,3) = C3(2,2)+3
```

В этом примере описываются два одномерных C1, C2 и один двумерный C3 массивы. Элементу массива C1(1) присваивается логическое значение .T., элементу массива C1(2) присваивается значение даты 1995/11/23. Всем элементам массива C2 присваивается значение нуля. Элементу двумерного массива C3(2,2) присваивается числовое значение 7, следующему элементу массива C3(2,3) присваивается числовое значение 10 (7+3=10). Элементы массивов C1(3), C3(1,1), C3(1,2), C3(1,3), C3(2,1) имеют значение .F., которое им было присвоено при описании массивов.

ВПП могут быть PUBLIC (глобальные), PRIVATE (частные), LOCAL (локальные). Все переменные, которые создаются в интерактивном режиме, являются глобальными. Если ВПП создаются в программе, то они являются частными или локальными. Эти переменные после выполнения программы уничтожаются, поэтому они допустимы для работы только в той программе, в которой созданы. Можно явно объявить характер ВПП, используя следующие команды:

PUBLIC <список_ВПП> (для глобальных ВПП)

или

PRIVATE <список_ВПП> (для частных ВПП)

Созданные ВПП можно просмотреть, используя команды:

LIST MEMORY [TO PRINT]

или

DISPLAY MEMORY [TO PRINT]

Необязательный параметр TO PRINT позволяет выводить информацию о ВПП на печать. Вид информации о ВПП на экране (или принтере) представлен в табл.6.2.

ВПП или элементу массива можно присвоить значение, взятое из указанного поля текущей записи таблицы данных, используя команду:

STORE <имя_поля> TO <имя_ВПП>

Можно выполнить обратное действие, используя команду замены REPLACE. Формат команды:

REPLACE [<диапазон>] <поле1> WITH <выражение1>

[,<поле2> WITH <выражение2>....] [FOR / WHILE <условие>]

Команда REPLACE для активной таблицы данных выполняет замену данных, находившихся в поле <поле1> на данные, приведенные в выражении <выражение1>, в поле <поле2> - на данные, приведенные в выражении

<выражение2> и т.д. Диапазоном по умолчанию является одна **текущая** запись. В этом случае указатель записи должен быть предварительно установлен на соответствующую запись, так как команда REPLACE не перемещает указатель записи. Использование операторов FOR (WHILE) позволяет выполнять замену данных только в тех записях, для которых выражение FOR <условие> истинно или до тех пор, пока выражение WHILE<условие> истинно.

Таблица 6.2

Обозначение ВПП	Тип ВПП	Значение ВПП
A1	N	5
A2	N	“ПЕТРОВ”
A3	C	7
B1	N	5
B2	C	“Петров”
B3	N	12
C1(1)	L	.T.
C1 (2)	D	11/23/95

Например, для занесения в поле FIO 5-й записи активной таблицы данных значения поля FIO 10-й записи этой же таблицы, необходимо выполнить следующую последовательность команд:

```
GOTO 10
STORE   FIO TO D1
REPLACE RECORD 5   FIO   WITH   D1
```

В этом примере указатель записи устанавливается на 10-ю запись. Содержимое поля FIO 10-й записи присваивается переменной D1. Командой REPLACE полю FIO 5-й записи присваивается значение переменной D1.

Созданные в оперативной памяти ВПП можно уничтожить, используя команду:

RELEASE <список_ВПП> / ALL [LIKE/EXCEPT <шаблон>]

Эта команда позволяет уничтожить ВПП, перечисленные в списке <список_ВПП> или все ВПП.

RELEASE ALL - эта команда уничтожает все ВПП, аналогична команде CLEAR MEMORY.

А для уничтожения переменных A1 и B1 необходимо использовать команду:

```
RELEASE A1, B1
```

Необязательные параметры позволяют уничтожить все ВПП, подобные (ALL LIKE) <шаблону>, или все ВПП, за исключением подобных (ALL EXCEPT) <шаблону>.

Например, для уничтожения всех ВПП, имена которых начинаются на букву С, можно использовать команду:

```
RELEASE   ALL LIKE C*
```

В ВПП можно сохранить всю информацию, находящуюся на экране, используя команду:

SAVE SCREEN TO <имя_ВПП>

При этом создается переменная типа S (Screen) с указанным именем, которая хранит образ экрана.

Восстановление образа экрана из ВПП осуществляется командой:

RESTORE SCREEN FROM <имя_ВПП>

Для сообщения СУБД о необходимости замены имени ВПП на ее значение, перед именем ВПП указывается функция макроподстановки **&.**

Например:

E1 = “STUD”

USE &E1

В этом примере ВПП с именем E1 первой командой присваивается значение “STUD”. Второй командой открывается таблица данных с именем STUD.

Массивы ВПП можно использовать для перемещения информации из таблицы данных в массивы и наоборот. Команда заполнения массива данными из текущей записи:

SCATTER [FIELDS <список_полей>] TO <имя_массива>

Тип каждого элемента массива определяется типом заносимого данного. Если размерность массива меньше количества заносимых полей, то автоматически создается массив необходимой размерности с тем же именем.

Занесение значений элементов массива в текущую запись таблицы данных осуществляется командой:

GATHER FROM <имя_массива> [FIELDS <список_полей>]

При этом типы элементов массива должны совпадать с типами соответствующих полей. Необязательный параметр FIELDS позволяет работать только с теми полями, имена которых указаны через запятую в перечне <список_полей>.

Например, чтобы заменить значения полей FIO и STIP 5-й записи активной таблицы на значения полей FIO и STIP 10-й записи той же таблицы, необходимо выполнить следующую последовательность команд:

```
DIMENSION F1(4)
GOTO 10
SCATTER FIELDS FIO, STIP TO F1
GOTO 5
GATHER FROM F1 FIELDS FIO, STIP
```

В этом примере описывается одномерный массив F1(4) и указатель записи устанавливается на 10-ю запись активной таблицы базы данных. Содержимое поля FIO и STIP 10-й записи присваивается элементам массива F1(1) и F1(2). Затем указатель записи устанавливается на 5-ю запись и значения элементов массива F1(1) и F1(2) заносятся в поля FIO и STIP 5-й записи.

6.8. СОЗДАНИЕ И ВЫПОЛНЕНИЕ ПРОГРАММ

Для организации различных видов представления и обработки данных, хранящихся в базах данных, разрабатываются **приложения** - прикладные программы. В программы можно включать все команды и операторы языка программирования xBase.

Имя файла программы имеет расширение **.PRG.**

Это текстовый файл, который можно создать и корректировать в любом текстовом редакторе или в среде СУБД, в которых имеется встроенный редактор. Его создание или вызов осуществляется командой:

MODIFY COMMAND <имя_файла>

Если имя файла не указано, то система выведет подсказку о вводе имени файла. Если указанный файл не существует, система создаст его. Для изменения программного файла необходимо использовать эту же команду.

При наборе программы на одной строке записывается только одна команда. Если команда не поместилась на одной строке, она переносится на следующую строку, а в конце предыдущей строки ставится символ **;** . Комментарии в конце строки указываются после двух символов **&&**, в начале строки - после символа ***** .

Программа не имеет признаков начала и конца, строки не нумеруются.

Чтобы набранная программа записалась в указанный файл, нужно одновременно нажать клавиши **Ctrl**+**End** или **Ctrl**+**W.**

Команда запуска программы имеет формат: DO <имя файла>.

Когда вызывается программный файл, команды в его составе выполняются в порядке их следования с учетом управляющих операторов xBase. Выполнение командного файла будет продолжаться до того момента, пока не встретится одна из команд RETURN, CANCEL, QUIT или не будет достигнут конец программы.

Команда CANCEL прерывает выполнение программы, при этом очищаются все собственные переменные памяти. RETURN является командой, завершающей программу, но ее использование необязательно, так как выполнение последней команды программы автоматически приведет к выполнению оператора RETURN. Команда QUIT осуществляет выход из СУБД.

При обнаружении ошибки выполнение программы автоматически прерывается и исходный текст программы представляется в окне текстового редактора, при этом строка с предполагаемой ошибкой выделяется. В диалоговом окне выдается сообщение об ошибке и предлагаются возможные действия пользователя.

В программе можно использовать **подпрограммы** и **процедуры.** Каждую подпрограмму отдельно создают командой MODIFY COMMAND <имя_подпрограммы>. Каждой подпрограмме соответствует свой файл на

диске с уникальным именем и расширением .PRG. Вызывается подпрограмма командой:

DO <имя_подпрограммы> [WITH<список_параметров>]

Параметр WITH позволяет передавать программе требуемые параметры. В этом случае в начале подпрограммы должны указываться параметры. Команда указания параметров подпрограммы имеет формат:

PARAMETERS <список_параметров>

Команда PARAMETERS позволяет назначить данным, передаваемым вызывающей программой, имена локальных переменных. Эта команда должна быть первой в вызванной подпрограмме. Параметром в списке параметров может быть любая переменная памяти с доступным именем. Параметры в списке параметров разделяются запятыми и должны соответствовать количеству параметров, приведенных в списке вызывающей программы.

Команда возврата управления из подпрограммы вызвавшей программе имеет формат:

RETURN [TO MASTER / <выражение>]

Параметр TO MASTER указывает, что управление передается в вызвавшую программу самого верхнего уровня, так как возможен вызов подпрограммы из подпрограмм. Если программа является процедурой - функцией, то возвращаемое значение указывается в выражении.

При использовании в программе большого количества подпрограмм, на обращение к ним требуется значительное время - время считывания их с диска. Для увеличения быстродействия вместо подпрограмм можно использовать процедуры. Они могут располагаться после основной программы или должны быть объединены в отдельный файл - процедурный. При использовании процедурного файла в основной программе должна быть выполнена команда установки:

SET PROCEDURE TO <имя_процедурного_файла>

Процедурный файл должен быть закрыт после его использования. Команда закрытия процедурного файла:

CLOSE PROCEDURE

Вызывается процедура, как и подпрограмма, командой:

DO <имя_процедуры> [WITH<список_параметров>]

Команда начала каждой процедуры в программе имеет формат:

PROCEDURE <имя_процедуры>

[PARAMETERS <список_параметров>]

Необязательная вторая команда (PARAMETERS) используется, если вызов прцедуры осуществляется с указанием передаваемых параметров. Команда окончания процедуры: RETURN.

При выполнении программ возникает необходимость отслеживания некоторых условий: появления ошибки, нажатия клавиши ESC, нажатия любой клавиши, нажатия определенной клавиши и т.д.

Команда отслеживания ошибки имеет формат:

ON ERROR [<команда>]

При появлении ошибки будет выполняться указанная команда.

Команда отслеживания нажатия клавиши **Esc** имеет формат:

ON ESCAPE [<команда>]

При нажатии на клавишу Esc будет выполняться указанная команда.

Команда отслеживания нажатия любой клавиши имеет формат:

ON KEY [<команда>]

При нажатии на любую клавишу выполняется указанная команда.

Команда отслеживания нажатия определенной клавиши имеет формат:

ON KEY=<числовое_выражение> <команда>

При нажатии клавиши, код которой соответствует числовому выражению, будет выполняться указанная команда.

Если используется несколько команд ON, их приоритет устанавливается в следующем порядке: ERROR, ESCAPE, KEY. Все команды остаются активными до тех пор, пока не будет обнаружена следующая команда ON, использующая то же условие. Отслеживание условий может быть отключено при использовании этих же команд без параметра <команда>:

ON ERROR / ON ESCAPE / ON KEY

Программа, составленная на языке программирования xBase, может быть выполнена различными способами. Исходные программы (командные файлы с расширением .PRG) выполняются в СУБД, использующей язык программирования xBase, при выполнении команды: DO <имя_программы>

Исходные программы (командные файлы с расширением .PRG) можно откомпилировать. При этом создаются новые файлы с теми же именами, но с другим расширением (например, в FoxPro с расширением **.FXP**). Откомпилированные программы загружаются с диска со скоростью их считывания, в то время как исходные программы должны компилироваться по мере их считывания, что приводит к дополнительным временным задержкам. Поэтому рекомендуется выполнять предварительную компиляцию программ.

Современные версии dBase-подобных СУБД позволяют создавать исполняемые файлы программ (с расширением **.EXE**). Эти файлы можно выполнять автономно без СУБД, но при наличии специальных библиотек.

6.9. УПРАВЛЯЮЩИЕ ОПЕРАТОРЫ

В языке программирования xBase имеется несколько управляющих операторов, широко используемых при написании программ: оператор условия, оператор выбора и операторы цикла. В интерактивном режиме использование операторов невозможно.

Оператор условия имеет формат:

IF <условие>
<группа_команд_1>
[ELSE <группа_команд_2>]
ENDIF

В этом операторе выражение <условие> анализируется и если оно истинно - выполняется <группа_команд_1>. При наличии параметра ELSE и ложности выражения <условие> выполняется <группа_команд_2>.

Если выражение <условие> ложно и параметр ELSE отсутствует, осуществляется выход из оператора IF...ENDIF.

Выражения IF <условие> могут быть вложены одно в другое, при этом для каждого IF требуется соответствующее ENDIF.

Например, возможно следующее применение оператора условия:

```
IF A1<0
A1=A1+1
ELSE   A1=A1-1
ENDIF
```

В этом примере оператором условия анализируется значение переменной A1 и если A1<0, значение A1 увеличивается на 1, в противном случае значение A1 уменьшается на 1.

При необходимости проверки нескольких условий вместо вложенных операторов условий удобнее использовать **оператор выбора**, в котором анализируются <условие_1>, <условие_2>, <условие_N> по одному в поисках первого логически истинного условия. При обнаружении истинного условия выполняется соответствующая <группа_команд_i>, после чего осуществляется выход из оператора. Оператор выбора имеет следующий формат:

DO CASE
CASE < условие_1>
<группа_команд_1>
CASE < условие_2>
<группа_команд_2>
.....
CASE < условие_N >
<группа_команд_N>
[OTHERWISE
<группа_команд>]
ENDCASE

В операторе DO CASE всегда выполняется только одна <группа_команд_i>, следующая за первым из обнаруженных истинных условий. Если ни одно из истинных условий не обнаружено, то выполняется <группа_команд>, следующая за параметром OTHERWISE. Если этот параметр отсутствует, то управление передается первой команде, следующей

за оператором ENDCASE. В этом случае ни одна <группа_команд_i>, входящая в структуру оператора DO CASE...ENDCASE, выполнена не будет.

Например, возможно использование следующего оператора выбора:

```
DO CASE
CASE C1=1
USE STUD
BROWSE
CASE C1=2
STORE 15 TO A1
OTHERWISE
WAIT   'Необходимо C1 присвоить значение 1 или 2'
ENDCASE
```

В данном примере анализируется переменная C1. Если значение C1 равняется 1, выполняется открытие и просмотр таблицы данных STUD. Если значение C1 равняется 2, выполняется присвоение переменной A1 значения 15. Если ни одно из выражений CASE не истинно, на экран выводится сообщение о том, что C1 необходимо присвоить значение 1 или 2.

Для организации циклов может использоваться один из трех операторов цикла: с условием, по параметру, по записям таблицы данных.

Оператор цикла с условием имеет формат:

```
DO WHILE <условие>
<группа_команд_1>
[ LOOP]
<группа_команд_2>
[ EXIT]
ENDDO
```

В этом операторе анализируется выражение <условие> и в случае его истинности (.T.) выполняются команды, указанные в теле цикла. В случае ложности выражения <условие> (.F.), осуществляется выход из оператора цикла.

LOOP является необязательным параметром, при обнаружении которого происходит переход к началу цикла, при этом <группа_команд_2> игнорируется. EXIT является необязательным параметром, который передает управление из цикла первой команде, следующей за ENDDO. Команды EXIT и LOOP могут быть размещены в любом месте внутри цикла и могут использоваться во всех операторах цикла.

Например:

```
USE STUD
C1=2
DO WHILE C1<=20
DISPLAY RECORD C1
C1=C1+2
ENDDO
```

В этом примере активизируется таблица данных STUD и переменной C1 присваивается числовое значение 2. В цикле до тех пор, пока значение C1 будет меньше или равно 20, командой DISPLAY выводятся все записи с номерами, равными C1. Значение C1 меняется с шагом, равным 2, поэтому будут выведены все четные записи.

Оператор цикла с параметром имеет формат:

FOR<имя_ВПП>=<выражение1>TO<выражение2>[STEP<выражение3>]
<группа_команд_1>
[LOOP]
<группа_команд_2>
[EXIT]
ENDFOR

В этом операторе параметром цикла является ВПП и команды, указанные в теле цикла, выполняются каждый раз, начиная с начального значения параметра (<выражение1>) и до конечного значения параметра (<выражение2>). Значение параметра изменяется автоматически после каждого цикла с шагом, равным значению < выражение3>. В случае отсутствия необязательного указания шага параметра шаг берется равным 1.

Например, для пометки на удаление нечетных записей активной таблицы данных, начиная с 11-й записи и кончая 1-й записью, можно использовать следующий оператор цикла с параметром:

```
FOR S1=11 TO 1   STEP -2
DELETE
ENDFOR
```

Оператор цикла по таблице данных осуществляет сканирование записей таблицы данных в указанном диапазоне с учетом заданного условия. Формат оператора цикла сканирования таблицы данных:

SCAN [<диапазон>] [FOR/WHILE<условие>]
<группа_команд_1>
[LOOP]
<группа_команд_2>
[EXIT]
ENDSCAN

Например, для просмотра всех записей в активной таблице со значением поля STIP больше 1500 можно использовать следующий оператор цикла:

```
SCAN FOR STIP > 1500
DISPLAY
WAIT
ENDSCAN
```

Команда WAIT используется в этом примере для приостановки выполнения команд в программе до нажатия любой клавиши.

6.10. КОМАНДЫ-ФУНКЦИИ И СТАНДАРТНЫЕ ФУНКЦИИ

В xBase существуют специальные **команды - функции**, к которым относятся команды подсчета записей в активной таблице данных, вычисления среднего значения и суммы по числовым полям активной таблицы данных.

Команда **подсчета записей** в активной таблице данных имеет формат:

COUNT [<диапазон>] [FOR/WHILE <условие>] [TO < ВПП>]

Записи подсчитываются в пределах диапазона (по умолчанию ALL). При использовании необязательных параметров FOR/WHILE в подсчете используются только те записи, которые удовлетворяют условию <условие>. Результат может быть занесен во временную переменную памяти <ВПП>.

Команда вычисления **среднего значения** для полей числового типа активной таблицы данных имеет формат:

AVERAGE [<диапазон>] [<список_выражений>]

[FOR/WHILE < условие >] [TO <список_ВПП>]

Обрабатываются значения всех числовых полей, если списком выражений не указано иначе. Получаемые значения могут быть занесены в переменные памяти (см.<список_ВПП>), число которых должно равняться числу обрабатываемых полей. Параметры FOR/WHILE указывают, какие записи следует обрабатывать. Диапазон по умолчанию имеет значение ALL.

Команда вычисления **сумм** для числовых полей имеет формат:

SUM [<диапазон>] [<список_выражений>]

[FOR/WHILE <условие>] [TO <список_ВПП>]

Диапазон по умолчанию имеет значение ALL. Операция может выполняться не над всеми числовыми полями, а лишь над полями, указанными в списке выражений. Параметры FOR/WHILE указывают, что в суммировании участвуют значения полей тех записей, которые удовлетворяют указанному условию. Полученные суммы могут быть занесены в переменные памяти (см. список_ВПП), число которых должно равняться числу обрабатываемых полей. Ниже приведены примеры описанных выше команд-функций:

```
COUNT TO A1
AVERAGE ZAH_KN FOR STIP > 1300   TO B1
SUM STIP TO C1
```

Первой командой в переменной A1 подсчитывается число записей (COUNT) таблицы. Вычисляется и записывается в переменную B1 среднее значение (AVERAGE) поля числового типа ZAH_KN для записей, имеющих значение поля STIP>1300. Вычисляется сумма (SUM) всех значений поля числового типа STIP, которая заносится в переменную C1.

Система программирования xBase содержит около 100 стандартных встроенных функций. Ниже приведены некоторые из них.

Функции для обработки числовых выражений

ABS (<числовое_выражение>) - получение абсолютного значения.

INT (<числовое_выражение>) - получение целого значения.

SQRT (<числовое_выражение>) - определение квадратного корня.

LOG (<числовое_выражение>) - вычисление натурального логарифма.

EXP (<**n**>) - вычисление экспоненциальной функции (**n** -показатель степени).

ROUND (<числовое_выражение, <**n**>) - округление до **n** дробных разрядов.

MIN (<**n**1>, <**n**2>) - определение минимального из двух чисел **n**1 и **n**2.

MAX (<**n**1>, <**n**2>) - определение максимального из двух чисел **n**1 и **n**2.

STR (<числовое выражение>,<**n**>[,<**k**>]) - преобразование числа в строку символов длиной **n** с сохранением **k** дробных значений.

Например:

```
A1=ABS(-7.21)
B1=INT(A1)
? MIN(A1, B1)
C1=STR(A1+B1, 4, 1)
```

В этом примере переменной A1 присваивается числовое значение 7.21, а переменной B1 - числовое значение 7. Командой ? выводится на экран числовое значение переменной B1, равное 7.00, так как B1<A1.Переменной C1 присваивается символьное значение “14.2”, полученное преобразованием числа 14.21 (A1+B1=14.21).

Функции для работы с датой

DATE () - определение текущей системной даты.

TIME () - определение текущего системного времени.

DAY (<дата>) - определение дня месяца для указанной даты.

MONTH (<дата>)- определение номера месяца для указанной даты.

YEAR (<дата>) - определение года для указанной даты.

DOW (<дата>) - определение номера дня недели для указанной даты (воскресенье - первый день недели, суббота - седьмой день недели).

CDOW (<дата>) - получение названия дня недели для указанной даты (на английском языке).

CMONTH (<дата>) - получение названия месяца для указанной даты (на английском языке).

CTOD (<символьное выражение даты>) - преобразование символьного представления даты в данные типа даты.

DTOC (<дата>) - преобразование даты в символьное выражение.

Например:

```
SET DATE GERMAN
A1=CTOD(“15.01.96”)
? DOW(A1)
B1=DAY(A1)
C1=MONTH(A1)
```

D1=YEAR(A1)

В этом примере устанавливается германский формат даты dd.mm.yy. Переменной A1 присваивается значение даты 15.01.96, полученное преобразованием символьного выражения “15.01.96”. Командой ? выводится на экран числовое значение дня недели, определенное из значения переменной A1. Так как день 15 января 1996 года является понедельником, то будет выведено число 2. Переменной B1 присваивается числовое значение 15 (dd) из даты A1. Переменной C1 присваивается числовое значение 01 (mm) из даты A1. Переменной D1 присваивается числовое значение года 1996 (yy), полученное из даты 15.01.96 путем применения функции YEAR().

Функции для обработки файлов

SELECT () - определение номера текущей рабочей области.

FILE (<имя файла>) - проверка существования файла (если файл с указанным именем есть, выводится значение .T.).

DBF ([<**n**>]) - определение имени активной таблицы данных (**n** - номер рабочей области СУБД, по умолчанию - текущая область).

BOF ([<**n**>]) - проверка условия начала файла для таблицы данных (**n** - номер рабочей области).

EOF ([<**n**>]) - проверка условия конца файла для таблицы данных (**n** - номер рабочей области).

FCOUNT ([<**n**>]) - определение количества полей в таблице данных (**n** - номер рабочей области).

FIELDS (<**n**1> [,<**n**2>]) - определение имени поля в таблице данных (**n**1 - номер поля, **n**2 - номер рабочей области).

RECCOUNT ([<**n**>]) - определение количества записей в таблице из рабочей области **n.**

RECNO ([<**n**>]) - определение номера текущей записи таблицы данных из рабочей области **n.**

RECSIZE ([<**n**>]) - определение размера текущей записи таблицы данных из рабочей области **n.**

NDX (<**n**>) - определение открытого индексного файла для активной базы данных (**n** - номер индексного файла).

Например:

USE STUD

A2=DBF()

? EOF(), FILE (“STUD.DBF”)

B2=FCOUNT()

В этом примере активизируется таблица данных STUD. Переменной A2 присваивается символьное значение “STUD.DBF”. Командой ? на экран выводятся логические значения .F. и .T., так как при открытии таблицы данных указатель записи устанавливается на первую запись (значение EOF() равно .F.) и существует файл с именем STUD.DBF. Переменной B2

присваивается числовое значение 5, равное количеству полей в таблице данных STUD.

Функции для обработки символьных выражений

LEN (< строка >) - определение длины строки.

AT (<строка1>, <строка2>) - поиск первого вхождения строки <строка1> в строку <строка2> (результатом является номер позиции, в которой вхождение найдено).

UPPER (<строка>) - преобразование в строке латинских букв нижнего регистра в буквы верхнего регистра.

LOWER (<строка>) - преобразование в строке латинских букв верхнего регистра в буквы нижнего регистра.

ISALPHA (<строка>) - выполнение проверки, является ли первый символ строки буквой.

ISLOWER (<строка>) - выполнение проверки, является ли первый символ строки буквой нижнего регистра.

ISUPPER (<строка>) - выполнение проверки, является ли первый символ строки буквой верхнего регистра.

LEFT (<строка>, <**n**>) - вывод подстроки длиной **n** символов из указанной строки <строка>, начиная с 1-й позиции.

RIGHT(<строка>, <**n**>) - вывод подстроки длиной **n** символов из указанной строки <строка>, начиная с последней позиции.

RTRIM (<строка>) - удаление завершающих пробелов в строке.

REPLICATE (<строка>, <**n**>) - копирование строки **n** раз.

SPACE (<**n**>) - формирование строки из **n** пробелов.

STUFF (<строка1>, <**n**>, <**k**>, <строка2>) - преобразование строки символов <строка1> путем удаления **k** символов и ввода строки <строка2> , начиная с **n** -го символа.

SUBSTR (<строка>, <**n**> [,<**k**>]) - формирование подстроки из строки <строка> длиной **k** символов,начиная с **n** -го символа.

VAL (<строка>) - преобразование строки символов, состоящей только из цифр, в числовое выражение.

Например:

```
E1=UPPER (‘FoxPro’)
E2=SUBSTR (E1, 4, 3)
E3=SPACE (5)
? LEN (E1), AT (E2, E1)
```

В этом примере переменной E1 присваивается символьное значение ‘FOXPRO’. Переменной E2 присваивается символьное значение “PRO”, а переменной E3 - значение ‘ ‘ (5 пробелов). Командой ? на экран выводятся числовые значения 6 (длина символьной строки ‘FOXPRO’) и 4 (номер позиции вхождения строки “PRO” в строку “FOXPRO”).

КОНТРОЛЬНЫЕ ВОПРОСЫ И УПРАЖНЕНИЯ

1. Какая команда языка xBase позволяет создать или изменить структуру файла таблицы данных с именем BAZA.dbf?
2. Какая команда языка xBase позволяет добавлять записи в конец файла таблицы данных?
3. Какая команда языка xBase добавляет пустую запись в конец файла таблицы данных?
4. Какая команда языка xBase позволяет вставлять записи в файл таблицы данных перед текущей записью?
5. Какие команды языка xBase позволяют вывести: десятую запись активной таблицы базы данных на экран; записи активной таблицы базы данных со значением поля STIP=1500 на принтер?
6. Какой командой языка xBase можно отредактировать первые пять записей активной таблицы базы данных?
7. Какая команда языка xBase отменяет пометку на удаление отмеченных записей активной таблицы базы данных, для которых значение поля ZAH_KN>990001?
8. Какая команда языка xBase устанавливает указатель записи на начало файла активной таблицы базы данных?
9. Какой командой языка xBase выполняется создание файла S1.DBF с полями FIO и STIP и записями активной таблицы данных, отсортированными в алфавитном порядке по полю FIO?
10. Какой командой языка xBase осуществляется создание индексного файла S2.IDX для записей активной таблицы данных со значением ZAH_KN<99100, упорядоченных по убыванию поля STIP?
11. Какая команда языка xBase заменяет значение поля STIP на 1800 для записей активной таблицы базы данных, у которых значение поля DATA_R равно {01/12/75}?
12. Какие операторы языка xBase позволяют проверять условия при выполнении программ, реализовать циклы?
13. Какая команда языка xBase суммирует в переменной D1 содержимое поля STIP тех записей, для которых значение поля DATA_R меньше {02/10/75}?
14. Какая функция языка xBase определит абсолютное значение числа -5.3?
15. Какая функция языка xBase выражение даты {09/02/96} преобразует в символьное?
16. Какая функция языка xBase определяет, достигнут ли конец файла?
24. Какие действия выполняют функции MIN(), LOG(), ALLTRIM(), MONTH())?

Глава 7. SQL В СУБД FOXPRO

7.1. ФОРМИРОВАНИЕ ЗАПРОСОВ

СУБД FoxPro во всех своих версиях, в той или иной мере, поддерживает **язык запросов SQL**. Команды SQL могут использоваться в интерактивном режиме работы и включаться в программы наряду с собственными командами FoxPro на языке xBase. При этом необходимо придерживаться правил написания команд, принятых в СУБД FoxPro (см. 6.11). В последних версиях СУБД FoxPro реализован стандарт SQL-92, позволяющий создавать полноценные клиентские приложения, работающие с наиболее распространенными серверами баз данных.

При формировании запросов в языке SQL при работе в СУБД FoxPro могут использоваться любые **стандартные функции** языка xBase и **собственные функции** языка SQL работы с полями (см. 3.3).

Команда SELECT и вообще команды SQL мало зависят от текущего состояния среды FoxPro. Они сами открывают нужные им БД и индексные файлы. Если необходимых для выполнения команды индексов нет, они будут созданы, а по завершении команды уничтожены. Однако, лучше, чтобы применялись готовые индексы - для этого они должны быть открыты.

Команда SELECT является мощным средством обработки запросов. С ее помощью из таблицы-источника выделяются необходимые данные и пересылаются на экран, в файл, массив, таблицу, курсор, принтер. Можно с помощью команды SELECT получать данные из нескольких таблиц, логически связанных между собой. Команда имеет массу опций - возможностей. Ввиду этого ниже приведен сначала ее предварительный синтаксис с основными операторами (см. стандарт SQL в 3.3):

SELECT <что выводится> FROM <откуда (источник)>
INTO <куда (получатель)>
WHERE <каким условиям должны отвечать выбираемые данные>
GROUP BY <колонки, по которым выполняется группирование>
HAVING <условие отбора группируемых записей>
ORDER BY <в каком порядке выводить данные>.

Термин "колонка" здесь очень близок к понятию "поле" таблицы данных, но может быть и выражением. Кроме того, вследствие выборки можно получить (получатель) как новую таблицу данных, так и текстовый файл или только отображение на экране, т.е. колонки. Команда SELECT допускает включение в себя других внутренних команд SELECT (формирование подзапросов).

Более полный формат команды SELECT включает дополнительные опции:

SELECT [DISTINCT] [<псевдоним>.]<выражение> [AS <колонка>]
[, [<псевдоним>]<выражение> [AS <колонка>]..]

FROM<имя_таблицы1>[<псевдоним1>] [,<имя_таблицы2>[<псевдоним2>].]

[[INTO <получатель>]/[TO FILE < имя_файла> [ADDITIVE]/TO PRINTER]]

[NOCONSOLE] [PLAIN] [NOWAIT]

[WHERE <условие_связи> [AND <условие_связи>]

[AND/OR <условие_связи > [AND/OR <условие_связи>...]]]

[GROUP BY <колонка> [, <колонка>...]] [HAVING <условие_отбора>]

[ORDER BY <колонка> [ASC/DESC] [,<колонка> [ASC/DESC]...]]

Включение опции DISTINCT исключает возможность вывода одинаковых строк в выборке. Перед словом FROM перечисляются отбираемые <выражения>, а после - имена таблиц, из которых берутся данные. <Выражение> может быть полем записи из таблицы, константой (выводимой в каждой строке выборки), функцией от переменных (полей) и т.п. Если <выражение> является именем поля, то оно может быть составным (с включением имени таблицы данных или псевдонима), в особенности если выборка делается из нескольких таблиц, где имена полей совпадают. Псевдонимом может быть не только официальный псевдоним (ALIAS) таблицы данных, но и любое другое имя, которое присваивается в команде SELECT. Это задаваемое временное имя указывается в опции <псевдоним> после слова FROM за именем таблицы и используется только в данной команде SELECT в других ее опциях (локальный псевдоним). Если необходимо построить выборку из всех полей таблицы данных, вместо их перечня можно указать символ *. В результате выполненной выборки получается совокупность колонок, заголовками которых могут быть имена исходных полей.

Например, для вывода всех записей из таблицы STUD.DBF используется команда:

SELECT * FROM STUD

Для вывода названий всех лабораторий организации (поле LAB) из таблицы данных KADR.DBF с предотвращением вывода дублирующих значений, используется команда:

SELECT DISTINCT LAB FROM KADR

Команду SELECT можно использовать для одновременной выборки данных из нескольких таблиц. Если имена полей, выбираемых из разных таблиц совпадают, то такие колонки получают совпадающие имена, к которым присоединяется одна из букв (по алфавиту), например, FAM_A, FAM_B и т.д. Аналогичным образом даются имена колонкам, полученным в результате вычисления выражений. Их имена состоят из слова EXP и последовательных чисел (EXP_1, EXP_2 и т.д.). Исключения составляют выражения, использующие собственные функции SQL: AVG, MIN, MAX, SUM, COUNT. Последняя функция может иметь в качестве аргумента здездочку (COUNT(*)), что означает подсчет всех записей, попавших в

выборку. Имена колонок в этом случае будут включать имена функций. Вместо имен, формируемых по умолчанию, можно назначить колонкам другие имена, указав их после слова AS в виде <выражение> AS <новое_имя_колонки>.

Например, для вывода минимального, максимального и среднего значений зарплаты (поле ZARP) из KADR.DBF необходимо использовать команду:

SELECT MIN(ZARP), MAX(ZARP), AVG(ZARP) FROM KADR

Для вывода из KADR.DBF фамилий (поле FAM) и табельных номеров (поле TAB) с другими именами колонок (FAMILII и TABEL) используется команда:

SELECT FAM AS FAMILII, TAB AS TABEL FROM KADR

Для вывода всех фамилий из двух таблиц данных KADR.DBF и STUD.DBF с использованием локальных псевдонимов P и T, а также с заменой при выводе имен колонок FAM_A и FAM_B на KFIO и STFIO соответственно, используется команда:

SELECT P.FAM AS KFIO, T.FAM AS STFIO FROM KADR P, STUD T

В команде SELECT можно задать достаточно сложные условия для выборки данных в запрос:

SELECT <выражение> FROM <имя_таблицы>

[WHERE <условие_выборки> [AND <условие_связи>...]

[AND/OR <условие_ выборки > [AND/OR <условие_ выборки >...]]]

Условие связи применяется в случае, если выборка делается более, чем из одной таблицы данных, и определяет критерий объединения данных из разных таблиц. В условии связи указываются поля из разных таблиц с псевдонимами и используются знаки отношений =, #, = =, >, >=, <, <=. Допускается задание нескольких критериев, соединенных знаком AND. Условие выборки строится аналогично, но из выражений только для одной таблицы, и допускается использование логических операторов OR, AND и NOT.

Условия, кроме любых функций xBase, могут содержать следующие операторы SQL: LIKE, BETWEEN, IN. Эти операторы можно комбинировать с помощью связок OR, AND, NOT и скобок.

Оператор LIKE позволяет построить условие сравнения по шаблону, где символ "**_**" указывает единичный неопределенный символ в строке, а символ "**%**" - любое их количество. Формат оператора LIKE:

<выражение> LIKE <шаблон>.

Оператор BETWEEN задает начальное и конечное значение диапазона и проверяет, находится ли выражение, стоящее слева от оператора, в указанном диапазоне. Формат оператора BETWEEN:

<выражение> BETWEEN <нижнее значение> AND <верхнее значение>.

Оператор IN проверяет, находится ли выражение, стоящее слева от слова IN, среди перечисленных справа от него. Формат оператора IN:

<выражение> IN (<выражение1>, <выражение2>,...)

Например, для выборки фамилий (поле FAM), начинающихся на букву А, всех мужчин (поле POL='M') из таблицы KADR. DBF используется команда:

SELECT FAM FROM KADR WHERE POL='M' AND FAM LIKE “A”

Для вывода из таблицы STUD фамилий студентов (поле FAM), получающих стипендии (поле STIP) от 1500 до 2000 тенге, используется команда:

SELECT FAM FROM STUD WHERE STIP BETWEEN 1500 AND 2000

Для вывода на экран из таблицы KADR.DBF фамилий сотрудников (поле FAM), а из таблицы TABEL.DBF - соответствующего количества рабочих дней (поле WD) для записей, у которых совпадают табельные номера, необходимо использовать команду с условием связи (в качестве псевдонимов таблиц указаны имена таблиц):

SELECT KADR.FAM, TABEL.WD FROM KADR, TABEL WHERE KADR.TAB=TABEL.TAB

Для упорядочения по заданной колонке или колонкам используется опция ORDER BY. По умолчанию сортировка выполняется по возрастанию (ASC), но может быть задана и по убыванию (DESC).

Сокращенный вариант команды SELECT с использованием опции ORDER BY выглядит следующим образом:

SELECT <выражение> FROM <имя_таблицы>

ORDER BY <колонка1> [ASC/DESC] [,<колонка2> [ASC/DESC]...]

Например, для вывода фамилий (поле FAM) и зарплаты (поле ZARP) из таблицы данных KADR.DBF в порядке убывания размера заработной платы используется команда:

SELECT FAM, ZARP FROM KADR ORDER BY ZARP DESC

Для вывода фамилий (поле FAM) и группы (поле GRUPPA) из таблицы данных STUD.DBF в порядке возрастания групп и фамилий необходимо использовать команду:

SELECT FAM, GRUPPA FROM STUD ORDER BY GRUPPA, FAM

Опция GROUP BY команды SELECT позволяет сгруппировать записи с одинаковым значением указанной колонки (или колонок):

SELECT <выражение> FROM <имя_таблицы>

GROUP BY <колонка1> [,<колонка2>...] [HAVING <условие_отбора>]

Опция GROUP BY задает колонки, по которым производится группирование выходных данных. Все записи таблицы, для которых значения колонок совпадают, отображаются в выборке единственной строкой. Группирование удобно для получения некоторых сводных характеристик группы (суммы, среднего значения, количества записей в группе и т.д.).

Опция HAVING <условие_отбора> задает критерий отбора данных в каждую сформированную в процессе выборки группу, т.е. выполняет роль опции WHERE, но для группируемых данных.

Например, для вывода названий всех групп (поле GRUPPA), количества студентов и суммы стипендий (поле STIP) для каждой группы из таблицы данных STUD.DBF используется команда:

```
SELECT GRUPPA, COUNT(FAM), SUM(STIP) FROM STUD
GROUP BY GRUPPA
```

Для вывода названий всех лабораторий (поле LAB), количества сотрудников и значения суммарной зарплаты (поле ZARP) из таблицы данных KADR.DBF используется следующая команда (информация выводится только для лабораторий, где количество сотрудников больше 5):

```
SELECT LAB, SUM(ZARP), COUNT(*) FROM KADR
GROUP BY LAB HAVING COUNT(*)>5
```

Для указания объекта получателя данных выборки используется опция INTO или TO.

Ниже приведен сокращенный вариант команды SELECT с опцией INTO/TO:

```
SELECT <выражение> FROM <имя_таблицы>
[INTO DBF/TABLE <имя_таблицы>] / [INTO CURSOR <имя_курсора>]
/
[INTO ARRAY <имя_массива>] / [TO FILE <имя_файла>
[ADDITIVE]] /
[TO PRINTER] [NOCONSOLE] [PLAIN] [NOWAIT]
```

Типы возможных получателей данных выборки описаны ниже.

DBF/TABLE <имя_таблицы> - получателем является новая таблица с указанным именем (слова DBF и TABLE здесь являются синонимами).

CURSOR <имя_курсора> - результат запроса помещается в **курсор** с указанным именем. Курсор - это **временный набор данных**, который может быть областью памяти или временным файлом FoxPro и имеет режим "только чтение". Данные курсора могут быть, например, предъявлены в команде BROWSE, напечатаны, из них может быть образовано меню и т.д. Курсор может быть обработан другой командой SELECT. К колонкам курсора надо обращаться по имени этих колонок с префиксом - именем курсора (через точку).

ARRAY <имя_массива> - в качестве получателя результата запроса будет использован новый двумерный массив с указанным именем.

Кроме того, данные выборки можно переслать в файл или на принтер. Для этого в команде указывается получатель TO FILE <имя_файла> [ADDITIVE] / TO PRINTER и выборка посылается в текстовый файл с указанным именем или на принтер. Если используется слово ADDITIVE, то выборка будет добавлена в конец существующего файла без его перезаписи.

Следующие опции имеют смысл только при выдаче на экран (команда используется без опций INTO или TO):

NOCONSOLE - выборка не выдается на экран,

PLAIN - заголовки колонок не выдаются,

NOWAIT - не делается пауза при заполнении экрана.

Например, для вывода фамилий (поле FAM) и зарплаты (поле ZARP) из таблицы KADR.DBF в таблицу FAMSZ.DBF необходимо использовать команду:

```
SELECT FAM, ZARP FROM KADR INTO TABLE FAMSZ
```

Для вывода фамилий (поле FAM) и стипендий (поле STIP) из таблицы STUDENT.DBF в курсор CURSTUD необходимо использовать команду:

```
SELECT FAM, STIP FROM STUDENT INTO CURSOR CURSTUD
```

Для занесения в текстовый файл FIO.TXT всех фамилий из двух таблиц данных KADR.DBF и STUD.DBF с использованием локальных псевдонимов S и T и заменой при выводе имен колонок FAM_A и FAM_B на KADRFIO и STUDFIO, соответственно, используется команда:

```
SELECT  S.FAM AS KADRFIO, T.FAM AS STUDFIO
FROM KADR S, STUD T  TO  FILE FIO.TXT
```

Для вывода на принтер названий лабораторий (поле LAB), сумм зарплат и количества сотрудников, получающих зарплату (поле ZARP) больше 9000, сгруппированных по каждой лаборатории необходимо использовать следующую команду (вывод данных в порядке убывания сумм зарплат):

```
SELECT LAB, SUM(ZARP), COUNT(*) FROM KADR TO PRINTER
WHERE ZARP>9000 GROUP BY LAB ORDER BY ZARP DESC
```

Использование подзапросов в команде SELECT позволяет создавать сложные запросы. Подзапрос заключается в круглые скобки и представляет собой вложенную команду SELECT. Причем, основной запрос и используемый в нем подзапрос могут обращаться как к одним и тем же таблицам, так и к разным таблицам баз данных. Например, для выборки фамилий сотрудников (поле FAM) из таблицы KADR.DBF с заработной платой (поле ZARP) выше средней можно использовать запрос с подзапросом (для таблицы KADR.DBF используется локальный псевдоним S1):

```
SELECT S1.FAM FROM KADR S1 WHERE ZARP >
(SELECT AVG(ZARP) FROM KADR)
```

Результаты нескольких выборок можно объединить в одном запросе, используя оператор объединения UNION. Результатом будет множество, состоящее из всех строк, входящих в какую-либо выборку или в несколько выборок. Но при этом результаты исходных выборок должны иметь одинаковое число полей (столбцов) и тип i-го поля одной выборки должен совпадать с типом i-го поля любой другой выборки. При использовании опции UNION часто оказывается полезным включение константы в

получаемый результат выборки. Например, текстовую константу можно использовать в качестве поясняющего текста при выборе из таблицы STUD.DBF фамилий студентов (поле FAM), получающих стипендию (поле STIP) больше 2000 или проживающих в городе Алматы (город проживания указывается в поле ADRESS):

```
SELECT FAM, “стипендия больше 2000” FROM STUD WHERE STIP> 2000
                    UNION
SELECT FAM, “город Алматы” FROM STUD WHERE ADRESS LIKE “%Алматы”
```

Оператором UNION можно соединить любое число команд SELECT, но опция ORDER BY в запросе с использованием оператора UNION может входить только в последнее предложение SELECT. При указания критерия упорядочивания указываются номера полей в получаемой выборке. Например, при выборке из таблицы KADR.DBF фамилий сотрудников (поле FAM), имеющих заработную плату (поле ZARP) меньше 10000 или проживающих в городе Алматы (город проживания указывается в поле ADRESS), можно сначала упорядочить данные по второй колонке (признак выборки), а затем по первой колонке (фамилии в алфавитном порядке):

```
SELECT FAM , “заработная плата меньше 1000”
FROM KADR WHERE ZARP<10000
UNION
SELECT FAM , “ город Алматы ” FROM KADR
  WHERE ADRESS LIKE “%Алматы”
ORDER BY    2,1
```

7.2. ОПРЕДЕЛЕНИЕ ДАННЫХ

Для **создания** базы данных используется команда CREATE DATABASE <имя_базы_данных>. Например, для создания БД с именем BASA1 используется команда: CREATE DATABASE BASA1

Для **определения** структуры таблицы данных используются команды создания (CREATE TABLE) и изменения (ALTER TABLE) структуры таблицы. Таблица создается в активной БД. Если нет активной БД, то создается свободная таблица.

Команда создания структуры таблицы данных имеет следующий формат:

CREATE TABLE <имя_таблицы> (<определение_поля1> [,<определение_поля2>...]) / FROM ARRAY <имя массива>

Команда создает новую таблицу данных (DBF-файл) с указанным именем. Для каждого поля задаются его имя и тип (одной из букв: C - символьный; N - числовой; F - с плавающей запятой; D - дата; L - логический; M - примечание и т.д.). Для типов полей, которые могут иметь

разную ширину, дополнительно необходимо указать требуемую ширину. Для числовых полей с плавающей запятой указывается также точность (число десятичных разрядов). Если создаваемое поле будет являться индексом, то необходимо указать тип индекса. Кроме того, в определении поля можно указать признак (NOT NULL) обязательного заполнения поля при вводе данных в таблицу, значения полей по умолчанию и ограничения на ввод значений.

Описание полей может быть задано и из массива при использовании опции FROM ARRAY. Такой двумерный массив из шести столбцов и строк в количестве, совпадающем с количеством полей, должен содержать информацию о имени, типе, ширине, числе десятичных разрядов, указании типа ключа, возможности использования значения NULL для каждого поля (шесть параметров поля).

Например, для создания структуры таблицы KADR.DBF, предназначенной для хранения следующей информации: фамилий (поле FAM, символьного типа, длиной 25 символов); табельных номеров (поле TAB, целого типа, является первичным ключом - индексом); дат рождения (поле DTR, типа даты); размера зарплаты (поле ZARP, числового типа, общей длиной 7 символов, из которых 2 символа под дробную часть числа) - необходимо использовать следующую команду:

```
CREATE TABLE KADR (FAM C(25), TAB I PRIMARY KEY,
  DTR D, ZARP N(7,2))
```

А чтобы создать структуру таблицы BD1, повторяющую структуру таблицы KADR.DBF, из сгенерированного массива MAS1 (функция AFIELDS() генерирует массив из описания полей открытой таблицы данных KADR.DBF), необходимо использовать следующую последовательность команд:

```
USE KADR.DBF
=AFIELDS(MAS1)
CREATE TABLE BD1 FROM ARRAY    MAS1
```

Можно создать временную таблицу, с которой можно работать как с обычной таблицей данных, называемую **курсором** (CURSOR). Курсор располагается в оперативной памяти в свободной рабочей области и доступен сразу после создания до тех пор, пока он не будет закрыт. При закрытии курсора временная таблица удаляется из памяти. Команда создания курсора CREATE CURSOR аналогична команде CREATE TABLE.

При изменении структуры таблицы используется команда ALTER TABLE с ключевыми словами, которые позволяют добавить, удалить, переименовать поле, изменить параметры поля. Эта команда имеет формат:

```
ALTER TABLE <имя_таблицы>
[ADD <определение_добавляемого_поля>]
[DROP <имя_удаляемого_поля>]
[RENAME <старое_имя_поля>    TO <новое_имя_поля>]
```

[ALTER <имя_изменяемого_поля>
<определение_изменяемых_параметров_поля>]

Например, для добавления символьного поля FAX шириной 12 позиций и удаления поля TELEFON в таблице KADR.DBF можно использовать команду:

ALTER TABLE KADR ADD FAX C (12) DROP TELEFON

Можно удалить не только поле (колонку) из таблицы, но и всю таблицу целиком командой:

DROP TABLE <имя_таблицы>

Например, следующая команда удалит таблицу BD1:

DROP TABLE BD1

Для удаления базы данных со всеми таблицами используется команда:

DROP DATABASE <имя_базы_данных>

Например, следующая команда удалит базу данных BASA1:

DROP DATABASE BASA1

7.3. МАНИПУЛИРОВАНИЕ ДАННЫМИ

Дополнять записями базу данных можно используя в качестве источника данных выражения, массивы, переменные, константы, результат запроса.

Для дополнения записей в таблицу БД используются команда INSERT, которая предварительно открывает закрытую таблицу БД. Команда SQL дополнения записями таблицы данных имеет следующий формат:

INSERT INTO <имя_таблицы> [(<имя_поля1> [, <имя_поля2> ...])]
VALUES (<выражение1> [, <выражение2> ...])

Команда добавляет записи в конец существующей таблицы, используя выражения, перечисленные после слова VALUES. Выражения заносятся в указанные поля. Если опущены имена полей, выражения будут записываться в последовательные поля таблицы данных в соответствии с ее структурой.

Например, результат следующих двух команд будет одинаковым:

INSERT INTO KADR (FAM, ZARP) VALUES ('Каиров', 8200)
INSERT INTO KADR (ZARP, FAM) VALUES (8200, 'Каиров')

Для дополнения записей в таблицу данных с использованием массивов используется следующая команда SQL:

INSERT INTO <имя_таблицы> FROM ARRAY <имя_массива>

Команда добавляет записи в конец таблицы данных, используя данные, содержащиеся в указанном массиве. Данные из массива заносятся в последовательные поля, начиная с первого. Типы соответствующих полей и элементов массива должны совпадать.

Например, для определения значения элементов массива C1 и добавления этих данных из массива в поля таблицы данных KADR.DBF необходимо использовать следующую последовательность команд:

C1(1)="Ахметов"
C1(2)=235
C1(3)={12/20/74}
C1(4)=8000
INSERT INTO KADR FROM ARRAY C1

Во многих СУБД SQL-команду INSERT можно использовать с подзапросом - вложенной командой SELECT, если множество дополняемых данных является результатом запроса:

INSERT INTO <имя_таблицы> [(<имя_поля1> [, <имя_поля2> ...])]
<подзапрос>

Например, дополнение в таблицу STUD1.DBF фамилий студентов (поле FAM) из таблицы STUD2.DBF осуществляется следующей командой:

INSERT INTO STUD1 (FAM) VALUES
(SELECT DISTINCT FAM FROM STUD2)

Для **модификации** данных в таблице БД используется команда UPDATE, которая имеет следующий формат:

UPDATE <имя_таблицы> SET <имя_поля1> = <выражение1>
[, <имя_поля2> = <выражение2>....] [WHERE <условие>]

Замена значений происходит в записях, удовлетворяющих условию, указанному после опции WHERE. Если эта опция отсутствует, то замена значений происходит во всех записях таблицы. Например, для замены фамилии (поле FAM) ‘Ахметов’ на ‘Ахмет’ в таблице STUD необходимо использовать следующую команду:

UPDATE STUD SET FAM = ‘Ахмет’ WHERE FAM=’Ахметов’

При модификации данных можно использовать подзапрос - вложенную команду SELECT для формирования операнда условия:

UPDATE <имя_таблицы> SET <имя поля> = <выражение>
[WHERE <операнд условия> <оператор условия> (<подзапрос>)]

Например, установка надбавки к стипендии студентов (поле STIP) на 25% в таблице STUD1.DBF, если фамилии этих студентов (поле FAM) имеются в таблице STUD2.DBF, реализуется следующей командой:

UPDATE STUD1 SET STIP=1.25*STIP WHERE FAM IN
(SELECT FAM FROM STUD2)

В FoxPro поддерживается команда SQL, которая **помечает** записи на удаление - DELETE, без физического удаления записей. Физическое удаление записей можно реализовать командой PACK. Команда DELETE имеет следующий формат:

DELETE FROM <имя_таблицы> [WHERE <условия>]

Помечаются на удаление записи, удовлетворяющие условию, указанному после опции WHERE. Если эта опция отсутствует, то

помечаются на удаление все записи из таблицы. Например, для пометки на удаление данных о сотруднике с табельным номером 65 (поле TAB) из таблицы KADR.DBF используется команда:

DELETE FROM KADR WHERE TAB=65

Для пометки на удаление всех записей из таблицы KADR.DBF необходима команда:

DELETE FROM KADR

При удалении можно использовать подзапрос - вложенную команду SELECT для формирования операнда условия:

DELETE FROM <имя_таблицы>

[WHERE <операнд_условия> <оператор_условия> (<подзапрос>)]

Например, необходимо пометить на удаление данные о студентах из таблицы STUD1.DBF, если фамилии этих студентов (поле FAM) имеются в таблице STUD2.DBF. Это можно реализовать следующей командой:

DELETE FROM STUD1 WHERE FAM IN
(SELECT FAM FROM STUD2)

7.4. УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ

В технологии баз данных **транзакция** представляет собой логическую единицу работы с данными в базе данных. В общем случае транзакция представляет собой последовательность нескольких операций, которая преобразует некоторое целостное состояние базы данных в другое целостное состояние, однако не гарантирует сохранения целостности во все промежуточные моменты времени.

Имеются команды управления транзакциями COMMIT (фиксация) и ROLLBACK (откат, аннулирование).

Команда COMMIT сигнализирует о успешном завершении транзакции. Она сообщает СУБД, что логическая единица работы была успешно завершена, база данных снова находится в целостном состоянии, все произведенные этой единицей работы обновления могут быть зафиксированы (приняты за совершившиеся).

Команда COMMIT устанавливает, так называемую, точку фиксации (синхронизации). Точка фиксации соответствует концу логической единицы работы. И выполнение команды ROLLBACK (откат) возвращает БД в состояние предыдущей точки фиксации.

Команда ROLLBACK сигнализирует о неудачном завершении транзакции. Она сообщает СУБД, что в результате аварийной ситуации, возможно, было нарушено целостное состояние БД. Поэтому для всех обновлений, произведенных транзакцией, необходимо выполнить откат, т.е. аннулировать их. Под обновлением в этом случае понимаются результаты выполнения команд INSERT, UPDATE, DELETE.

Наличие команд COMMIT и ROLLBACK является гарантией того, что транзакция будет выполнена до конца или, в противном случае, будут аннулированы все обновления, произведенные незавершившейся транзакцией.

КОНТРОЛЬНЫЕ ВОПРОСЫ И УПРАЖНЕНИЯ

1. Какие собственные функции языка SQL используются в командах SQL в СУБД FoxPro?
2. Составьте SQL - команду, позволяющую создать структуру файла таблицы данных с именем KADR1.DBF, содержащую следующие поля: FAM (фамилия) – символьного типа (20 позиций); SEM – (семейное положение) – логического типа; DET (количество детей) – числового типа (2 позиции).
3. Составьте команду, позволяющую создать структуру файла таблицы данных с именем KADR2.DBF, содержащую следующие поля: FAM (фамилия) – символьного типа (15 позиций); TAB (табельный номер) – числового типа (4 позиции); PER (перемещение по службе) – примечания, LAB (лаборатория) – символьного типа (15 позиций).
4. Составьте SQL команду, добавляющую в конец существующего файла таблицы данных KADR1.DBF следующую запись: “Юн” (поле FAM (фамилия) – символьного типа), ‘T’ (true - истина), (поле SEM – семейное положение, логического типа), 2 (поле DET – количество детей, числового типа).
5. Составьте команду, добавляющую в конец существующего файла таблицы данных KADR2.DBF следующую запись: ‘Исин’ (поле FAM – фамилия, символьного типа), 5 (поле TAB – табельный номер, числового типа), ‘15.04.97 переведен в лабораторию ТО’ (поле PER - примечания).
6. Составьте SQL-команду добавления записи с данными о студенте Ахметове, получающего стипендию 1400 тенге в конец файла таблицы данных с именем STUD, содержащего следующие поля: поле FAM (фамилия) - символьного типа; поле GODR (дата рождения) - типа даты; поле STIP (стипендия) - числового типа.
7. Составьте команду вывода фамилий студентов (поле FAM), стипендии (поле STIP) из таблицы данных STUD.DBF в массив FS.
8. Составьте команду вывода фамилии (поле FAM) и группы (поле GRUPPA) из таблицы данных STUD.DBF в порядке возрастания групп и фамилий в курсор CUR1.
9. Составьте команду вывода минимального, максимального и среднего значения стипендии студентов (поле STIP) из таблицы данных STUD.DBF на экран.
10. Составьте команду вывода из таблицы данных KADR2.DBF фамилий (поле FAM) и названий лабораторий (поле LAB) в новую таблицу базы данных KDR.

11. Составьте команду вывода из таблицы данных KADR1.DBF фамилий (поле FAM) и количество детей (поле DET) с другими именами полей (FAMILII и DETI).
12. Составьте команду вывода из таблицы данных KADR2.DBF фамилий сотрудников (поле FAM), работающих в лаборатории “TO” (LAB = “TO”), в текстовый файл.
13. Составьте команду вывода из таблицы данных KADR2.DBF фамилий сотрудников (поле FAM), работавших или работающих в лаборатории “TO”. Поиск ведется в мемо – поле PER (сведения о перемещениях по службе).
14. Составьте команду вывода из таблицы данных KADR2.DBF названий лабораторий (поле LAB) и количества сотрудников, сгруппированных по лабораториям, для лабораторий с количеством сотрудников больше 5.
15. Составьте команду вывода из таблицы данных KADR2.DBF и KADR1.DBF названий лабораторий (поле LAB) и суммарного количества детей сотрудников (поле DET) по каждой лаборатории.
16. Составьте команду вывода из таблицы данных KADR1.DBF (псевдоним A) фамилий сотрудников (поле FAM), а из таблицы данных KADR2.DBF (псевдоним B) – названий лабораторий, где они работают (поле LAB), для семейных сотрудников.
17. Составьте команду вывода из таблицы данных KADR1.DBF фамилий сотрудников с указанием их семейного положения (поля FAM и SEM, в качестве псевдонима взять имя таблицы данных), а из таблицы данных KADR2.DBF – их табельных номеров и названий лабораторий (поля TAB и LAB, в качестве псевдонима взять имя таблицы данных) для записей, у которых совпадают фамилии.
18. Составьте команду пометки на удаление данных о сотрудниках из таблицы данных KADR1.DBF, если эти сотрудники работают в лаборатории ‘TO’ (поле LAB таблицы KADR2.DBF).
19. Составьте команду вывода из таблиц базы данных KADR1.DBF и KADR2.DBF фамилий тех сотрудников (поле FAM), кто работает в лаборатории ‘TO’ (поле LAB таблицы KADR2.DBF) или имеет более трех детей (поле DET таблицы KADR1.D

Глава 8. MICROSOFT SQL SERVER

8.1. АРХИТЕКТУРА MICROSOFT SQL SERVER

Microsoft SQL Server (MS SQL Server) работает под управлением операционной системы Windows и является одним из широко используемых на персональных компьютерах SQL-серверов. Эта СУБД предназначена для использования в качестве сервера в архитектуре клиент-сервер. Основное ее назначение, как и всех серверов баз данных, - обеспечение высокоэффективного хранения данных и обслуживания клиентов. В качестве клиента может выступать любое приложение, поддерживающее язык управления данными SQL: приложения пользователя, приложения поставщика инструментальных средств для создания и выполнения других приложений. В MS SQL Server реализован диалект SQL, называемый Transact-SQL. Transact-SQL добавляет к базовому SQL ключевые слова, позволяющие производить формирование инструкций (команд) выборки, сохранения данных и манипуляций над ними. Несмотря на дополнительные возможности и расширения, внесенные в Transact-SQL, его синтаксис очень близок к стандарту SQL (см. 3.3).

В силу своего функционального назначения MS SQL Server не имеет средств создания прикладных программ, но включает в себя инструменты проектирования и управления базами данных, такие как SQL Interprise Manager. SQL Interprise Manager - это тоже клиент MS SQL Server.

В состав MS SQL Server входит множество сервисов и компонентов, которые могут располагаться и на компьютере-клиенте, и на компьютере-сервере. Эти сервисы и компоненты могут взаимодействовать друг с другом, а также с сервисами и компонентами Windows. Ниже приведено краткое описание основных компонентов MS SQL Server.

Distributed Transaction Coordinator - координирует выполнение распределенных транзакций, т.е. транзакций, выполняющихся на нескольких серверах.

SQL Server Agent - управляет задачами (jobs), оповещениями (alerts), операторами (operators), уведомлениями (notifications), репликацией данных (replication). Этот компонент позволяет планировать резервирование и задачи административной поддержки.

SQL Mail - позволяет отправлять и получать сообщения, состоящие из короткой текстовой строки и (или) присоединенного файла.

Logins (регистрации, входы в систему) - набор параметров для определения имени и пароля пользователя, связанных с ними уровней доступа к базам данных.

Server Roles (роли сервера) - компонент, позволяющий объединить пользователей в группы с назначением определенных полномочий для всей группы, использует систему безопасности Windows NT.

Remote Servers - предоставляет возможность настройки удаленных серверов.

Linked Servers - предоставляет возможность конфигурирования связанных серверов (для выполнения распределенных запросов).

Backup Devices - устройства, которые используются для резервирования и последующего восстановления баз данных и журнала транзакций (см. 4.3). Резервирование осуществляется путем записи копии данных в виде архива на специально предназначенное для этого устройство.

Web Assistant Jobs - мастер создания стандартных файлов HTML (Web-страниц) из данных MS SQL Server.

Database Maintenance Plans - мастер создания задач эксплуатации баз данных: реорганизация страниц данных и индексов, сжатие файлов базы данных после удаления из нее пустых страниц, выполнение проверки согласованности данных, резервирование базы данных и журнала транзакций.

SQL Server Logs (журналы транзакций) - файлы, куда MS SQL Server записывает все транзакции, выполняемые в базе данных, перед тем как зафиксировать изменения в самой базе данных. Для каждой базы данных определяется свой журнал транзакций.

Databases - базы данных, состоящие из таблиц с данными и других объектов, обеспечивающих дополнительную поддержку действий, выполняемых над данными.

Ниже рассматривается архитектура данных в последних версиях MS SQL Server (7.0 и 2000). Организацию баз данных можно рассматривать с двух сторон: логической и физической. Данные Microsoft SQL Server организованы в логические компоненты, видимые пользователю. Логическая структура базы данных описывает структуру таблиц, взаимоотношения между ними, список пользователей, хранимые процедуры, правила, умолчания и другие объекты базы данных. Физическая структура базы данных включает в себя описание файлов базы данных и журнала транзакций, их первоначальный размер, шаг прироста (если разрешено автоматическое увеличение размера базы данных), максимальный размер и т.д. Физическая реализация баз данных представляет собой два или больше файлов на диске (рис.8.1).

Для работы MS SQL Server использует 4 системные базы данных (**MASTER, MSDB, TEMPDB и MODEL**). В комплект поставки также включены две пользовательские базы данных (**PUBS и NORTHWIND**), специально предназначенные для экспериментов. Кроме того, может быть определено несколько пользовательских баз данных для различных предметных областей (рис.8.2). Все базы данных, как пользовательские, так и системные физически организованы одинаково.

База данных MASTER - эта системная база данных является главной базой данных MS SQL Server. Остальные системные базы данных имеют

второстепенное значение и их можно считать вспомогательными. В базе данных MASTER хранится вся системная информация о параметрах конфигурации сервера, имеющихся на сервере пользовательских баз данных, имеющих доступ к серверу пользователях и другая системная информация.

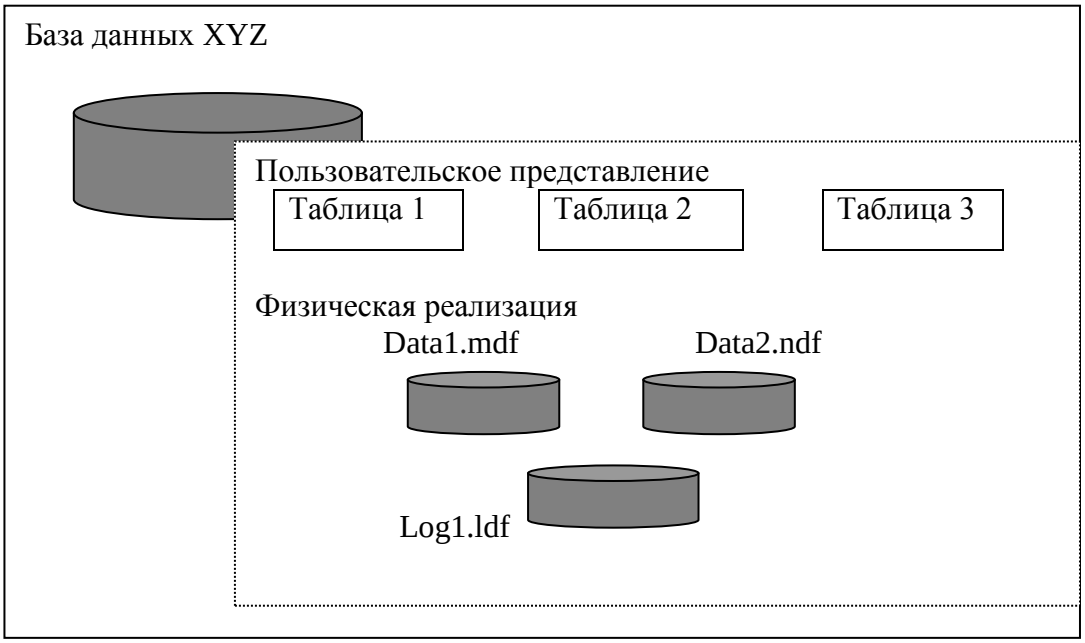


Рис. 8.1. Логическая и физическая базы данных

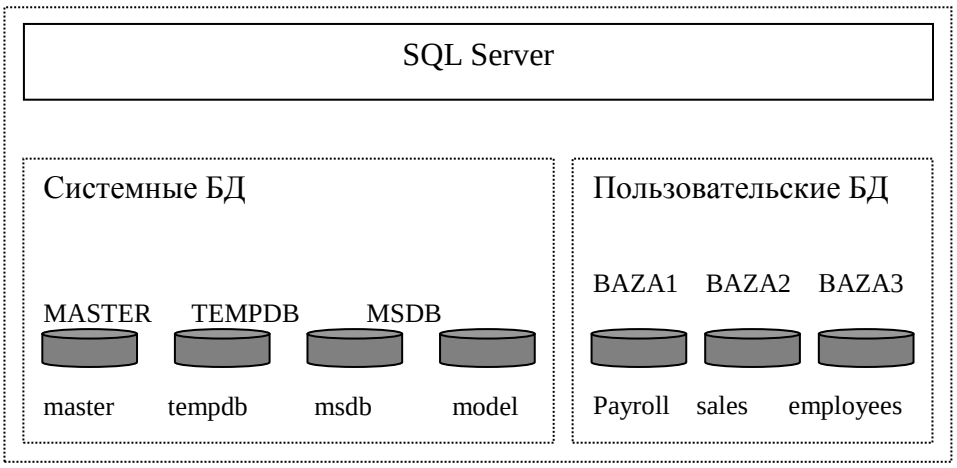


Рис.8.2. Базы данных MS SQL Server

База данных MODEL - эта системная база данных является шаблоном для создания новых баз данных. Технология создания новой базы данных в MS SQL Server построена следующим образом: сервер копирует базу данных MODEL в указанное место и изменяет ее имя соответствующим образом. Если при создании базы данных не указаны никакие параметры, кроме ее имени, то новая база данных будет являться полной копией базы данных MODEL. Если же размер и состав файлов создаваемой базы данных указан

явно, то скопированная база данных изменяется соответствующим образом.

Изменяя параметры базы данных MODEL, можно управлять параметрами по умолчанию создаваемых баз данных. Кроме того, базу данных MODEL можно использовать в качестве корпоративного стандарта на содержимое и свойства базы данных. Можно ускорить создание множества однотипных таблиц со специализированной конфигурацией, если соответствующим образом изменить базу данных MODEL.

База данных TEMPDB используется для хранения временных объектов, создаваемых пользователем во время сеанса работы. Пользователям иногда необходимо создавать временные таблицы, представления, курсоры и другие объекты для сохранения промежуточных результатов. MS SQL Server позволяет создавать такие временные объекты. Для создания временной таблицы достаточно добавить перед ее именем символ # или ##. Сервер автоматически создаст временную таблицу. Временные объекты могут быть локальными или глобальными. Отличительной особенностью базы данных TEMPDB является то, что она уничтожается каждый раз, когда происходит останов сервера. При следующем запуске MS SQL Server база данных TEMPDB создается заново на основе системной базы данных MODEL, так же как и пользовательские базы данных.

База данных MSDB предназначена для хранения всей информации, относящейся к автоматизации администрирования и управления MS SQL Server, а также информации об операторах и событиях. Кроме того, в этой базе данных хранится информация о расписании автоматического запуска заданий. То есть в базе данных MSDB размещается вся системная информация, используемая службой SQL Server Agent.

При подключении клиента к MS SQL Server соединение ассоциируется с конкретной базой данных на сервере, которая называется текущей базой данных. Переключаться между базами данных из клиентского приложения можно с помощью команды Transact-SQL USE имя_базы_данных или используя функции API, которые изменяют содержание текущей базы данных. MS SQL Server, начиная с версии 7.0, позволяет отсоединять базу данных от сервера и присоединять ее к другому серверу или опять к тому же самому. Независимо от того, является база данных системной или пользовательской, все они подчиняются одним и тем же определенным правилам.

8.2. ФИЗИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ

Каждая база данных MS SQL Server имеет минимум два файла: один для журнала транзакций и один для базы данных. Эти файлы имеют различную структуру и при работе с ними применяются разные правила. Более сложные базы данных имеют несколько файлов для данных и журнала транзакций. Эти файлы могут объединяться в группы для упрощения

администрирования базы данных. Каждый файл состоит из страниц.

Разные файлы одной базы данных могут располагаться на разных физических дисках. Такой подход очень эффективен при использовании множества физических дисков и контроллеров. В MS SQL Server база данных и журнал транзакций хранятся отдельно, что повышает надежность системы. Если произойдет сбой и файлы базы данных будут повреждены, то можно будет восстановить базу данных из резервной копии, а затем восстановить сделанные изменения из журнала транзакций, который останется неповрежденным. Помимо обеспечения отказоустойчивости, правильное использование файлов и групп файлов может повысить производительность выполнения обращений к базе данных. Так как дисковые операции ввода/вывода при использовании отдельных контроллеров выполняются на всех дисках параллельно, то производительность в некоторых случаях может повыситься в несколько раз.

Каждый файл, используемый в базе данных, имеет два имени:

Logical File Name - логическое имя файла. Это имя используется в командах Transact-SQL при ссылке на конкретный файл.

OS File Name - имя файла в операционной системе. Под этим именем файл хранится на жестком диске.

База данных MS SQL Server может иметь в своем составе файлы следующих типов:

Primary - основной файл. Файл этого типа содержит системную информацию о самой базе данных и ее объектах. В этом файле размещаются системные таблицы и описание объектов базы данных. Например, на девятой странице primary-файла содержится информация об атрибутах базы данных. В primary-файле хранятся также данные. Каждая база данных имеет в своем составе файл типа primary, причем только один файл в базе данных может относиться к этому типу. По умолчанию файлам типа primary присваивается расширение .MDF (рис.8.1).

Secondary - вторичный файл. Файлы этого типа не содержат системную информацию и используются только для хранения данных. Все данные, которые не разместились в файле primary, располагаются в файлах secondary. База данных может вообще не иметь файла типа secondary или, наоборот, иметь их несколько. Часто в базе данных создается множество файлов secondary, чтобы распределить данные среди нескольких физических дисков. По умолчанию файлам типа secondary присваивается расширение .NDF (рис.8.1).

Transaction Log - журнал транзакций. Файлы этого типа используются для хранения информации о транзакциях, выполненных в базе данных. Каждая база данных имеет как минимум один файл журнала транзакций. Можно использовать несколько файлов для ускорения операций дискового ввода/вывода, так как данные о транзакциях записываются параллельно во все файлы. По умолчанию файлам типа transaction log присваивается

расширение .LDF (рис.8.1).

Расширения, назначаемые файлам базы данных, рекомендованные, но не являются обязательными. Разрешается использовать произвольные расширения для каждого файла.

В последние версии MS SQL Server введена возможность автоматического роста базы данных. Все файлы самой базы данных (как файлы данных, так и файлы журнала транзакций) могут автоматически увеличиваться (auto grow). Эта возможность определяется при создании файла. Шаг прироста можно устанавливать в процентах от первоначального размера или в мегабайтах. Дополнительно можно указать максимальный размер, до которого файлу разрешено увеличиваться. Если максимальный размер не указывается, файл будет увеличиваться до тех пор, пока не займет все свободное пространство на диске. Возможность автоматического увеличения файлов снимает необходимость постоянного контроля размера файлов базы данных. Пользователи, не имеющие административных полномочий, могут беспрепятственно работать с базой данных, не испытывая неудобств при ее критическом заполнении.

Файлы могут объединяться в группы для упрощения администрирования и управления физическим размещением объектов базы данных. В некоторых системах правильное использование групп файлов может дать значительное увеличение производительности. Каждый файл может быть включен только в одну группу файлов. Кроме того, нельзя определять группы файлов для файлов журнала транзакций. Группы файлов бывают следующих типов:

Primary File Group - основная группа файлов. Эта группа файлов содержит primary-файл и все файлы, не включенные в другие группы. Все страницы системных таблиц базы данных располагаются в основной группе файлов. База данных может иметь только одну основную группу файлов.

User File Group - пользовательская группа файлов. В пользовательские группы файлов включаются все файлы, указанные в параметре FILEGROUP при создании (команда CREATE DATABASE) или изменении (команда ALTER DATABASE) базы данных. В базе данных можно создать несколько пользовательских групп файлов с произвольным набором файлов или ни одной.

Default File Group - группа файлов по умолчанию. Одна из групп файлов, созданных в базе данных, назначается в качестве группы файлов по умолчанию. Если явно не указано иное, то группой файлов по умолчанию при создании базы данных назначается основная группа файлов. Если при создании объекта базы данных (таблицы или колонки) не указано явно, к какой группе файлов он будет принадлежать, то этот объект создается в группе файлов по умолчанию. Владелец базы данных может назначить любую группу файлов в качестве группы файлов по умолчанию. Только одна группа файлов может быть назначена в качестве группы файлов по

умолчанию.

Когда данные записываются в какую-то группу файлов, то данные распределяются между всеми файлами в этой группе. То есть вместо того, чтобы записывать данные сначала в первый файл, пока он не заполнится, потом во второй файл и т. д., производится параллельная запись данных во все файлы группы, что обеспечивает повышение производительности операций ввода/вывода. Причем количество данных, записываемых в каждый файл, пропорционально размеру файла. Например, если в группе определено три файла размером, соответственно, 1 Мбайт, 2 Мбайт и 3 Мбайт, то при записи 24 строк данных они будут распределены следующим образом: в первый файл будет записано 4 строки, во второй - 8 строк, а в третий - 12 строк. Если в группе определено множество файлов, то файлы не будут увеличиваться до тех пор, пока все они не будут полностью заполнены. Если один из файлов будет заполнен, то и другие файлы также будут заполнены. Увеличение размера файлов в группе выполняется циклическим способом. Например, пусть группа состоит из 3 файлов, разрешенных для автоматического увеличения. Когда все файлы в группе заполнены, MS SQL Server увеличивает размер первого файла на установленный для него шаг приращения (в процентах или мегабайтах). Все новые данные будут записываться в первый файл, пока он снова не заполнится. После этого сервер увеличивает размер второго файла и записывает данные в него. После того как третий файл также будет увеличен и заполнен, сервер снова увеличит первый файл.

Использование групп файлов не является обязательным требованием при проектировании базы данных. Если в базе данных не создаются группы файлов, то все объекты располагаются в основной группе файлов, создаваемой по умолчанию. Если в базе данных был создан единственный файл, то он будет содержать все данные.

Для любой группы файлов, включая основную, может быть установлен режим "только для чтения"(read only). В этом режиме никакие изменения, добавления и удаления данных в файлах не разрешаются. Пользователи могут только считывать информацию, что обеспечивает защиту от несанкционированного изменения данных. Если для основной группы файлов установлен режим “только для чтения”, то в базу данных не могут быть добавлены никакие новые объекты, нельзя также удалять или изменять ранее созданные объекты. Резервная копия группы файлов, помеченной “только для чтения”, может быть восстановлена на другом сервере без необходимости восстановления журнала транзакций.

Использование возможности отсоединения базы данных совместно с установкой режима ”только для чтения” позволяет безболезненно переносить базу данных на множество серверов.

8.3. ОРГАНИЗАЦИЯ ФАЙЛОВ БАЗЫ ДАННЫХ

Основная единица хранения данных на уровне файла базы данных - это страница (page). Страницы могут объединяться в группы страниц - экстенты. В виртуальной системе хранения MS SQL Server страница представляет собой блок фиксированной длины, обрабатываемый при дисковых операциях ввода/вывода как единое целое. Поэтому размер страницы определяет максимальный размер некоторых типов данных. Если пользователь обращается к одной строке данных, с диска будет считана вся страница, даже если в ней помимо требуемой строки записано еще 20 других строк. MS SQL Server выделяет место в базе данных страницами. Начиная с версии 7.0, размер страницы в MS SQL Server увеличен до 8 Кбайт. Соответственно, в одном мегабайте содержится 128 страниц (в предыдущих версиях SQL Server размер страницы составлял 2 Кбайт). При этом файлы журнала транзакций не имеют страниц и экстентов. Они содержат только последовательность записей транзакций, выполняемых в базе данных.

Каждая страница имеет объем 8192 байта. Первые 96 байт страницы отводятся под заголовок, в котором хранится системная информация о типе страницы, объеме свободного места на странице и идентификационном номере объекта (таблицы или индекса), являющегося владельцем страницы (ID object owner). Имеется шесть следующих типов страниц:

Data - в страницах этого типа хранятся данные, исключая данные типа text, ntext и image.

Index - страницы этого типа используются для хранения информации об индексах таблиц и представлений.

Text/Image - в страницах этого типа хранятся данные типа text, ntext и image. Одна страница позволяет хранить в колонках типа text, ntext и image данные объемом до 8060 байт. Данные этого типа могут занимать более одной страницы.

Global Allocation Map - этот тип страниц, как и последующие, является чисто служебным и используется для хранения информации об использовании экстентов (групп страниц).

Page Free Space - в страницах этого типа хранится служебная информация о свободном пространстве на страницах.

Index Allocation Map – служебный тип страниц для хранения информации об экстентах, используемых таблицами или индексами.

Страницы типа Data содержат информацию о данных, исключая данные типа text, ntext и image. Для каждой строки данных таблицы на странице выделяется фиксированное пространство, куда заносятся данные из колонок строки. Строки располагаются на странице последовательно, сразу же после заголовка, в конце каждой страницы имеется таблица смещений строк (row offset table), в которой указывается смещение начала каждой строки относительно начала страницы. Смещения строк располагаются в

порядке, обратном порядку размещения строк на странице, то есть последнее значение в таблице смещений соответствует первой строке на странице, предпоследнее значение - второй строке и т. д. Страница является минимальным блоком, который сервер выделяет для хранения данных.

Для более эффективного управления страницами MS SQL Server использует группы страниц или экстенты (extent). Экстент является базовым блоком, используемым для хранения данных таблиц и индексов, то есть экстент существует только для страниц типа Data, Index и Text/Image. Каждый экстент содержит 8 страниц и занимает 64 Кбайт.

Каждая страница имеет своего владельца (таблицу, индекс и т. д.). В зависимости от того, принадлежат ли все страницы одному владельцу, используется один из двух нижеописанных типов экстентов.

Mixed - экстенты этого типа могут содержать страницы, принадлежащие разным владельцам. Это позволяет сократить расход свободного пространства. Например, если в таблице содержится всего несколько строк и они занимают не более половины страницы, было бы нецелесообразно выделять для их хранения целый экстент, так как 74 % выделенного пространства было бы не использовано. Первоначально для таблицы или индекса выделяется экстент типа Mixed.

Uniform экстент - всегда содержит страницы, являющиеся частью одного объекта. Когда для хранения данных одного владельца выделено 8 страниц в различных экстентах типа Mixed, они автоматически объединяются в экстент типа Uniform. В результате данные в таблице становятся более упорядоченными, что позволяет оптимизировать операции дискового ввода/вывода, ускоряет операции поиска и массивного обновления.

MS SQL Server использует страницы типа Global Allocation Map для хранения информации о свободных экстентах. При этом используются два типа битовых полей:

- Global Allocation Map (GAM). Поля этого типа хранят информацию о том, используется конкретный экстент для хранения данных или не используется. Каждый бит в GAM соответствует одному экстенту. Если бит установлен в 1, то экстент свободен (пуст) и может быть выделен. Если бит установлен в 0, то экстент содержит какие-то данные. Одна страница GAM может хранить информацию об использовании 64 000 экстентов, что соответствует примерно 4 Гбайт данных.

- Shared Global Allocation Map (SGAM). Подобно GAM, одна страница типа SGAM может описать 64 000 экстентов или около 4 Гбайт данных. Каждый бит страницы соответствует одному экстенту. Если бит для конкретного экстента установлен в 1, то этот экстент имеет тип Mixed и как минимум одна страница в этом экстенте свободна и может быть использована. Если бит установлен в 0, то либо экстент имеет тип Uniform (а значит полностью занят), либо все страницы экстента типа Mixed заняты.

Использование битовых полей GAM и SGAM позволяет быстро найти свободную страницу в экстенте типа Mixed и упрощает управление информацией об использовании экстентов. Кроме того, использование двух самостоятельных полей позволило уменьшить объем управляющей информации и повысить производительность. Поля GAM и SGAM используются совместно, дополняя друг друга. Для каждого участка длиной в 4 Гбайт используется по одному полю GAM и SGAM. В табл.8.1 приведены значения полей GAM и SGAM при различных состояниях экстента.

Таблица 8.1

Использование полей GAM и SGAM

Состояние экстента	Значение в поле GAM	Значение в поле SGAM
Неиспользуемый экстент	1	безразлично
Экстент типа Uniform или полностью занятый экстент типа Mixed	0	0
Экстент типа Mixed, имеющий свободные страницы	0	1

Если экстент типа Mixed был заполнен полностью, но после удаления строк одна страница освободилась, то достаточно установить в "1" соответствующий бит в поле SGAM. Если же экстент освобождается полностью, то в поле GAM устанавливается в "1" бит, соответствующий нужному экстенту. Для выделения экстента типа Mixed сервер ищет в поле GAM бит, установленный в "1", и устанавливает его в "0", затем в поле SGAM устанавливает соответствующий бит в "1" (если не все страницы экстента заняты полностью) или в "0" (все страницы экстента заняты). Для выделения экстента типа Uniform сервер ищет в поле GAM бит, установленный в "1", и устанавливает его в "0". В поле SGAM соответствующий бит устанавливается в "0".

На основании сказанного можно сделать вывод, что битовые поля GAM и SGAM используются только для быстрого выделения страниц. Хранимая в них информация не дает ответа, какому объекту принадлежит экстент.

Битовые поля GAM и SGAM содержат информацию только о том, используется ли конкретная страница. Для хранения информации о количестве свободного пространства на страницах типа Data, Index и Text/Image используются страницы типа Page Free Space (PFS). Каждая такая страница содержит информацию о свободном пространстве на 8000 страниц. Каждый байт страницы PFS соответствует отдельной странице и может описывать следующие состояния: страница не используется (0 % занято),

страница заполнена на 1-50 %, страница заполнена на 51-80 %, страница заполнена на 81-95 %, страница заполнена на 96-100 %. Страницы PFS используются для быстрого поиска страниц, имеющих необходимое количество свободного пространства.

Предположим, что необходимо записать 17 строк данных. Имеется последовательный набор страниц, имеющих свободное пространство для хранения, соответственно, 7, 4, 9, 23 и 3 строк данных. Если записывать строки последовательно, начиная с первой свободной страницы, то чтобы сохранить 17 строк, понадобится использовать три страницы (7 + 4 + 9). В худшем случае для этого понадобится выполнить три операции дискового ввода/вывода (если страницы не копированы). Если же записывать данные сразу в четвертую страницу, позволяющую сохранить 23 строки, то понадобится всего одна операция дискового ввода/вывода. Если для поиска страницы, имеющей достаточно свободного пространства, последовательно считывать с диска все страницы, это может занять много времени. При использовании страниц типа Page Free Space поиск свободных страниц становится очень эффективным, так как сервер перебирает только страницы PFS.

Страницы PFS облегчают поиск свободных страниц, но они не имеют информации о том, какому владельцу принадлежит та или иная страница. Для отслеживания информации о страницах, принадлежащих одному объекту-владельцу, используются страницы типа Index Allocation Map (IAM). Для каждого владельца создается минимум одна страница IAM. Каждая страница IAM имеет заголовок, в котором хранится информация об идентификационном номере объекта - владельца страниц, а также смещении первого экстента, с которого начинается область IAM. Страницы IAM создаются при необходимости и могут располагаться в произвольном месте файла. Структура IAM представлена битовым полем, каждый из битов которого описывает единственный экстент. Если бит установлен в "1", то соответствующий экстент содержит страницы, принадлежащие описанному в IAM владельцу. Если же бит установлен в "0", то данный экстент не имеет страниц, принадлежащих нужному владельцу.

Каждая страница IAM покрывает диапазон размером 64 000 экстентов, начиная с указанного в заголовке первого экстента. Если объект-владелец имеет принадлежащие ему экстенты за пределами области, охватываемой одной страницей IAM, то создаются дополнительные страницы IAM, которые связываются в цепочку. Каждая страница IAM в цепочке, кроме последней страницы, имеет ссылку на следующую в цепочке страницу IAM. Если владелец имеет принадлежащие ему экстенты в разных файлах базы данных, то в каждом таком файле создается минимум одна страница IAM. Страницы в разных файлах также связываются в цепочку.

При необходимости вставить несколько строк в таблицу, MS SQL Server просматривает страницы IAM для поиска экстентов, содержащих

принадлежащие этой таблице страницы. Когда экстент найден, просматриваются соответствующие страницы PFS для поиска свободных страниц. Если ни одна из страниц не имеет достаточного места для вставки требуемого количества строк, то для объекта (в данном случае - для таблицы) выделяется новый экстент. Причем, если база данных имеет несколько файлов, экстенты выделяются в каждом файле в равных пропорциях. Например, если база данных содержит два файла и второй файл в три раза больше первого файла, то при выделении экстента в первом файле во втором файле будет выделено три экстента. Это сделано для того, чтобы файлы базы данных заполнялись равномерно.

Как видно из всего сказанного, MS SQL Server имеет достаточно эффективную структуру хранения данных, обеспечивающую высокую скорость операций создания, поиска и удаления строк данных. Страницы в редких случаях бывают заполнены на 100 %. MS SQL Server специально резервирует свободное место на странице для ускорения операций вставки строк.

Каждый файл данных базы данных имеет приблизительно одинаковую физическую структуру. Все страницы в файле нумеруются последовательно, начиная с 0 для самой первой страницы. Каждый файл базы данных имеет уникальный идентификационный номер (ID). Комбинация номера страницы и идентификационного номера файла позволяет однозначно идентифицировать страницу в базе данных. В начале файла располагается страница заголовка файла (Page 0), в которой хранятся атрибуты файла. Сразу же за ней располагается страница PFS (Page 1), в которой хранится информация об использовании страниц в экстенте. Затем располагаются страницы битовых полей GAM (Page 2) и SGAM (Page 3). Страницы IAM создаются в любом месте файла. На каждые 8 000 страниц данных создается новая страница PFS. Страницы GAM и SGAM создаются на каждые 64 000 экстентов или на каждые 512 000 страниц.

8.4. ЛОГИЧЕСКАЯ СТРУКТУРА БАЗЫ ДАННЫХ

После того как база данных создана физически, то есть созданы и размещены файлы и группы файлов, можно приступать к проектированию и созданию логической структуры базы данных. Логическая структура включает в себя таблицы, представления, хранимые процедуры, триггеры и другие объекты, а также пользователей и роли базы данных, включая выданные им права доступа. Планирование логической структуры базы данных является более сложной задачей, чем планирование физической структуры.

Каждая база данных имеет несколько системных таблиц, в которых хранится описание всех объектов, созданных в этой базе данных. Каждый объект описывается в нескольких таблицах. Для создания и управления

объектами разрешается напрямую обращаться к тем или иным системным таблицам, но разработчик - фирма Microsoft не рекомендует прибегать к подобным методам, а предлагает пользоваться стандартными командами и хранимыми процедурами. Помимо обеспечения надежности, такой подход гарантирует, что программа, написанная для одной версии MS SQL Server, будет корректно работать и с последующими версиями. Например, по сравнению с MS SQL Server 6.x в системные таблицы следующих версий MS SQL Server были внесены значительные изменения. В некоторых колонках таблиц теперь используется стандарт Unicode. Программы MS SQL Server 6.x, обращавшиеся напрямую к системным таблицам, не будут корректно работать с MS SQL Server 7.0. В то же время программы MS SQL Server 6.x, использующие команды Transact-SQL и хранимые процедуры, будут успешно работать и в MS SQL Server 7.0.

Полное имя объекта базы данных содержит четыре идентификатора: **имя сервера, имя базы данных, имя владельца и собственно имя объекта.** Можно не указывать все четыре части имени. При отсутствии имени сервера предполагается, что пользователь обращается к локальному серверу. Если опускается имя базы данных, то сервер предполагает, что используется текущая база данных. Если пользователь не обращался к другим базам данных, то текущей будет база данных по умолчанию, установленная в свойствах этого пользователя. Если опускается имя владельца объекта, то сервер предполагает, что объект принадлежит пользователю, выполнившему обращение. Если такой объект не найден, то сервер ищет объект с указанным именем, принадлежащий владельцу базы данных (dbo). Если владелец базы данных не имеет объекта с требуемым именем, то выдается сообщение об ошибке. При обращении к объектам, не принадлежащих владельцу базы данных или работающему пользователю, необходимо всегда указывать имя владельца объекта.

Для ссылки на колонку таблицы или представления после имени объекта добавляется имя колонки.

Когда пользователь создает объект и указывает только имя объекта, то этот объект создается на локальном сервере в текущей базе данных и принадлежит он пользователю, установившему соединение.

В MS SQL Server имеются следующие виды объектов, к которым могут обращаться пользователи: Tables - таблицы, Views - представления, Indexes - индексы, Triggers - триггеры, User-defined data types - пользовательские типы данных, Defaults - умолчания, Rules - правила, Constraints - ограничения целостности, Stored procedures - хранимые процедуры, Keys – ключи для ограничения целостности, User-defined functions – определенные пользователем функции. К этим объектам базы данных пользователи могут обращаться для выполнения задач обработки данных. Администратор базы данных или владелец объекта может ограничить доступ пользователей к объектам базы данных, разрешая или запрещая пользователям выполнять те

или иные действия. Например, для таблиц можно устанавливать доступ на чтение, изменение, добавление или удаление данных. В Transact-SQL существуют специальные команды для создания этих объектов.

База данных MS SQL Server также имеет несколько объектов, выполняющих административные функции: Database User - пользователи базы данных, Database Role - роли базы данных, Diagram - диаграммы, Publications - публикации, Pull Subscription - репликации с использованием подписки по запросу, Full-Text Catalog - полнотекстовый каталог. К этим объектам не устанавливаются права доступа. Для управления перечисленными объектами пользователя необходимо включить в ту или иную фиксированную роль базы данных, что даст ему необходимые права доступа.

Папки Publications и Pull Subscription предназначены для управления репликацией. Эти папки создаются автоматически после того, как база данных будет разрешена для публикации или подписки.

Папка Full-Text Catalog используется для полнотекстовых индексов. Эта папка появится в структуре базы данных после того, как будет инсталлирована поддержка полнотекстового поиска, устанавливаемого в виде самостоятельной службы Full-Text Search Service. Эта служба устанавливается только на MS SQL Server, работающем под управлением операционной системы Windows NT Server 4.0. и выше.

Диаграммы предназначены для облегчения понимания структуры базы данных и связей между таблицами. На самом деле диаграмма не представляет собой объекта базы данных - она существует только на уровне Enterprise Manager, который сохраняет описание диаграммы в таблице dtproperties текущей базы данных. Поэтому в Transact-SQL нет команд или хранимых процедур для управления диаграммами. Для управления ими используется только Enterprise Manager, который считывает определение диаграммы из таблицы dtproperties и сам ее генерирует. Диаграммы позволяют легко управлять базой данных и ее объектами. Используя диаграмму, можно управлять таблицами и их колонками, связями между таблицами, индексами, ограничениями целостности, хранимыми процедурами и триггерами.

8.5. ОСНОВНЫЕ ОБЪЕКТЫ ПОЛЬЗОВАТЕЛЯ

База данных является базовым элементом MS SQL Server и своего рода контейнером, в котором располагаются объекты и данные. Любой объект должен принадлежать базе данных. Каждая база данных имеет свою систему безопасности, связанную с системой безопасности MS SQL Server, причем MS SQL Server является системой, удовлетворяющей самым жестким требованиям к безопасности информации, так как плохо продуманная система безопасности не сможет предотвратить кражу или повреждение

данных. Любой пользователь при обращении к серверу работает с какой-то базой данных. Каждой базе данных сопоставлен пользователь, который является ее владельцем (database owner). Этот пользователь имеет имя "dbo" и ему предоставлены максимальные права в базе данных.

Создание базы данных можно выполнить командой Transact-SQL:

CREATE DATABASE <имя_базы_ данных>

<характеристики_базы_данных>

Создание базы данных и ее объектов можно выполнить с использованием средств Transact-SQL или с помощью графического интерфейса специальной утилиты Enterprise Manager. Enterprise Manager позволяет не только создавать базы данных, но и управлять ими, а также удалять их. Работа с Enterprise Manager не требует знания Transact-SQL, что облегчает работу с базами данных.

Объектом, который содержит все данные, хранящиеся в базе данных, является таблица (Tables). Никакие другие объекты не могут хранить данные, но они могут обращаться к данным в таблице. Для создания таблицы средствами Transact-SQL используется команда CREATE TABLE. Если необходимо изменить какие-то параметры таблицы, используется команда ALTER TABLE. Для удаления таблиц применяется команда DROP TABLE.

Представления (Views) представляют собой виртуальные таблицы, определяемые запросом. Физически представление реализовано в виде SQL-запроса, на основе которого проводится выборка данных из одной или нескольких таблиц или представлений. Для создания представлений средствами Transact-SQL используется команда CREATE VIEW. Для удаления представлений применяется команда DROP VIEW.

Индексы (Indexes) предназначены для ускорения операций поиска данных. Индекс определяется для одной или нескольких колонок, называемых индексированными колонками. Использование индексов может существенно повысить производительность поиска данных. Создание индексов может быть выполнено в процессе создания таблицы или после создания таблицы командой CREATE INDEX. Для удаления индексов применяется команда DROP INDEX.

Хранимые процедуры (Stored Procedures) MS SQL Server представляют собой группу команд Transact-SQL и вызовов других хранимых процедур. Эта группа компилируется и выполняется как единое целое. Каждая хранимая процедура имеет уникальное в пределах базы данных имя, по которому пользователи могут к ней обращаться. Создание хранимых процедур может быть выполнено командой Transact-SQL:

CREATE PROCEDURE <имя_ процедуры>

AS <набор команд Transact-SQL>

Триггерами (Triggers) называется специальный класс хранимых процедур, автоматически запускаемых при добавлении, изменении и удалении данных из таблицы. В зависимости от выполняемых пользователем

действий, приводящих к запуску триггера, они делятся на три категории: триггеры изменения (UPDATE TRIGGER), триггеры вставки (INSERT TRIGGER) и триггеры удаления (DELETE TRIGGER). В последних версиях MS SQL Server возможно определять для таблицы несколько триггеров каждого типа. Создание триггеров может быть выполнено командой Transact-SQL:

CREATE TRIGGER <имя_ триггера>

AS <набор команд Transact-SQL>

Правила (Rules) используются для ограничения значений, хранимых в колонке таблицы или в пользовательском типе данных. Правила оставлены для обеспечения обратной совместимости с предыдущими версиями MS SQL Server. В MS SQL Server 7.0 и 2000 правила встроены в определение таблицы. Тем не менее, иногда бывает необходимо использовать правила как отдельные объекты базы данных. Для колонки таблицы можно определить только одно правило. Создание правила не может выполняться в одном пакете с другими командами Transact-SQL. Для создания правила используется следующая команда Transact-SQL:

CREATE RULE <имя_правила>

AS <логическое_выражение>

Логическое выражение определяет условие, накладываемое на значения. В качестве условия можно использовать любые логические команды, арифметические операции, встроенные функции и предикаты (например, IN, BETWEEN, LIKE). В логическом выражении нельзя ссылаться на колонки таблиц или на любые другие объекты базы данных. Встроенные функции также не должны ссылаться на объекты базы данных. Правило может быть создано только в текущей базе данных. Само по себе правило не выполняет контроля данных. Созданное правило необходимо связать с колонкой таблицы или пользовательским типом данных. Правило не может быть связано с системным типом данных и колонками, имеющими типы данных timestamp, text и image. При связывании правила с колонкой таблицы или пользовательским типом данных следует убедиться, что типы данных в правиле и в объекте привязывания совпадают.

Умолчание (Default) – это значение, которое будет присвоено колонке таблицы при вставке строки, если в команде вставки явно не указано значение для этой колонки. Созданное умолчание само по себе не выполняет никаких действий. Его необходимо связать с какой-нибудь колонкой таблицы или пользовательским типом данных. Тип данных, используемый в умолчании, должен соответствовать типу данных, установленному для колонки, с которой связывается умолчание. Для создания умолчания используется следующая команда:

CREATE DEFAULT <имя_умолчания>

AS <значение_по_умолчанию>

При выборе имени умолчания следует придерживаться установленных

правил именования объектов. Кроме того, имя умолчания должно быть уникально для каждого владельца. Указание имени владельца умолчания при его создании необязательно.

В качестве значения по умолчанию можно использовать не только константы, но и любые математические выражения и встроенные функции. Строки символов и даты должны заключаться в одиночные кавычки. Перед бинарными данными должен использоваться префикс 0x, а перед денежным типом – символ $. Если размер значения по умолчанию превышает размер, заданный для данных в колонке, выполняется усечение значения по умолчанию до требуемого размера.

8.6. ОГРАНИЧЕНИЯ ЦЕЛОСТНОСТИ ДАННЫХ

Ограничения целостности (Constraints) – это механизмы MS SQL Server, обеспечивающие автоматический контроль соответствия данных установленным условиям (или ограничениям), то есть обеспечивающие целостность данных. MS SQL Server обеспечивает автоматическое отслеживание связей между таблицами, не разрешая удалять данные до тех пор, пока не будут удалены все связанные строки из зависимой таблицы.

Ограничения целостности имеют приоритет над триггерами, правилами и значениями по умолчанию. Оптимизатор запросов использует ограничения целостности для построения оптимального плана исполнения запроса. Ограничения целостности могут применяться на уровне колонки или на уровне таблицы. Ограничения целостности, применяемые на уровне колонки, влияют только на значения, вводимые в эту колонку. Если одно ограничение целостности применяется для нескольких колонок, то оно работает на уровне таблицы. Например, если ограничение целостности PRIMARY KEY включает одну колонку, то оно работает на уровне колонки. Если же PRIMARY KEY состоит из нескольких колонок, то это ограничение целостности работает на уровне таблицы.

Имеются пять классов ограничений целостности, различающихся по функциональности и области применения.

Ограничение целостности NULL. Действует на уровне колонки и пользовательского типа данных. Устанавливая для них ограничение целостности NULL или NOT NULL, можно, соответственно, разрешать или запрещать хранение значений NULL.

Ограничение целостности CHECK (проверка). Это ограничение целостности действует на уровне колонки и ограничивает диапазон значений, которые могут быть сохранены в колонке. При определении этого ограничения целостности для колонки указывается логическое условие для вводимых данных. При вводе или изменении данных вводимое значение подставляется в условие. Если в результате проверки возвращается значение «истина» (TRUE), то изменения данных принимаются. Если же проверка

возвращает «ложь» (FALSE), то изменения данных отвергаются и генерируется сообщение об ошибке. Для одной колонки можно создать несколько ограничений целостности CHECK.

Ограничение целостности UNIQUE (уникальность). Действует на уровне колонки и гарантирует уникальность значения в колонке. В таблице не может быть двух строк, имеющих одинаковое значение в колонке с установленным ограничением целостности UNIQUE. В отличие от первичного ключа, ограничение целостности UNIQUE допускает хранение значений NULL.

Ограничение целостности PRIMARY KEY (первичный ключ). Действует на уровне колонки или таблицы. Первичный ключ состоит из одной или нескольких колонок, уникально идентифицирующих строку в пределах таблицы. Если для генерации первичного ключа используется единственная колонка, то для этой колонки должно быть установлено ограничение целостности UNIQUE для гарантии уникальности значений в колонке. Ни для одной из колонок, включенных в первичный ключ, не должно быть установлено свойство NULL. В таблице может быть создан только один первичный ключ.

Ограничение целостности FOREIGN KEY (внешний ключ). Создается на уровне таблицы. Внешний ключ связывается с первичным ключом в другой таблице. Таблица, в которой определен внешний ключ, называется зависимой, а таблица с первичным ключом – главной. В зависимую таблицу нельзя вставить строку, если внешний ключ не имеет соответствующего значения в главной таблице. Кроме того, из главной таблицы нельзя удалить строку, если с ней связана хотя бы одна строка в зависимой таблице. Перед удалением строки из главной таблицы необходимо предварительно удалить все строки из зависимой таблицы. Для этого можно написать специальную хранимую процедуру или создать триггер.

База данных содержит таблицы (каждому объекту предметной области соответствует своя таблица) с установленными связями – ограничениями целостности Primary key (первичный ключ) – Foreign key (внешний ключ), которые через эти связи осуществляют взаимодействие друг с другом, и организуют работу целиком с базой данных.

8.7. АДМИНИСТРИРОВАНИЕ MS SQL Server

Процесс администрирования MS SQL Server распадается на две задачи: администрирование сервера и администрирование баз данных. Каждая база данных может иметь своего администратора. Должности администратора сервера и администратора баз данных обычно объединяются в лице одного человека. Кроме того, этот же человек может заниматься администрированием сети организации. Такой подход позволяет одному человеку быть в курсе всех проблем и оптимально управлять объединенной

системой безопасности MS SQL Server и Windows NT.

В обязанности администратора сервера входят:

Установка и обновление версий сервера. Для установки сервера необходимо скоординировать действия с системным администратором. Администратор должен следить за выходом новых версий и обновлений MS SQL Server. В обязанности администратора также входит установка сервисных пакетов (Service Pack) и исправлений (Patches).

Запуск и остановка сервера. Никто, кроме системного администратора, не имеет прав на запуск и остановку процессов MS SQL Server. Остановка сервера может понадобиться для проведения профилактики или модернизации.

Управление настройками связанных серверов, процедурами, стартующими при запуске MS SQL Server. Если в сети имеется несколько серверов SQL Server, которые обращаются друг к другу, то администратор должен обеспечить их нормальную работу.

Выполнение конфигурирования и настройки сервера. Администратор сервера должен выполнять конфигурирование параметров запуска и работы MS SQL Server; конфигурирование памяти, выделяемой для работы сервера, и потоков Windows NT; настройку пользовательских соединений, сетевых библиотек и др.

Управление учетными записями MS SQL Server и Windows NT для разрешения или запрещения доступа к серверу.

Выполнение резервного копирования системных и пользовательских баз данных. Администратор сервера должен отслеживать изменения в системных и пользовательских базах данных и своевременно выполнять резервное копирование данных. Исходя из обстоятельств, администратор должен избрать соответствующий способ копирования.

Управление процессами MS SQL Server. Администратор должен выполнять настройку учетных записей Windows NT, под которыми работают службы сервера, а также следить, чтобы эти учетные записи имели необходимые права доступа в сети.

Создание и модифицирование баз данных. Администратор должен следить за использованием базами данных дискового пространства, а при их создании планировать наиболее оптимальное размещение файлов и индексов на дисках.

Управление файлами MS SQL Server. Администратор должен просматривать файлы журналов ошибок, выполнять присоединение и отсоединение баз данных, устанавливать права доступа к файлам MS SQL Server.

Выполнение конфигурирования устройств резервного копирования. Администратор должен планировать использование устройств резервного копирования в системах с несколькими устройствами, вести учет лент с резервными копиями и составлять расписание повторного использования

лент.

Выполнение мониторинга и оптимизации работы сервера, то есть обеспечение сбора данных обо всех аспектах работы сервера, устранение слабых мест в конфигурации баз данных, оптимизация операций ввода-вывода.

Администратор баз данных SQL Server выполняет следующие обязанности:

Создает объекты баз данных. Администратор баз данных осуществляет реализацию, а иногда и разработку базы данных, имеет права на создание таблиц, представлений, правил, умолчаний и т. д.

Создает пользователей и роли баз данных. Администратор баз данных решает, какие пользователи имеют доступ к базе данных, а каким пользователям необходимо запретить его. Он разрабатывает структуру системы безопасности в базе данных, планирует использование ролей для облегчения администрирования.

Разрешает пользователям доступ к объектам баз данных. Администратор баз данных реализует разработанную систему безопасности через предоставление соответствующих прав разрешения или запрещения выполнения команд Transact-SQL, хранимых процедур и доступа к объектам базы данных.

Выполняет резервное копирование пользовательских данных. Администратор баз данных разрабатывает стратегию резервного копирования базы данных с использованием полной либо разностной копии и копии журнала транзакций.

Следит за целостностью и достоверностью данных. В задачи администратора баз данных входит отслеживание целостности данных с использованием правил, умолчаний, первичных и внешних ключей, уникальных колонок и т. д.

Создает роли для приложений. Если доступ к базе данных осуществляется через специализированное приложение, то администратор может предоставить доступ к данным с использованием роли приложений, предоставив только ей необходимые права и ограничив возможность пользователей напрямую изменять или просматривать данные.

Производит мониторинг выполнения запросов. В случае, если пользователи сообщают, что их запросы исполняются слишком медленно, администратор должен выполнять мониторинг запросов для выяснения причины. При необходимости администратор может создать дополнительные индексы либо перенести «тяжелые» данные на отдельный диск.

Последние версии MS SQL Server поддерживают различные типы резервного копирования (на основе создания архивов), позволяя администратору очень гибко управлять данным процессом. Существуют: полная копия, разностная копия, копия журнала транзакций, резервное копирование файлов и групп файлов.

Полная копия базы данных (database backup) является стандартным типом резервного копирования. Этот тип резервирования предполагает полное копирование всей информации, имеющейся в базе данных. В качестве приемника данных может выступать либо обычный файл, либо специальное устройство резервного копирования, такое как стример, магнитооптический диск или zip-диск.

Полное копирование данных имеет свои недостатки и преимущества. К последним можно отнести то, что для приведения системы в рабочее состояние достаточно восстановить лишь один архив, так как вся необходимая информация содержится в единственной резервной копии. К недостаткам же относится длительное время создания архива даже в случае внесения незначительных изменений в базу данных. MS SQL Server автоматически отслеживает изменения в уже скопированных данных и обновляет их в архиве. В результате к моменту завершения резервного копирования состояние данных в архиве будет соответствовать информации в исходной базе данных. Это гарантирует целостность и достоверность данных.

Разностное (дифференциальное) резервное копирование (differential database backup) было разработано с целью уменьшения времени, необходимого для получения копии базы данных. В основе дифференциальной копии лежит отслеживание изменений, вносимых пользователями в базу данных. Создание дифференциальной копии состоит из двух этапов: создание полной копии данных, создание собственно дифференциальной копии.

Полная копия базы данных является отправной точкой, начиная с которой система может отслеживать изменения. Изменения отслеживаются на уровне страниц. Каждая страница имеет флаг архивирования, который сбрасывается при создании полной копии и устанавливается, если данные на странице были изменены. Это можно сравнить с архивированием файлов. Дифференциальная копия создается только для страниц, которые были изменены.

Резервное копирование журнала транзакции (transaction log backup) позволяет сохранять информацию обо всех транзакциях, выполненных в базе данных. В итоге резервная копия содержит непрерывную цепочку изменений, которые претерпели данные со времени последнего архивирования. Такая цепочка позволяет восстановить систему в состоянии, в котором она была в любой момент времени, и этот момент отображен в резервной копии. При восстановлении резервной копии журнала транзакций MS SQL Server последовательно выполняет все изменения, выполненные пользователями в базе данных.

При резервном копировании журнала транзакций MS SQL Server автоматически выполняет усечение журнала транзакций после завершения создания архива. То есть размер журнала транзакций будет автоматически

поддерживаться на нормальном уровне.

Резервное копирование файлов и групп файлов - последний из типов резервного копирования MS SQL Server. База данных состоит из одного или более файлов данных и одного или более файлов журнала транзакций. Ни полное, ни резервное копирование не позволяют архивировать только часть данных, например, только данные без индексов или только столбцы типа image, text и ntext. Резервная копия журнала транзакций отображает лишь операции изменения данных. А резервное копирование файлов и групп файлов позволяет выполнить частичное архивирование.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что собой представляет MS SQL Server?
2. Какие основные компоненты существуют в MS SQL Server?
3. Какие системные базы данных используются в MS SQL Server?
4. Для чего используются системные базы данных MS SQL Server
5. Какие файлы обязательны для базы данных MS SQL Server?
6. Какие типы файлов используются в составе базы данных MS SQL Server?
7. Какие группы файлов используются в составе базы данных MS SQL Server?
8. Что собой представляют страницы и какие их типы используются в файле базы данных MS SQL Server?
9. Что собой представляют экстенты и группы экстентов, их типы в MS SQL Server?
10. Какой механизм используется в MS SQL Server для поиска и выделения свободных страниц?
11. Что собой представляет логическая структура базы данных в MS SQL Server?
12. Какие пользовательские объекты существуют в MS SQL Server и для чего они предназначены?
13. Какие команды Transact-SQL используются для создания пользовательских объектов MS SQL Server?
14. Что собой представляют ограничения целостности в MS SQL Server?
15. Какие классы ограничений целостности используются в MS SQL Server?
16. Какие обязанности у администратора сервера MS SQL Server?
17. Какие обязанности у администратора базы данных MS SQL Server?
18. Что такое резервное копирование и для чего оно используется?
19. Какие типы резервного копирования возможны в MS SQL Server?

Глава 9. ВИЗУАЛЬНОЕ ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ

9.1. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Объектно-ориентированное программирование (ООП) основывается на понятии объекта, в то время как традиционное **процедурное программирование** основывается на процедурах. Понятие объекта, являющееся основным в ООП, позволяет очень эффективно использовать модульный принцип составления программ и сделало возможным визуальное программирование и проектирование БД и их приложений.

Основная особенность объектно-ориентированного языка программирования состоит в том, что программа организуется по объектам программирования. Объекты содержат инструкции (называемые методами) и данные (называемые свойствами), которые определяют поведение объекта. **Метод** – это функция или процедура, которая управляет работой объекта. Внешний вид и работа объекта определяются характеристиками, которые являются свойствами объекта. **Свойства** объекта могут быть изменяемыми и неизменяемыми, но их значения всегда можно прочитать.

Объект можно определить как совокупность программно связанных методов и свойств, выполняющих одну функционально связанную задачу и представляющих собой единое целое. В процессе работы программы объект выполняет свое стандартное поведение и в случае необходимости изменяет данные для отражения влияния назначенного ему действия. Объектом можно назвать любой предмет: окно, поле ввода, кнопку и т.д.

При объектно-ориентированном программировании на первый план выступают данные, несущие информацию об объектах и осуществляющие обмен информацией между объектами. В модели электронной схемы в роли управляющих данных могут быть электрические сигналы, в модели экономики - количество товаров на рынке, в модели многооконной операционной оболочки - все происходящие в системе события. В ООП управляющие данные принято называть **сообщениями.**

В ООП используется событийная модель управления. **Событие** представляет собой некоторое действие, которое активизирует стандартную реакцию объекта. Событие обрабатывается методом. В качестве события может рассматриваться нажатие кнопки мыши, выбор пункта меню, открытие таблицы и т.д. Порядок выполнения действий определяется прежде всего событиями, возникающими в системе, и реакцией на них объектов. Ниже приведена последовательность выполнения действий в системе при условии предварительно запрограммированных реакций системы.

1. В системе возникает событие (нажатие клавиши, изменение значения поля и т.д.).

2. Определяется связанный с этим событием объект (поле таблицы, кнопка, форма и т.д.).

3. Вызывается соответствующий событию метод объекта, который содержит действия по обработке возникшего события.

Помимо ключевого понятия объект в объектно-ориентированных языках программирования существует и широко применяется понятие **класс.** Объект можно создать программно или визуально на основе класса. Классы и объекты тесно связаны между собой, но эти понятия не тождественны. Класс содержит информацию о том, как должен выглядеть объект и определяет выполняемые им действия. Объект является экземпляром класса, который наследует характеристики класса.

Выделяют три основные характеристики объектно-ориентированного программирования: наследование, инкапсуляция, полиморфизм.

Наследование. Все объекты создаются на основе классов и наследуют свойства и методы классов. Классы могут, в свою очередь, создаваться на основе других классов. Такие классы называются подклассами. Они наследуют все свойства и методы своих родительских классов. Дополнительно можно определить для подкласса новые методы и свойства. Подклассы позволяют сократить объем программирования и максимально использовать предыдущий опыт работы. Кроме того, изменение свойств и методов родительского класса отслеживается в подклассах, созданных на основе этого класса, а также в объектах, созданных на основе подклассов. Таким образом можно без особых затрат изменить характеристики всего приложения.

Инкапсуляция. Методы и свойства, объединенные в объекте, не могут существовать вне его. При копировании объект копируется как единое целое. Принцип объединения в единое целое данных и программного кода, описывающего поведенческие свойства объекта, известен под термином инкапсуляция. При этом можно пользоваться как специальным типом данных, введенным в языки программирования для реализации технологии ООП (например, в C++, Simula, Prolog), так и обычными синтаксическими конструкциями.

Полиморфизм. При традиционном процедурном программировании имя вызываемой подпрограммы или функции однозначно определяет выполняемый код. В объектно-ориентированном программировании можно использовать одни и те же имена методов для выполнения совершенно разных действий. Например, метод open() может использоваться как для открытия формы, так и открытия базы данных. Выполняемое действие зависит от типа объекта, к которому применяется данный метод. Такой подход значительно упрощает программирование, позволяя использовать одинаковые смысловые наименования для выполнения разнообразных действий.

Классом называют **шаблон**, который описывает методы и свойства, используемые для определённого типа объектов. Классы имеют иерархическую структуру. На основе класса системы (базового класса) можно создать класс разработчика приложений, который, в свою очередь, может быть родительским классом для подклассов пользователя. Классы, как базовые, так и созданные, хранятся в библиотеках классов. При создании объектов можно использовать базовые классы системы, а также созданные новые специальные классы и подклассы.

Существуют разные типы классов. Большинство базовых классов являются видимыми, некоторые классы используются для объединения объектов и не отображаются в форме. Кроме того, одни классы допускают вложение других классов, другие - не допускают. Ниже дается краткая характеристика основным типам классов.

Классы - элементы управления, позволяют создавать объекты, с которыми можно обращаться только как с единым целым (линия, кнопка, список, таймер и другие элементы управления).

Классы - контейнеры, позволяют создавать объекты, которые могут содержать внутри себя другие объекты, позволяя к тому же манипулировать этими внутренними объектами. Например, контейнер - таблица содержит столбцы и заголовки столбцов, контейнер - форма содержит различные элементы управления.

Различают **визуальные** классы - видимые на экране и составляющие основу пользовательского интерфейса (например, кнопка, таблица, форма и т.д.) и **не визуальные**, которые видны только в процессе проектирования (например, таймер).

Едва ли можно найти предметную область, объекты которой нельзя было бы описать в терминах иерархических структур или классификаций. Классификации удобны тем, что позволяют разнести свойства объектов по иерархическим уровням, т.е. структурировать их. При этом реализуется механизм наследования - свойства старшего класса автоматически переносятся на дочерний уровень.

В объектно-ориентированных языках программирования, где механизм наследования реализуется при помощи специальных синтаксических конструкций, дочерние классы автоматически наследуют все свойства объекта, в том числе и функции, описывающие его поведение.

Для создания объектов на основе классов применяют специальные средства визуального программирования. Визуальное программирование донесло основные положения объектно-ориентированного программирования до конечного пользователя. Вместо того, чтобы писать многие строки кодов, определяющие поведение объекта, пользуются прототипами необходимых объектов - классами. Средства визуального программирования предоставляют соответствующие классы для создаваемых объектов, которые можно настроить по своему усмотрению. Например,

практически все визуально программируемые языки содержат прототипы кнопки, которая утопает и всплывает при нажатии на нее пользователем. Это дает возможность не писать многие строки кода для создания кнопки, а воспользоваться уже готовым прототипом кнопки.

Языки программирования современных СУБД являются объектно-ориентированными, визуально программируемыми языками, управляемыми по событиям, и в полной мере соответствуют требованиям, предъявляемым к средствам проектирования приложений. Можно перечислить все элементы интерфейса современных СУБД и большинство из них окажутся объектами. Специальные инструментальные средства СУБД позволяют **визуально проектировать** базы данных и другие компоненты приложений баз данных.

9.2. ОБЪЕКТЫ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ И ВИЗУАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В реляционных СУБД данные представляются в табличном виде. Данные можно организовывать либо в автономные таблицы, либо в таблицы базы данных (в зависимости от требований конкретной СУБД). Таблицы и базы данных - это основные объекты СУБД. В общем случае базы данных представляют собой совокупность логически связанных таблиц данных. Поэтому отношения между таблицами также хранятся в базе данных. Для ускорения поиска данных в таблице используются индексы, входящие в состав баз данных. В большинстве СУБД реализован механизм триггеров. Триггеры хранятся в базе данных и позволяют централизованно обрабатывать события, возникающие при любых изменениях в базе данных. Можно также создавать хранимые процедуры, которые являются частью базы данных и используются при описании таблиц для проверки введённых данных, определения значения по умолчанию и т.п.

Помимо таблиц для отображения и (или) редактирования данных в приложениях баз данных широко используются и другие объекты - компоненты СУБД: представления, формы, отчёты, запросы, программы. Эти объекты, в отличие от других объектов (кнопок, переключателей, списков и т.д.), являются самостоятельными компонентами СУБД наряду с таблицами данных. Их создание и использование выполняется независимо от других объектов или в зависимости только от таблиц данных. Формы, отчёты, запросы, программы могут храниться в одном файле с таблицами (например, в СУБД Access) или в отдельных файлах (например, в СУБД Visual FoxPro). Во втором случае, в зависимости от содержащегося в файле объекта, СУБД автоматически присваивает каждому файлу расширение, которое помогает в идентификации объекта.

Черезвычайно удобным и полезным средством доступа к базе данных являются **представления данных** (виды, просмотры данных). Представления

данных позволяют объединять данные таблиц и отображать их в более удобном виде. Можно выбрать только интересующие поля таблиц, объединить несколько полей в одно поле, вычислить итоговые значения, задать новые имена полей таблицы и т.д.

Как правило, количество представлений в базе данных намного превосходит количество таблиц. По мере эксплуатации базы данных их число непрерывно растёт. Во многих информационных системах доступ к данным (включая просмотр, добавление и редактирование) осуществляется только с помощью представлений данных. Такой подход позволяет осуществить гибкое управление доступом к информации. Редактирование данных, включённых в представление, возможно только при определённых условиях.

При выборке данных из представлений в формы, отчёты, запросы, а также в программах применяются те же правила, что и для таблиц.

В приложениях баз данных широко используются формы и отчёты. Они являются составными объектами, так как состоят из более мелких объектов (таких, как поля, кнопки, диаграммы, рамки, OLE – компоненты и прочие объекты), которые называются объектами интерфейса.

Формы - объекты, предназначенные в основном для программирования графического интерфейса пользователя, для просмотра и редактирования информации из баз данных, для отображения и ввода данных с применением различных элементов управления. Например, при разработке приложения для бухгалтерии возможно создание форм, которые на экране будут выглядеть точно так же, как стандартные бланки бухгалтерских документов.

Формы предоставляют возможность использования широкого набора объектов, настройки этих объектов посредством определения их свойств и заданием требуемого поведения объектов посредством описания их методов. Простота при определении действий объектов, как для системных событий, так и для событий, определяемых пользователем, позволяет легко, эффективно и быстро описывать интерфейс пользователя и другие процедуры управления приложением.

Отчеты – используются для форматирования, вычисления итогов и печати определенных данных из базы данных. Отчёты создаются в том случае, если информацию необходимо иметь в печатном виде. Примерами отчётов являются прайс-листы товаров, списки покупателей, различные ведомости и т.д. Как правило, отчеты можно предварительно просматривать на экране.

Запросы – являются средством выборки данных из одной или нескольких таблиц. Практически во всех реляционных СУБД для создания запроса используется специализированный язык Structured Query Language - **SQL** (см.3.3). Результаты выполнения запроса могут отображаться в форме, выводиться в виде отчётов и диаграмм или сохраняться в указанной таблице.

Можно создать запрос не только на выборку, но и на обновление, удаление, добавление данных.

Программы, написанные на языке программирования СУБД, являются объектно-ориентированными. С их помощью обрабатываются события в форме, создаются объекты, изменяются их свойства, осуществляются различные вычисления, выполняется управление базой данных. Для удобства работы можно объединить программы в библиотеки.

Визуальное проектирование БД и их приложений использует визуальное программирование, основу которого составляет манипулирование объектами будущего приложения на экране в виде, максимально приближенном к их реальному воспроизведению. Визуальное программирование обеспечивается объектно-ориентированным подходом, который позволяет очень эффективно использовать модульный принцип составления программ.

Во всех современных СУБД используется объектно-ориентированный подход, что позволяет применять технологию визуального проектирования БД и их приложений. Эта технология реализуется с помощью специальных средств: конструкторов, генераторов, редакторов, мастеров, построителей и т.д.

Конструкторы (генераторы, редакторы) используются для проектирования компонентов БД и пользовательских приложений БД. Они имеют набор специальных средств (инструменты, кнопки, функциональные окна, команды меню и т.д.), которые позволяют с учетом специфики каждого объекта максимально облегчить и сделать наглядным процесс разработки.

Во многих современных программных системах существуют процедуры, которые в русскоязычной литературе называют Мастерами, а в англоязычной – **Wizards** (Волшебники). Они предназначены для упрощения процесса разработки сложных объектов. Особенность Мастеров состоит в том, что они используют готовые объекты для создания стандартизованных объектов.

Мастера - это интерактивные визуальные средства проектирования, которые через заранее определенную последовательность диалоговых окон (шагов) уточняют задачу и разрабатывают требуемый компонент, используя стандартное решение задачи. Так как это стандартное решение, то при этом упускаются из виду специфические особенности создаваемого объекта или данных, но сам процесс разработки упрощается. После запуска мастера достаточно ответить на вопросы в текущем диалоговом окне, после чего перейти к следующему шагу работы мастера.

Построители - это те же мастера или конструкторы, но они обычно вызываются из основных конструкторов и помогают решать дополнительные задачи, возникающие при работе конструкторов.

Один и тот же компонент может быть создан с использованием разных средств. Например, для создания отчётов (форм) в СУБД Visual FoxPro

используются мастер и конструктор отчётов (форм). С помощью мастера отчётов можно быстро создать собственный отчёт на основе имеющихся шаблонов. Применяя конструктор отчётов, можно создавать отчёты произвольной сложности, включая многоуровневую группировку данных и размещение вычисляемых полей. Чтобы создавать формы в СУБД Visual FoxPro, можно использовать не только базовые классы, но и создать собственные (например, определить класс форм, в котором задан определённый цвет фона и стандартный набор кнопок для управления данными). Чтобы стандартизировать разработку, полезно иметь один или несколько пользовательских классов для каждого базового класса.

При создании приложений БД во многих СУБД (и не только в СУБД) используется понятие проекта. **Проект** - средство объединения отдельных элементов приложения. Для создания проекта используются специальные средства (например, диспетчер проектов).

9.3. СОЗДАНИЕ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ

В состав полнофункциональных СУБД включаются средства, позволяющие проектировать и создавать приложения баз данных. Обычно СУБД также предоставляют дополнительные средства разработки приложений, которые могут работать не только с собственными структурами данных, но и с форматами других наиболее распространенных СУБД. В большинстве реляционных СУБД предусмотрена возможность обработки данных из электронных таблиц, текстовых файлов, файлов баз данных различных серверов SQL. Это означает, что СУБД можно использовать для создания такого приложения, которое будет работать с данными, поступающими с сетевого сервера SQL или из базы данных SQL на большой ЭВМ.

Приложения баз данных представляют собой информационные системы различного типа, ориентированные на конечного пользователя. Они должны быть удобны в эксплуатации и иметь дружественный комфортный **пользовательский интерфейс**. Обычно приложения баз данных создаются для специалистов, не обладающих никакими знаниями в сфере программирования, и знакомых с работой компьютера лишь поверхностно на уровне оператора. Простота, удобство, надежность, сохранение целостности данных, учет требований и пожеланий пользователя в соответствии с поставленными задачами, а также легкость восприятия информации - это основные требования при создании приложения. В связи с этим, можно сформулировать ряд требований к разрабатываемому приложению, изложенные ниже.

Автономность эксплуатации. При эксплуатации система не должна быть связанной ни с какими другими программными средствами.

Функциональность. Приложение должно полностью соответствовать своему функциональному назначению. Необходима возможность выбора пользователем функций, выполняемых приложением.

Экономичность. Приложение должно экономно использовать вычислительные ресурсы для обеспечения его работы на небольших, экономичных конфигурациях аппаратного обеспечения без потери функциональности.

Простота эксплуатации. При работе с приложением не должно требоваться специальных знаний по программированию, должны отсутствовать требования к профессиональной подготовке пользователя в области вычислительной техники. Достаточно, чтобы пользователь мог запустить компьютер, работать с клавиатурой, совершать некоторые действия, такие, как установка дискеты, копирование информации и т.д.

Дружественность интерфейса. Работа приложения должна сопровождаться подсказками, ориентирующими пользователя на выполнение определенных действий. В подсказках должна содержаться информация о том, как выполнить то или иное действие. Должна быть обеспечена возможность получения вспомогательной информации “Помощь” в зависимости от контекста. В случае значительных затрат времени на обработку информации (затягивание ответа) в системе должна быть предусмотрена возможность периодического информирования пользователя о ходе работы. 4. В приложении должна быть предусмотрена возможность контроля за действиями пользователя, диагностика ошибочных действий и выдача на экран рекомендаций по их устранению.

Комфортность интерфейса. Отображаемая информация должна быть структурированной для обеспечения комфортных условий взаимодействия в диалоговой системе и согласованной с возможностями человека по восприятию и обработке данных. Система отображения и комментариев (подсказок) должна строиться на однотипной формальной грамматике, рациональном размещении информации на экране видеотерминала (монитора), сообщения должны быть короткими, терминология понятна пользователю.

Надежность и защита данных. Должно быть обеспечено сохранение целостности и непротиворечивости данных, ограничение несанкционированного доступа к ним. Работа системы должна обеспечивать возможность резервного копирования и создания архива данных с целью обеспечения надёжности их хранения и восстановления в случае порчи.

Современные визуальные технологии обеспечивают быструю разработку приложений (RAD - Rapid Application Designer), отвечающих вышеперечисленным требованиям. Но самое главное - это не непосредственно разработка, а предшествующие ей этапы проектирования - планирования.

В главе 4 рассматриваются общие вопросы проектирования приложений баз данных - информационных систем и теоретические вопросы проектирования баз данных. В главе 6 рассматривается язык программирования xBase, применяемый в dBase-подобных СУБД для создания приложений. В главе 7 рассматривается реализация языка управления данными SQL в СУБД FoxPro, широко используемого как для создания локальных приложений, так и приложений типа клиент-сервер. В главе 10 рассматриваются визуальные средства проектирования компонентов приложения и принципы их работы в среде Visual FoxPro.

Ниже рассматриваются практические этапы разработки приложений баз данных в среде современных реляционных СУБД с проблемно-ориентированным языком программирования.

Этап 1. Анализ предметной области и требований пользователя:

- уточнение задач и требований пользователя;
- анализ (определение) входной информации и входных документов;
- анализ технологии сбора информации;
- определение типов обработки информации;
- анализ (определение) выходной информации и выходных документов.

Этап 2. Определение данных и потоков данных создаваемого приложения:

- определение характеристик входных, выходных и обрабатываемых данных;
- определение технологии сбора входных данных и выдачи выходных данных.

Этап 3. Создание компонентов приложения:

- определение структуры таблиц, баз данных, индексов, представлений, нормализация баз данных;
- определение запросов, отчетов, входных и выходных форм;
- создание и отладка созданных компонентов приложения.

Этап 4. Создание проекта приложения:

- разработка пользовательского интерфейса;
- создание приложения;
- тестирование и усовершенствование приложения.

Понимание различия между данными и информацией в СУБД (в более строгом значении этих терминов) облегчит выявление сведений, которые необходимо хранить в базе данных. Данные - это статические значения, хранящиеся в таблицах баз данных, а информация - это сведения, которые запрашиваются пользователем и предоставляются ему в наиболее удобном виде. Можно сказать, что данные - это информация, структурированная в соответствии с требованиями конкретной СУБД. А информация - это представление данных в соответствии с требованиями пользователя.

Анализ предметной области и требований пользователя (этап 1) заключается в изучении и анализе имеющейся системы обработки данных

(ручной или автоматизированной). В случае отсутствия подобной системы изучается и описывается та часть предметной области, которая представляет интерес для конечного пользователя. До начала работы над конкретным приложением существует некоторое представление о том, для чего оно предназначено. Необходимо определить список всех основных задач, которые должны решаться этим приложением, - включая те, которые не нужны сегодня, но могут появиться в будущем. Под основными задачами понимаются функции, которые должны быть представлены в формах или отчетах приложения. Изучаются и уточняются требования пользователей, стандартизуется терминология.

После формирования списка задач составляется подробный перечень всей информации, необходимой для решения каждой задачи. Для каждой задачи существует некоторая исходная информация (например, зарплата, установленная для определенной категории служащих, необходима для подсчета общей суммы отчислений зарплаты для данной категории). Кроме того, некоторую информацию задача может удалять (например, информацию об оплаченных счетах) или добавлять (например, дополнительные сведения о зарплате). И, наконец, часть информации должна быть получена в результате вычислений и отображена, но не сохранена.

Определяется, как осуществляется сбор информации, что делается с информацией после ее сбора, как она видоизменяется, какая информация и каким образом вводится в систему. Изучаются выходные документы, какая информация и в какой форме выводится.

Определение данных и потоков данных создаваемого приложения (этап 2) заключается в определении структуры для входных и выходных данных, обрабатываемых данных, границ их значений. Устанавливают параметры элементов данных, порядок и технологию сбора входных данных, порядок и технологию выдачи выходных данных.

Создание компонентов приложения (этап 3) включает определение набора функциональных компонент приложения, их создание и отладку. Функциональные компоненты приложения обеспечивают ввод, хранение, обработку и вывод необходимых данных в требуемом виде, обеспечение целостности и защиту данных. К компонентам приложения относятся таблицы, базы данных, индексы, представления, запросы, отчеты, формы, программы обработки данных и т.д.

После предварительного анализа всех необходимых для приложения элементов данных, все данные упорядочивают по объектам, и объекты соотносят с таблицами, представлениями и запросами базы данных. Таблицы подвергаются нормализации, в результате чего реализуется наиболее эффективный и гибкий способ хранения данных. Выполняются требования уникальности полей, выбора первичного ключа, независимости полей, эффективных связей между таблицами и т.д. Определяются структура баз данных, необходимые индексы для связи таблиц и ускорения поиска в них

данных. Входные документы соотносят с входными формами, выходные документы - с выходными формами, отчетами, запросами. Определяются средства реализации функциональности компонент: свойства, методы, хранимые процедуры, триггеры и т.д. Выполняется создание функциональных компонент приложения и их отладка в среде СУБД. Используя визуальное проектирование можно создавать на экране достаточно реальные, необходимые в приложении формы и отчеты, периодически переключаясь для контроля своих действий из режима проектирования в режим предварительного просмотра.

Создание проекта приложения (этап 4) заключается в разработке удобного пользовательского интерфейса с использованием средств визуального проектирования и объединении функциональных компонент приложения в единое целое. В случае очень простых задач простое объединение компонент является практически законченным приложением. Но обычно законченное приложение имеет некоторую **основную программу** - меню, форму, запрос или любую другую основную программу. Для небольших приложений в качестве основной программы используют **меню** или **формы с меню**. Запуск приложения осуществляется путем запуска, соответственно, меню или формы. Дерево меню организуется на основе функций, выполняемых приложением (например, **Продажа товара, Поступление товара** и т.д.) или на основе таблиц, используемых в приложении.

При создании крупных приложений баз данных основная программа, которая вызывает компоненты приложения, часто является просто **программным кодом**. В этом случае она включает процедуры, которые запоминают текущие установки системы, устанавливают необходимые параметры системы, определяют глобальные массивы и переменные. После чего, в случае управления по событиям (парадигма объектно-ориентированного программирования), приложение переводится в режим ожидания событий.

Приложение тестируется разработчиком для каждого из возможных режимов работы и проект приложения передается пользователю для проверки функционирования и получения мнения о необходимости тех или иных изменений. В процессе знакомства с работой приложения у пользователей практически всегда возникают дополнительные предложения по усовершенствованию, какой бы тщательной ни была предварительная проработка проекта. Часто обнаруживается, что некоторые средства, о которых в процессе постановки задач говорится как об очень важных и необходимых, на самом деле практически не используются. Выявление необходимых изменений на ранних стадиях разработки приложения позволит существенно сократить время на последующие переделки. Поэтому по мере разработки автономных разделов приложения желательно передать их

пользователям для проверки функционирования и получения мнения о необходимости тех или иных изменений.

Передача приложения в эксплуатацию отнюдь не означает, что процесс усовершенствования завершен. Большинство разработчиков программного обеспечения после завершения работы над очередной версией часто видят возможность дальнейшего улучшения и расширения возможностей программного продукта.

Многие СУБД обеспечивают возможность создания **единого файла приложения**, часто автономно исполнимого (т.е. без наличия самой СУБД). Автономные исполнимые файлы приложений обычно требуют наличия на компьютере специальных определенных **библиотек** (программ).

Современные СУБД воспринимают множество самых разнообразных форматов данных, включая файловые структуры других СУБД. Можно осуществлять импорт и экспорт данных из текстовых файлов, электронных таблиц, из таблиц баз данных других СУБД. При разработке приложений БД в СУБД под Windows широко применяются такие технологии, как DDE, OLE, ODBC, ActiveX и т.д.

ОС Windows обеспечивает механизмы коллективного пользования данными из различных приложений. Сначала это был временный буфер Windows (Clipboard), затем в набор протоколов связи был добавлен динамический обмен данными (DDE), а затем появился механизм связывания и внедрения объектов (OLE).

DDE (Dynamic Data Exchange, динамический обмен данными) позволяет выполнять функции и производить обмен данными между любыми, поддерживающими DDE, приложениями Windows.

OLE (Object Linking and Embedding, связь и внедрение объектов) является более совершенной **технологией Microsoft**, которая, в частности, позволяет устанавливать связи с объектами другого приложения или внедрять некоторые объекты в базу данных. Это могут быть рисунки, диаграммы, электронные таблицы или документы из других приложений Windows, поддерживающих технологию OLE (так называемые OLE-объекты). Первоначальной целью OLE было позволять пользователям связывать непохожие файлы или внедрять фрагменты одного файла в другой. Теперь OLE обеспечивает прямую связь между файлами, используя специальный язык, новую наглядную спецификацию и выборочный контроль, что может быть использовано для расширения функциональности среды разработчика приложений БД.

Следующий список содержит ряд терминов, относящихся к технологии OLE.

OLE-объект - это единица информации, которую можно связать или внедрить. Ею может быть текст, графика, звуковое сопровождение. OLE является протоколом, который позволяет связывать документы. Если объект связывается с таблицей БД, то он запоминается в приложении-сервере или

файле, создаваемом этим приложением, а в таблице содержится только ссылка на объект. При связывании СУБД сохраняет только ту информацию, которая необходима для манипулирования этим объектом в СУБД. Характеристикой OLE является также его способность внедрять один объект в другой. Если объект внедряется в таблицу БД, он хранится в этой таблице вместе с именем приложения, используемом при его создании.

Вложенность. Используя механизм OLE, один документ может содержать фрагменты другого. Например, можно хранить в таблицах фрагменты документа текстового процессора, электронной таблицы и другие документы.

Приложение-сервер - исходное приложение (приложение источника), которое было использовано для создания внедренного или связанного объекта. Например, если в таблицу БД Visual FoxPro внедряется изображение Paint, то Paint является приложением-сервером для редактирования, вывода на экран или выполнения OLE-объекта. В случае связанного объекта данные объекта запоминаются в сервере или файле, созданном этим приложением.

Приложение-клиент - это приложение, создающее документ, в который внедряется или с которым связывается OLE-объект. Этот документ называется документом-клиентом или документом-контейнером.

Клиент-сервер. OLE использует архитектуру клиент-сервер для создания своих протоколов связи. Приложения могут выполнять роль клиента или сервера, или их обоих. Клиенты OLE используют сервер OLE для обработки своих данных.

Регистрация. Если устанавливаемое приложение является сервером OLE, то оно должно зарегистрировать себя в Windows при его установке.

СУБД может действовать как клиент OLE, и как сервер механизма управления объектами OLE (**OLE Automation**), позволяя открывать свои объекты (таблицы, запросы и формы) в других приложениях Windows, и работать там с ними.

Для улучшения работы приложения можно включать в формы и отчеты многих СУБД специальный тип объекта OLE, так называемый **элемент управления ActiveX**. ActiveX образует большой класс специальных элементов управления, которые можно применять как в приложениях Microsoft Office, так и на страницах Web в сети Интернет. Они основаны на технологии OLE и являются довольно сложными объектами, позволяющими представлять данные в удобном графическом виде. Многие элементы управления ActiveX предлагают большой набор действий (методов), которые можно использовать при установке значений свойств и для управления внешним видом и поведением элемента.

Такие распространенные СУБД, как Microsoft Access, Paradox, Borland dBASE, Microsoft FoxPro поддерживают стандарт ODBC (Open Database Connectivity - Открытый доступ к данным), который является открытым

стандартом доступа к БД. ODBC поддерживает стандартный протокол для SQL-серверов БД. Этот стандарт обеспечивает им как взаимный доступ к базам данных, так и доступ к серверам БД, включая Microsoft SQL Server, Oracle, Informix, Sybase, InterBase, DB2, Rdb и другие серверы БД.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какое понятие является основным в объектно-ориентированном программировании?
2. Что собой представляют метод и свойство объекта?
3. Какая модель управления используется в объектно-ориентированном программировании?
4. Что собой представляет событие?
5. Какие характеристики присущи объектно-ориентированному программированию?
6. Какое определение можно дать объекту?
7. Что означает понятие класса в объектно-ориентированном программировании?
8. Какие существуют основные типы классов?
9. В чем отличие класса- контейнера от класса-элемента управления?
10. В чем состоит особенность визуального программирования и проектирования?
11. Какие объекты относятся к основным объектам СУБД?
12. Что собой представляют объекты-формы и объекты-отчеты, для чего они используются?
13. Что собой представляют объекты-запросы и объекты-программы, для чего они используются?
14. Какой специализированный язык используется в запросах?
15. Какие существуют специальные средства, позволяющие применять технологию визуального проектирования баз данных и их приложений?
16. Для чего используются конструкторы, мастера, построители?
17. Какие требования предъявляются к приложениям баз данных?
18. Какие задачи решаются на этапе анализа предметной области при проектировании приложений баз данных?
19. В чем заключается этап создания компонентов приложения?
20. На каком этапе разработки приложений баз данных выполняется разработка пользовательского интерфейса?
21. Что может выступать в качестве основной программы приложения баз данных?
22. Какие технологии используются при разработке приложений баз данных в СУБД под Windows?
23. Что собой представляют технологии DDE, OLE, ActiveX?
24. Что такое OLE-объект, OLE-клиент, OLE-сервер?

10. СУБД VISUAL FOXPRO

10.1.ОСНОВНЫЕ ХАРАКТЕРИСТИКИ VISUAL FOXPRO

Microsoft Visual FoxPro - это функционально полная реляционная СУБД. В ней предусмотрены все необходимые средства для определения и обработки данных, а также для управления ими при работе с большими объемами информации. Visual FoxPro является развитием dBase-подобной СУБД FoxPro с языком программирования - **xBase** (см. главу 6). Она относится к СУБД с **визуальными средствами проектирования** на основе объектно-ориентированного программирования, является типичным клиентом сервера БД и поддерживает **язык управления данными SQL.** Пользовательский интерфейс FoxPro всегда был гибким и дружественным, а мощность и скорость ставили его вне конкуренции.

С появлением Visual FoxPro производительность не ухудшилась. Зато стало легче получать доступ к файлам ODBC-совместимых баз данных. ODBC (Open DataBase Connectivity) является стандартным протоколом для серверов баз данных, реализует концепцию открытых баз данных. Поэтому можно напрямую, без преобразования работать с данными формата СУБД Access, Paradox, dBase и т.д.

В среде СУБД Visual FoxPro легко создавать пользовательские интерфейсы и SQL-запросы для обращения к данным. В СУБД Visial FoxPro можно создавать клиентские приложения для архитектуры клиент-сервер. Приложение может работать с локальными и сетевыми ресурсами одновременно. Есть возможность хранить и обрабатывать информацию на серверных платформах, таких как IBM AS/400 и др.

Язык программирования Visual FoxPro имеет среду для интерактивного выполнения команд и выполнения скомпилированных программ. Это означает, что для разработки собственных прикладных программ необходимо наличие полной версии Visual FoxPro, но созданные программы можно передавать другим пользователям, наличие у которых собственных копий Visual FoxPro необязательно.

Visual FoxPro спроектирована таким образом, что она может быть использована как в качестве самостоятельной СУБД на отдельной рабочей станции, так и в сети в режиме клиент-сервер. Поскольку в Visual FoxPro доступ к данным могут иметь одновременно несколько пользователей, в ней предусмотрены надежные средства защиты и обеспечения целостности данных. В Visual FoxPro применяется механизм автоматической блокировки для избежания одновременного изменения объекта несколькими пользователями.

Язык Visual FoxPro является объектно-ориентированным, визуально

программируемым языком, **управляемым по событиям**, и в полной мере соответствует требованиям, предъявляемым к современным средствам проектирования приложений. В Visual FoxPro 6.0 можно использовать **компоненты ActiveX**, которые представляют собой 32-разрядные компоненты, созданные в соответствии с открытой спецификацией, разработанной для стандартизации взаимодействия программных компонентов. Использование этих компонентов позволяет расширить возможности приложений и ускорить их создание.

В Visual FoxPro 6.0 имеется возможность создавать приложения типа **Active Document**. Аналогично обычным приложениям Visual FoxPro они содержат формы, отчёты, запросы, программы, но их можно запускать не только из Visual FoxPro, но и из Web - броузеров, таких как Internet Explorer. Наиболее часто используемые в Интернет-приложениях графические форматы GIF (Graphics Interchange Format) и JPEG (Joint Photographic Electronic Group) поддерживаются теперь и в Visual FoxPro.

В Visual FoxPro 6.0 добавлены новые методы для работы со свойствами объектов **Assign** (Назначение) и **Access** (Доступ). Первый из них выполняется при изменении значения свойства, а второй - при чтении значения свойства.

Для упорядоченного хранения компонентов приложений (классов, форм, кнопок, проектов) предоставляется возможность использовать галерею компонентов. Галерея компонентов также содержит большое количество классов, поставляемых в составе Visual FoxPro.

Для обнаружения ошибок в приложении и анализа процесса его выполнения теперь можно использовать инструмент Coverage Profiler. С помощью этого средства создаётся файл, в котором содержится информация о процессе выполнения приложения.

Компания Microsoft разработала новый стандарт отображения справочной информации в формате **HTML**. Используя программу **Help Workshop**, входящую в комплект поставки HTML, можно создавать файлы **справки HTML** для приложений Visual FoxPro.

В Visual FoxPro 6.0 добавлены новые мастера и построители, функции и команды, которые облегчают разработку приложений. Как и во многих других программах, в Visual FoxPro 6.0 полностью решена проблема 2000 года.

В Visual FoxPro имеется широкий набор **базовых классов** (прототипов объектов) - **контейнеров** и **элементов управления**, с помощью которых можно определить свои объекты. Большинство базовых классов являются видимыми (визуальными), некоторые классы используются для объединения объектов и не отображаются в форме (не визуальные). Основные базовые классы представлены в табл.10.1.

Для каждого объекта Visual FoxPro существует свой **набор событий**, используемый для управления программой. Ниже описываются основные события, используемые в событийной модели управления Visual FoxPro.

Таблица 10.1

Наименование	Назначение	Тип класса
ToolBar	Панель инструментов	Контейнер, визуальный
Form	Форма	Контейнер, визуальный
FormSet	Группа форм	Контейнер, не визуальный
Page	Страница формы	Контейнер, визуальный
PageFrame	Набор страниц	Контейнер, не визуальный
Grid	Таблица	Контейнер, визуальный
CommandGroup	Набор кнопок управления	Контейнер, визуальный
OptionGroup	Набор переключателей	Контейнер, визуальный
Header	Заголовок столбца таблицы	Элемент управления, визуальный
Command	Кнопка управления	Элемент управления, визуальный
OptionButton	Переключатель	Элемент управления, визуальный
CheckBox	Флажок	Элемент управления, визуальный
Spinnner	Счетчик	Элемент управления, визуальный
TextBox	Поле ввода	Элемент управления, визуальный
EditBox	Область редактирования	Элемент управления, визуальный
Label	Надпись	Элемент управления, визуальный
ComboBox и **ListBox**	Комбинированный список и Список	Элементы управления, визуальные
Line и **Shape**	Линия и Обрамление	Элементы управления, визуальные
Image	Изображение	Элемент управления, визуальный
Timer	Таймер	Элемент управления, не визуальный

Load - событие возникает, когда форма или набор форм загружаются в память.

UnLoad - событие возникает, когда форма или набор форм выгружаются из памяти.

Init - событие возникает при инициализации объекта.

Destroy - событие возникает, когда объект уничтожается.

Activate - событие возникает при активизации объекта.

Click - событие возникает при щелчке мышью на объекте.

DblClick - событие возникает при двойном щелчке мышью на объекте.

RightClick - событие возникает при щелчке правой кнопкой мыши на объекте.

MouseDown - событие возникает при нажатии клавиши мыши, когда курсор мыши указывает на объект.

MouseUp - событие возникает при освобождении клавиши мыши, когда курсор мыши указывает на объект.

KeyPress – событие возникает для активного объекта при нажатии любой клавиши клавиатуры.

InteractiveChange - событие возникает, когда значение объекта изменяется в интерактивном режиме.

ProgrammaticChange - событие возникает, когда значение объекта изменяется в программном режиме.

GetFocus - событие возникает при любой активизации объекта.

LostFocus - событие возникает, когда объект перестаёт быть активным.

Error - событие наступает при возникновении ошибки, связанной с объектом.

Для отдельных объектов не все из перечисленных выше событий имеют место, но каждый объект обязательно имеет три события: **Init** (наступает при инициализации объекта), **Destroy** (возникает, когда объект уничтожается) и **Error** (наступает при возникновении ошибки, связанной с объектом).

Настройка конструируемого на основе классов объекта на конкретное применение осуществляется с помощью свойств. **Свойства** можно считать физическими атрибутами объекта. Каждый атрибут описывает объект и делает его уникальным. Перечень свойств объекта, созданного на основе базового класса, определяется свойствами базового класса (принцип наследования). Свойства подобны переменным, однако в отличие от последних, они не существуют вне объекта. Каждый базовый класс обладает своим набором свойств, которыми обладают все классы и объекты, созданные на его основе. Но есть свойства, которыми обладают все классы:

Class - тип класса;

BaseClass - базовый класс, на основе которого создан данный класс (объект);

ClassLibrary - библиотека классов, в которой хранится данный класс;

ParentClass - определенный пользователем класс, на основе которого создан данный класс (объект).

Возможность изменения свойств объектов в Visual FoxPro является мощным инструментом управления данными в интерактивном режиме. Например, возможно:

- изменить свойство экранной формы (заголовок, размер и т.п.);
- изменить цвета полей вывода, формат отображения;
- запретить ввод в поля, переход на них или же сделать их невидимыми и др.

Для того чтобы определить какие-либо из свойств объекта в интерактивном режиме проектирования, необходимо выбрать требуемый объект и нажать правую кнопку мыши, а из открывшегося меню объекта выбрать команду **Свойства** (Properties). В результате на экране появится окно свойств текущего объекта. Значения свойств объектов могут быть **символьными**, **логическими** и **числовыми**. Некоторые из них могут принимать произвольные значения (например, заголовок поля, координаты объекта поля). Другие свойства могут принимать значения из определенного списка.

Visual FoxPro позволяет изменять свойства объектов в процессе выполнения программы. Например, при переходе из режима ввода в режим редактирования можно изменить название заголовка формы **ВВОД**, которое является одним из свойств формы, на **РЕДАКТИРОВАНИЕ.**

Ниже перечислены основные свойства объектов Visual FoxPro.

Border Color - задаёт цвет рамки.

BackColor - задаёт цвет фона объекта.

ForeColor - задаёт цвет выводимых символов.

FontName, **FontSize** - используются для задания имени и размера шрифта символов (по умолчанию используется шрифт Arial, 10 пунктов).

Height и **Width** - высота и ширина объекта.

Left и **Top** - расстояние объекта от левого и верхнего края объекта - контейнера.

StatusBarText - задаёт текст в строке состояний.

Name - задает программное имя объекта.

Caption - задаёт название заголовка формы, название кнопки и т.д.

Picture - используется для задания имени файла с изображением или имени поля типа General.

MouseIcon - определяет имя файла для указания вида курсора мыши (файлы с расширением ICO, ANI, CUR).

ConrtolSource - указывает источник, откуда берутся данные (поле таблицы, переменная, элемент массива и т.д.).

ReadOnly - используется для задания возможности изменения данных, имеет два значения: **True** - не разрешается изменять данные; **False** - разрешается изменять данные в источнике данных.

Visible - видимость объекта, имеет два значения: **True** - объект виден; **False** - объект не виден, но он существует.

Enable - доступность объекта, имеет два значения: **True** - объект доступен пользователю; **False** - объект не доступен пользователю.

BaseClass - имя базового класса объекта.

Parent - позволяет сослаться на контейнер, который включает данный элемент управления.

Свойство можно задать не только при проектировании приложения, но и при его работе программным путём, указав: **имя_объекта.имя_свойства = значение_свойства.**

Каждый объект Visual FoxPro имеет не только свойства, наследованные из класса, на основе которого создается объект, но и **методы**. Список методов, определенных для объекта, отражается во вкладке **Methods** окна свойств объекта.

Методы - это действия, которые объекты могут выполнять. Очевидно, что объект формы имеет по крайней мере два метода - открыть и закрыть. При создании формы предполагается ее когда-нибудь открыть, а открытую форму затем нужно закрыть. Следовательно, формы содержат процедуры, сообщающие Visual FoxPro, как открывать и закрывать их.

Существуют методы, имеющие общий характер для объектной модели Visual FoxPro. Например: метод **Release** позволяет удалить объект, метод **Refresh** - обновить объект, метод **SetFocus** - активизирует указанный элемент управления, метод **ResetToDefault (‘имя свойства’)** - возвращает для указанного свойства значение по умолчанию.

Для каждого события (перечень основных событий приведен выше) определен соответствующий ему метод, который выполняется при возникновении данного события. Например, метод **Click** выполняется при нажатии кнопки мыши. Список доступных методов обработки событий определяется базовым классом. Именно эти методы обеспечивают обработку всех действий пользователя (нажатие клавиш, управление мышью и другие действия). Visual FoxPro предоставляет возможность создания и модифицирования методов-процедур (с использованием объектно-ориентированного языка программирования и языка программирования xBase), активизируемых событиями данного объекта.

Для каждого события объекта выполняется определенное базовым классом действие. В некоторых случаях можно отключать это действие, тогда в начале метода вставляется ключевое слово **NoDefault.**

В объектно-ориентированной СУБД Visual FoxPro широко применяются поля типа **General** и типа **Memo**. Поле типа General является специальной разновидностью Memo-полей.

Memo-поля позволяют хранить длинные текстовые строки (более 254 символов). Они также обеспечивают переменный размер сохраняемой строки на основе блочной структуры. Блок состоит из фиксированного числа символов (по умолчанию - 64).

Visual FoxPro сохраняет Memo-поля не в файле с расширением **.DBF**, а в отдельном файле с расширением **.FPT**. Независимо от числа Memo-полей в таблице все они хранятся в одном **FPT**-файле.

Если есть поля типа General, то они хранятся в том же файле. Поля типа General используются для **OLE-объектов.** OLE-объекты могут храниться и отображаться только в полях типа General.

Поскольку Visual FoxPro работает в многозадачной среде совместно с другими приложениями Windows, можно воспользоваться OLE – механизмом для вывода на экран и редактирования объектов из этих приложений, не выходя из Visual FoxPro. Например, можно выводить на экран и редактировать изображение, подготовленное в Microsoft Paint, документ, созданный в Microsoft Word для Windows, или блок электронной таблицы, созданной в Microsoft Excel. Можно также вставить в Visual FoxPro звуковое оформление, видеоклипы, графику и прочие OLE-объекты.

Первым шагом в добавлении OLE-объектов в приложения Visual FoxPro является создание контейнера для OLE-объектов. Контейнер Visual FoxPro для OLE-объектов представляет собой поле типа General в таблице Visual FoxPro. Для внедрения или связывания OLE-объекта в таблице Visual FoxPro создается предварительно поле типа General, а затем оно используется для хранения OLE-объекта. Если поле типа General пусто, то при просмотре таблицы в нем появится пометка **gen**. Если поле содержит OLE-объект, то пометка примет вид **Gen.**

При помощи поля типа General, Visual FoxPro позволяет внедрять OLE-объекты в свои таблицы. OLE-объект (или ссылка на связанный объект) всегда хранится в поле General. В дальнейшем это поле можно использовать для вывода на экран или в отчет. Можно хранить в таблицах фрагменты документа текстового процессора, электронной таблицы и другие документы.

Visual FoxPro может являться клиентом OLE связи. Если объект связывается с таблицей Visual FoxPro, то он запоминается в приложении-сервере или файле, создаваемом этим приложением, а в таблице Visual FoxPro содержится только ссылка на объект. Когда связывается файл, Visual FoxPro сохраняет только ту информацию, которая необходима для манипулирования этим объектом в Visual FoxPro. Например, если в таблицу Visual FoxPro внедряется изображение Paint, то Paint является приложением-сервером для редактирования, вывода на экран или выполнения OLE-объекта.

Для заполнения и редактирования полей типа General обычно пользуются командами меню **Edit**. Открывают в режиме **Browse** таблицу, содержащую поле данного типа, и дважды щелкают мышью на поле типа General. Откроется окно редактирования этого поля, в котором и выполняются различные действия с OLE-объектами.

Для манипулирования этими объектами программным путем в приложениях Visual FoxPro используются команды **MODIFY GENERAL** и **APPEND GENERAL**.

Команда **MODIFY GENERAL** открывает окно редактирования для указанного поля типа General текущей записи таблицы. При открытии окна все команды управления OLE-объектом становятся доступными, и можно вставлять, изменять или удалять внедренные или связанные объекты. При выполнении этой команды открывается окно редактирования и программа приостанавливает выполнение до выхода из него.

Команда **APPEND GENERAL** импортирует OLE-объекты из файла и помещает его в указанное поле типа General текущей записи. Если в этом поле уже есть OLE-объект, то он будет замещен новым OLE-объектом. Команда может быть использована для внедрения OLE-объекта или создания связи с файлами OLE-объекта в приложении-сервере. Большинство приложений-серверов поддерживают и внедрение и связывание. Некоторые из приложений-серверов (например, Microsoft Graph) поддерживают только внедрение.

10.2.СРЕДСТВА ВИЗУАЛЬНОГО ПРОЕКТИРОВАНИЯ В VISUAL FOXPRO

В языке Visual FoxPro есть два способа организации данных. Организовывать данные можно либо в автономные таблицы, либо в базу данных Visual FoxPro. В Visual FoxPro **база данных** – это совокупность таблиц (Table), представлений (View), связей между ними, хранимых процедур (Stored Procedures), соединений с внешними источниками данных (Connections). БД соответствуют файлы с расширением .DBC, которые хранят информацию о входящих в БД компонентах. В Visual FoxPro можно работать и со свободными таблицами (файлы с расширением .DBF).

СУБД Visual FoxPro обеспечивает визуальное проектирование баз данных, свободных таблиц, представлений, запросов, форм, отчетов и других компонентов системы и работу с этими компонентами.

Для создания компонентов системы в Visual FoxPro существует большое количество мастеров (**Wizards**), краткая характеристика которых приведена ниже.

Мастер баз данных (Database Wizard) - облегчает построение баз данных за счёт использования стандартных типовых баз данных.

Мастер таблиц (Table Wizard) - облегчает построение таблиц за счёт использования стандартных типовых (около 40) таблиц.

Мастер запросов (Query Wizard) - позволяет создать запрос из нескольких таблиц, указав условия для выбора данных и условия для связи между таблицами, при этом постоянные связи между таблицами БД игнорируются.

Мастер перекрёстных таблиц (Cross Table Wizard) - позволяет создать запрос в виде электронной таблицы, которую можно экспортировать в формат электронных таблиц MS Excel, Lotus 1,2,3 и работать с нею в табличном процессоре.

Мастер диаграмм(Graph Wizard) - позволяет создать по данным таблицы график (диаграмму) MS Graph и поместить этот график в специальную таблицу с именем **VFPgraph** в поле типа General с именем **olegraph**. При создании графика данные одного поля таблицы используются для оси абсцисс, для оси ординат, в зависимости от типа графика используются данные от одного до нескольких полей. Затем этот график можно использовать в отчете, в форме.

Мастер локальных представлений (Local View Wizard) и **Мастер внешних представлений** (Remote View Wizard) - создают, соответственно, локальные и удаленные представления (виртуальные таблицы - виды), которые сохраняются в БД. Эти представления можно просматривать, но через них нельзя изменить данные в источнике данных. Чтобы источники данных обновлялись, представления необходимо отредактировать в Конструкторе представлений. Мастер внешних представлений использует технологию ODBC (Open Database Connectivity) для создания видов из внешних данных, то есть данных не в формате **DBF**.

Мастер форм (Form Wizard) - создаёт форму на основе данных одной таблицы.

Мастер форм 1:M (One-to-Many Form Wizard) - позволяет создать форму с данными из двух таблиц, связанных отношением 1:M, причём данные из дочерней таблицы располагаются в виде объекта Grid (таблицы). Мастер форм для отображения данных более, чем из двух таблиц, не существует.

Мастер отчётов (Report Wizard) - создает простой отчёт с упорядоченными данными.

Мастер отчётов с группировкой данных (Group/Total Report Wizard) - позволяет дополнительно указать условия группировки данных (3 уровня) и подведение итогов по каждой группе.

Мастер отчётов 1:M (One-to-Many Report Wizard) - создает отчёт на базе двух таблиц, связанных отношением 1:M. Указываются поля из родительской таблицы, затем из дочерней, данные из которых будут печататься следом за записью родительской таблицы.

Мастер ярлыков (Label Wizard) - позволяет создать этикетку (наклейку, ярлык) на основе данных одной таблицы и правильно расположить этикетки на листе для печати.

Кроме этих, традиционных для многих СУБД мастеров, существует ряд специфических мастеров, перечисленных ниже.

Мастер почтовой рассылки (Mail Merge Wizard) - создаёт источник данных для документа почтовой рассылки текстового процессора Microsoft

Word. MS Word должен быть инсталлирован на компьютере, так как после окончания работы мастера он автоматически запускается. Составляется текст документа и указывается, куда вставить выбранные поля из источника данных. Этот мастер используется также для создания текстового файла, который может быть использован в любом текстовом процессоре.

Мастер сводных таблиц (Pivot Table Wizard) - позволяет создать электронную таблицу как объект MS Excel или как объект формы Visual FoxPro. В первом случае в конце работы мастера автоматически запускается MS Excel. Во втором случае - автоматически создается форма со встроенной таблицей и открывается в Конструкторе форм.

Мастер импорта данных (Import Wizard) - из таблиц Excel, Lotus, СУБД Paradox, текстовых файлов импортирует данные в таблицу формата DBF. Можно создать новую таблицу или добавить данные в существующую таблицу.

Мастер документирования (Documenting Wizard) - подготавливает программу для распечатки.

Мастер создания дистрибутива (Setup Wizard) - обеспечивает создание дистрибутива разработанного приложения.

Мастер создания Internet страницы (WWW Search Page Wizard) - позволяет представить данные в виде статичного Web-документа (страница в формате HTML).

Мастера наращивания (Upsizing Wizard) для MS SQL Server и Oracle Server - позволяют перенести БД с локального компьютера на сервер БД.

Мастер проектов или мастер создания приложений (Application Wizard) - обеспечивает построение проекта приложения.

Мастера обеспечивают быстрое и упрощенное создание компонентов системы в режиме диалога по стандартным шаблонам. А для создания компонентов, максимально учитывающих специфику задачи, используются конструкторы Visual FoxPro: Конструктор БД, Конструктор таблиц, Конструктор представлений, Конструктор запросов, Конструктор форм, Конструктор отчетов, Конструктор ярлыков и т.д. Компоненты Visual FoxPro, созданные с помощью мастера, можно открыть в соответствующем конструкторе и модифицировать для придания требуемой функциональности.

Конструкторы имеют свои меню с набором соответствующих команд, специфические панели инструментов, функциональные окна, кнопки и другие средства, которые позволяют создавать компоненты с заданными функциями, свойствами, методами. Некоторые компоненты Visual FoxPro (например, меню) можно создать только в конструкторе. При запуске любого конструктора в системное меню Visual FoxPro добавляется дополнительный пункт меню с командами для работы с соответствующим компонентом.

10.3. ВИЗУАЛЬНОЕ ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

В информационных системах – приложениях баз данных, значительная часть затрат на разработку падает на проектирование базы данных как основного средства структурного хранения информации и организации доступа к данным. В базе данных поддерживается целостность данных с помощью первичных ключей и связей между таблицами. Для обработки событий добавления, удаления или изменения записей таблиц используются средства самой базы данных - триггеры и хранимые процедуры.

БД в Visual FoxPro может быть создана в интерактивном режиме с помощью визуального инструментария создания базы данных - **Мастера БД** или **Конструктора БД** (Database Designer), который имеет свою панель инструментов. Конструктор баз данных, имеющий больше возможностей по сравнению с Мастером БД, позволяет: создавать, модифицировать и удалять компоненты БД (таблицы, хранимые процедуры, представления данных, соединения с внешним источником данных); добавлять свободные таблицы в БД; определять для таблиц БД индексы; устанавливать отношения между таблицами, которые будут поддерживаться при создании форм и отчётов.

Кнопки панели инструментов Конструктора БД продублированы в меню **Database**, которое добавляется в системное меню Visual FoxPro при запуске конструктора.

Для создания базы данных необходимо открыть окно Конструктора БД, воспользовавшись системным меню Visual FoxPro. Для этого в меню **File** (Файл) выбирается команда **New** (Новый). В открывшемся диалоговом окне **New** необходимо установить опцию **Database** (База данных) и нажать кнопку **New file** (Новый файл). В окне **Create** в поле **Enter** задать имя создаваемой базы данных. Убедившись, что в поле **Тип файла** установлен тип сохраняемого файла **DBC**, а в раскрывающемся списке **Сохранить в** правильно указана папка, в которой будет располагаться создаваемая база данных, нажать кнопку **Сохранить.**

Откроется пустое окно базы данных **Database Designer** - Конструктор БД (рис.10.1) с панелью инструментов **Database Designer**. Используя кнопки панели инструментов Database Designer, команды меню Database и контекстного меню, в окне Конструктора БД можно добавлять имеющиеся свободные таблицы, создавать новые таблицы и представления, модифицировать и удалять существующие, создавать для них индексы, устанавливать отношения между таблицами.

Панель инструментов Database Designer содержит кнопки для выполнения наиболее часто используемых операций над базой данных. Если панель инструментов Database Designer не видна на экране, в меню **View**

(Вид) нужно выбрать команду **Toolbars** (Панель инструментов). Откроется диалоговое окно **Toolbars**, в котором необходимо установить флажок напротив пункта **Database Designer**.

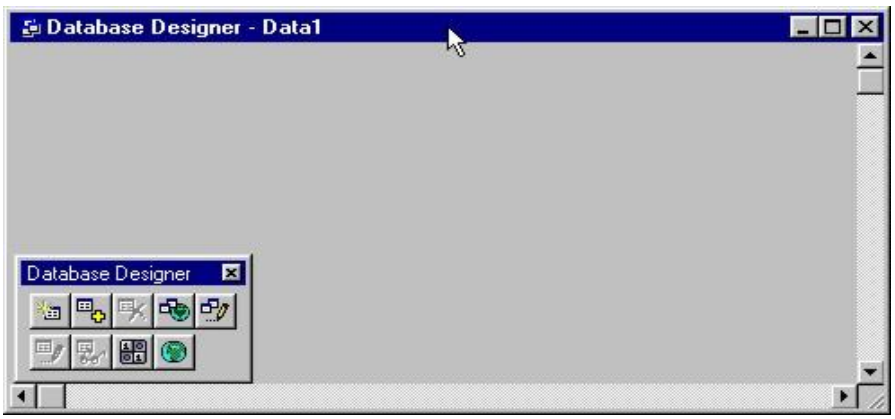


Рис.10.1. Окно Database Designer

В начале создания базы данных окно Конструктора БД пусто. Для создания в Конструкторе баз данных новых объектов и модификации существующих, можно использовать **команды меню Database** или **кнопки панели инструментов Database Designer**:

- New Table (Новая таблица) создаёт новую таблицу (первая кнопка на панели инструментов Database Designer);

- Add Table (Добавить таблицу) добавляет созданную ранее свободную таблицу в базу данных (вторая кнопка на панели инструментов Database Designer);

- Remove Table (Удалить) удаляет таблицу из базы данных (третья кнопка на панели инструментов Database Designer);

- New Remote View (Новое удаленное представление) создаёт удалённое представление данных (четвертая кнопка на панели инструментов Database Designer);

- New Local View (Новое локальное представление) создаёт локальное представление данных (пятая кнопка на панели инструментов Database Designer);

- Modify Table (Модифицировать) открывает таблицу в конструкторе таблиц (шестая кнопка на панели инструментов Database Designer);

- Browse Table (Обзор таблицы) показывает содержимое таблицы в режиме Browse, который дает возможность редактировать записи, добавить запись (сочетание клавиш **Ctrl+Y**), пометить запись на удаление (сочетание клавиш **Ctrl+T**) (седьмая кнопка на панели инструментов Database Designer);

- Find Object (Найти объект) находит указанный объект в окне Конструктора баз данных;

- Rebuild Table Indexes (Перестроить индексы), перестраивает индексы;

- **Remove Deleted Records** (Удалить помеченные записи) физически удаляет из таблицы помеченные для удаления записи;

- **Edit Relationship** (Редактирование отношений) редактирует отношения между таблицами;

- **Edit Referential Integrity** (Редактирование условия целостности) определяет условия целостности данных;

- **Edit Stored Procedures** (Редактирование хранимых процедур) открывает окно редактирования хранимых процедур (восьмая кнопка на панели инструментов Database Designer);

- **Connections** (Соединения) выводит на экран диалоговое окно Connections, в котором создаются или модифицируются соединения с удалёнными данными (девятая кнопка на панели инструментов Database Designer);

- **Arrange** (Упорядочить) упорядочивает объекты по имени или типу и выравнивает их по горизонтали или вертикали;

- **Refresh** (Обновить) обновляет информацию в окне Конструктора БД;

- **Properties** (Свойства) выводит на экран диалоговое окно **Database Properties**;

- **Clean Up Database (**Очистка базы данных) очищает базу данных от помеченных на удаление объектов.

Для работы в окне Конструктора БД можно использовать контекстное меню, вызываемое нажатием правой кнопки мыши. Оно содержит наиболее часто используемые команды меню Database, команду вызова справочной системы, а также команды **Expand All** (Развернуть все) и **Collapse All** (Свернуть все), предназначенные для раскрытия и свёртывания уровней вложенности объектов в окне Конструктора баз данных.

Основной объект БД – таблица (свободная или в составе БД) создаётся с помощью **Мастера таблиц** (Table Wizard) или **Конструктора таблиц** (Table Designer), которые можно запустить из системного меню или из Конструктора БД.

Для создания таблицы из Конструктора БД нужно выбрать команду **New Table** из меню Database (или из контекстного меню) или нажать кнопку **New Table** на панели инструментов Database Designer. Затем в появившемся диалоговом окне **New Table** (Новая таблица) нажать кнопку **New File** (для запуска Конструктора таблиц).

В появившемся окне **Create** в поле **Enter** задать имя создаваемой таблицы данных и нажать кнопку **Сохранить.** В результате выполнения этих действий откроется окно Конструктора таблиц - **Table Designer**. При создании таблиц в базе данных есть дополнительные возможности для придания таблице большей функциональности. Окно Конструктора таблиц в этом случае содержит дополнительные области для задания параметров, свойств, функций таблицы.

Окно Конструктора таблиц - **Table Designer** (рис.10.2) содержит три вкладки, предназначенные для определения следующих параметров: **Fields** (Поля) - для описания характеристик полей таблицы; **Indexes** (Индексы) - для указания индексов таблицы; **Table** (таблица) - для задания условия достоверности вводимых данных, а также триггеров добавления, удаления и модификации.

Вкладка **Fields** (Поля) используется при определении полей таблицы, позволяет помимо основных параметров указать для каждого поля дополнительные параметры, которые будут определять условия ввода данных в него, а также краткое описание, которое поможет разработчику при модификации таблицы в процессе создания приложения или его сопровождения (рис.10.2).

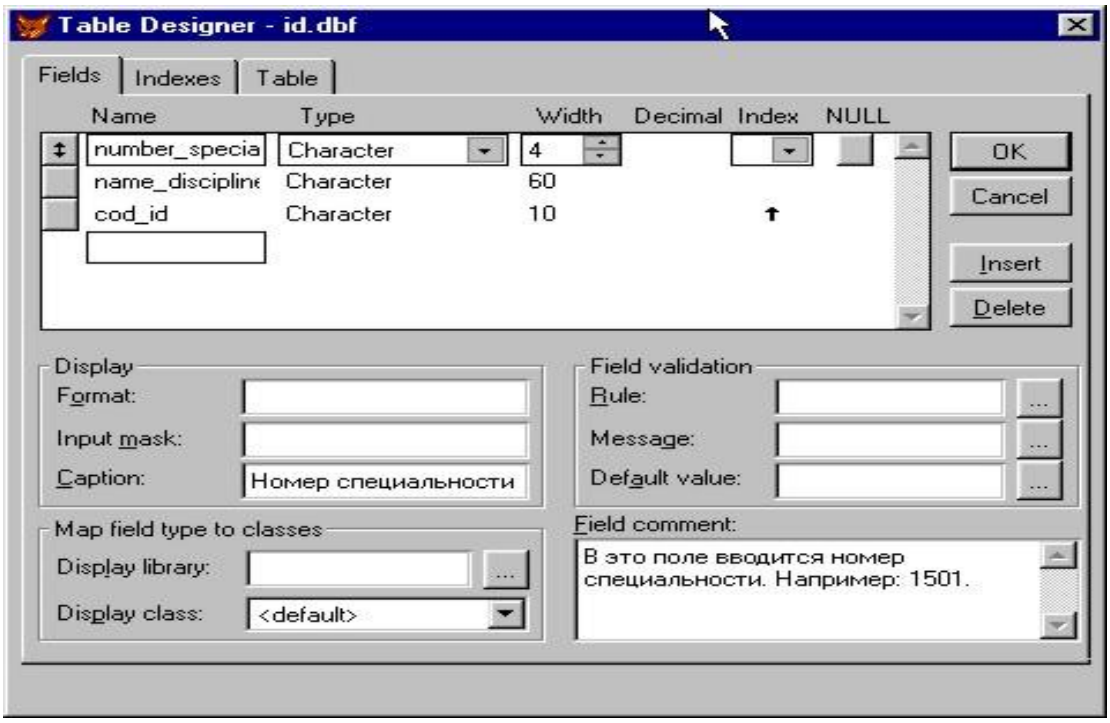


Рис.10.2. Вкладка Fields Конструктора таблиц с дополнительными областями

Поля таблицы предназначены для хранения данных: чисел, текстов, дат, графики и т. д. При определении полей таблицы используется вкладка Fields, позволяющая ввести наименование поля, тип размещаемых данных, ширину поля. Для числовых полей необходимо также задать количество десятичных знаков. Наименования полей таблицы вводятся в строке ввода столбца **Name** (Имя). При задании названий полей можно использовать только буквы, цифры и знак подчёркивания. Все попытки ввести специальные символы Visual FoxPro проигнорирует.

Для определения типа данных, размещаемых в поле, используется тип

поля, его ширина и количество знаков после запятой. Для их ввода предназначены столбцы **Type** (Тип), **Width** (Ширина) и **Decimal** (Десятичные) вкладки **Fields** Конструктора таблиц.

В Visual FoxPro используется достаточно широкий набор типов полей, краткая характеристика которых дана ниже.

Текстовый тип (**Character, Character binary**) – текстовые поля могут содержать буквы, цифры и специальные символы. Максимальная ширина поля составляет 254 символа. Тип Character binary используется в том случае, если не требуется учитывать кодовую страницу отображаемых данных.

Числовой тип (**Integer, Numeric, Float, Double**) – числовые поля содержат числа. Тип Integer отображает целые числа от –2147483647 до + 2147483647. Числовые поля типа Numeric и Float отображают данные с фиксированной точкой в диапазоне от $-0{,}9999999999*10^{+19}$ до $+0{,}9999999999*10^{+20}$. Тип данных Double используется для хранения данных с высокой точностью в диапазоне от $\pm 4{,}94065648541247*10^{-324}$ до $\pm 1{,}79769313486232*10^{+308}$.

Денежный тип (**Currency**) – в поле денежного типа могут содержаться числа от –922337203685477,5807 до 922337203685477,5807.

Тип данных Дата (**Date**) – в поле этого типа может содержаться любая дата от 01.01.0001 до 31.12.9999 года.

Тип данных Дата и Время (**DateTime**) – в поле типа DateTime может содержаться любая дата от 01.01.0001 до 31.12.9999 г. и время от 00:00:00 a.m. (до полудня) до 11:59:59 p.m. (после полудня).

Логический тип (**Logical**) – содержит логическое значение True (.T. - Истина) или False (.F. - Ложь).

Текстовое поле (**Memo, Memo binary**) – Memo-поле содержит символьные данные большого объёма.

Двоичное поле произвольной длины (**General**) – поле данного типа предназначено для хранения в таблицах изображений и других двоичных данных.

Для каждого поля можно определить признак, разрешающий при вводе данных оставлять это поле пустым. Для этого используется опция **NULL** в описании поля таблицы. Столбец **Index** используется для указания упорядочения записей по данным этого поля (по возрастанию или убыванию).

Ввод полей в окне Конструктора таблиц осуществляется последовательно. После определения всех необходимых параметров первого поля осуществляется переход на новую строку и определяются характеристики следующего поля таблицы.

На вкладке **Fields** справа расположены четыре кнопки. Кнопка **OK** предназначена для закрытия окна Конструктора таблиц и сохранения всех изменений, внесённых в структуру таблицы. Если структура таблицы была изменена, но от этого нужно отказаться, необходимо воспользоваться

кнопкой **Cancel** (Отмена). Для добавления в таблицу нового поля нужно установить курсор на поле, выше которого предполагается разместить новое поле, и нажать кнопку **Insert** (Вставить). Будет добавлена пустая строка, в которую можно ввести параметры нового поля. Для удаления поля таблицы необходимо перейти на строку с описанием данного поля и нажать кнопку **Delete** (Удалить).

В нижней части вкладки Fields Конструктора таблиц расположены области, позволяющие задать для каждого поля таблицы свойства, которые будут использоваться при вводе данных в эти поля.

Область **Display** (Отображение) - содержит поля, позволяющие задать форматы ввода и отображения данных:

- **Format** (Формат) задаёт формат отображения данных в формах, отчётах и окне Browse.

- **Input mask** (Маска ввода) задаёт формат ввода данных;

- **Caption** (Надпись) определяет выводимый заголовок поля.

Область **Map field type to classes** (Используемые типы полей для классов) предназначена для указания библиотеки и имени класса, который будет использоваться для создания объектов при размещении данного поля таблицы в форме:

- **Display library** (Показывать библиотеку) задаёт местоположение и имя файла библиотеки классов;

- **Display class** (Показывать класс) задаёт имя класса из выбранной библиотеки.

Область **Field Validation** (Проверка правильности ввода) позволяет задать следующие параметры:

- **Rule** (Условие) - условие правильности ввода данных;
- **Message** (Сообщение) - сообщение, выводимое при неправильном вводе данных в поле;
- **Default Value** (Значение по умолчанию) - значение, вводимое в поле по умолчанию.

В текстовом поле **Field comment** (Комментарий) можно ввести краткое описание поля, которое может потребоваться при последующих модификациях структуры таблицы и сопровождении проекта.

Среди требований, предъявляемых к СУБД, основное – это возможность быстрого поиска необходимой информации. Отличным средством, позволяющим решить эту проблему, является использование индексов, которые в зависимости от количества полей, используемых при создании, делятся на простые и составные. В отличие от многих реляционных баз данных при определении составного индекса в Visual FoxPro можно использовать не просто список индексных полей, а индексное выражение, при формировании которого используются поля таблицы.

Индексы в Visual FoxPro предназначаются также для создания первичных ключей, используемых при определении отношений между

таблицами и условий целостности данных. В этом случае индексы должны быть уникальными, т.е. если для создания уникального индекса используется простой индекс, значения поля должны быть неповторяющимися. Когда для создания уникального индекса используется составной индекс, значения в каждом из полей, входящих в индекс, могут повторяться, но, несмотря на это, индексное выражение должно быть уникальным.

Значения индексов хранятся в индексном файле, в котором для каждого индекса существует уникальная ссылка, указывающая на местонахождение записи, соответствующей индексу, что обеспечивает при поиске нужной информации прямой доступ к записи на основании упорядоченных значений индекса, а не последовательного просмотра всей таблицы. Такой подход ускоряет поиск данных в БД.

Вкладка Конструктора таблиц **Indexes** служит для задания типов в составном структурном индексном файле (рис.10.3). Этот файл (с расширением **.CDX**) создаётся и открывается автоматически вместе с таблицей БД. Можно задать новые и переопределить имеющиеся индексы в таблице.

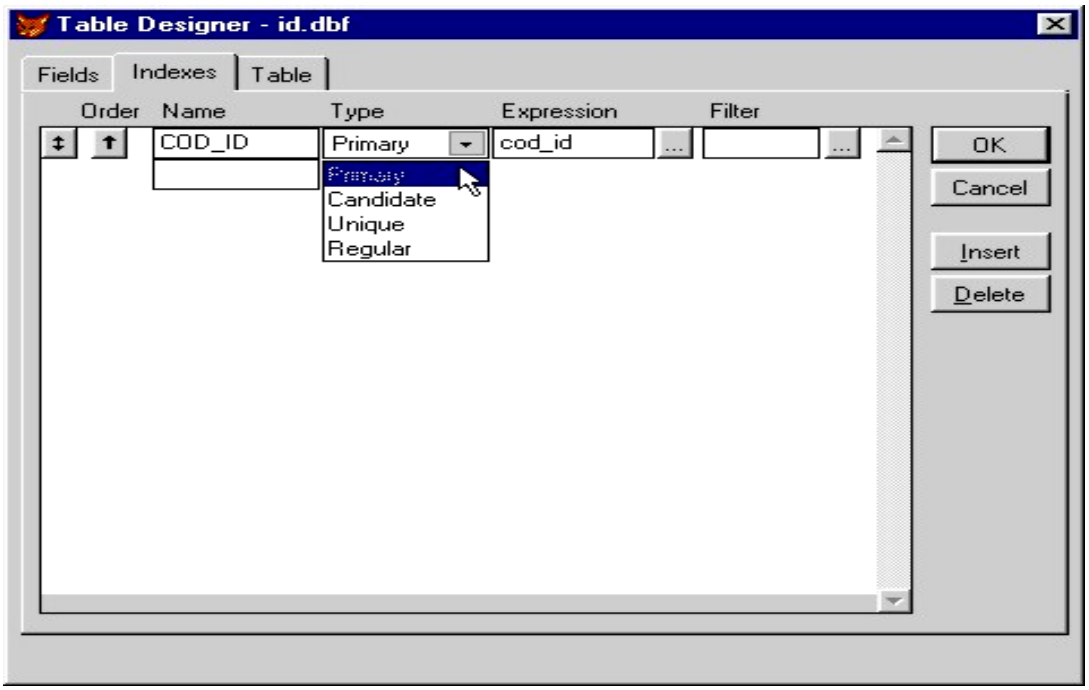


Рис.10.3. Вкладка Indexes Конструктора таблиц

В таблице БД в отличие от свободной таблицы можно задать первичный ключ - **Primary Key**. При этом нужно помнить, что поле может использоваться для задания первичного ключа таблицы только в том случае, если оно содержит неповторяющиеся значения. С помощью кнопок **Insert** и **Delete** добавляют новые индексы в таблицу и удаляют существующие.

Все индексы в Visual FoxPro имеют имена, задаваемые в поле **Name**

(Имя). Слева от имени индекса в столбце **Order** (Упорядочение) располагается переключатель, определяющий как будут упорядочены значения индексного выражения. По умолчанию при создании индекса в данном поле появляется стрелка, направленная вверх, которая показывает, что значения индексного выражения упорядочены по возрастанию. Если стрелка направлена вниз, это говорит о том, что значения упорядочены по убыванию. Для изменения способа упорядочения можно использовать клавишу **Spacebar** или щелкнуть мышью на стрелке.

Список **Type** (Тип) используется для задания типа создаваемого индекса и содержит следующие значения:

- **Primary** (Первичный) – создаётся уникальный индекс, который используется для связывания таблиц и определения условий целостности данных. Поля, входящие в первичный ключ, не должны допускать ввода пустых значений. Таблица может иметь первичный ключ, если она входит в состав БД. Первичный ключ в таблице может быть только один.
- **Candidate** (Кандидат) – создаётся уникальный индекс, в том числе и в свободной таблице, который не содержит полей с пустыми значениями. Этот индекс обладает всеми качествами первичного ключа и не является им только по той причине, что таблица не может содержать более одного первичного ключа. Количество индексов типа Candidate в одной таблице не ограничено.
- **Regular** (Обычный) – создаётся индекс, в котором для каждой записи таблицы хранится значение индексного выражения. Если несколько записей имеют одинаковое значение индексного выражения, то каждое значение хранится отдельно и содержит ссылку на связанную с ней запись
- **Unique** (Уникальный) – создаётся индекс, в котором хранятся только неповторяющиеся значения индексного выражения. Если две или более записей содержат одинаковое значение индексного выражения, то будет храниться только одно значение и ссылка на первую из записей с одинаковым значением индексного выражения. Таблица может иметь несколько индексов Unique.

Значение индекса или индексного выражения вводится в поле **Expression** (Выражение). Можно ввести индексное выражение непосредственно в поле ввода или для формирования выражения использовать диалоговое окно конструктора выражений **Expression Builder** (Построитель выражения), для запуска которого необходимо нажать кнопку, расположенную справа от поля Expression.

Для определения свойств самой таблицы предназначена вкладка **Table** (Таблица) конструктора. Для таблиц, входящих в базу данных, можно задать два имени. Одно вводится в диалоговом окне **Create**, а второе – на вкладке **Table** окна Конструктора таблиц. Имя, вводимое в диалоговом окне Create при создании таблицы, и будет именем файла, в котором таблица сохраняется на диске. При задании этого имени необходимо придерживаться

ограничений, накладываемых операционной системой на количество символов в имени файла. Наименование таблицы может содержать буквы, цифры и знак подчёркивания. Создавая новую таблицу, необходимо помнить, что в базе данных не может быть двух таблиц с одинаковыми именами.

Второе имя таблицы является внутренним и хранится в базе данных. Внутреннее имя таблицы может содержать до 128 символов. Оно вводится в поле **Name** на вкладке **Table** в верхней части окна Конструктора таблиц. Это имя будет отображаться в окне проекта, а также использоваться при создании форм, запросов и отчётов. Используя поле **Table Comment** вкладки Table, можно ввести описание таблицы. Для определения условия проверки правильности ввода информации на уровне записей, гарантирующих достоверность данных, вводимых в таблицу, используются поля области **Record validation**. Для задания правила проверки значения записи при вставке, удалении, изменении данных таблицы предназначены поля области **Triggers**. Эти правила называются триггерами вставки, удаления, изменения, соответственно.

Структуру таблицы, созданную с помощью Мастера таблиц или Конструктора таблиц, можно модифицировать, то есть изменить наименование любого поля и его тип, вставить новое поле или удалить существующее, изменить порядок следования полей в таблице. Чтобы модифицировать таблицу, её нужно открыть в Конструкторе таблиц.

Для этого в меню **File** (Файл) выбирается команда **Open** (Открыть). В появившемся окне Open в поле **Тип файлов** установить тип открываемого файла *.DBF. Убедившись, что в раскрывающемся списке **Папка** правильно указана папка, в которой располагается таблица, выделить модифицируемую таблицу и нажать кнопку **ОК.** В результате выполнения этих действий откроется указанная таблица данных, а в системном меню **View** появится команда **Table Designer**. Выбор этой команды приведет к открытию окна Конструктора таблиц - Table Designer со структурой данной таблицы.

Открыть окно Конструктора таблиц со структурой активной (предварительно открытой) таблицы можно также и через командное окно, используя команду MODIFY STRUCTURE. В результате на экране открывается диалоговое окно **Table Designer** (Конструктор таблиц), содержащее структуру модифицируемой таблицы.

10.4. ВИЗУАЛЬНОЕ ПРОЕКТИРОВАНИЕ ЗАПРОСОВ

Запросы являются мощным средством управления данными. С помощью запросов можно просматривать, анализировать и изменять данные из нескольких таблиц. Они также используются в качестве источника данных для форм и отчетов. С их помощью возможно отображение данных из нескольких таблиц, сортировка в нужном порядке, выполнение нужных вычислений над группами записей и вывод полученых результатов в виде

таблицы, отчёта, ярлыка и т.д.

Для создания запросов в Visual FoxPro используется язык управления данными SQL, являющийся стандартом для работы с реляционными базами данных в архитектуре клиент-сервер.

Наиболее часто используется запрос на выборку (SQL-команда SELECT). При его выполнении данные, удовлетворяющие условиям отбора, выбираются из одной или нескольких таблиц и выводятся в определенном порядке. Запрос на выборку возвращает данные из одной или нескольких таблиц, а также результаты, которые при желании пользователь может изменить (с некоторыми ограничениями). Также можно использовать запрос на выборку, чтобы сгруппировать записи для вычисления сумм, средних значений, пересчета и других действий.

Реализация SQL в FoxPro описана в главе 7. Но в Visual FoxPro существуют специальные визуальные средства, облегчающие создание запросов на выборку: **Мастер запросов** (Query Wizard) и **Конструктор запросов** (Query Designer).

Для создания запроса с помощью Конструктора запросов Visual FoxPro в меню File (Файл) выбирается команда New (Новый). В открывшемся диалоговом окне New необходимо установить опцию Query (Запрос) и нажать кнопку New file (Новый файл). Откроется окно Query Designer - Конструктор запросов (рис.10.4).

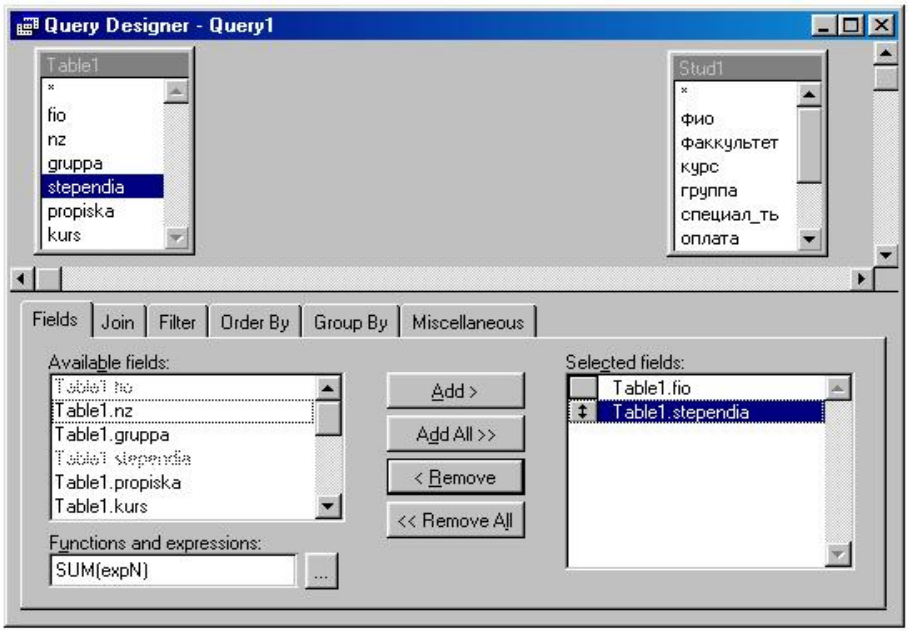


Рис.10.4. Окно Конструктора запросов с выбранной базой данных

В окне Конструктора запросов первоначально отсутствуют источники данных для запроса. После открытия Конструктора запросов появляется окно Add Table or View (добавить таблицу или представление), где можно выбрать базу данных, свободную таблицу или представление, из которых будут выбираться данные в запрос. После выбора таблиц, представлений или базы

данных они появляются в окне Query Designer. На рис.10.4 представлено окно Конструктора запросов с выбранной базой данных, имеющей две таблицы данных.

Окно **Query Designer** имеет своё меню и панель инструментов **Query Designer**, с помощью которых можно добавить таблицы в окно Конструктора запроса, создать запрос, сохранить его в виде программного файла с расширением .**QPR**, включить в запрос различные выражения и функции, указать все опции команды **SELECT**. На рис.10.5 представлена панель инструментов **Query Designer** и назначение кнопок панели.



- **Add table** (добавить таблицу) открывает окно Add Table or View.
- **Remove table** (вернуть таблицу) возвращает таблицу из списка выбранных.
- **Add join** (добавить связь) для описания связей между таблицами
- **Show the SQL window** (показать окно языка управления данными SQL).
- **Maximize the table view** (убирает вкладки с окна Конструктора запросов).
- **Query Destination** позволяет выбрать получателя результатов запроса.

Рис.10.5. Панель инструментов Query Designer

Используя последнюю кнопку (Query Destination) панели инструментов Query Designer можно через диалоговое окно Query Destination (рис.10.6) выбрать получателя данных выборки: виртуальную таблицу (**Cursor**), физическую таблицу (**Table**), экран монитора (**Screen**), отчёт (**Report**), ярлык (**Label**). По умолчанию результаты выполнения запроса выводятся в режиме **Browse**.

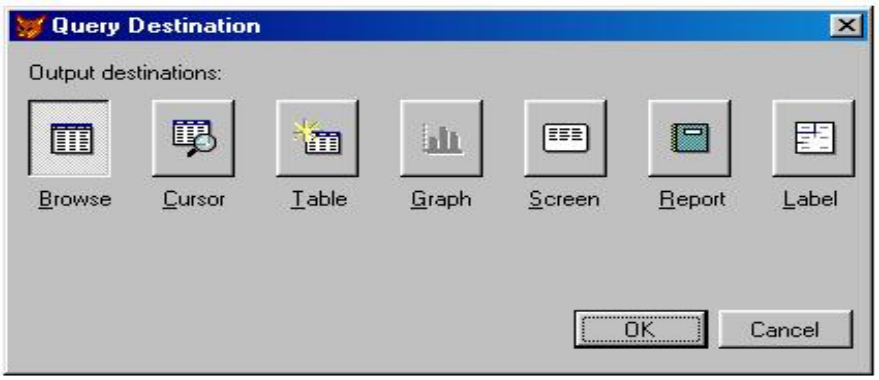


Рис.10.6. Окно Query Destination

Окно Конструктора Запросов имеет несколько вкладок, которые служат для визуального задания опций команды SELECT. Ниже

рассматривается назначение этих вкладок и работа с ними.

Вкладка **Fields** (рис.10.4) позволяет задать имена полей, данные из которых выбираются в запрос.В левом списке доступных полей таблицы (Available fields) выделяется необходимое поле, а кнопка **Add** (добавить) позволяет это поле перенести в правый список выбранных полей (Selected fields). Кнопка **Add All** (добавить все) используется для выбора всех полей таблицы. Кнопка **Remove** (вернуть) - для удаления столбца из списка выбранных полей. Кнопка **Remove All** (вернуть все) возвращает все выбранные столбцы из списка выбранных полей. В поле **Functions and expressions** формируются вычисляемые выражения как вручную, так и с помощью **Построителя Выражений** (Expression Builder). При этом можно использовать различные функции и выражения. В выражении можно указать все стандартные функции Visual FoxPro, являющиеся функциями языка программирования xBase (см. главу 6), а также собственные функции SQL для работы со столбцами. Кнопка **Add** (добавить) позволяет это выражение добавить в правый список выбранных полей.

Вкладка **Join** (рис.10.7) позволяет описать связи между таблицами по какому-либо критерию (условию).

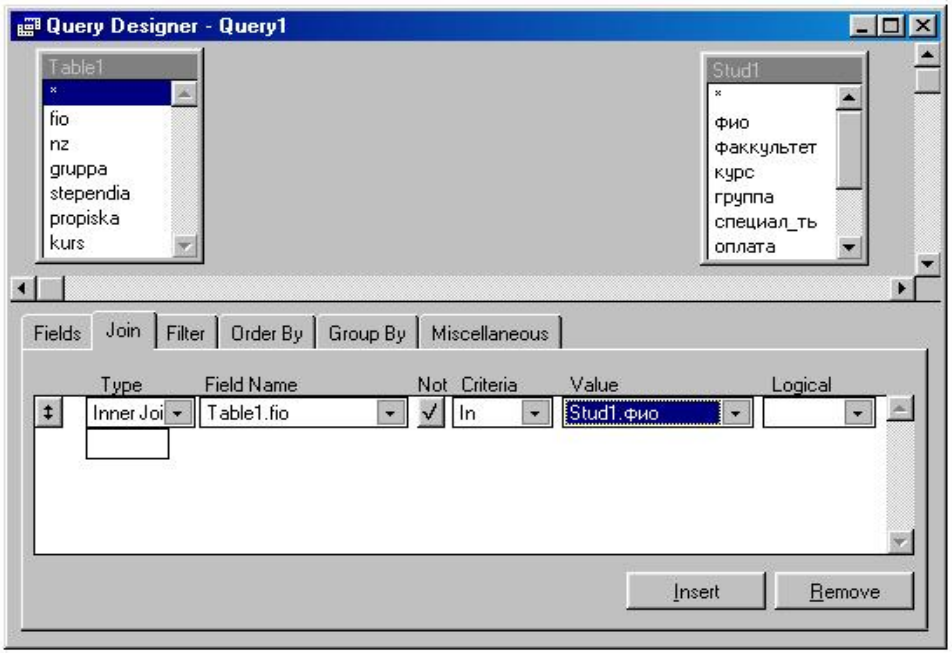


Рис.10.7. Вкладка Join окна Конструктора запросов

Возможны следующие типы связей (**Type**):

- **Inner Join** – эта связь позволяет включить в выборку только те значения из первой и второй таблицы, которые удовлетворяют указанному условию;
- **Left Outer Join** – эта связь позволяет включить в выборку записи из первой и второй таблицы удовлетворяющие указанному условию и записи не удовлетворяющие указанному условию из первой таблицы;

- **Right Outer Join** – эта связь позволяет включить в выборку записи из первой и второй таблицы, удовлетворяющие указанному условию, и записи, не удовлетворяющие указанному условию, из второй таблицы;
- **Full Outer Join** – эта связь позволяет включить в выборку все записи, удовлетворяющие и не удовлетворяющие указанному условию.

Также во вкладке **Join** указывается имя поля (**Field Name**), по которому будет происходить связь, и условие - критерий связи (**Criteria**): **=, ==, LIKE** (похожий), >, <, >=, <=, **BETWEEN** (между), **IN** (в) и др. Указывается также значение критерия (**Value**) и логическая операция (**Logical**) для объединения простых условий в сложные: **And** (И), **Or** (ИЛИ).

Вкладка **Filter** позволяет установить условия выборки - критерий отбора данных в запрос, поле или выражение, по которому будет происходить выборка: **=, ==, LIKE** (похожий), >, <, >=, <=, **BETWEEN** (между), **IN** (в) и др. Указывается также значение критерия (**Value**) и логическая операция (**Logical**) для объединения простых условий в сложные: **And** (И), **Or** (ИЛИ). В выражении можно указать все стандартные функции Visual FoxPro, являющиеся функциями языка программирования xBase (см. главу 6), а также собственные функции SQL для работы со столбцами.

Вкладка **Order By** (упорядочить по) позволяет указать поля, по которым необходимо выполнить упорядочение данных, а также тип упорядочивания: **Ascending** (по возрастанию), **Descending** (по убыванию).

Вкладка **Group By** (группировать по) используется для задания поля, по которому необходимо выполнить группировку. Кнопка **Having** этой вкладки используется для выбора способа группировки, указания имени поля, из которого будут выбираться данные для группировки и различных условий группировки.

Вкладка **Miscellaneous** позволяет задать дополнительные условия отбора (не допускать повторения) и установить количество записей, попадающих в выборку.

Если щёлкнуть правой кнопкой мыши в окне Конструктора запросов, то появится контекстно зависимое меню, где можно:

- выбрав команду **Run Query,** запустить на выполнение подготовленный запрос;
- командой **View SQL** просмотреть созданный запрос на языке управления реляционными базами данных SQL;
- командой **Remove Table** вернуть назад выбранную таблицу;
- командой **Add Table** добавить таблицу в окно Конструктора запросов;
- командой **Output Settings** установить получателя результатов запроса;
- командой **Help** просмотреть справочную информацию.

Сохранённый запрос можно запустить из командного окна или из программного файла командой: **DO <имя запроса> .QPR**.

10. 5. ВИЗУАЛЬНОЕ ПРОЕКТИРОВАНИЕ ФОРМ

Для просмотра, ввода и редактирования данных, хранящихся в таблицах, используются формы, являющиеся наглядным средством представления информации. Формы, предназначенные для ввода документов, должны выглядеть на экране точно так же, как стандартные бланки этих документов. Важным преимуществом форм является и то, что они позволяют работать не с одной, а с несколькими связанными таблицами.

Пользователю приложения нет необходимости знать, что такое СУБД, какие команды используются для добавления или удаления записей в таблицах. Он может вообще не знать, с использованием каких программных средств создавалось приложение. Для него главным является поиск и просмотр данных, перемещение по таблице, добавление новых записей, редактирование и удаление имеющихся. Все эти возможности предоставляют формы. При создании форм в Visual FoxPro можно использовать **Мастер форм** (Form Wizard) или **Конструктор форм** (Form Designer).

Мастер форм обеспечивает построение стандартных форм для работы с таблицами данных путем использования диалоговых окон для уточнения требований пользователя. Для создания формы с помощью мастера форм необходимо выполнить команду **New** (Новый) из меню **File** (Файл) или нажать кнопку **New** на стандартной панели инструментов Visual FoxPro. В открывшемся диалоговом окне **New** выбрать опцию **Form** (Форма) и нажать кнопку **Wizard** (Мастер). В каждом из открывающихся окон мастера форм указываются запрашиваемые параметры для формы, после чего нажимается кнопка **Next** для перехода к следующему шагу (окну) построения формы. На последнем шаге создания формы нажатием кнопки **Finish** подтверждается окончание формирования формы. Форма сохраняется в файле с заданным именем и расширением **.SCX** (**SCT**).

Процесс создания формы в Конструкторе форм состоит в размещении в форме объектов и определении свойств, а также связанных с ними событий и выполняемых действий. Для открытия окна Конструктора форм при создании формы необходимо выполнить команду **New** (Новый) из меню **File** (Файл) или нажать кнопку **New** на стандартной панели инструментов Visual FoxPro. В открывшемся диалоговом окне **New** выбрать опцию Form (Форма) и нажать кнопку **New File** (Новый файл).

Окно Конструктора форм показано на рис.10.8. Оно содержит панели инструментов **Color Palette** (Цветовая палитра), **Layout** (Расположение), **Form Designer** (Конструктор форм) и **Form Controls** (Элементы управления формы), используемые при работе в конструкторе. В окне конструктора находится и новая форма (Form1), с которой можно начинать работать. Если необходимые панели инструментов отсутствуют, для их отображения на экране в меню **View** (Вид) нужно выбрать команду **Toolbars** (Панель инструментов). Откроется диалоговое окно **Toolbars**, в котором необходимо установить флажки выбора соответствующих панелей инструментов.

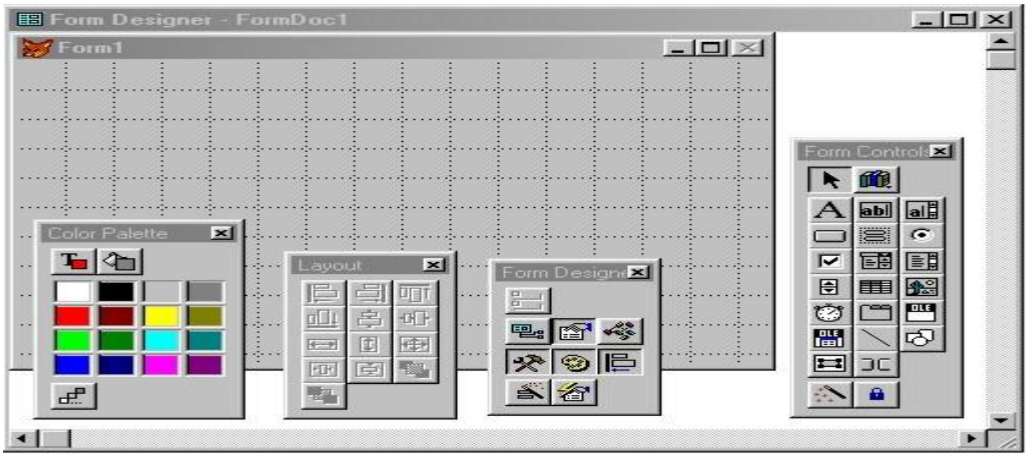


Рис.10.8. Окно Конструктора форм с панелями инструментов

Панель инструментов **Form Designer** (Конструктор форм) содержит кнопки вызова панелей инструментов Form Controls (Элементы управления формы), Color Palette (Цветовая палитра), Layout (Расположение). С помощью панели **Form Designer** можно выполнять и некоторые дополнительные действия по управлению формой. Краткое назначение кнопок данной панели инструментов приводится ниже:

- **Set Tab Order** (Порядок объектов) переключает Конструктор форм в режим установления порядка обхода объектов формы;
- **Data Environment** (Окружение данных) открывает окно определения среды окружения формы;
- **Code Window** (окно кода) открывает окно просмотра исходного кода формы;
- **Color Palette Toolbar** (Панель инструментов **Цветовая палитра**) отображает на экране панель инструментов Color Palette;
- **Form Builder** (Построитель форм) вызывает построитель формы;
- **Properties Window** (Окно свойств) открывает на экране окно свойств объектов формы;
- **Forms Controls Toolbar** (Панель инструментов **Элементы управления формы**) вызывает на экран панель инструментов Form Controls;
- **Layout** (Расположение) вызывает на экран панель инструментов Layout;
- **Auto Format** (Автоформат) вызывает построитель автоформата для выбранных объектов формы.

Панель инструментов **Form Controls** (Элементы управления формы) используется для размещения в форме объектов. Краткое описание кнопок этой панели приводится ниже:

- **Select Objects** (Выбор объектов) позволяет выбирать в форме объекты;

 - **View Classes** (Просмотр классов) позволяет выбрать класс для создаваемых в форме объектов;

 **- Label** (Метка) создаёт в форме текстовый объект;

 - **TextBox (**Поле ввода) создаёт в форме поле ввода;

 - **EditBox** (Поле редактирования) создаёт в форме поле редактирования;

 - **CommandButton** (Кнопка) создаёт в форме кнопку управления;

 - **OptionGroup** (Переключатель) создаёт в форме зависимый переключатель;

 - **CheckBox** (Флажок) создаёт в форме флажок;

 - **Grid** (Таблица) создаёт в форме объект в виде таблицы для размещения полей таблицы данных;

 - **ComboBox** (Раскрывающийся список) создаёт в форме раскрывающийся список;

 - **ListBox** (Список) создаёт в форме список;

 - **Spinner** (Счётчик) создаёт в форме поле ввода значения в виде счётчика;

 - **Line** (Линия) создаёт в форме линию;

 **- Shape** (Контур) создаёт в форме контур;

 - **Container** (Контейнер) создаёт в форме контейнер;

 - **Image** (Изображение) размещает в форме рисунок;

 - **CommandGroup** (Группа кнопок) размещает в форме группу кнопок;

 - **Timer** (Таймер) создаёт в форме объект типа таймера;

 - **PageFrame** (Вкладка) размещает в форме страницы с вкладками;

 - **ActiveX Bound Control** (OleBoundControl – ActiveX объект) отображает содержимое OLE-объекта, хранящегося в поле типа General;

 - **ActiveX Control** (OleControl - ActiveX объект) создаёт OLE-объект;

 - **Separator** (Разделитель) размещает на панели инструментов разделитель кнопок;

 - **Builder Lock** (Закрепитель построителя) закрепляет выбор построителя;

 - **Button Lock** (Закрепитель кнопки) закрепляет выбранную кнопку на панели инструментов.

Для размещения в форме указанных выше объектов на панели инструментов **Form Controls** выбирается щелчком мыши необходимая кнопка (она переходит в нажатое состояние). Курсор мыши переводят на форму, в в левом верхнем углу предполагаемого размещения объекта

выполняется щелчок мыши, который оканчивается в правом нижнем углу размещения объекта, после чего на форме появляется соответствующий объект. Для работы с объектом его необходимо выделить щелчком мыши. Для выделенного объекта можно установить свойства и методы, используя мышь можно изменить его размеры и местоположение. Для выравнивания объектов, размещённых в форме, удобно использовать панель инструментов Layout (Расположение) со следующими инструментами:

 - **Align Left Sides** (По левому краю) выравнивает выбранные объекты по левому краю самого левого объекта;

 - **Align Top Edges** (По верхнему краю) выравнивает выбранные объекты по верхнему краю самого верхнего объекта;

 - **Align Right Sides** (По правому краю) выравнивает выбранные объекты по правому краю самого правого объекта;

 - **Align Bottom Edges** (По нижнему краю) выравнивает выбранные объекты по нижнему краю самого нижнего объекта;

 - **Align Vertical Center** (По вертикали) выравнивает выбранные объекты по вертикали;

 - **Align Horizontal Center** (По горизонтали) выравнивает выбранные объекты по горизонтали;

 - **Center Vertically** (Относительно вертикали, проходящей через центр) центрирует выбранные объекты относительно вертикальной средней линии формы;

 - **Center Horizontally** (Относительно горизонтали, проходящей через центр) центрирует выбранные объекты относительно горизонтальной средней линии формы;

 - **Same Width** (Одинаковая ширина) устанавливает одинаковую ширину для выбранных объектов формы;

 - **Same Size** (Одинаковый размер) устанавливает одинаковую ширину и высоту для выбранных объектов формы;

 - **Same Height** (Одинаковая высота) устанавливает одинаковую высоту для выбранных объектов формы;

 - **Send to Back** (Позади) направляет выбранный объект на самый нижний слой формы;

 - **Bring to Form** (Поверх) направляет выбранный объект на самый верхний слой формы.

Процесс создания формы состоит из следующих действий:

- Настройка параметров формы;
- Определение среды окружения, то есть выбор используемых в форме таблиц и установка связей между ними;

- Размещение в форме объектов: текста, полей различных типов, линий, рисунков, кнопок управления и т.д.;
- Настройка свойств размещённых в форме объектов.

Форма, как и все располагаемые в ней объекты, имеет свойства, используя которые можно задать её размер, координаты верхнего левого угла, стиль рамки обрамления, заголовок, цвет и т. д. Настройка параметров формы осуществляется в окне свойств **Properties** (Свойства), для открытия которого нужно установить курсор на свободную от объектов поверхность формы и выбрать команду **Properties** (Свойства) из меню **View** (Вид). На рис.10.9 показано главное окно Visual FoxPro с панелями инструментов и открытым окном Конструктора форм, для которого открыты окно свойств **Properties** и окно среды окружения **Data Environment**.

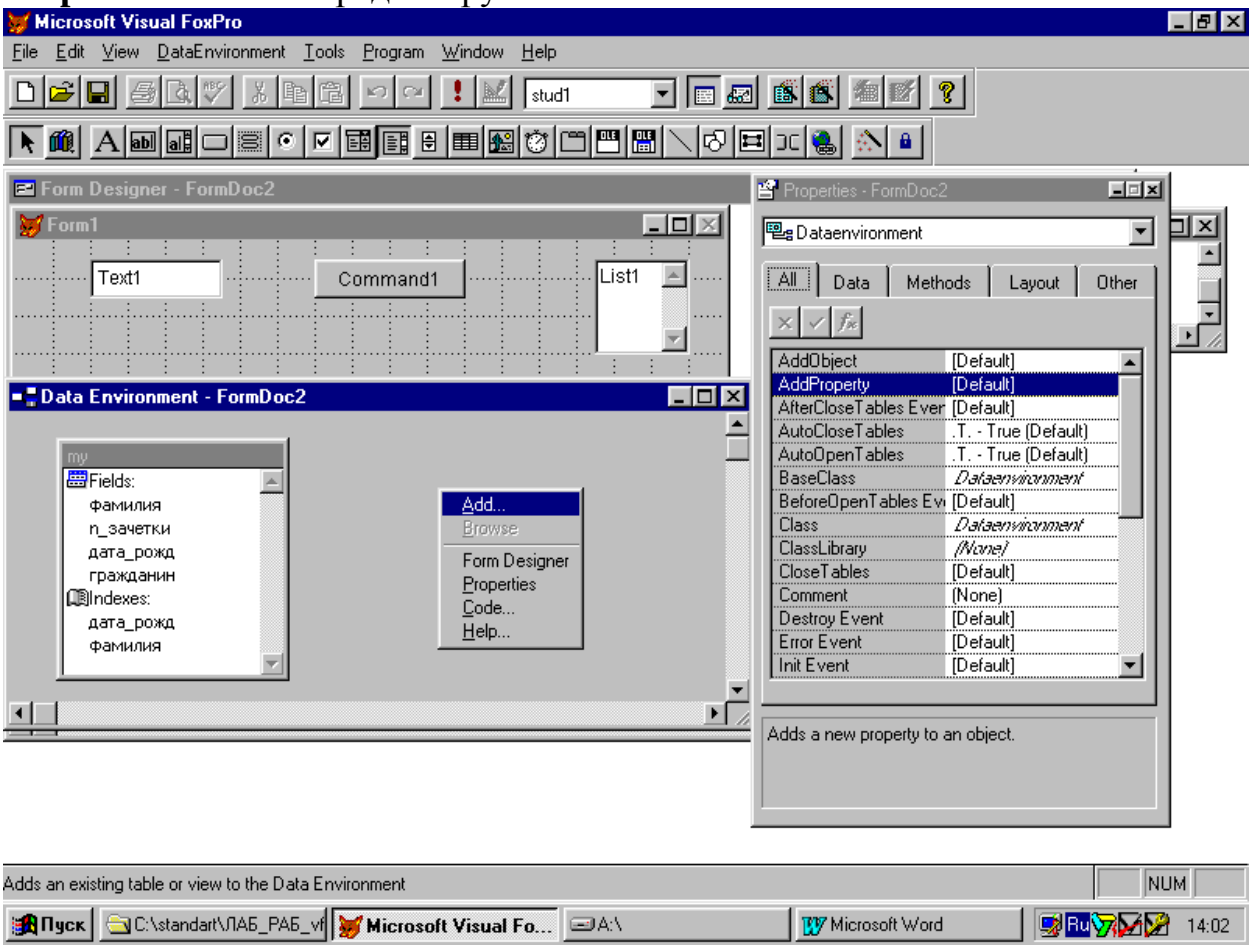


Рис.10.9. Окно Конструктора форм, окно Data Environment и окно Properties

После создания форма, по умолчанию, располагается в верхнем левом углу главного окна Visual FoxPro. Для изменения её положения можно использовать свойства **Left** (Левый) и **Top** (Верхний), указывающие расстояние в пикселях от левого и верхнего края, соответственно, а также мышь. Если использовать мышь для изменения положения формы, то нужно установить курсор на заголовок формы, нажать левую кнопку мыши и, удерживая её, переместить форму в окне Конструктора форм в место её

предполагаемого расположения. Для размещения формы в центре главного окна Visual FoxPro необходимо в окне свойств установить для свойств **AutoCenter** (Автоцентр) значение **True** (Истина).

Для задания текста заголовка, располагающегося в верхней части формы, предназначено свойство **Caption** (Надпись) окна свойств. Чтобы отредактировать заголовок, нужно открыть окно Properties, выделить свойство Caption и в поле ввода, ставшее активным, ввести с клавиатуры заголовок формы. Если появится необходимость, чтобы форма вообще не содержала заголовок, то необходимо установить для свойства **TitleBar** (Строка заголовка) значение **Off.** Свойство **Name** используется для задания программного имени объекта (в том числе и формы), по которому можно обратиться к данному объекту.

Стиль обрамления формы можно задать с помощью свойства **BorderStyle** (Стиль рамки), он может принимать следующие значения:

0 – **No Border** (Нет рамки), форма не имеет рамки;

1 – **Fixed Single** (Одинарная рамка), неизменяемая одинарная рамка;

2 – **Fixed Dialog** (Двойная рамка), неизменяемая двойная рамка;

3 – **Sizable** (Default) (Изменяемая), изменяемая рамка (размеры формы можно изменять при выполнении).

После создания форма по умолчанию располагается в верхнем левом углу главного окна Visual FoxPro. Для изменения её положения можно использовать свойства **Left** (Левый) и **Top** (Верхний), указывающие расстояние в пикселях от левого и верхнего края, соответственно, а также мышь. Если использовать мышь для изменения положения формы, то нужно установить курсор на заголовок формы, нажать левую кнопку мыши и, удерживая её, переместить форму в окне конструктора в место её предполагаемого расположения. Для размещения формы в центре главного окна Visual FoxPro необходимо в окне свойств установить для свойств **AutoCenter** (Автоцентр) значение **True** (Истина).

Для задания текста заголовка, располагающегося в верхней части формы, предназначено свойство **Caption** (Надпись) окна свойств. Чтобы отредактировать заголовок, нужно открыть окно Properties, выделить свойство Caption и в поле ввода, ставшее активным, ввести с клавиатуры заголовок формы. Если появится необходимость, чтобы форма вообще не содержала заголовок, то необходимо установить для свойства **TitleBar** (Строка заголовка) значение **Off.** Свойство **Name** используется для задания программного имени объекта (в том числе и формы), по которому можно обратиться к данному объекту.

Стиль обрамления формы можно задать с помощью свойства **BorderStyle** (Стиль рамки), он может принимать следующие значения:

0 – **No Border** (Нет рамки), форма не имеет рамки;

1 – **Fixed Single** (Одинарная рамка), неизменяемая одинарная рамка;

2 – **Fixed Dialog** (Двойная рамка), неизменяемая двойная рамка;

3 – Sizable (Default) (Изменяемая), изменяемая рамка (размеры формы можно изменять при выполнении).

При помощи свойства **BackColor** (Цвет фона) задается цвет фона формы.

Свойство **WindowState** (Состояние окна) определяет размер формы при её вызове и может принимать одно из следующих значений:

- **Normal** (Обычное) – форма имеет размеры, определённые её свойствами;
- **Minimized** (Windows only) – форма сворачивается в значок;
- **Maximized** (Максимизированное) – форма разворачивается на весь экран.

При создании формы, предназначенной для редактирования или просмотра данных таблиц, в Конструкторе форм необходимо определить среду окружения, т.е. задать таблицы, используемые в форме, и установить связи между ними. Следует учесть, что при создании форм с помощью мастера и размещении объектов в форме с помощью построителя, среда окружения создаётся Visual FoxPro автоматически без участия разработчика.

При определении среды окружения нужно выполнить следующие действия: добавить таблицы и представления, используемые в форме; установить индексы для таблиц; установить для таблицы отношения, необходимые для создания формы. Вся эта информация, относящаяся к среде окружения, хранится в файле описания формы. Для создания среды окружения формы предназначено диалоговое окно **Data Environment** (Среда окружения) (рис.10.11), открыть которое можно выбрав команду **Data Environment** из меню **View** (или из контекстного меню формы**)** или нажав кнопку **Data Environment**  на панели инструментов **Form Designer** (Конструктор форм).

При открытии окна среды окружения **Data Environment** в основное меню добавляется пункт **Data Environment**. Для работы в окне Data Environment можно использовать команды из меню Data Environment или контекстного меню, позволяющие добавить в окружение таблицы, просмотреть их в режиме Browse, открыть окно свойств окружения для задания различных параметров.

Для добавления таблицы в среду окружения необходимо выбрать команду контекстного меню **Add** или выбрать команду **Add** из меню **Data Environment**. При этом откроется диалоговое окно **Add Table or View** (Добавить таблицу или представление данных), содержащее список таблиц открытой базы данных. В поле **Database** выбирается база данных, а в поле **Tables in database** находится таблица, которую следует выделить и нажимается кнопка **Add**. При этом выбранная таблица размещается в среде окружения формы. Кнопка **Close** позволяет закрыть диалоговое окно. Кнопка **Other** открывает окно **Open**, с помощью которого, можно добавить отсутствующую базу данных и таблицы. Опция **Views** (Представления

данных) области **Select** (Выбор) позволяет разместить в среде окружения представления данных, созданные в базе данных.

Разместить на форме данные из полей таблицы (или представления) можно путем перетаскивания из окна Data Environment на форму соответствующих полей таблицы (представления) (при нажатой правой клавише мыши). При этом на форме автоматически размещаются объекты редактирования и просмотра данных таблиц (**TextBox, EditBox, ListBox, ComboBox** и т.д.), связанные с соответствующими полями таблиц.

Если на форме уже размещены объекты, предназначенные для редактирования и просмотра данных таблиц, то необходимо их связать с соответствующими полями таблиц, указав источник данных через окно **Properties.** Сохранение формы в файле осуществляется командой **Save as** меню **File**, запуск формы на выполнение осуществляется командой меню **DO FORM <имя формы>.**

10.6. ВИЗУАЛЬНОЕ ПРОЕКТИРОВАНИЕ МЕНЮ

Меню в прикладной программе – это первое, что видит пользователь, решив запустить приложение. Чаще всего используется многоуровневое иерархическое меню. В любом приложении рекомендуется иметь строку меню. Строка меню в приложении Visual FoxPro содержит команды, предназначенные для вызова форм, формирования отчётов, запросов и т. д. Строкой меню называется горизонтальное меню, располагаемое в верхней части экрана, являющееся главным меню в случае многоуровневого меню. Примерами меню являются основное меню Visual FoxPro и меню программ, работающих в среде Windows. Созданное меню может замещать основное меню Visual FoxPro или добавляться к нему.

При разработке меню особое внимание следует обратить на то, чтобы заголовок меню содержал явную информацию о его назначении. При этом желательно использовать общеупотребительные термины и стараться не использовать легко забываемые сокращения. Функционально связанные группы команд нужно выделять с помощью разделителей. Для часто используемых команд в меню нужно задать “горячие клавиши”. Это позволит пользователю выполнять эти команды, не отрывая рук от клавиатуры.

Обычно при разработке приложения создаются все требуемые компоненты приложения - база данных, входящие в неё таблицы, представления, формы, отчёты, запросы и т.д., которые затем объединяются с помощью меню. В качестве визуального инструментария создания меню используется Конструктор меню - **Menu Designer**.

Для создания меню необходимо выполнить следующие действия:

- открыть окно Конструктора меню;
- описать вид меню, текст, пункты меню и его атрибуты;

- определить действия, которые будут выполняться при выборе пунктов меню;
- сгенерировать меню, используя команду Generate (Генерация) из меню Menu.

При этом создаётся программа, которую в результате пользователь и запускает на выполнение.

Для открытия окна Конструктора меню можно воспользоваться системным меню Visual FoxPro. В меню **File** (Файл) выбрать команду **New** (Новый). В открывшемся диалоговом окне **New** установить опцию **Menu** (Меню) и нажать кнопку **New File** (Новый файл) (рис.10.10). После выполнения действий этого пункта создания меню появится диалоговое окно **New Menu** (рис.10.10). Кнопка **Menu** диалогового окна **New Menu** для выбора типа меню предназначена для создания меню в виде строки, им можно заменить главное меню Visual FoxPro или добавить свои элементы в него, также меню можно сделать главным элементом приложения, установив атрибут меню в **состояние Set Main** (см.10.7).

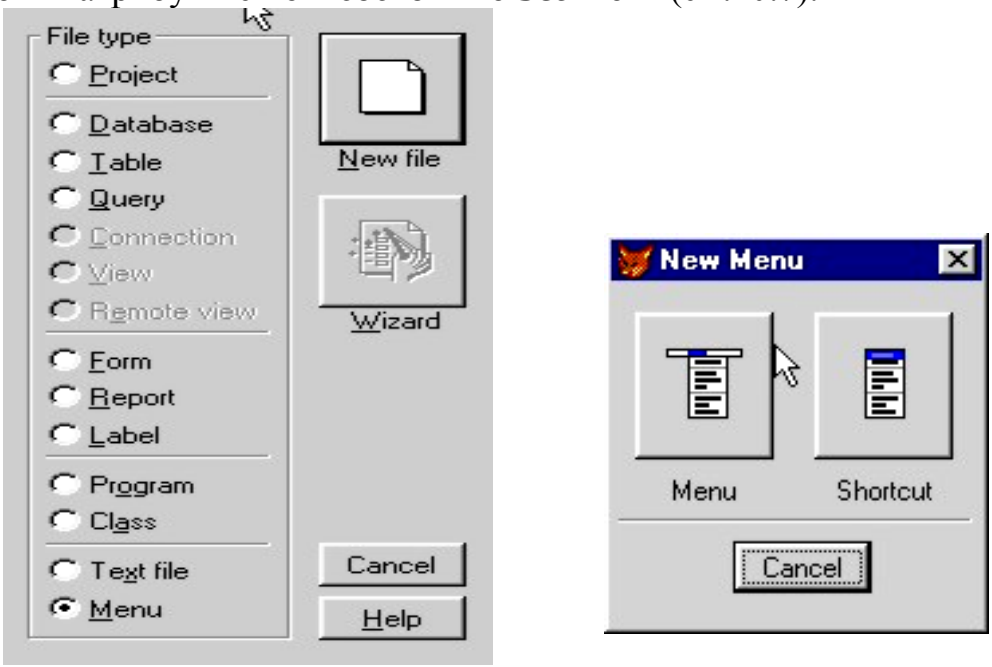


Рис.10.10. Окно New для выбора создаваемого элемента и окно New Menu для выбора типа меню

Кнопка **Shortcut** позволяет создать всплывающее меню. Вызвать всплывающее меню можно, щёлкнув правой кнопкой в нужном месте формы. Способ создания этого меню аналогичен созданию горизонтального меню в виде строки. Для этого меню, как и для обычного в виде строки, можно определить “горячие” клавиши для получения быстрого доступа к командам и процедурам всплывающего меню. В всплывающем меню основные пункты расположены по вертикали, рекомендуется разделять их на группы команд при помощи разделителей. В качестве разделителей используются символы

¥-, в меню именно они образуют разделительную линию между командами или группами команд.

Кнопка **Cancel** предназначена для выхода из диалогового окна New Menu и прекращения создания меню.

Для создания многоуровневого меню приложения используется кнопка **Menu** (рис.10.10). После выбора одной из кнопок (Menu или Shortcut) на рабочем столе Visual FoxPro появится окно **Menu Designer** (рис.10.11).

Область окна, над которой размещены надписи к столбцам **Prompt** (Приглашение), **Result** (Результат) и **Option** (Опции), предназначена для определения пунктов меню. В поле **Prompt** вводится наименование пункта меню. После ввода наименования нужно нажать клавишу **Enter** или **Tab** (либо воспользоваться мышью) для перехода на следующее поле. Курсор окажется в столбце **Result**, каждое поле которого выполнено как раскрывающийся список, используемый для указания типа пункта меню - типа действия на выбор данного пункта меню.

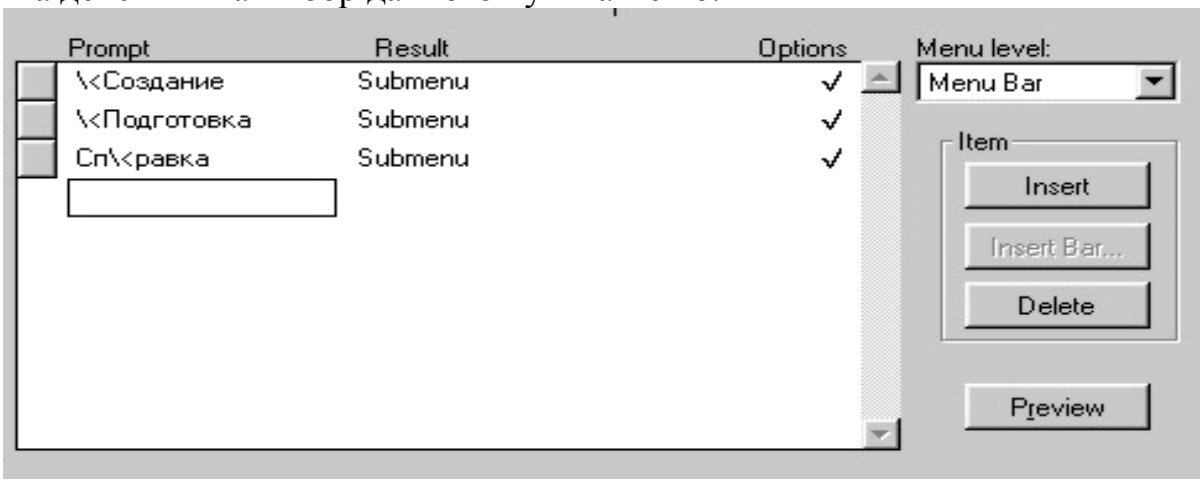


Рис.10.11. Окно Конструктора меню

В поле **Result** возможно указание одного из следующих типов пункта меню:

- **Command** (Команда) – при выборе пункта меню будет выполняться связанная с ним команда, которая вводится в поле справа от поля Result;
- **Submenu** (Подменю) – при выборе пункта меню раскрывается связанное с данным пунктом ниспадающее меню;
- **Procedure** (Процедура) – при выборе пункта меню вызывается процедура, определённая для данного пункта меню.

При использовании значения типа **Command** с правой стороны появляется поле для ввода команды, выполняемой при выборе данного пункта меню. Это может быть команда вызова формы (например, **DO FORM ID**). После выбора пункта меню, которому присвоена эта команда, будет происходить открытие формы (в данном случае формы ID).

Если при выборе пункта меню должна выполняться некоторая последовательность команд, необходимо в столбце **Result** (Результат) окна

Конструктора меню выбрать для пункта меню тип **Procedure** (Процедура) и определить требуемые команды. Для этого нужно нажать появившуюся кнопку **Create** (Создать) и в открывшемся окне редактирования процедуры ввести фрагмент кода, связанный с элементом строки меню. Например, для пункта меню **Справка** можно указать тип **Procedure** и в окне создания процедуры задать имя файла справки и команду вызова справочной системы:

```
SET HELP TO TEST.HLP
HELP
```

Если при выборе пункта меню должно открываться подменю, то в столбце **Result** (Результат) окна Конструктора меню из списка выбирается тип пункта меню **Submenu**, что дает возможность создавать и редактировать подменю. Для создания подменю необходимо нажать появившуюся рядом с Submenu кнопку **Create** (Создать). На экране появится пустое окно Конструктора меню для создания подменю. Список **Menu Level** (Уровень меню) нового окна будет содержать наименование текущего пункта меню. В поле **Prompt** (Приглашение) вводятся команды подменю. Используя список **Result** (Результат), задаются типы созданных пунктов подменю.

Кнопки в столбце **Options** открывают окно **Prompt Options** (Опции элемента меню) (рис.10.12). В этом окне можно определить дополнительные параметры текущего пункта меню (подменю): “горячие” клавиши, сообщение при выборе пункта меню, программное имя пункта меню и т. д. Для выбора пункта меню (подменю) используются клавиши перемещения курсора или мышь.

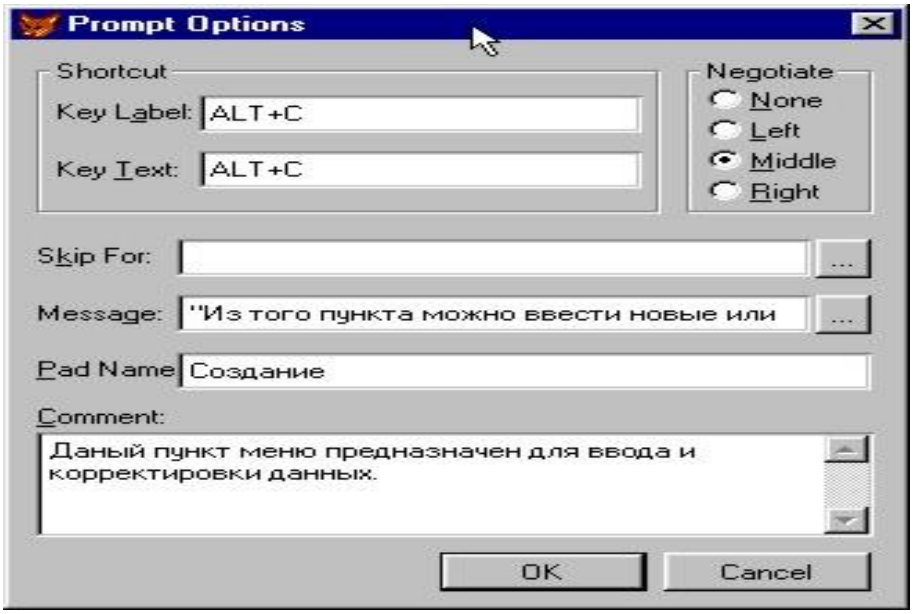


Рис.10.12. Окно Prompt Options для настройки элемента меню

Для пункта меню “горячей” клавишей является та, нажатие которой вместе с клавишей **Alt** активизирует этот пункт. “Горячие” клавиши предоставляют пользователю возможность ускоренного выбора пункта меню. Для строки горизонтального меню при генерации они назначаются Visual

FoxPro по умолчанию, с использованием первых букв элементов строки меню. Символы, используемые в качестве “горячей” клавиши, выделяются в строке меню подчёркиванием. Если два элемента меню начинаются с одинаковой буквы, то обоим элементам строки меню в качестве “горячей” клавиши назначается одинаковый символ.

Для определения “горячей” клавиши необходимо включить в наименование пункта меню перед активизирующим символом следующие два символа: ¥ и <. В качестве “горячей” клавиши можно использовать как первый символ наименования пункта меню (предпочтительнее), так и любой другой символ.

В открывшемся диалоговом окне **Prompt Options** (Опции элемента меню) в области **Shortcut** поле **Key Label** (Метка) содержит фразу **press the key**. Установив курсор в поле **Key Label**, необходимо нажать клавишу или комбинацию клавиш, используемую в качестве быстрого вызова, и в поле появится метка, соответствующая нажатым клавишам. В поле ввода **Key Text** (Пояснение) можно ввести краткое пояснение к определяемой клавише.

Поле **Skip For** (Пропустить для), позволяет задать условие блокировки пункта меню. Например, можно использовать условие блокировки для ограничения доступа определённых пользователей к отдельным пунктам меню или для блокировки незавершенных пунктов меню.

Используя поле ввода **Message** (Сообщение) диалогового окна **Prompt Options** (Опции элемента меню), можно определить для пункта меню сообщение, которое будет отображаться в строке состояния при выделении данного пункта меню (сообщение нужно заключить в кавычки). В качестве сообщения можно использовать не только строку текста, но и произвольные символьные выражения. В этом случае нужно нажать кнопку вызова конструктора выражения, расположенную справа от поля, и в открывшемся диалоговом окне **Expression Builder** (Построитель выражений) создать необходимое выражение.

В поле **Pad Name** (Имя пункта меню) можно задать программное имя пункта меню. По умолчанию при генерации программы Visual FoxPro в качестве имён пунктов меню создаёт уникальные имена. Для облегчения понимания программы можно определить эти имена явным образом, т.е. ввести их вручную с клавиатуры, используя для этого поле **Pad Name** (Имя пункта) диалогового окна **Prompt Options** (Опции элемента меню).

Поле **Comment** (Комментарий) предназначено для ввода комментария, который поясняет назначение команды или пункта меню. Его содержимое используется только при отладке приложения, для того чтобы получить краткую или подробную информацию о конкретном элементе меню.

В списке **Menu Level** (Уровень меню) окна Конструктора меню указывается уровень текущего меню (рис.10.11).

В окне Конструктора меню размещены кнопки (рис.10.11):

- **Insert** (Вставить) – добавляет в меню новый пункт;

- **Insert Bar** (Вставить команды системного меню) – открывает диалоговое окно **Insert System Menu Bar**, содержащее команды системного меню Visual FoxPro, позволяя разместить их в создаваемом пользовательском меню;
- **Delete** (Удалить) – удаляет текущий пункт меню;
- **Preview** (Просмотр) – размещает создаваемое меню на экране, позволяя просмотреть его внешний вид , но не активизирует.

Для строки меню можно определить процедуры, которые будут выполняться перед запуском меню, а также после выхода из него. Кроме того, можно указать место расположения строки меню. Для этого нужно использовать диалоговое окно **General Options**, открываемое при выборе команды **General Options** (Основные опции) из меню **View** (Вид).

В группе **Location** (Размещение) этого диалогового окна можно выбрать один из следующих вариантов размещения строки созданного меню:

- **Replace** (Замещать) – меню приложения замещает основное меню Visual FoxPro;
- **Append** (Добавить) – меню приложения добавляется в основное меню Visual FoxPro;
- **Before** (Перед) – меню вставляется перед указанным пунктом основного меню Visual FoxPro;
- **After** (После) – меню размещается за указанным пунктом основного меню Visual FoxPro.

В поле **Procedure** (Процедура) можно ввести фрагмент программы, которая будет выполняться при активизации строки меню.

Флажки **Setup** и **Cleanup** используются для открытия окна редактирования процедуры, вызываемой при запуске сгенерированной программы с описанием меню, и процедуры, вызываемой перед завершением этой программы. При установленном флажке **Top-Level-Form** меню будет отображаться в отдельном окне. В противном случае созданное меню будет отображаться в строке системного меню окна Visual FoxPro.

После того как меню было создано, его необходимо сохранить. Для сохранения созданного меню нужно выбрать команду **Save as** (Сохранить как) в меню **File** (Файл). В открывшемся диалоговом окне **Save as** из списка **Папка** выбрать папку, в которой предполагается сохранить файл и открыть её. В поле **Save Menu** (Сохранить меню) ввести имя сохраняемого меню и нажать кнопку **Сохранить.**

Чтобы можно было использовать меню в приложениях, его необходимо предварительно сгенерировать. Для этого необходимо выполнить следующие действия:

- в меню Menu выбрать команду **Generate** (Генерация), откроется диалоговое окно **Generate Menu** (Генерация меню) (рис.10.13);

- в поле **Output File** ввести имя файла, который будет создан в результате генерации, и для запуска генерации программного кода меню нажать кнопку **Generate (**Генерация).



Рис. 10.13. Диалоговое окно Generate Menu.

После завершения генерации можно запустить программу меню на выполнение. Для этого в меню **Program** (Программа) выбрать команду **Do** (Выполнить). В открывшемся диалоговом окне **Do** открыть папку, в которой был сохранён файл меню, выбрать файл меню с расширением .MPR и нажать кнопку **Do** в этом же окне. На экране появится созданное меню, которое заменит основное меню Visual FoxPro или дополнит его. Это зависит от того, какие опции были выбраны в диалоговом окне **General Options**. При создании меню создается несколько файлов с одним и тем же именем, но разным расширением: **.MNT** – мемо файл меню; **.MNX** – файл образа меню; **.MPR** - сгенерированная программа меню; **.MPX** - откомпилированная программа.

Для того, чтобы после запуска пользовательского меню восстановить системное (главное) меню Visual FoxPro, нужно ввести команду **SET SYSMENU TO DEFAULT** в командном окне (Command Window).

10.7. ДИСПЕТЧЕР ПРОЕКТОВ

При создании приложений во многих инструментальных средах создания приложений, в том числе и в Visual FoxPro, используется понятие проекта. **Проект** – средство объединения отдельных элементов приложения. Для создания проекта Visual FoxPro используется специальный **Диспетчер Проектов - Project Manager**. Функции проекта:

- запоминание расположения каждого включённого в него компонента;
- осуществление поиска и сбор файлов, на которые есть ссылки в проекте;
- сохранение объектного кода компонентов в MEMO полях специальных таблиц, что уменьшает количество файлов;
- упрощение тиражирования путем генерации проекта, который позволяет все файлы приложения объединить в один файл ***.APP** или ***.EXE**.

Сам проект хранится в обычной таблице формата DBF**,** но расширение этот файл имеет **.PJX** (**.PJT**). Для запуска Диспетчера Проектов выполняется команда **New** (Новый) из меню **File** (Файл) или нажимается кнопка **New** в стандартной панели инструментов Visual FoxPro, затем в открывшемся диалоговом окне **New** выбирается опция **Project** (Проект) и нажимается

кнопка **New File** (Новый файл). Появляется окно **Project Manager** (рис.10.14).

При открытии окна проекта в системном меню появляется новый пункт меню **Project**, команды которого позволяют выполнять различные действия по созданию проекта. В окне Диспетчера Проектов представлена иерархическая структура построения всего проекта. Каждая папка объединяет однотипные элементы проекта и ее можно открыть в окне Диспетчера Проектов, как и в Проводнике Windows. Слева в окне Диспетчера Проектов находится набор кнопок, с помощью которых можно добавить новый компонент, модифицировать компонент через запуск соответствующего конструктора, создать компонент, запустить компонент для выполнения, выполнить генерацию проекта, создавая единый файл приложения с расширением **.APP** или **.EXE**. Последнее допустимо лишь в профессиональной версии Visual FoxPro.

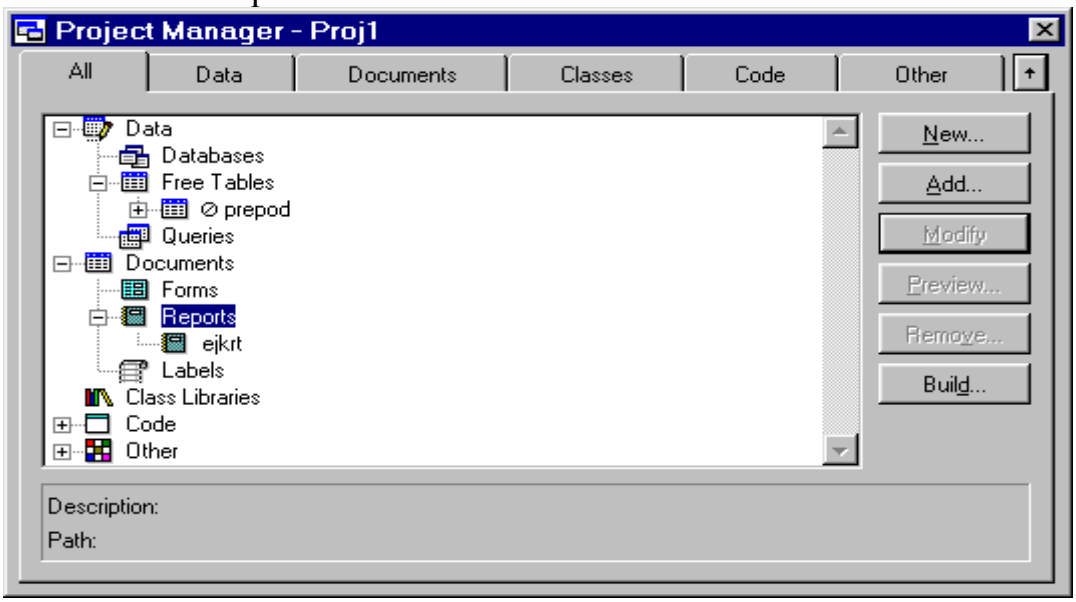


Рис.10.14. Окно диспетчера проекта Visual FoxPro

Окно Диспетчера Проектов имеет шесть вкладок, используемых для выборочного показа элементов проекта определенного типа. Для того чтобы увидеть все элементы проекта, используется вкладка **All.**

Вкладка **Data** служит для показа данных проекта, к которым относятся базы данных - папка **Databases**, свободные таблицы - папка **Free Tables**, запросы - папка **Queries**. Если раскрыть эти папки, то можно увидеть конкретные базы данных, свободные таблицы, запросы. Папка конкретной базы данных, в свою очередь, содержит таблицы, локальные и внешние представления, хранимые процедуры, соединения с внешними источниками данных, которые входят в состав данной базы данных. Вкладка **Documents** позволяет отобразить документы, используемые в проекте: формы, отчёты, ярлыки. Вкладка **Classes** используется для показа всех библиотек пользовательских классов, созданных в Конструкторе Классов. Вкладка **Code**

отображает программы, библиотеки API. Вкладка **Other** используется, когда необходимо увидеть папки с остальными элементами проекта: меню, текстовые файлы, файлы переменных памяти и другие файлы.

В проекте обязательно должна быть основная программа. В качестве основной программы может использоваться любая программа, меню, форма, запрос. При создании проекта необходимо указать, какая программа является основной. Для этого в диспетчере проектов выделяется компонент проекта, который будет являться основным, и нажимается правая кнопка мыши. В появившемся контекстно-зависимом меню выбирается команда **Set Main,** после чего имя основного компонента проекта должно выделиться жирным шрифтом.

В СУБД Visual FoxPro используется команда ожидания событий - **Read event.** Процедура выхода из приложения использует команду **Clear event**. Запуск приложения осуществляется путем запуска этой программы.

В качестве основной программы могут использоваться формы. В СУБД Visual FoxPro необходимо при создании основной формы установить значение свойства формы **Show Window** равным 2 (**Show Window = 2**). Если в основной форме необходимо запустить меню, то в событие **Init** формы необходимо записать команду: **Do <имя меню>.MPR with this, .F.** В этом случае запуск приложения осуществляется путем запуска основной формы.

При генерации проекта по кнопке **Build** диспетчера проекта создаётся единый файл приложения. При этом в появляющемся диалоговом окне выбирается тип создаваемого файла: **Build Application** (создается файл приложения с расширением **.APP**) или **Build executable** (создается автономный исполнимый файл приложения с расширением **.EXE**).

Перед генерацией проекта необходимо включить все используемые файлы в проект. Например, вручную добавить (кнопка **Add** диспетчера проекта) файлы с изображениями, используемыми в приложении.

Все компоненты, включенные в единый файл приложения, компилируются в объектный код и не могут изменяться. Поэтому базы данных, которые необходимо в процессе работы приложения изменять, перед генерацией проекта исключаются из приложения, иначе их можно будет только читать. Для этого необходимо в диспетчере проектов выделить базу данных, исключаемую из приложения, и нажать правую кнопку мыши. В появившемся контекстно-зависимом меню выбирается команда **Exclude**, после чего имя компонента, исключаемого из проекта, помечается символом ∅. Исключаются из проекта также все файлы, связанные с базами данных.

Для исполнения файла приложения с расширением **.APP** требуется наличие на компьютере СУБД Visual FoxPro. Приложение запускается на выполнение использованием в командном окне команды **Do** или через команду **Do** меню **Program** Visual FoxPro с выбором в списке файлов необходимого файла приложения. Автономный исполнимый файл приложения с расширением **.EXE** не требует самой СУБД, но требует

специальных библиотек из ее состава.

КОНТРОЛЬНЫЕ ВОПРОСЫ И УПРАЖНЕНИЯ

1. Какие основные базовые классы, события и методы используются в Visual FoxPro, их назначение?
2. Какие основные свойства имеют объекты Visual FoxPro, их назначение?
3. Какие средства визуального проектирования используются в Visual FoxPro?
4. Что собой представляет проект и какие вкладки имеются в окне Диспетчера проектов Visual FoxPro, их назначение?
5. Выберите предметную область, определите для данной предметной области два объекта и их свойства, представьте эти объекты в терминах реляционных отношений в виде таблиц с атрибутами (столбцами). Используя Мастер таблиц данных Visual FoxPro, создайте первую свободную таблицу данных. Используя Конструктор таблиц данных, создайте вторую свободную таблицу данных. Поработайте с таблицами данных, выполняя просмотр, редактирование, модификацию структуры таблиц и т.д.
6. Используя Конструктор БД Visual FoxPro, создайте базу данных, включающую две таблицы: одну таблицу (см. п. 5) добавьте в БД, вторую таблицу создайте с учетом дополнительных возможностей (задайте триггеры, индексы и т.д.). Сохраните БД и поработайте с ней, выполняя просмотр, редактирование, модификацию структуры таблиц и т.д.
7. Используя Мастер запросов и Конструктор запросов Visual FoxPro, создайте запросы к базе данных для получения из таблиц справочной и статистической информации (не менее 10 типов), используя простые запросы, запросы с условиями, запросы с группировкой данных, запросы с итогами, отправляя результат выборки в таблицу базы данных, текстовый файл, в массив, на принтер и т.д. Сохраните запросы под разными именами, запустите их на выполнение.
8. Используя Мастер форм Visual FoxPro, создайте форму для работы с одной таблицей, сохраните ее и запустите на выполнение. Используя Конструктор форм модифицируйте форму, созданную с помощью Мастера форм, для работы с двумя таблицами данных, сохраните ее и запустите на выполнение.
9. Используя Конструктор меню Visual FoxPro, создайте двухуровневое меню (с подменю) для работы с таблицами данных, запросами, формами, сгенерируйте меню и запустите его на выполнение.
10. Используя Диспетчер Проектов Visual FoxPro, объедините созданные компоненты приложения в проект, определите меню в качестве основной программы, запустите проект на выполнение.

Глава 11. СИСТЕМЫ НА ОСНОВЕ ЗНАНИЙ

11.1. СИСТЕМЫ БАЗ ЗНАНИЙ

В настоящее время все большее распространение, наряду с БД, получают **базы знаний** (БЗ). БЗ используются в **системах баз знаний** (СБЗ), которые также называются системами на основе знаний или **интеллектуальными системами**. СБЗ могут быть различного назначения: системы распознавания образов, понимания естественного языка, пустые и прикладные экспертные системы, обучающие системы, интеллектуальные тренажёры и т.д. Наука, посвященная проблеме создания интеллектуальных систем, называется искусственным интеллектом. Терминология в области искусственного интеллекта еще не устоялась, поэтому в разной литературе применяются разные термины. Ниже использованы наиболее часто применяющиеся термины и определения.

Интеллектуальные системы – это искусственные системы, созданные человеком на базе вычислительных систем и имитирующие решение человеком сложных творческих задач, возникающих в процессе его жизнедеятельности. Для выполнения этих задач базы знаний, кроме данных, должны содержать знания из определенной предметной области. Системы управления базами знаний, по аналогии с СУБД, можно назвать СУБЗ.

В СБЗ можно выделить три основные части: **базу знаний** (БЗ), **машину логического вывод**а (МЛВ) или **механизм получения решения** (МР), **интерфейс** (ИФ).Каждая из этих частей может быть построена по-разному в различных системах, при этом отличия могут быть глобальными (в принципах построения) и несущественными (в деталях). Между составными частями СБЗ нет чётких границ, они могут пересекаться, т.к. разделение это условно. Различные варианты построения систем на основе знаний зависят от того, какая из трёх составных частей СБЗ играет в ней доминирующую роль.

Основным в СБЗ являются **знания**. Знания можно разделить на алгоритмические и неалгоритмические.

Алгоритмические знания (процедурные) - это алгоритмы, т.е. программы, процедуры, вычисляющие функции, выполняющие преобразования, решающие конкретные задачи. Например, любая система программирования, операционные системы включают в себя базу алгоритмических знаний. В данном случае можно говорить о знаниях, оформленных в виде алгоритма.

Неалгоритмические знания состоят из понятий (мысленные объекты), правил и фактов. Выделяют два основных вида неалгоритмических знаний: концептуальные (понятийные) и фактуальные (предметные).

Из концептуальных знаний можно составить модель предметной области. Модель предметной области является концептуальной частью БЗ. Предметные знания составляют БД, которая является частью БЗ.

База знаний - это понятия, факты и правила из определённой предметной области знаний. На рис.11.1 представлена структура БЗ. Пакеты прикладных программ (ППП) - образуют процедурные знания, которые в свою очередь составляют алгоритмическую часть БЗ. Базы данных - образуют фактуальную часть БЗ, которая состоит из предметных знаний. Система управления базой знаний связывает между собой составные части БЗ и обеспечивает их взаимодействие

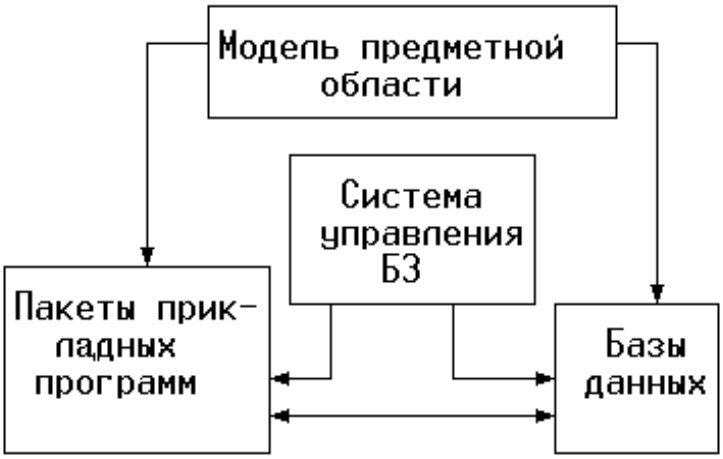


Рис.11.1. Структура БЗ

Таким образом все знания (алгоритмические и неалгоритмические) можно разделить на четыре основных класса:

- понятия (математические и нематематические);
- факты;
- правила, зависимости, связи;
- алгоритмы, процедуры.

Факты играют роль элементарных "единиц знаний" (простые утверждения о характеристиках объектов). По аналогии с терминологией БД можно определить факты как записи об объектах и их атрибутах.

Правила - это знания о том, как из известных фактов вывести новые факты. Правила служат для выражения связей и зависимостей между фактами и их комбинациями.

Для представления знаний наибольшее распространение получили четыре модели представления знаний: фреймы, семантические сети, правила продукции и логическое представление знаний.

Механизм получения решения обеспечивает прямое использование знаний из БЗ для решения задач. МР также называют машиной вывода или процедурой поиска, планирования, решения и т.д. МР даёт возможность извлекать из БЗ ответы на вопросы, получать решения задач, формулируемых в терминах понятий, хранящихся в БЗ. Принципы работы МР тесно связаны со способами представления знаний в БЗ, но в любом случае МР всегда содержит алгоритм решения, т.е. специальное алгоритмическое знание. Если

знания в БЗ представлены в виде уравнений, то МР - процедура решения уравнения. Если же знания представлены логическими формулами или правилами, то МР - механизм вывода (принятия решения).

Интерфейс обеспечивает работу с БЗ и МР на языке достаточно высокого уровня, приближённого к профессиональному языку специалистов в той прикладной области, к которой относится СУБЗ. Для этого в интерфейс вводится соответствующий языковый процессор.

В функции интерфейса, помимо вышеуказанного, входит поддержка диалога пользователя с системой, что даёт возможность пользователю получать объяснения действий системы, участвовать в поиске решения, пополнять, корректировать БЗ.

Для представления знаний и создания интеллектуальных систем могут использоваться разные языки и системы программирования:

- машинно-ориентированные и алгоритмические языки программирования (Ассемблер, Си, Паскаль и т.д.);
- символьные и функциональные языки программирования (ЛИСП, Рефал и т.д.);
- специальные языки инженерии знаний (Пролог, OPS, LogLisp и т.д.).

Специальные языки инженерии знаний имеют встроенный механизм логического вывода, что облегчает реализацию систем, основанных на знаниях. В Европе и Японии в качестве языка инженерии знаний наибольшее распространение получил язык логического программирования Пролог.

11.2. МОДЕЛИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Теория фреймов - это парадигма для представления знаний любого типа с целью использования этих знаний компьютером. Впервые была предложена М.Минским в 1974 г. Минский разработал такую схему, в которой информация содержится в специальных ячейках, называемых фреймами, объединенными в сеть, называемую системой фреймов. Новый фрейм активизируется с наступлением новой ситуации. Отличительной его чертой является то, что он одновременно содержит большой объем знаний и в то же время является достаточно гибким для того, чтобы быть использованным как отдельный элемент БЗ. Минский также ввел терминологию, которая используется при изучении фреймовой модели представления знаний (**фреймы, слоты, терминалы** и т.д.).

Фреймы – это фрагменты знания, предназначенные для представления стандартных ситуаций. Фреймы имеют вид структурированных компонентов ситуаций, называемых **слотами** (ячейками). Конечные слоты называются терминалами. Слот может указывать на другой фрейм, устанавливая, таким образом, связь между двумя фреймами. Могут устанавливаться общие связи типа “связи по общению”. С каждым фреймом ассоциируется разнообразная информация (в том числе и процедуры). Например, ожидаемые процедуры

ситуации, способы получения информации о слотах, значения принимаемые по умолчанию, правила вывода.

Формальная структура фрейма имеет вид

f [<N1, V1>, <N2, V2>, …, <Nk,Vk>],

где f – имя фрейма; пара <Ni, Vi> - i-й слот, Ni – имя слота и Vi – его значение.

Значение слота может быть представлено последовательностью

<Ki ><Li>;...; <Kn><Ln>; <Rj>; …; <Rm>,

где Ki – имена атрибутов, характерных для данного слота; Li – значение этих атрибутов, характерных для данного слота; Rj – различные ссылки на другие слоты.

Каждый фрейм, как структура, хранит знания о предметной области (фрейм–прототип), а при заполнении слотов знаниями превращается в конкретный фрейм события или явления.

Фреймы можно разделить на три группы: **фреймы визуальных образов**; **семантические фреймы** (для понимания слов); **фреймы-сценарии** (для рассуждений, повествований, действий). Среди последних можно выделить фреймы-описания и ролевые фреймы. Ниже рассмотрен пример, показывающий разницу между фреймом-описанием и ролевым фреймом.

Фрейм описание: [<программное обеспечение>; <программа 1С бухгалтерия, версия 7.5>; <программа 1С торговля, версия 7.5>; <правовая программа «Консультант +» проф.>].

Ролевой фрейм: [<заявка на продажу>; <что, установка и покупка программы 1С торговля, версия 7.5>; <откуда, фирма ВМИ>, <куда, фирма «Лукойл»>; <кто, курьер Иванова>; <когда, 27 октября 1998г.>].

Во фрейме-описании в качестве имен слотов задан вид программного обеспечения, а значение слота характеризует массу и производителя конкретного вида продукции. В ролевом фрейме в качестве имен слотов выступают вопросительные слова, ответы на которые являются значениями слотов. Для данного примера представлены уже описания конкретных фреймов, которые могут называться либо фреймами-примерами, либо фреймами-экземплярами. Если в приведенном примере убрать значения слотов, оставив только имена, то получим так называемый фрейм-прототип.

Основное достоинство фреймов заключается в том, что представления во многом основываются на включении в него предположений и ожиданий. Это достигается за счет присвоения по умолчанию слотам фрейма стандартных ситуаций. В процессе поиска решений эти значения могут быть заменены более достоверными. Некоторые переменные организованы таким образом, что их значения определяются пользователем в процессе диалога с системой. Часть переменных определяется посредством встроенных процедур, называемых внутренними. По мере присвоения переменным определенных значений осуществляется вызов других процедур. Этот тип представления комбинирует декларативные и процедурные знания.

Фреймовые модели обеспечивают требования структурированности и связанности, что достигается за счет свойств наследования и вложенности, которыми обладают фреймы: в качестве слотов может выступать система имен слотов более низкого уровня, слоты могут быть также использованы как вызовы каких-либо процедур для выполнения.

Родственные фреймы объединяются в системы фреймов. Системы, состоящие из связанных между собой фреймов, объединяются в сеть. Фрейм - это **модель знаний**, которая активизируется в определенной ситуации и служит для ее объяснения и предсказания. При несоответствии активизированного фрейма реальной ситуации, в системе фреймов активизируется другой фрейм. Если в данной системе отсутствует фрейм, подходящий для данной ситуации, то по сети происходит переход к другой системе фреймов.

Для многих предметных областей фреймовые модели являются основным способом формализации знаний.

Семантические сети получили широкое распространение в конце 70-х годов. Семантическая сетевая модель - это интегрированное представление данных, категорий данных, свойств категорий и операций над данными. А семантическая сеть - это средство с помощью которого удобно описывать структуру отношений между фактами и понятиями. Базовые структуры в этих моделях представляются графом, множество вершин и дуг которого образует сеть. Графу семантической сети приписана некоторая **семантика** - содержание. Вершины и дуги графа определяют различные понятия в зависимости от использования семантической сети. Например, при моделировании ассоциативной природы человеческой памяти ассоциации представляются вершинами и дугами между ними. Или вершинам могут соответствовать слова, а дугам - связи между ними (представление одних слов посредством других). С помощью семантических сетей можно описывать любые знания, но наибольшее применение они нашли в системах понимания естественного языка.

Самые первые семантические сети были разработаны для систем машинного перевода, а многие современные версии семантических сетей до сих пор сходны по своим характеристикам с естественным языком. Однако последние версии семантических сетей стали более мощными и гибкими и составляют конкуренцию фреймовым системам, логическому программированию и другим представлениям знаний.

В последние годы были созданы и применены на практике десятки вариантов семантических сетей. Несмотря на некоторые различия, существуют особенности, присущие всем семантическим сетям.

Вершины в семантической сети могут быть разных видов: **вершины - классы** для представления типов объектов (категорий), **вершины - концепты** для представления конкретных экземпляров объектов (данных), **вершины - ситуации**. Дуги отражают **бинарные отношения** между

классами, концептами, ситуациями, классом и концептом, классом и ситуацией и т.д. Ниже представлен пример простейшей семантической сети:



В этом примере вершина БОБ является концептом, вершины МАЛЬЧИК и РЕБЕНОК являются классами, дуги выражают наиболее часто используемые в семантических сетях бинарные отношения **IS-A** (есть, является) между концептом и классом (БОБ есть МАЛЬЧИК) и между классом и классом (МАЛЬЧИК есть РЕБЕНОК).

В системах искусственного интеллекта семантические сети используются для создания естественно-языковых систем (ответы на различные вопросы, изучение процессов обучения, запоминания и рассуждения).

Семантические сети могут быть реализованы практически на любом языке программирования (Fortran, Pascal, C, Prolog, Lisp и т.д) и на любой машине. Есть специальные языки, разработанные для представления семантических сетей. Например, язык PLNLP (Programming Language for Natural Language Processing) - язык Программирования для Обработки Естественного Языка, созданный Хайдерном.

Помимо специальных языков для семантических сетей было разработано и специальное аппаратное обеспечение.

Продукционные правила используют правила типа **ЕСЛИ <условие>, ТО <заключение> (**правила **если-то**), называемые также продукциями. Каждое такое правило является, вообще говоря, некоторым условным утверждением, но возможны и различные другие интерпретации. Вот примеры:

•ЕСЛИ предварительное условие P, ТО заключение (вывод) C.

•ЕСЛИ ситуация S, ТО действие A.

•ЕСЛИ выполнены условия C1 и C2, ТО не выполнено условие C.

Правила **если-то** обычно оказываются весьма естественным выразительным средством представления знаний. Кроме того, они обладают следующими свойствами:

1. Модульность: каждое правило описывает небольшой, относительно независимый фрагмент знаний;

2. Возможность инкрементного наращивания: добавление новых правил в базу знаний происходит относительно независимо от других правил;

3. Удобство модификации (как следствие модульности): старые правила можно изменять и заменять на новые относительно независимо от других правил;

4. Применение правил способствует прозрачности системы.

Под прозрачностью понимается способность системы к объяснению принятых решений и полученных результатов, что очень важно для экспертных систем, одного из многочисленного класса интеллектуальных

систем. Применение правил **если-то** облегчает получение ответов на следующие основные типы вопросов пользователя:

•Вопросы типа "как": Как вы пришли к этому выводу?

•Вопросы типа "почему": Почему вас интересует эта информация?

Правила **если-то** часто применяют для определения логических отношений между понятиями предметной области. Про чисто логические отношения можно сказать, что они принадлежат к "категорическим знаниям", "категорическим" - потому, что соответствующие утверждения всегда абсолютно верны. Однако в некоторых предметных областях, таких как медицинская диагностика, преобладают "мягкие" или вероятностные знания. Эти знания являются "мягкими" в том смысле, что говорить об их применимости к любым практическим ситуациям можно только до некоторой степени ("часто, но не всегда"). В таких случаях используют модифицированные правила **если-то**, дополняя их логическую интерпретацию вероятностной оценкой. Например:

ЕСЛИ условие А, ТО заключение В с уверенностью F.

Предположим, существует небольшая база знаний, которая может помочь локализовать неисправности в простой электрической схеме, состоящей из электрических приборов и предохранителей. Вот одно из возможных правил БЗ:

ЕСЛИ лампа_1 включена и лампа_1 не работает
и предохранитель_1 заведомо цел,
ТО лампа_1 заведомо неисправна.

Вот другой пример правила:

ЕСЛИ радиатор работает,
ТО предохранитель_1 заведомо цел.

Эти два правила опираются на некоторые факты (относящиеся к конкретной схеме), а именно: лампа_1 соединена с предохранителем_1, лампа_1 и радиатор имеют общий предохранитель. Для другой схемы понадобится еще один набор правил. Поэтому было бы лучше сформулировать правила в более общем виде так, чтобы они были применимы к любой схеме, а затем дополнять их информацией о конкретной схеме. Например, вот одно из полезных правил: если прибор включен, но не работает, а соответствующий предохранитель цел, то прибор неисправен. На формальный язык это правило записывается так:

ЕСЛИ прибор включен и не (прибор работает) и
прибор соединен с предохранителем и предохранитель заведомо цел,
ТО прибор заведомо неисправен.

Продукционные системы могут быть созданы на любом языке программирования на любой машине. Но есть и специальные языки инженерии знаний, которые используют правила продукции (например, продукционная система OPS).

При **логическом представлении знаний** используется исчисление предикатов первого порядка. Правильно построенные выражения в исчислении предикатов называются правильно построенными формулами (ППФ). В ППФ используются элементарные компоненты языка исчисления предикатов первого порядка:

- символы констант (относятся к термам);
- символы переменных (относятся к термам);
- предикатные символы;
- функциональные символы;
- вспомогательные символы (круглые и квадратные скобки, используемые для разделения основных символов).

Из этих компонент строятся простые (атомные) ППФ, которые используются для представления знаний. Например, формула **писать (вольтер, кандид)** представляет знание о том, что Вольтер написал Кандида.

Формулы, полученные в результате объединения атомных ППФ с помощью логических связок (**и, или, импликация**), также являются ППФ.
Логическое представление знаний используется в широко распространенном языке инженерии знаний – языке (системе) логического программирования Пролог.

Язык Пролог является языком декларативного типа, т. е. Пролог-программа представляет из себя набор логических описателей, определяющих цель программы.
Программа на Прологе состоит из множества утверждений, каждое из которых является либо фактом о заданной информации, либо правилом, указывающим как решение связано с заданными фактами. Используя имеющиеся факты и правила система выводит новые факты.

11.3. ЭКСПЕРТНЫЕ СИСТЕМЫ

Экспертные системы (ЭС) являются одним из классов интеллектуальных систем, в реализации которых достигнуты наибольшие успехи. ЭС обычно определяют как компьютерные программы, моделирующие действия эксперта - человека при решении задач в узкой предметной области на основе накопленных знаний, составляющих базу знаний. В ЭС, в отличие от большинства других интеллектуальных систем, знания всегда отделены от механизма вывода. Цель ЭС - выдача рекомендаций, советов, консультаций.

Экспертные системы:

-размышляют, имея частичную информацию или информацию, в которой пользователь не уверен;
-различают нужную и ненужную информацию;
-дают логичный совет;
-объясняют ход процесса аргументации;

-формализуют специальные знания людей для обучения других.
Хотя часто считают экспертные системы новой технологией, концепции их построения стары, как род человеческий. Еще ученые древности пытались систематизировать все накопленные человечеством знания. Примером экспертной системы древности может служить система медицинской диагностики, некоторые правила которой дожили и до сегодняшних дней.

Современную же экспертную систему, основанную на правилах, описал, по существу, Дж. Дэйви. Кроме него большое влияние на развитие ЭС оказали идеи А.Тьюринга. Важная основополагающая концептуальная работа была выполнена А.Ньюэллом и Г.Саймоном, которые предложили схему процессов, лежащих в основе решения проблем человеком.

Экспертная система решает проблему и дает квалифицированную консультацию, используя имеющиеся в наличии знания конкретной предметной области (базу знаний) и специальные средства для логических выводов (машину логических выводов).

Знания, хранимые в экспертной системе, обычно представлены в форме правил, у которых есть посылка и заключение. Правило – это какой-то объем знаний, в котором говорится, какие предпринимать действия (указывается в заключении) при выполнении определённых условий (указывается в посылке). Для конкретной ситуации предпринимается конкретное действие. Если экспертная система определяет, что посылка верна, то правило признается подходящим для данной консультации и оно запускается в действие. Запуск правила означает принятие заключения данного правила в качестве составной части процесса консультации, которую экспертная система дает пользователю.

Если экспертной системе нужны дополнительные подробности, она может спросить о них и, в случае необходимости, объяснит, почему она требует эту дополнительную информацию. После того, как проблема будет решена, экспертная система выдаст решение или рекомендацию и объяснит, как она пришла к этому решению. Знания и опыт, хранящиеся в экспертной системе, можно обновлять и расширять, так же как знания и опыт человека.

Пользователь получает объяснения действий системы, участвует в поиске решения, пополняет и корректирует базу знаний, используя интерфейс системы. Интерфейс обеспечивает работу с базой знаний и машиной логического вывода на языке высокого уровня, приближенного к профессиональному языку специалистов в той прикладной области, к которой относится экспертная система. Наиболее простым языком для любого пользователя является естественный разговорный язык. Поэтому интерфейс должен также обеспечивать поддержку диалога пользователя с системой на естественном языке.

Экспертные системы разрабатываются специалистами в области баз знаний и вычислительной техники, которые работают непосредственно с экспертами - людьми. Специалист в области баз знаний (инженер знаний)

использует знания и мысленные процессы эксперта и преобразует их в ряд операторов или правил, представляющих знания этого эксперта. При разработке серьезной экспертной системы для некоторой выбранной предметной области инженер знаний должен провести консультации с экспертами в этой области и многое узнать о ней сам. Достигнуть определенного понимания предметной области после общения с экспертами и чтения литературы, а затем облечь это понимание в форму представления знаний в рамках выбранного формального языка – искусство, называемое инженерией знаний. Как правило, это сложная задача, требующая больших усилий.

Применение ЭС позволяет:

- принимать быстрые и качественные решения;
- готовить опытных специалистов за относительно короткий промежуток времени;
- сохранять “ноу-хау”, так как пользователям ЭС недоступны опыт и знания, содержащиеся в ЭС;
- использовать опыт и знания высококвалифицированных специалистов на непрестижных, скучных, опасных и тому подобных рабочих местах.

К недостаткам ЭС можно отнести ограниченные возможности использования здравого смысла.

Для создания экспертных систем могут использоваться различные инструментальные средства: машинно-ориентированные и алгоритмические языки программирования, специальные языки инженерии знаний, оболочки ЭС - инструментальные экспертные системы.

В течение почти десяти лет эта мощная технология не выходила за рамки стен университетских и военных лабораторий. В настоящее время интерес к практическим экспертным системам, питаемый значительными финансовыми инвестициями из частного сектора, достиг максимума. В результате колоссально возросло число программных продуктов, предназначенных для создания систем, основанных на знаниях, с диапазоном применений от дегустации вин до оценки способностей служащих.

В 70-е годы были разработаны первые ЭС, имеющие практическое применение. В 1971 г. была создана первая медицинская диагностическая система, а в 1974 г. - программа MYCIN (диагноз бактериальных инфекций и рекомендации по курсу эффективной терапии).

В 80-х годах в США начала реализовываться программа расширенных исследований в области обороны (DARPA). Одним из результатов таких исследований стало появление танка-робота, способного распознавать рельеф местности.

Логистический отдел ВВС США использует экспертную систему IMA (Inventory Management Assistant) в складском хозяйстве. Отдел обслуживает

свыше 19 000 самолетов по всему миру. Складская система отдела содержит 916 000 наименований запасных частей для самолетов. Цель создания IMA – помощь персоналу складов при решении задач, связанных с управлением запасами.

Использование ЭС позволяет на 8-10% повысить эффективность решения обычных проблем. Эффективность решения вопросов в сложных ситуациях возрастает на 15-18%.

Постоянный прогресс в области технологий и информации несомненно будет требовать дальнейшего развития экспертных систем как одной из самых важных частей искусственного интеллекта.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для решения каких задач предназначены интеллектуальные системы?
2. Из каких основных частей состоят системы, основанные на знаниях?
3. Что собой представляет механизм логического вывода в ИС?
4. Что собой представляет база знаний?
5. Что собой представляет интерфейс интеллектуальных систем?
6. Что собой представляют неалгоритмические знания?
7. Для представления каких знаний используются фреймы?
8. Для представления каких знаний удобно использовать семантические сети?
9. Что собой представляют продукционные правила?
10. Что лежит в основе логического представления знаний?
11. Какие символы используются для представления отношений в логическом представлении знаний?
12. В каких системах база знаний и механизм вывода не разделены?
13. К какому типу вершин семантической сети относятся: вершина "школа", вершина "дом №3, вершина "собственность"?
14. Какой признак отличает экспертную систему от любой другой интеллектуальной системы?
15. Какую функцию выполняет "инженер знаний" в технологии построения экспертных систем?
16. Какие инструментальные средства позволяют создать интеллектуальные системы с минимальными трудозатратами?

ЗАКЛЮЧЕНИЕ

Обработка информации на основе технологии баз данных применяется во всех сферах деятельности человека. Эта перспективная технология обеспечила решение проблем, связанных с эффективной обработкой больших объемов данных, коллективным использованием данных, построением корпоративных хранилищ данных, интегрированием данных разного типа. Большинство систем баз данных используют **реляционную модель**. Для традиционных приложений баз данных - банковских систем, систем резервирования и т.д. - это позволяет проектировать экономные по памяти БД с предельно понятной структурой.

Однако с появлением эффективных реляционных СУБД их стали пытаться использовать и в менее традиционных прикладных системах - системах автоматизации проектирования (САПР), системах искусственного интеллекта и т.д. Такие системы обычно оперируют сложно структурированными объектами, для реконструкции которых из плоских таблиц реляционной БД приходится выполнять запросы, почти всегда требующие соединения отношений. Поэтому применение классических реляционных систем управления базами данных в САПР, интеллектуальных системах обучения и других системах, основанных на знаниях, часто является затруднительным.

Поэтому исследователи в области баз данных выполняют многочисленные проекты, основанные на идеях, выходящих за пределы реляционной модели данных. Следует заметить, что тематика современных исследований, относящихся к базам данных, исключительно широка: **объектно-ориентированные** СУБД; **темпоральные** (хронологические) СУБД; СУБД**, основанные на правилах; генераторы** СУБД; **дедуктивные** (экспертные) СУБД и т.д.

В объектно-ориентированных СУБД используется **объектно-ориентированное моделирование данных**. Но развитого математического аппарата, на который могла бы опираться общая объектно-ориентированная модель данных, не существует. Объектно-ориентированными являются СУБД O2, СУБД ORION, СУБД GemStone, СУБД Iris.

В **темпоральных** системах для любого объекта данных, созданного в момент времени t1 и уничтоженного в момент времени t2, в БД сохраняются (и доступны пользователям) все его состояния во временном интервале [t1,t2]. Возможна выборка информации, хранившейся в базе данных в указанное время, в указанном временном интервале и т.д. Это возможно, благодаря тому, что при модификациях кортежа изменения производятся не на месте его хранения, а заводится новая запись, куда помещаются измененные поля. Новая запись подшивается в список к изменявшемуся кортежу.

Исследования и построения прототипов темпоральных СУБД обычно выполняются на основе некоторой реляционной СУБД. Темпоральная СУБД - это надстройка над реляционной системой. Примером темпоральной СУБД может служить СУБД Postgres.

Генераторы СУБД генерируют системы с ориентацией на конкретную предметную область (специализированные СУБД), максимально соответствующие потребностям приложения. На сегодняшний день на коммерческом рынке такие "генерационные" системы отсутствуют. Существуют экспериментальные прототипы, например, Genesis, Exodus. Системы состоят из минимального ядра (развитой файловой системы в случае Exodus) и технологического механизма программирования дополнительных модулей. Вместо готовой СУБД предоставляется набор "полуфабрикатов" с согласованными интерфейсами, из которых можно сгенерировать систему, максимально отвечающую потребностям приложения.

Дедуктивная **СУБД** управляет БД, состоящей из двух частей: **экстенциональной**, содержащей факты, и **интенциональной**, содержащей правила для логического вывода новых фактов на основе экстенциональной части и запроса пользователя.

При таком общем определении SQL-ориентированную реляционную СУБД можно отнести к дедуктивным системам: даные БД (факты) - это экстенциональная часть, а определенные в схеме реляционной БД представления можно рассматривать как интенциональную часть БД. Основным отличием реальной дедуктивной СУБД от реляционной является то, что и правила интенциональной части БД, и запросы пользователей могут содержать рекурсию. Возможность определения рекурсивных правил и запросов дает возможность простого решения в дедуктивных базах данных проблем, которые сложно решаются в реляционных системах. С другой стороны, именно возможность рекурсии делает реализацию дедуктивной СУБД очень сложной и во многих случаях неразрешимой эффективно проблемой.

Связь дедуктивных СУБД с реляционными СУБД заключается в том, что для реализации дедуктивной СУБД обычно применяется реляционная система. Такая система выступает в роли хранителя фактов и исполнителя запросов, поступающих с уровня дедуктивной СУБД.

В СУБД, основанных на правилах, правила играют существенно более важную роль, чем в традиционных реляционных системах. Такая СУБД представляет собой некоторый интерпретатор системы правил и набор модулей-действий, вызываемых в соответствии с этими правилами.

Выше указанные направления работ являются лишь малой частью тех работ, которые выполняются в области развития и совершенствования теории и практики баз данных в мире, что еще раз подчеркивает перспективность обработки информации на основе технологии баз данных.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Дейт К. Введение в системы баз данных /Пер.с англ. - М.: Вильямс, 2001.
2. Хансен Г., Хансен Д. Базы данных. Разработка и управление. - М.: Бином, 2000.
3.Диго С.М. Проектирование и использование баз данных.- М.: Финансы и статистика, 1995.
4.Ульман Дж., Уидом Дж. Введение в системы баз данных /Пер. с англ.- М.: Лори, 2000.
5.Мейер Д. Теория реляционных баз данных /Пер.с англ.- М.: Мир,1987.
6.Четвериков В.Н., Ревунков Г.И., Самохвалов Э.Н.Базы и банки данных.- М.: Высшая школа, 1987.
7.Горев А., Ахаян Р., Макашарипов С. Эффективная работа с СУБД. - СПб.: Питер, 1997.
8. Горев А. Visual FoxPro 5.0. Книга для программистов. - М.: Эдэль,1997.
9. Пэддок Р.,Петерсен Дж., Тэлмейдж Р., Ренфт Э. Visual FoxPro 6. Разработка корпоративных приложений /Пер. с англ. - М.:ДМК, 1999.
10.Каратыгин С.А.,Тихонов А.Ф., Тихонова Л.Н. Visual FoxPro 6.0. - М.: Бином, 1999.
11. Ланг К., Чоу Дж. Публикация БД в Internet /Пер. с англ. - СПб.: Символ-Плюс, 1998.
12. Омельченко Л. Самоучитель Visual FoxPro 6.0. - СПб.: БХВ - Санкт-Петербург, 2000.
13. Боуман Дж., Дарновски Э. Практическое руководство по SQL /Пер. с англ. - Киев: Диалектика, 1997.
14. Бекаревич Ю., Пушкина Н. Самоучитель Microsoft Access 2000.-СПб.: БХВ - Санкт-Петербург, 1999.
15.Шумаков П.В., Фаронов В.В. Delphi 4. Руководство разработчика баз данных. - М.: Нолидж, 1999.
16. Когаловский М.Р. Технология баз данных на персональных компьютерах. - М.: Финансы и статистика, 1993.
17. Шнайдер Р. Microsoft SQL Server. Проектирование высокопроизводительных баз данных /Пер. с англ.- М.: Лори, 1998.
18. Архангельский А.Я. Работа с локальными базами данных в Delphi 5. - М.: Бином, 2000.
19. Мамаев Е.В. Microsoft SQL Server. - СПб.: БХВ-Санкт-Петербург, 2001.
20. Фролов А., Фролов Г. Базы данных в Интернете: практическое руководство по созданию Web-приложений с базами данных. - М.: Русская редакция, 2000.
21.Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. - СПб.: Питер, 2000.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....3
Глава 1.ОСНОВНЫЕ СВЕДЕНИЯ О СИСТЕМАХ БАЗ ДАННЫХ
1.1.Особенности технологии систем баз данных.....5
1.2.Архитектура систем баз данных.....7
1.3.Взаимодействие компонентов систем баз данных.....12
1.4.Классификация систем баз данных.....16
Контрольные вопросы.....19
Глава 2.МОДЕЛИ ДАННЫХ
2.1.Структуры и схемы данных.....20
2.2.Операции над данными и ограничения целостности.....24
2.3.Иерархическая и сетевая модели данных.....26
2.4.Реляционная модель данных.....31
Контрольные вопросы.....35
Глава 3.СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ
3.1.Реляционные системы баз данных.....36
3.2.Реляционные СУБД.....40
3.3.Язык управления данными SQL.....44
3.4.Серверы баз данных.....48
Контрольные вопросы.....54
Глава 4. ЗАЩИТА И БЕЗОПАСНОСТЬ ИНФОРМАЦИИ В СИСТЕМАХ БАЗ ДАННЫХ
4.1.Основные понятия защиты и безопасности информации.....55
4.2.Транзакции и блокировки.....62
4.3.Резервирование баз данных.....64
4.4.Администрирование баз данных.....66
4.5.Ограничения целостности.....68
Контрольные вопросы.....71
Глава 5.ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ
5.1.Общие вопросы проектирования информационных систем.....72
5.2.Проектирование баз данных.....76
5.3.Сущности и связи.....82
5.4.Публикация баз данных в сети Интернет.....85
Контрольные вопросы и упражнения.....91
Глава 6. ЯЗЫК ПРОГРАММИРОВАНИЯ XBASE
6.1.Основные положения.....92
6.2.Создание и изменение структуры таблицы данных.....94
6.3.Ввод и просмотр записей в таблице данных.....97
6.4.Редактирование таблицы данных.....100
6.5.Удаление записей в таблице данных.....104
6.6.Сортировка и индексирование таблиц данных.....105
6.7.Временные переменные памяти.....107

6.8.Создание и выполнение программ.....111
6.9.Управляющие операторы.....113
6.10.Команды-функции и стандартные функции.....117
Контрольные вопросы и упражнения.....121
Глава 7. SQL В СУБД FOXPRO
7.1.Формирование запросов.....122
7.2.Определение данных.....128
7.3.Манипулирование данными.....130
7.4.Управление транзакциями.....132
Контрольные вопросы и упражнения.....133
Глава 8.MICROSOFT SQL SERVER
8.1.Архитектура данных MS SQL Server.....135
8.2.Физическая структура базы данных.....138
8.3.Организация файлов базы данных.....142
8.4.Логическая структура базы данных.....146
8.5.Основные объекты пользователя.....148
8.6.Ограничения целостности.....151
8.7.Администрирование MS SQL Server.....152
Контрольные вопросы.....156
Глава 9.ВИЗУАЛЬНОЕ ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЙ
БАЗ ДАННЫХ
9.1.Объектно-ориентированное программирование.....157
9.2.Объекты приложений баз данных и визуальное проектирование..160
9.3.Создание приложений баз данных.....163
Контрольные вопросы.....170
Глава 10.СУБД VISUAL FOXPRO
10.1.Основные характеристики Visual FoxPro.....171
10.2.Средства визуального проектирования в Visual FoxPro.....178
10.3.Визуальное проектирование баз данных.....181
10.4.Визуальное проектирование запросов.....189
10.5.Визуальное проектирование форм.....194
10.6.Визуальное проектирование меню.....201
10.7.Диспетчер проектов.....207
Контрольные вопросы и упражнения.....210
Глава 11.СИСТЕМЫ НА ОСНОВЕ ЗНАНИЙ
11.1.Системы баз знаний.....211
11.2.Модели представления знаний.....213
11.3.Экспертные системы.....218
Контрольные вопросы.....221
ЗАКЛЮЧЕНИЕ.....222
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....224

Учебное издание

Айтхожаева Евгения Жамалхановна

Системы баз данных

Учебник

Зав.РИО З.А.Губайдулина
Редактор С.Н. Яниева
Техн. редактор Н.Н. Передереева
Корректор Е.М.Кудайкулов
Компьютерная верстка Г.Оразалиева

Подписано в печать ….. 2002 г.
Тираж 500 экз. Формат 60 х 84 1/16. Бумага типографская №1.
Объем 14,0 уч.-изд. л. Заказ № …. Цена договорная.

Издание Казахского национального технического
университета им.К.И.Сатпаева

Издательский центр КазНТУ
Алматы, ул. Сатпаева, 22