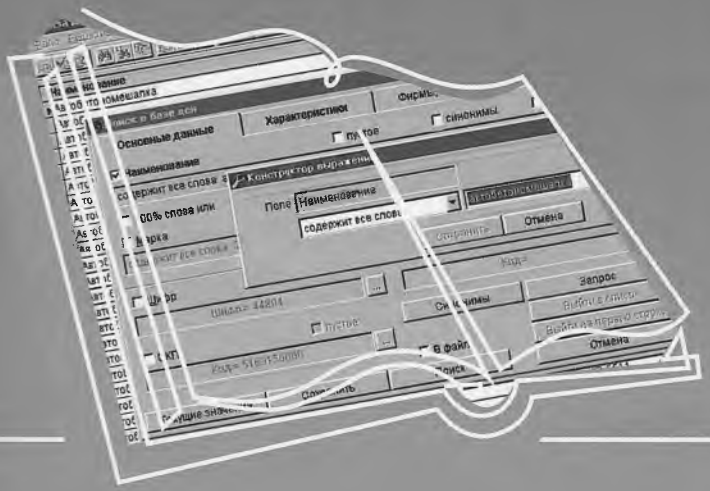
Высшее профессиональное образование Учебное пособие

А. В. Кузин
С. В. Левонисова

БАЗЫ ДАННЫХ

5-е издание



БАКАЛАВРИАТ

ИНФОРМАТИКА
И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ACADEMA

Высшее профессиональное образование

БАКАЛАВРИАТ

А. В. КУЗИН, С. В. ЛЕВОНИСОВА

БАЗЫ ДАННЫХ

Допущено
Учебно-методическим объединением вузов
по университетскому политехническому образованию
в качестве учебного пособия для студентов высших учебных заведений,
обучающихся по направлению подготовки
«Информатика и вычислительная техника»

5-е издание, исправленное

ACADEMA

Москва
Издательский центр «Академия»
2012

УДК 681.3(075.8)
ББК 32.81я73
К89

Рецензенты:
директор Красноярского оптико-электронного колледжа, д-р пед. наук *В. М. Демин*;
ведущий научный сотрудник ВЦ РАН, проф., д-р техн. наук *С. К. Дулин*

Кузин А. В.

К89 Базы данных : учеб. пособие для студ. высш. учеб. заведений / А. В. Кузин, С. В. Левонисова. — 5-е изд., испр. — М. : Издательский центр «Академия», 2012. — 320 с. — (Сер. Бакалавриат).
ISBN 978-5-7695-9308-6

Учебное пособие создано в соответствии с Федеральным государственным образовательным стандартом по направлению подготовки 230100 «Информатика и вычислительная техника» (квалификация «бакалавр»).

Рассмотрены базовые вопросы теории проектирования баз данных, использование СУБД Access для создания баз данных, особенности разработки пользовательских приложений на основе СУБД Microsoft Access, а также архитектура системы баз данных.

Для студентов учреждений высшего профессионального образования.

УДК 681.3(075.8)
ББК 32.81я73

ISBN 978-5-7695-9308-6

ПРЕДИСЛОВИЕ

За последние годы в нашей стране произошли значительные перемены, которые не могли не затронуть области информатики и вычислительной техники. Всего каких-нибудь десять лет назад работа с базами данных и электронными таблицами была уделом профессиональных программистов.

Системы управления базами данных (СУБД) не были предназначены для широкого пользователя.

Их основным потребителем был военно-промышленный комплекс.

С появлением огромного числа банков, акционерных обществ и частных компаний ситуация резко изменилась.

В настоящее время обработка и хранение информации являются важнейшими задачами.

Потеря информации или ее несвоевременное получение могут обернуться потерей денег. Именно этими обстоятельствами можно объяснить столь быстрый рост компьютерной техники и стремительное развитие электронных таблиц и систем управления базами данных в нашей стране и за рубежом.

Для оперативного, гибкого и эффективного управления предприятиями, фирмами и организациями различных форм собственности, телекоммуникационными средствами гражданского и военного назначения, информационно-вычислительными, экологическими, радиолокационными и радионавигационными системами широко внедряются системы автоматизированного управления, ядром которых являются базы данных (БД). При большом объеме информации и сложности производимых с ней операций проблема эффективности средств организации хранения, доступа и обработки данных приобретает особое значение.

Важность и значимость баз данных в современной жизни определяют серьезные требования, предъявляемые к

квалификации специалистов, создающих приложения на их основе.

В предлагаемом учебном пособии, предназначенном для студентов, обучающихся по направлению «Информатика и вычислительная техника», рассматриваются базовые вопросы теории проектирования баз данных и особенности разработки пользовательских приложений на основе СУБД Microsoft Access.

ГЛАВА 1
ОСНОВЫ ТЕОРИИ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ

1.1. Определение и назначение баз данных. Системы управления базами данных

С самого начала развития вычислительной техники образовались два основных направления ее использования.

Первое направление — применение вычислительной техники для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную.

Второе направление — использование средств вычислительной техники в автоматических или автоматизированных информационных системах. В самом широком смысле информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация имеет довольно сложную структуру. Классическими примерами информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т.д.

Второе направление возникло несколько позже первого. Это связано с тем, что на заре вычислительной техники компьютеры обладали ограниченными возможностями. Надежное и долговременное хранение информации возможно только при наличии запоминающих устройств, сохраняющих информацию после выключения электрического питания. Оперативная память этим свойством обычно не обладает. Используемые в ранних ЭВМ два вида устройств внешней памяти — магнитные ленты и барабаны — были несовершенными. Магнитные ленты обладали достаточно большой емкостью, но по своей физической природе обеспечивали лишь последовательный доступ к данным. Магнитные барабаны, обеспечивая возможность произвольного доступа к данным, имели ограниченный размер. Появление новых носителей данных — в первую очередь, жестких дисков — дало толчок к работам по созданию информационных компьютерных систем.

Основу любой информационной системы составляет база данных, т.е. набор данных, организованных специальным образом.

В действующем в настоящее время Законе РФ «О правовой охране программ для электронных вычислительных машин и баз данных» от 23.09.92 № 3523-1 дается следующее определение: «База данных — это объективная форма представления и организации совокупности данных (например, статей, расчетов), систематизированных таким образом, чтобы эти данные могли быть найдены и обработаны с помощью ЭВМ».

Файл — это место фактического хранения информации. В файле различают структуру и собственно данные. Структура файла остается неизменной, а информация (данные) может изменяться при операциях обращения к нему.

В качестве основной структурообразующей единицы хранимых в файле данных принимается *хранимая запись*. Хранимые записи состоят из фиксированной совокупности *полей*, служащих для представления значений какого-либо типа (чисел, литерных строк, дат, булевских значений, денежных единиц и т.д.), и могут иметь формат фиксированной или переменной длины. Полям, как правило, присваиваются уникальные в данной базе имена, ассоциируемые с предметной областью. Если в качестве примера базы данных рассмотреть картотеку сотрудников некоторого абстрактного предприятия, то единицей хранимых данных может быть запись персональной информации по каждому сотруднику с полями: табельный номер (формат поля — целое число); фамилия, имя, отчество (формат поля — литерная строка определенной длины); дата рождения (формат поля — дата); заработная плата (формат поля — действительное число) и т.д.

Информационные системы ориентированы главным образом на хранение, выбор и модификацию постоянно существующей информации.

Структура информации зачастую очень сложна, и хотя структуры данных различны в разных информационных системах, между ними часто бывает много общего. На начальном этапе использования вычислительной техники для управления информацией проблемы структуризации данных решались индивидуально в каждой информационной системе.

Поскольку информационные системы содержат сложные структуры данных, дополнительные индивидуальные средства управления этими данными, являясь существенной частью информационных систем, практически повторялись от одной системы к другой. Стремление выделить общую часть информационных систем, ответственную за управление сложно структурированными данными, явилось первой побудительной причиной создания систем управления базами данных.

Компоненты наиболее полного варианта СУБД следующие:

• среда пользователя, дающая возможность непосредственного управления данными с клавиатуры;

• алгоритмический язык для программирования прикладных систем обработки данных, реализованный как интерпретатор (последний позволяет быстро создавать и отлаживать программы);

• компилятор для придания завершенной программе вида готового коммерческого продукта в форме независимого EXE-файла;

• программы-утилиты для быстрого программирования рутинных операций (генераторы отчетов, форм, таблиц, экранов, меню и других приложений).

Собственно СУБД — это инструментальная оболочка пользователя, а ввиду того, что такая среда ориентирована на немедленное удовлетворение запросов пользователя, — это всегда система-интерпретатор. Наличие в СУБД языка программирования позволяет создавать сложные системы обработки данных, ориентированные на конкретные задачи и конкретного пользователя.

1.2. Области применения баз данных

Автоматизированные информационные системы (АИС), основу которых составляют базы данных, появились в 60-х годах XX века в военной промышленности и бизнесе — там, где были накоплены значительные объемы полезных данных. Первоначально АИС были ориентированы лишь на работу с информацией фактического характера — числовыми или текстовыми характеристиками объектов. Затем по мере развития техники появилась возможность обработки текстовой информации на естественном языке.

Принципы хранения разных видов информации в АИС аналогичны, но алгоритмы ее обработки определяются характером информационных ресурсов. Соответственно различают два класса АИС: документальные и фактографические.

Документальные АИС служат для работы с документами на естественном языке. Наиболее распространенный тип документальных АИС — информационно-поисковые системы, предназначенные для накопления и подбора документов, удовлетворяющих заданным критериям. Эти системы могут выполнять просмотр и подборку монографий, публикаций в периодике, сообщений пресс-агентств, текстов законодательных актов и т.д.

Фактографические АИС оперируют фактическими сведениями, представленными в формализованном виде, и используются для решения задач обработки данных.

Обработка данных — специальный класс решаемых на ЭВМ задач, связанных с вводом, хранением, сортировкой, отбором и группировкой записей данных однородной структуры. К задачам этого класса относятся: учет товаров в магазинах и на складах;

начисление зарплаты; управление производством, финансами, телекоммуникациями и т. п.

Различают фактографические АИС оперативной обработки данных, подразумевающие быстрое обслуживание относительно простых запросов от большого числа пользователей, и фактографические АИС аналитической обработки, ориентированные на выполнение сложных запросов, требующих проведения статистической обработки исторических (накопленных за некоторый промежуток времени) данных, моделирования процессов предметной области и прогнозирования развития этих процессов.

Таким образом, АИС применяются в следующих областях:

- организация хранилищ данных;
- системы анализа данных;
- системы принятия решений;
- мобильные и персональные базы данных;
- географические базы данных;
- мультимедиа базы данных;
- распределенные информационные системы;
- базы данных для всемирной сети World Wide Web.

1.3. Информационная модель данных и ее состав

Каждая информационная система в зависимости от назначения имеет дело с той или иной частью конкретного мира, которую принято называть ее *предметной областью*. Анализ предметной области является необходимым начальным этапом разработки любой информационной системы. Именно на этом этапе определяются информационные потребности всей совокупности пользователей будущей системы, которые, в свою очередь, предопределяют содержание ее базы данных. Предметная область конкретной информационной системы рассматривается, прежде всего, как некоторая совокупность реальных *объектов*, которые представляют интерес для ее пользователей. Примерами объектов предметной области могут служить персональные ЭВМ, программные продукты и их пользователи. Каждый из этих объектов обладает определенным набором *свойств* (атрибутов). Так, например, компьютер характеризуется названием фирмы-производителя, идентификатором модели, типом микропроцессора, объемом оперативной и внешней памяти, типом графической карты и т. д.

Информационный объект — это описание некоторой сущности предметной области, т. е. *реального объекта, процесса, явления или события*. Информационный объект (сущность) образуется совокупностью логически взаимосвязанных атрибутов (свойств), представляющих собой качественные и количественные характеристики объекта (сущности).

Между объектами предметной области могут существовать связи, имеющие различный содержательный смысл. Эти связи могут быть обязательными или факультативными (необязательными).

Если вновь порожденный объект оказывается по необходимости связанным с каким-либо объектом предметной области, то между этими двумя объектами существует обязательная связь. В противном случае связь является факультативной.

Например, обязательная связь *Замещает* существует между двумя объектами СОТРУДНИК и ДОЛЖНОСТЬ в предметной области кадровой информационной системы, т.е. каждый принимаемый в организацию сотрудник зачисляется на какую-либо должность и не может быть сотрудника, не замещающего какой-либо должности. В то же время связь *Замещает* между типами объектов СОТРУДНИК и ДОЛЖНОСТЬ является факультативной, поскольку могут существовать вакантные должности.

Совокупность объектов предметной области и связей между ними характеризует *структуру* предметной области.

Множество объектов предметной области, значения атрибутов объектов и связи между ними могут изменяться во времени. Изменения могут сводиться к появлению новых или исключению из рассмотрения некоторых существующих объектов в предметной области, установлению новых или разрушению существующих связей между ними. Следовательно, с каждым моментом времени можно сопоставить некоторое *состояние* предметной области.

Информационно-логическая модель (*ИЛМ*) — *это совокупность информационных объектов* (*сущностей*) *предметной области и связей между ними.*

Процесс создания информационной модели начинается с определения концептуальных требований будущих пользователей БД.

Требования отдельных пользователей интегрируются в едином обобщенном представлении, которое называют концептуальной моделью данной предметной области (рис. 1.1). Такая модель ото-

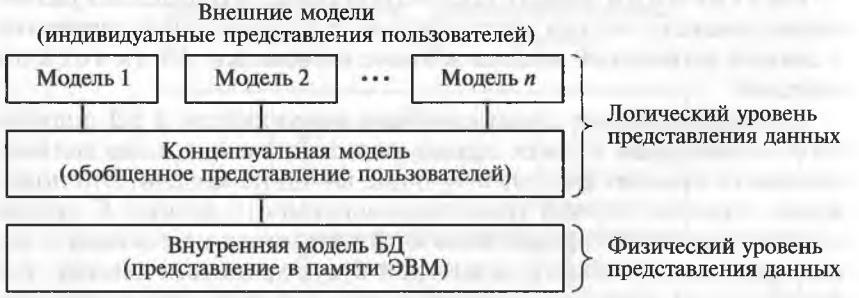


Рис. 1.1. Многоуровневое представление БД

бражает предметную область в виде взаимосвязанных объектов без указания способов их физического хранения.

Концептуальная модель представляет собой интегрированные концептуальные требования всех пользователей к базе данных данной предметной области. При этом усилия разработчика должны быть направлены в основном на структуризацию данных, принадлежащих будущим пользователям БД и выявление взаимосвязей между ними.

Возможно, что отраженные в концептуальной модели взаимосвязи между объектами окажутся впоследствии нереализуемыми средствами выбранной СУБД, что потребует ее изменения. Версия концептуальной модели, которая может быть реализована конкретной СУБД, называется *логической моделью.*

Логическая модель, отражающая логические связи между атрибутами объектов вне зависимости от их содержания и среды хранения, может быть реляционной, иерархической или сетевой.

Таким образом, *логическая модель отображает логические связи между информационными данными в данной концептуальной модели.*

Различным пользователям в информационной модели соответствуют различные подмножества ее логической модели, которые называются внешними моделями пользователей.

Таким образом, внешняя модель пользователя представляет собой отображение его концептуальных требований в логической модели и соответствует тем представлениям, которые этот пользователь получает о предметной области на основе логической модели. Следовательно, насколько хорошо спроектирована внешняя модель, настолько полно и точно информационная модель отображает предметную область и настолько полно и точно работает автоматизированная система управления этой предметной областью.

Логическая модель отображается в физическую память, которая может быть построена на электронных, магнитных, оптических, биологических или других принципах.

Внутренняя модель предметной области определяет размещение данных, методы доступа к ним и технику индексирования в данной логической модели и иначе называется физической моделью.

Информационные данные любого пользователя в БД должны быть независимы от всех других пользователей, т.е. не должны оказывать влияния на существующие внешние модели. Это положение отражает первый уровень независимости данных. С другой стороны, внешние модели пользователей никак не связаны с типом физической памяти, в которой будут храниться данные, и с физическими методами доступа к этим данным. Это положение отражает второй уровень независимости данных.

1.4. Три типа логических моделей баз данных

Ядром любой базы данных является модель данных. *Модель данных — это совокупность структур данных и операций их обработки.*

По способу установления связей между данными различают иерархическую, сетевую и реляционную модели.

Иерархическая модель позволяет строить базы данных с древовидной структурой, где каждый узел содержит свой тип данных (сущность). На верхнем уровне дерева в этой модели имеется один узел — корень, на следующем уровне располагаются узлы, связанные с этим корнем, затем узлы, связанные с узлами предыдущего уровня и т.д.

При этом каждый узел может иметь только одного предка (рис. 1.2).

Поиск данных в иерархической системе всегда начинается с корня. Затем производится спуск с одного уровня дерева на другой, пока не будет достигнут искомый уровень. Перемещения по системе от одной записи к другой осуществляются с помощью ссылок.

Основные достоинства иерархической модели — простота описания иерархических структур реального мира и быстрое выполнение запросов. Однако не всегда удобно каждый раз начинать поиск нужных данных с корня, а другого способа перемещения по базе в иерархических структурах нет.

Указанный недостаток снят в *сетевой* модели, где (по крайней мере, теоретически) возможны связи всех информационных объектов со всеми.

В примере, приведенном на рис. 1.3, каждый преподаватель может обучать многих (теоретически всех) студентов и каждый сту-

Рис. 1.2. Иерархическая древовидная структура модели БД

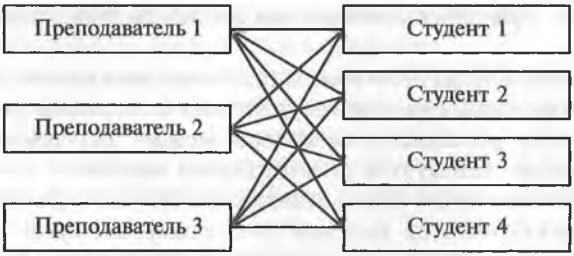


Рис. 1.3. Сетевая структура модели БД

дент может обучаться у многих (теоретически у всех) преподавателей. Поскольку на практике это, естественно, невозможно, приходится прибегать к некоторым ограничениям.

Использование иерархической и сетевой моделей ускоряет доступ к информации в базе данных. Однако, поскольку каждый элемент данных должен содержать ссылки на некоторые другие элементы, требуются значительные ресурсы как дисковой, так и основной памяти ЭВМ. Недостаточность основной памяти, конечно, снижает скорость обработки данных. Кроме того, для таких моделей характерна сложность реализации системы управления базами данных.

Реляционная модель (от англ. relation — отношение) была разработана в начале 70-х годов XX в. Коддом. Простота и гибкость этой модели привлекли к ней внимание разработчиков, и уже 80-х годах XX в. она получила широкое распространение. Таким образом реляционные СУБД оказались промышленным стандартом.

Реляционная модель опирается на систему понятий реляционной алгебры, важнейшими из которых являются таблица, строка, столбец, отношение и первичный ключ, а все операции в этом случае сводятся к манипуляциям с таблицами.

В реляционной модели информация представляется в виде прямоугольных таблиц, каждая из которых состоит из строк и столбцов и имеет имя, уникальное внутри базы данных.

Таблица отражает объект реального мира — *сущность*, а каждая ее строка (запись) отражает один конкретный экземпляр объекта — *экземпляр сущности*. Каждый столбец таблицы имеет уникальное для данной таблицы имя. Располагаются столбцы в соответствии с порядком следования их имен, принятом при создании таблицы.

В отличие от столбцов строки не имеют имен, порядок их следования в таблице не определен, а число — логически не ограничено. Так как строки в таблице не упорядочены, невозможно выбрать строку по ее позиции. Номер, имеющийся в файле у каждой

Название таблицы

Таблица 1. СОТРУДНИК

Номер пропуска	ФИО	Должность	Название отдела	Телефон

Первичный ключ табл. 1

Внешний ключ табл. 1

Название таблицы

Таблица 2. ОТДЕЛ

Название отдела	Расположение отдела	Назначение отдела

Первичный ключ табл. 2

Рис. 1.4. Организация ссылки от одной таблицы к другой

строки, не характеризует ее, так как его значение изменяется при удалении строк из таблицы. Логически не существует первой и последней строк.

Реляционные системы исключили необходимость сложной навигации, поскольку данные представлены в них не в виде одного файла, а независимыми наборами, и для отбора данных используются операции реляционной алгебры — прикладной теории множеств.

В каждой таблице реляционной модели должен быть столбец (или совокупность столбцов), значение которого однозначно идентифицирует каждую ее строку. Этот столбец (или совокупность столбцов) и называется *первичным ключом* таблицы (рис. 1.4).

Если таблица удовлетворяет требованию уникальности первичного ключа, она называется *отношением*. В реляционной модели все таблицы должны быть преобразованы в отношения. Отношения реляционной модели связаны между собой. Связи поддерживаются внешними ключами. *Внешний ключ* — это столбец (совокупность столбцов), значение которого однозначно характеризует значения первичного ключа другого отношения (таблицы).

Говорят, что отношение, в котором определен внешний ключ, ссылается на соответствующее отношение, в котором та же совокупность столбцов является первичным ключом.

В приведенном на рис. 1.4 примере отношение СОТРУДНИК ссылается на отношение ОТДЕЛ через название отдела.

Схема реляционной таблицы (отношения) представляет собой совокупность имен полей, образующих ее запись:

НАЗВАНИЕ ТАБЛИЦЫ (Поле 1, Поле 2, ..., Поле *n*).

Например, для таблиц, показанных на рис. 1.4, имеем следующие схемы (курсивом выделены первичные ключи):

СОТРУДНИК (*Номер пропуска*, ФИО, Должность, Название отдела, Телефон);
ОТДЕЛ (*Название отдела*, Расположение отдела, Назначение отдела).

Объектно-ориентированная модель баз данных начала разрабатываться в связи с появлением объектно-ориентированных языков программирования в 90-е годы XX века. Такого рода базы хранят методы классов, а иногда и постоянные объекты классов, что позволяет осуществлять беспрепятственную интеграцию между данными и их обработкой в приложениях.

Доминирование реляционной модели в современных СУБД определяется:

наличием развитой теории (реляционной алгебры);

наличием аппарата сведения других моделей данных к реляционной модели;

наличием специальных средств ускоренного доступа к информации;

наличием стандартизированного высокоуровневого языка запросов к БД, позволяющего манипулировать ими без знания конкретной физической организации БД во внешней памяти.

1.5. Типы взаимосвязей в модели

На практике часто используются связи, устанавливающие различные виды соответствия между объектами «связанных» типов, — это один к одному (1:1), один ко многим (1:M), многие ко многим (M:M).

Связь *один к одному* означает, что каждому экземпляру первого объекта (*A*) соответствует только один экземпляр второго объекта (*B*) и, наоборот, каждому экземпляру второго объекта (*B*) соответствует только один экземпляр первого объекта (*A*).

Связь *один ко многим* означает, что каждому экземпляру одного объекта (*A*) может соответствовать несколько экземпляров другого объекта (*B*), а каждому экземпляру второго объекта (*B*) может соответствовать только один экземпляр первого объекта (*A*).

Связь *многие ко многим* означает, что каждому экземпляру одного объекта (*A*) могут соответствовать несколько экземпляров второго объекта (*B*) и, наоборот, каждому экземпляру второго

объекта (*B*) могут соответствовать тоже несколько экземпляров первого объекта (*A*).

Пример 1.1. Рассмотрим совокупность следующих информационных объектов:

СТУДЕНТ (*Номер студента*, ФИО, Дата рождения, Номер группы);
СТИПЕНДИЯ (*Номер студента*, Размер стипендии);
ГРУППА (*Номер группы*, Специальность);
ПРЕПОДАВАТЕЛЬ (*Код преподавателя*, ФИО, Должность).

Здесь информационные объекты СТУДЕНТ и СТИПЕНДИЯ связаны отношением один к одному, так как каждый студент может иметь только одну стипендию и каждая стипендия может быть назначена только одному студенту.

Информационные объекты ГРУППА и СТУДЕНТ связаны отношением один ко многим, так как одна группа может включать в себя много студентов, в то время как каждый студент может обучаться только в одной группе.

Информационные объекты СТУДЕНТ и ПРЕПОДАВАТЕЛЬ связаны отношением многие ко многим, так как один студент может обучаться у многих преподавателей и один преподаватель может обучать многих студентов.

1.6. Обеспечение непротиворечивости и целостности данных в базе

Для пользователей АИС важно, чтобы база данных отображала предметную область однозначно и непротиворечиво, т.е. чтобы она удовлетворяла условию целостности.

Выделяют два основных типа ограничений по условию целостности данных в базе.

1. Каждая строка таблицы должна отличаться от остальных ее строк значением хотя бы одного столбца.

Пример 1.2. Сотрудники одного отдела могут оказаться полными тезками, иметь одинаковые должность и телефон. Наличие в табл. 1.4 столбца *Номер пропуска* превращает ее в отношение. Таким образом, первое ограничение по условию целостности данных в базе обеспечивается наличием в таблице-отношении первичного ключа.

2. Внешний ключ не может быть указателем на несуществующую строку той таблицы, на которую он ссылается. Это ограничение называется ограничением целостности данных в базе по ссылкам.

Пример 1.3. В столбце *Название отдела* таблицы СОТРУДНИК (см. рис. 1.4) хранятся сведения о принадлежности сотрудников к отделу, т.е. этот столбец является внешним ключом для ссылки на таблицу ОТДЕЛ. Для обеспечения ограничения целостности данных по ссылкам каждое

название отдела из таблицы СОТРУДНИК должно принадлежать конкретному столбцу из таблицы ОТДЕЛ.

В реальных базах данных названия не делают ключевыми из-за их длины, замедляющей процесс поиска, и возможности изменения, создающей сложности с сопровождением системы.

1.7. Основы реляционной алгебры

Поскольку каждая таблица в реляционной БД является отношением, действия над ними базируются на операциях реляционной алгебры. Исключение составляют лишь операции создания и заполнения таблиц данными (присваивания), а также операции описания и переименования столбцов таблицы.

В теории реляционной алгебры отношение рассматривается как множество, строки таблицы называются *кортежами*, столбцы — *атрибутами*. Над отношениями выполняются традиционные операции теории множеств.

1. Ограничение отношения (выборка) — создание нового отношения отбором в него строк отношения-операнда (исходного отношения), которые удовлетворяют условию ограничения.

2. Проекция отношения — создание нового отношения отбором в него определенных столбцов отношения-операнда.

3. Объединение отношений — создание нового отношения, содержащего все кортежи отношений-операндов. При этом операнды должны иметь одинаковые атрибуты.

Пример 1.4. Объединить поступающие из цехов отчеты о выпуске новой продукции за прошедший месяц, содержащие следующие данные: номер цеха, код продукции, дату выпуска и количество выпущенной продукции, с данными общей таблицы ВЫПУСК ПРОДУКЦИИ, имеющей ту же структуру. Для этого к кортежам

ВЫПУСК ПРОДУКЦИИ (Номер цеха, Код продукции, Дата выпуска, Количество)

добавляют кортежи

НОВАЯ ПРОДУКЦИЯ (Номер цеха, Код продукции, Дата выпуска, Количество).

Поскольку атрибуты приведенных операндов совпадают, таблица НОВАЯ ПРОДУКЦИЯ объединяется с исходной.

4. Пересечение отношений — создание нового отношения, содержащего строки, общие для сравниваемых операндов. При этом операнды должны иметь одинаковые атрибуты.

Пример 1.5. Рассмотрим пересечение отношений с выполнением операций ограничения и проекции.

Таблица 1.1

ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ по математике

Группа	Номер зачетной книжки	ФИО студента	Дата	Дисциплина	Оценка
1	1	Иванов И. И.	10.01.05	Математика	Отлично
1	2	Петров П. П.	10.01.05	Математика	Хорошо
1	3	Сидоров С. С.	10.01.05	Математика	Удовлетворительно
1	4	Прохоров Н. И.	10.01.05	Математика	Отлично
1	5	Симонов В. В.	10.01.05	Математика	Хорошо

Таблица 1.2

ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ по физике

Группа	Номер зачетной книжки	ФИО студента	Дата	Дисциплина	Оценка
1	1	Иванов И. И.	17.01.05	Физика	Отлично
1	2	Петров П. П.	17.01.05	Физика	Хорошо
1	3	Сидоров С. С.	17.01.05	Физика	Удовлетворительно
1	4	Прохоров Н. И.	17.01.05	Физика	Отлично
1	5	Симонов В. В.	17.01.05	Физика	Отлично

Пусть имеется набор экзаменационных ведомостей — отношений с совпадающими атрибутами (табл. 1.1, 1.2):

ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ (Группа, Номер зачетной книжки, ФИО студента, Дата, Дисциплина, Оценка).

Требуется подготовить список студентов, получивших только отличные оценки, в виде таблицы со столбцами *Номер зачетной книжки* и ФИО *студента*.

Таблица 1.3

Результат операции ограничения для ведомости по математике

Группа	Номер зачетной книжки	ФИО студента	Дата	Дисциплина	Оценка
1	1	Иванов И. И.	10.01.05	Математика	Отлично
1	4	Прохоров Н. И.	10.01.05	Математика	Отлично

Таблица 1.4

Результат операции ограничения для ведомости по физике

Группа	Номер зачетной книжки	ФИО студента	Дата	Дисциплина	Оценка
1	1	Иванов И. И.	17.01.05	Физика	Отлично
1	4	Прохоров Н. И.	17.01.05	Физика	Отлично
1	5	Симонов В. В.	17.01.05	Физика	Отлично

Таблица 1.5

Результат операции проекции для ведомости по математике

Номер зачетной книжки	ФИО студента
1	Иванов И. И.
4	Прохоров Н. И.

Таблица 1.6

Результат операции проекции для ведомости по физике

Номер зачетной книжки	ФИО студента
1	Иванов И. И.
4	Прохоров Н. И.
5	Симонов В. В.

Для экзаменационных ведомостей нужной группы сначала выполним ограничение исходных отношений, отобрав из каждого из них в новое отношение кортежи, удовлетворяющие следующему условию:

Оценка = Отлично.

В результате получим списки отличников группы по дисциплинам (табл. 1.3, 1.4).

Теперь выполним проекцию полученных отношений, отобрав из каждого из них только атрибуты *Номер зачетной книжки* и *ФИО студента*. Получим новые списки отличников, в которых остались только номера зачетных книжек и фамилии студентов (табл. 1.5, 1.6).

Таким образом получим искомое отношение — СПИСОК ОТЛИЧНИКОВ, содержащее номера зачетных книжек и фамилии, общие для всех списков отличников (табл. 1.7).

Таблица 1.7

СПИСОК ОТЛИЧНИКОВ

Номер зачетной книжки	ФИО студента
1	Иванов И. И.
4	Прохоров Н. И.

5. Разность отношений — создание нового отношения, содержащего строки 1-го операнда, отсутствующие во 2-м операнде. При этом операнды должны иметь одинаковые атрибуты.

Пример 1.6. Требуется, используя ежемесячные отчеты цехов (см.

пример 1.4), подготовить сведения о выпуске новых видов продукции за последний квартал.

Для решения этой задачи выполняем ограничение отношения ВЫПУСК ПРОДУКЦИИ по следующему условию: дата выпуска меньше последней даты прошлого квартала.

Результат такого ограничения поместим в исходную таблицу.

Затем выполним следующее ограничение для исходной таблицы: дата выпуска меньше первой даты прошлого квартала. Полученный результат занесем в конечную таблицу.

Разность отношений исходной и конечной таблиц даст искомые сведения.

6. Произведение отношений — создание нового отношения, в котором имеются все атрибуты 1-го и 2-го операндов, а строки получены попарным сцеплением строк их отношений. Число кортежей — мощность нового отношения — равна произведению мощностей 1-го и 2-го отношений. При этом множества атрибутов отношений не должны пересекаться.

Произведение отношений используется при решении задач подбора пар из двух множеств, например поставщиков и потребителей. Для этого сначала составляют все возможные пары, а затем по конкретному критерию отбирают из них подходящие.

Пример 1.7. По двум заданным отношениям (табл. 1.8, 1.9) требуется найти произведение (табл. 1.10).

Таблица 1.8

ПОСТАВЩИК

Поставщик
Поставщик 1
Поставщик 2

Таблица 1.9

ПОТРЕБИТЕЛЬ

Потребитель
Потребитель 1
Потребитель 2

Таблица 1.10

Результат операции произведения

Поставщик	Потребитель
Поставщик 1	Потребитель1
Поставщик 1	Потребитель 2
Поставщик 2	Потребитель 1
Поставщик 2	Потребитель 2

7. Деление отношений — создание нового отношения, содержащего атрибуты 1-го операнда, отсутствующие во 2-м операнде, и кортежи 1-го операнда, которые совпали с кортежами 2-го операнда. Для выполнения этой операции 2-й операнд должен содержать лишь атрибуты, совпадающие с атрибутами 1-го.

Пример 1.8. Требуется отобрать студентов группы, получающих стипендию, используя список, содержащий следующие сведения: ФИО, дата рождения, шифр группы и признак наличия стипендии (да, нет).

Для решения задачи создадим вспомогательное отношение с атрибутами *Шифр группы* и *Признак наличия стипендии*. Затем заполним один кортеж этого отношения, поместив в него шифр заданной группы и отметку о получении стипендии (да).

В результате деления исходного списка на вспомогательное отношение получим искомый список с атрибутами *ФИО* и *Дата рождения*.

8. Соединение отношений — создание нового отношения, кортеж которого является результатом сцепления кортежей операндов (исходных отношений).

Различают соединения отношений двух видов: естественное и по условию.

При соединении отношений по условию производятся сцепление строк их операндов и проверка полученной строки на соответствие заданному условию. Если условие выполнено, то полученная строка включается в результирующее отношение.

При естественном соединении отношений производятся сцепление строк их операндов и включение полученной строки в результирующее отношение без проверки. Такое соединение используют, когда отношения-операнды обладают общими атрибутами.

Пример 1.9. Требуется соединить отношения СТУДЕНТ (табл. 1.11) и ОЦЕНКА (табл. 1.12), для которых общим атрибутом является *Номер зачетной книжки*.

Результат операции соединения представлен в табл. 1.13.

Таблица 1.11

СТУДЕНТ

ФИО	Дата рождения	Номер зачетной книжки
Иванов И. И.	22.12.80	1234
Петров П. П.	12.05.80	1235
Сидоров С. С.	30.09.80	1236

Таблица 1.12

ОЦЕНКА

Код дисциплины	Номер зачетной книжки	Оценка
1	1234	4
1	1235	3
2	1234	4
2	1235	3

Таблица 1.13

Результат операции соединения

ФИО	Дата рождения	Номер зачетной книжки	Код дисциплины	Номер зачетной книжки	Оценка
Иванов И. И.	22.12.80	1234	1	1234	4
Иванов И. И.	22.12.80	1234	2	1234	4
Петров П. П.	12.05.80	1235	1	1235	3
Петров П. П.	12.05.80	1235	2	1235	3
Сидоров С. С.	30.09.80	1236			

1.8. Нормализация баз данных

Одни и те же данные могут группироваться в таблицы различными способами. Группировка атрибутов в отношениях должна быть рациональной, т. е. минимизирующей дублирование данных и упрощающей процедуры их обработки и обновления. Устранение избыточности данных, являющееся одной из важнейших задач при проектировании баз данных, обеспечивается нормализацией.

Нормализация — это формальный аппарат ограничений на формирование таблиц (отношений), который позволяет устранить дублирование, обеспечивает непротиворечивость хранимых данных и уменьшает трудозатраты на ведение (ввод, корректировку) базы данных.

Процесс нормализации заключается в разложении (декомпозиции) исходных отношений БД на более простые отношения. При этом на каждой ступени этого процесса схемы отношений приводятся в нормальные формы. Для каждой ступени нормализации имеются наборы ограничений, которым должны удовлет-

ворять отношения БД. Тем самым удаляется из таблиц базы избыточная неключевая информация.

Процесс нормализации основан на понятии функциональной зависимости атрибутов: атрибут *A* зависит от атрибута *B* ($B \rightarrow A$), если в любой момент времени каждому значению атрибута *B* соответствует не более одного значения атрибута *A*.

Зависимость, при которой каждый неключевой атрибут зависит от всего составного ключа и не зависит от его частей, называется *полной функциональной зависимостью*. Если атрибут *A* зависит от атрибута *B*, а атрибут *B* зависит от атрибута *C* ($C \rightarrow B \rightarrow A$), но обратная зависимость при этом отсутствует, то зависимость *C* от *A* называется *транзитивной*.

Общее понятие нормализации подразделяется на несколько нормальных форм.

Информационный объект (сущность) находится в ***первой нормальной форме*** (1НФ), когда все его атрибуты имеют единственное значение. Если в каком-либо атрибуте есть повторяющиеся значения, объект (сущность) не находится в 1НФ, и упущен, по крайней мере, еще один информационный объект (еще одна сущность).

Например, задано следующее отношение:

ПРЕДМЕТ (*Код предмета*, Название, Цикл, Объем часов, Преподаватели).

Это отношение не находится в 1НФ, так как атрибут *Преподаватели* подразумевает возможность наличия нескольких фамилий преподавателей в записи, относящейся к какому-то конкретному предмету, что соответствует участию нескольких преподавателей в ведении одной дисциплины.

Переведем атрибут с повторяющимися значениями в новую сущность, назначим ей первичный ключ (*Код преподавателя*) и свяжем с исходной сущностью ссылкой на ее первичный ключ (*Код предмета*). В результате получим две сущности, причем во вторую сущность добавятся характеризующие ее атрибуты:

ПРЕДМЕТ (*Код предмета*, Название, Цикл, Объем часов);
ПРЕПОДАВАТЕЛЬ (*Код преподавателя*, ФИО, Должность, Оклад, Адрес, Код предмета).

Полученные выражения соответствуют случаю, когда несколько преподавателей могут вести один предмет, но каждый преподаватель не может вести более одной дисциплины. А если учесть, что на самом деле один лектор может читать более одной дисциплины, так же как одну и ту же дисциплину могут читать несколько лекторов, необходимо отказаться от жесткой привязки препо-

давателя к предмету в сущности ПРЕПОДАВАТЕЛЬ, создав дополнительную сущность ИЗУЧЕНИЕ, которая будет показывать, как связаны между собой преподаватели и предметы:

ПРЕДМЕТ (*Код предмета*, Название, Цикл, Объем часов);
ПРЕПОДАВАТЕЛЬ (*Код преподавателя*, ФИО, Должность, Оклад, Адрес);
ИЗУЧЕНИЕ (*Код предмета*, *Код преподавателя*).

Информационный объект находится во ***второй нормальной форме*** (2НФ), если он уже находится в первой нормальной форме и каждый его неидентифицирующий (описательный) атрибут зависит от всего уникального идентификатора информационного объекта. Если некий атрибут не зависит полностью от уникального идентификатора информационного объекта, значит, он внесен в состав этого информационного объекта ошибочно и его необходимо удалить. Нормализация в этом случае производится путем нахождения существующего информационного объекта, к которому данный атрибут относится, или созданием нового информационного объекта, в который атрибут должен быть помещен.

Возвратившись к последнему примеру, заметим, что атрибут *Цикл* в сущности ПРЕДМЕТ, характеризующий принадлежность предмета к циклу гуманитарных, естественно-научных, общепрофессиональных или специальных дисциплин, не полностью зависит от уникального идентификатора *Код предмета*, так как разные предметы могут иметь одно и то же значение атрибута *Цикл*. Перенесем этот атрибут в новую сущность ЦИКЛ и получим четыре взаимосвязанных сущности:

ПРЕДМЕТ (*Код предмета*, Название, Объем часов, Код цикла);
ЦИКЛ (*Код цикла*, Название цикла);
ПРЕПОДАВАТЕЛЬ (*Код преподавателя*, ФИО, Должность, Оклад, Адрес);
ИЗУЧЕНИЕ (*Код предмета*, Код преподавателя).

Информационный объект находится в ***третьей нормальной форме*** (3НФ), если он уже находится во второй нормальной форме и ни один его описательный атрибут не зависит от каких-либо других описательных атрибутов. Атрибуты, зависящие от других неидентифицирующих атрибутов, нормализуются путем перемещения зависимого атрибута и атрибута, от которого он зависит, в новый информационный объект.

В данном случае неключевые атрибуты *Должность* и *Оклад* находятся в транзитивной зависимости. Опасность такой зависимости состоит в том, что несколько человек могут работать в одной и той же должности. При изменении должностного оклада в этом

случае нужно будет менять данные в каждой записи, содержащей эту должность, следовательно, требуется создать новую сущность ДОЛЖНОСТЬ с находящимися в транзитивной зависимости атрибутами *Название должности* и *Оклад* и сделать ссылку от сущности ПРЕПОДАВАТЕЛЬ на сущность ДОЛЖНОСТЬ:

ПРЕДМЕТ (*Код предмета*, Название, Объем часов, Код цикла);
ЦИКЛ (*Код цикла*, Название цикла);
ПРЕПОДАВАТЕЛЬ (*Код преподавателя*, ФИО, Код должности, Адрес);
ДОЛЖНОСТЬ (*Код должности*, Название должности, Оклад);
ИЗУЧЕНИЕ (*Код предмета*, Код преподавателя).

1.9. Средства ускоренного доступа к данным

Чтобы пользователь чувствовал себя комфортно, время ожидания ответа на запрос к БД не должно превышать нескольких секунд. В связи с этим требованием специально разрабатываются методы ускорения выборки, позволяющие обойтись без полного перебора строк при выполнении реляционных операций модификации отношений и отбора данных.

Наиболее эффективны методы индексирования и хеширования значений ключей отношения.

Индексирование — логическая сортировка строк таблицы — заключается в создании вспомогательных файлов, содержащих упорядоченные списки значений ключей отношения со ссылками на строку отношения, в которой они находятся. Индексные файлы занимают дополнительную память, но резко ускоряют поиск благодаря применению метода половинного деления. Для одного отношения может быть создано несколько индексов. Кроме того, можно создать индекс для нескольких отношений, если они содержат одинаковые атрибуты, что позволит ускорить выполнение операций соединения этих отношений.

Индексы позволяют находить строки, в которых значения ключевых полей совпадают с заданным значением или принадлежат заданному интервалу.

Хеширование (hashing) — использование хэш-функций, которые вычисляют вес строки таблицы по значению ее ключевых атрибутов. Результат вычисления хэш-функции — целое число в диапазоне физических номеров строк таблицы.

Идеальная хэш-функция должна давать разные значения веса для разных ключевых атрибутов. Но это не всегда возможно. На практике обычно используют простые хэш-функции, например $f(k) = k \bmod p$, где k — целое число, первичный ключ отношения; p — простое целое число; mod — операция, вычисляющая оста-

ток при целочисленном делении. Если ключевой атрибут — строка символов, то для вычисления $f(k)$ выбирается один из методов преобразования строки в число, например вычисление контрольной суммы.

Для организации доступа к данным при хешировании создается таблица с пустыми строками, которая заполняется следующим образом:

- по первичному ключу новой строки вычисляется значение хэш-функции $f(k)$ и результат трактуется как номер строки в созданной таблице;
- если строка уже занята, производится проверка следующих строк по специальному алгоритму до тех пор, пока не будет обнаружено свободное место.

Аналогично производится поиск нужной строки:

- если после вычисления $f(k)$ на месте в таблице, которое соответствует вычисленному значению, оказывается пустая строка, значит, искомой строки просто нет;
- если значение ключа совпало с искомым, поиск заканчивается;
- если же значение ключа не совпало с искомым, проверяются следующие строки таблицы до обнаружения строки с нужным ключом (в этом случае искомая строка найдена) или пустой строки (в этом случае искомая строка отсутствует).

Если таблица заполнена не более чем на 60 %, то для размещения новой или поиска существующей строки необходимо проверить в среднем не более двух ячеек. Хеширование используют для поиска строк по точному совпадению значения ключевого атрибута кортежа с нужным значением ключа.

1.10. Этапы проектирования баз данных

На этапе проектирования базы данных разработчик должен определить, из каких таблиц должна состоять база данных, какие данные нужно поместить в каждую таблицу и как связать таблицы.

Следовательно, в результате проектирования определяются логическая структура базы данных, т. е. состав реляционных таблиц, их структура и межтабличные связи.

Для создания базы данных необходимо располагать описанием выбранной предметной области, охватывающим реальные объекты и процессы, а также определить все необходимые источники информации для удовлетворения предполагаемых запросов пользователей и потребности в обработке данных. На основе такого описания определяются состав и структура данных предметной области, которые должны находиться в базе и обеспечивать выполнение необходимых запросов и задач пользователей. Структура дан-

ных предметной области может отображаться информационно-логической моделью, на основе которой легко создается реляционная база данных.

Этапы проектирования и создания базы данных:

- построение информационно-логической модели данных предметной области;
- определение логической структуры реляционной базы данных;
- конструирование таблиц базы данных;
- создание схемы данных;
- ввод данных в таблицы (создание записей);
- разработка необходимых форм, запросов, макросов, модулей, отчетов;
- разработка пользовательского интерфейса.

В процессе разработки модели данных необходимо выделить информационные объекты, соответствующие требованиям нормализации данных, и определить связи между ними. Полученная модель позволит создать реляционную базу данных без дублирования, в которой обеспечиваются однократный ввод данных при первоначальной загрузке и корректировках, а также целостность данных при внесении изменений.

При разработке модели данных используются два подхода.

1. Сначала определяются основные задачи, для решения которых строится база, выявляются потребности задач в данных и соответственно определяются состав и структура информационных объектов.

2. Сразу устанавливаются типовые объекты предметной области.

Наиболее рационально сочетание этих подходов, так как на начальном этапе, как правило, нет исчерпывающих сведений обо всех задачах. Использование такой технологии тем более оправдано, что гибкие средства создания реляционной базы данных допускают на любом этапе разработки внесение изменений и модифицирование ее структуры без ущерба для введенных ранее данных.

Процесс выделения информационных объектов предметной области, отвечающих требованиям нормализации, может производиться на основе интуитивного или формального подхода. Теоретические основы формального подхода разработаны известным американским ученым Дж. Мартином и изложены в его монографиях по организации баз данных.

При интуитивном подходе легко выявить информационные объекты, соответствующие реальным объектам, однако получаемая при этом информационно-логическая модель, как правило, требует дальнейших преобразований, в частности преобразования много-многозначных связей между объектами. При таком подходе в случае отсутствия достаточного опыта возможны существенные ошибки. Последующая проверка выполнения требований нор-

мализации обычно показывает необходимость уточнения информационных объектов.

Рассмотрим формальные правила выделения информационных объектов:

на основе описания предметной области выявить документы и их атрибуты, подлежащие хранению в базе данных;

определить функциональные зависимости между атрибутами;

выбрать все зависимые атрибуты и указать для каждого все его ключевые атрибуты, т.е. атрибуты, от которых он зависит;

сгруппировать атрибуты, одинаково зависимые от ключевых атрибутов. (Полученные группы зависимых атрибутов вместе с их ключевыми атрибутами образуют информационные объекты.)

При определении логической структуры реляционной базы данных на основе модели каждый информационный объект адекватно отображается реляционной таблицей, а связи между этими таблицами соответствуют связям между информационными объектами.

В процессе создания БД сначала конструируются таблицы, соответствующие информационным объектам построенной модели данных. Далее может создаваться схема данных, в которой фиксируются существующие логические связи между таблицами, соответствующие связям информационных объектов. В схеме данных могут быть заданы параметры поддержания целостности базы данных, если модель была разработана в соответствии с требованиями нормализации. Целостность данных означает, что в БД установлены и корректно поддерживаются взаимосвязи между записями разных таблиц при загрузке, добавлении и удалении записей в связанных таблицах, а также при изменении значений ключевых полей.

После формирования схемы данных осуществляется ввод непротиворечивых данных из документов предметной области.

На основе созданной базы формируются необходимые запросы, формы, макросы, модули, отчеты, производящие требуемую обработку данных и их представление.

С помощью встроенных средств и инструментов базы данных создается пользовательский интерфейс, позволяющий управлять процессами ввода, хранения, обработки, обновления и представления информации.

1.11. Проектирование базы данных на основе модели типа объект — отношение

Имеется целый ряд методик создания информационно-логических моделей. Наиболее популярна в настоящее время методика с использованием ERD (entity-relationship diagram). В русскоязычной литературе эти диаграммы называют объект — отношение либо

сущность — связь. Модель с использованием ERD была предложена Ченом в 1976 г. К настоящему времени разработано несколько ее разновидностей, но все они базируются на графических диаграммах, предложенных Ченом, которые конструируются из небольшого числа компонентов и благодаря наглядности представления широко используются в CASE-средствах (Computer-Aided Software/System Engineering).

Рассмотрим используемые при проектировании терминологию и обозначения.

Сущность (Entity) — реальный либо воображаемый объект, имеющий существенное значение для рассматриваемой предметной области, информация о котором подлежит хранению.

Каждая сущность должна обладать уникальным идентификатором. Каждый экземпляр сущности должен однозначно идентифицироваться и отличаться от всех других экземпляров данного типа, т. е. каждая сущность должна:

- иметь уникальное имя, причем это имя должно всегда однозначно интерпретироваться (определять сущность), и наоборот, одна интерпретация не может применяться к различным именам, если только они не являются псевдонимами;
- обладать одним или несколькими атрибутами, которые либо принадлежат ей, либо наследуются ею через связь;
- обладать одним или несколькими атрибутами, которые однозначно идентифицируют каждый ее экземпляр.

Сущность может быть независимой и зависимой (рис. 1.5). Признаком зависимой сущности служит наличие у нее наследуемых через связь атрибутов.

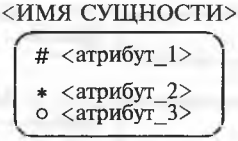
Каждая сущность может обладать любым количеством связей с другими сущностями модели.

Связь (relation) — поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области. При этом одна из участвующих в связи сущностей — независимая — называется родительской, а другая — зависимая — называется дочерней, или сущностью-потомком. Как правило, каждый экземпляр родительской сущности ассоциирован с произвольным (в том числе нулевым) числом экземпляров дочерней сущности. Каждый экземпляр сущности-потомка ассоциирован в точности с одним экземпляром сущности-родителя. Таким образом, экзем-

Рис. 1.5. Графическое обозначение зависимой сущности *A* и независимой сущности *B*

Нотация	Обозначение связи
IDEF1	много одна
IE	много одна
Любая	необязательная обязательная

Рис 1.6. Графическое изображение связей



ключевой атрибут
* обязательный атрибут
o необязательный атрибут

Рис 1.7. Графическое отображение характеристики атрибута

пляр сущности-потомка может существовать только при существовании сущности родителя.

Связи дается имя, выражаемое глаголом и помещаемое возле линии связи. Имя каждой связи между двумя данными сущностями должно быть уникальным, но имена связей в модели не обязаны быть уникальными. Каждая связь имеет определение. Определение связи образуют соединением имени сущности-родителя, имени связи, выражения степени связи и имени сущности-потомка.

Например, связь продавца с контрактом может быть определена следующим образом:

- продавец может получить вознаграждение за один или более контрактов;
- контракт должен быть инициирован одним продавцом.

На диаграммах связь изображается отрезками. Концы этих отрезков с помощью специальных обозначений (рис. 1.6) указывают степень связи. Кроме того, характер линии (штриховая или сплошная) указывает обязательность связи.

Атрибут — любая характеристика сущности, значимая для рассматриваемой предметной области. Он предназначен для квалификации, идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут представляет собой тип характеристик (свойств), ассоциированных с множеством реальных или абстрактных объектов (людей, мест, событий, состояний, идей, пар предметов и т.д.). Экземпляр атрибута — это определенная характеристика конкретного экземпляра сущности. Экземпляр атрибута определяется типом характеристики (например, цвет) и ее значением (например, лиловый), называемым значением атрибута. В ERD-модели атрибуты ассоциируются с конкретными сущностями. Каждый экземпляр сущности должен обладать одним конкретным значением для каждого своего атрибута.

Атрибут может быть либо обязательным, либо необязательным (рис. 1.7). Обязательность означает, что атрибут не может принимать

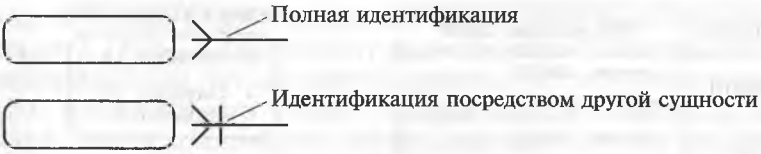


Рис 1.8. Графическое отображение характера идентификации

неопределенных значений (null values). Атрибут может быть либо описательным (т.е. обычным дескриптором сущности), либо входить в состав уникального идентификатора (первичного ключа).

Уникальный идентификатор — это атрибут или совокупность атрибутов и/или связей, однозначно характеризующая каждый экземпляр данного типа сущности. В случае полной идентификации экземпляр данного типа сущности полностью идентифицируется своими собственными ключевыми атрибутами, в противном случае в идентификации участвуют также атрибуты сущности-родителя.

Характер идентификации отображается в диаграмме на линии связи (рис. 1.8).

Каждый атрибут идентифицируется уникальным именем, выражаемым существительным, описывающим представляемую атрибутом характеристику. Атрибуты изображаются в виде списка имен внутри блока ассоциированной сущности, причем каждый атрибут занимает отдельную строку. Атрибуты, определяющие первичный ключ, размещаются вверху списка и выделяются знаком #.

Каждая сущность должна обладать хотя бы одним возможным ключом. Возможный ключ сущности — это один или несколько атрибутов со значениями, однозначно определяющими каждый ее экземпляр. При существовании нескольких возможных ключей один из них обозначается в качестве первичного, а остальные — как альтернативные.

В настоящее время на основе подхода Чена созданы IDEF1X-диаграммы, разработанные с учетом таких требований, как простота в изучении и возможность автоматизации. IDEF1X-диаграммы используются рядом распространенных CASE-средств (в частности, ERwin, Design/IDEF).

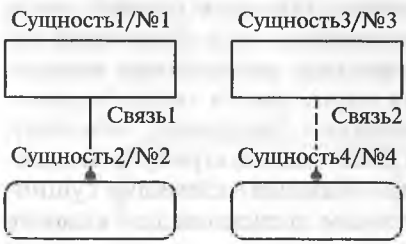


Рис 1.9. Изображение сущностей и связей

Сущность в IDEF1X-диаграммах называется независимой от идентификаторов, или просто независимой, если каждый ее экземпляр может быть однозначно идентифицирован без определения отношений этого экземпля-

ра с другими сущностями. Сущность называется зависимой от идентификаторов, или просто зависимой, если однозначная идентификация ее экземпляра зависит от отношения этого экземпляра к другой сущности (рис. 1.9).

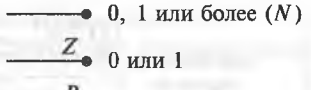


Рис 1.10. Мощность связи

Каждой сущности присваиваются уникальное имя и номер, разделяемые косой чертой и помещаемые над блоком.

Если экземпляр сущности-потомка однозначно определяется своей связью с сущностью-родителем, то связь является идентифицирующей, в противном случае связь неидентифицирующая.

Идентифицирующая связь между сущностью-родителем и сущностью-потомком изображается сплошной линией (см. рис. 1.9, где № 2 — зависимая сущность, Связь1 — идентифицирующая связь). Сущность-потомок в идентифицирующей связи является зависимой от идентификатора. Сущность-родитель в идентифицирующей связи может быть как независимой, так и зависимой от идентификатора (что определяется ее связями с другими сущностями).

Неидентифицирующая связь изображается пунктирной линией (см. рис. 1.9, где № 4 — независимая сущность; Связь2 — неидентифицирующая связь). Сущность-потомок в неидентифицирующей связи будет независимой от идентификатора, если она не является также сущностью-потомком в какой-либо идентифицирующей связи.

Связь может дополнительно определяться степенью или мощностью (числом экземпляров сущности-потомка, которое может существовать для каждого экземпляра сущности-родителя). В IDEF1X возможны следующие мощности связей:

- каждый экземпляр сущности-родителя может иметь нуль, один или более связанных с ним экземпляров сущности-потомка;
- каждый экземпляр сущности-родителя должен иметь не менее одного связанного с ним экземпляра сущности-потомка;
- каждый экземпляр сущности-родителя должен иметь не более одного связанного с ним экземпляра сущности-потомка;
- каждый экземпляр сущности-родителя связан с некоторым фиксированным числом экземпляров сущности-потомка.

Мощность связи обозначается, как показано на рис. 1.10 (мощность по умолчанию — *N*).

Атрибуты изображаются в виде списка имен внутри блока сущности. Атрибуты, определяющие первичный ключ, размещаются вверху списка и отделяются от других атрибутов горизонтальной чертой (рис. 1.11).

Сущности могут иметь также внешние ключи (Foreign Key). При идентифицирующей связи они используются в качестве части или

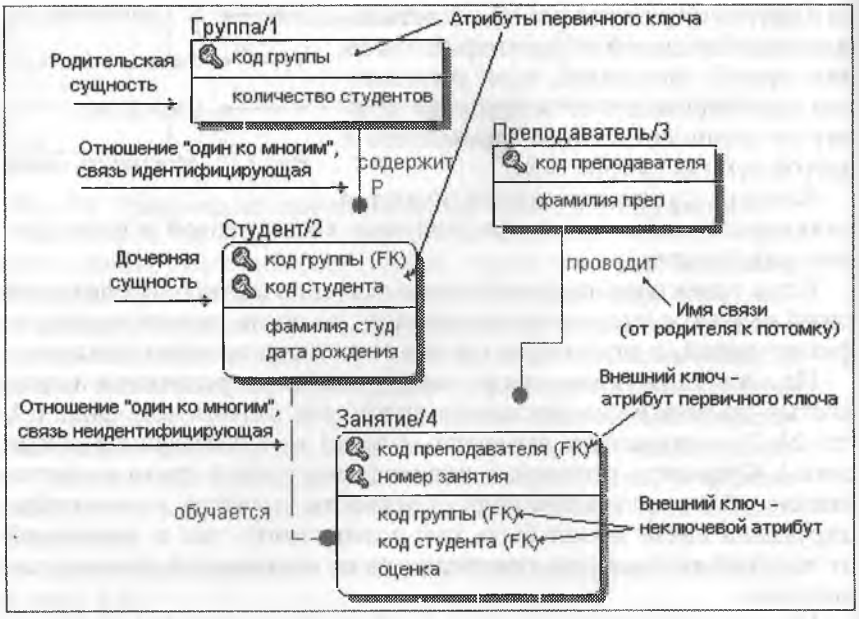


Рис 1.11. Атрибуты и первичные ключи

целого первичного ключа, при неидентифицирующей — служат неключевыми атрибутами.

В списке атрибутов внешний ключ отмечается буквами FK в скобках.

В результате описанных действий получается информационно-логическая модель, которая используется рядом распространенных CASE-средств, таких как ERwin, Design/IDEF. В свою очередь, CASE-технологии, имея высокие потенциальные возможности, позволяют при разработке баз данных и информационных систем обеспечить увеличение производительности труда, улучшение качества программных продуктов, поддержку унифицированного и согласованного стиля работы.

Контрольные вопросы и упражнения

1. Дать определение и описать назначение базы данных.
2. Дать определение и описать назначение системы управления базой данных.
3. Каковы основные функциональные возможности СУБД?
4. Какие модели данных вы знаете?
5. Определить понятие реляционной базы данных.
6. Пояснить назначение ключевых полей в реляционной базе данных.
7. Определить понятие ключа. Какие типы ключей вы знаете?
8. Что называется информационно-логической моделью базы данных?
9. Какие виды связей между объектами вам известны?

10. В чем заключается принцип нормализации отношений?

11. Каким требованиям должны отвечать отношения, находящиеся в первой, второй и третьей нормальных формах?

12. Каковы основные этапы проектирования баз данных?

13. Описать процесс проектирования базы данных на основе модели типа объект—отношение.

14. Разработать информационно-логическую модель БД информационной системы контроля за ходом выполнения учебного процесса в вузе, в которой должны храниться следующие сведения:

- преподаватели, работающие на кафедрах факультетов (ФИО, дата рождения, ученая степень, должность);
- студенты, обучаемые в составе учебных групп (ФИО, дата рождения, номер учебной группы);
- учебные группы (номер учебной группы, число студентов в группе, название специальности);
- изучаемые дисциплины (название дисциплины, число отводимых на изучение дисциплины часов);
- распределение преподавателей по дисциплинам (ФИО преподавателя, название дисциплины);
- результаты обучения (ФИО студента, название дисциплины, оценка).

15. Объединить два заданных отношения *A* и *B* с информацией о сотрудниках.

Отношение *A*

Табельный номер	ФИО	Зарплата, р
1	Иванов И. И.	1000
2	Петров П. П.	2000
3	Сидоров С. С.	3000

Отношение *B*

Табельный номер	ФИО	Зарплата, р
1	Иванов И. И.	1000
2	Петров П. П.	2500
4	Сидоров С. С.	3000

16. Найти пересечение отношений *A* и *B* из задания 15.

17. Найти разность отношений *A* и *B* из задания 15.

18. Найти произведение двух отношений *A* и *B* с информацией о поставщиках и деталях.

Отношение *A* (поставщики)

Номер поставщика	Наименование поставщика
1	Иванов И. И.
2	Петров П. П.
3	Сидоров С. С.

Отношение *B* (детали)

Номер детали	Наименование детали
1	Болт
2	Гайка
3	Винт

19. Выполнить выборку сотрудников с зарплатой меньше 3000 р. из информационного отношения *A*.

Отношение *A*

Табельный номер	Фамилия	Зарплата, р.
1	Иванов	1000
2	Петров	2000
3	Сидоров	3000

20. Выполнить проекцию отношения *A*, отбирая только номер поставщика и город поставщика.

Отношение *A* (поставщики)

Номер поставщика	Наименование поставщика	Город поставщика
1	Иванов И. И.	Уфа
2	Петров П. П.	Москва
3	Сидоров С. С.	Москва
4	Сидоров С. С.	Челябинск

ГЛАВА 2
ИСПОЛЬЗОВАНИЕ СУБД ACCESS ДЛЯ СОЗДАНИЯ БАЗ ДАННЫХ

2.1. Основные характеристики и возможности СУБД Access

Группа реляционных СУБД представлена на рынке программных продуктов очень широко. Это, например, такие системы, как Paradox, Clarion, dBASE, FoxBASE, FoxPro, Clipper, Access. Важнейшей характеристикой любой СУБД является используемый в ней тип транслятора (интерпретатор или компилятор). Программы, написанные для системы-интерпретатора, не работают без наличия самой этой системы. В настоящее время скорость работы таких программ не уступает скорости программ, сгенерированных компилятором. Бесспорным преимуществом интерпретаторов для программистов является удобство разработки и отладки программных продуктов, а также освоение языка. Из перечисленных СУБД dBASE, FoxPro, Access являются интерпретаторами, а Clipper — компилятором. В пакетах dBASE и FoxPro имеется компилятор, позволяющий при желании сформировать EXE-файлы готовых программ. Недостатком систем-компиляторов являются большие суммарные затраты времени на многократную компиляцию и сборку (линковку) исходных модулей программы при ее отладке.

СУБД Access (фирма Microsoft) имеет достаточно высокие скоростные характеристики и входит в состав чрезвычайно популярного в нашей стране и за рубежом пакета Microsoft Office. Набор команд и функций, предлагаемых разработчикам программных продуктов в среде Access, по мощи и гибкости отвечает большинству современных требований к представлению и обработке данных. В Access поддерживаются разнообразные всплывающие и многоуровневые меню, работа с окнами и мышью, реализованы функции низкоуровневого доступа к файлам, управления цветами, настройки принтера, представления данных в виде электронных таблиц и т. п. Система также обладает средствами быстрой генерации экранов, отчетов и меню, поддерживает язык управления запросами SQL, имеет встроенный язык Visual Basic for Applications (VBA), хорошо работает в сети. СУБД Access позволяет использовать другие компоненты пакета Microsoft Office, такие как текстовый процессор Word for Windows, электронные таблицы Excel и т.д.

Перечисленные факторы определили выбор СУБД Access в качестве среды для практического изучения вопросов проектирования баз данных в данной книге.

Приведем некоторые из средств Microsoft Access, существенно упрощающие разработку приложений.

1. **Процедуры обработки событий и модули форм и отчетов.** На встроенном языке VBA можно писать процедуры обработки событий, возникающих в формах и отчетах. Процедуры обработки событий хранятся в модулях, связанных с конкретными формами и отчетами, в результате чего код становится частью макета формы или отчета. Кроме того, существует возможность вызова функции VBA свойством события.

2. **Свойства, определяемые в процессе выполнения.** С помощью макроса или процедуры обработки событий можно определить практически любое свойство формы или отчета в процессе выполнения в ответ на возникновение события в форме или отчете.

3. **Модель событий.** Модель событий, похожая на используемую в языке Microsoft Visual Basic, позволяет приложениям реагировать на возникновение различных событий, например нажатие клавиши на клавиатуре, перемещение мыши или истечение определенного интервала времени.

4. **Использование обработки данных с помощью VBA.** С помощью языка VBA можно определять и обрабатывать различные объекты, в том числе, таблицы, запросы, поля, индексы, связи, формы, отчеты и элементы управления.

5. **Построитель меню.** Предназначен для помощи при создании специальных меню в приложениях. Кроме того, специальные меню могут содержать подменю.

6. **Улучшенные средства отладки.** Помимо установки точек прерывания и пошагового выполнения программ на языке VBA, можно вывести на экран список всех активных процедур. Для этого следует выбрать команду *Вызовы* в меню *Вид* или нажать кнопку [Вызовы] на панели инструментов.

7. **Процедура обработки ошибок.** Помимо традиционных способов обработки ошибок возможно использование процедуры обработки события Error для перехвата ошибок при выполнении программ и макросов.

8. **Улучшенный интерфейс защиты.** Команды и окна диалога защиты упрощают процедуру защиты и смены владельца объекта.

9. **Программная поддержка механизма OLE.** С помощью механизма OLE можно обрабатывать объекты из других приложений.

10. **Программы-надстройки.** С помощью VBA можно создавать программы-надстройки, например нестандартные мастера и построители. Мастер — средство Microsoft Access, которое сначала задает пользователю вопросы, а затем создает объект (таблицу, запрос, форму, отчет и т.д.) в соответствии с его указаниями.

Диспетчер надстроек существенно упрощает процедуру установки программ-надстроек в Microsoft Access.

Мастера Access

Access позволяет даже мало подготовленному пользователю создать свою БД, обрабатывать данные с помощью форм, запросов и отчетов, проводить анализ таблиц БД и выполнять ряд других работ. Практически для любых работ с БД в Access имеется свой мастер, который помогает их выполнять.

Мастер по анализу таблиц позволяет повысить эффективность базы данных за счет нормализации данных. Он разделяет ненормализованную таблицу на две или несколько таблиц меньшего размера, в которых данные сохраняются без повторения.

Мастера по созданию форм и отчетов упрощают и ускоряют процесс создания многотабличных форм и отчетов. Новые форма и отчет могут наследовать примененный к таблице-источнику записей фильтр. Мастера по разработке форм и отчетов автоматически создают инструкцию SQL, определяющую источник записей для формы или отчета, поэтому отпадает необходимость в создании запроса.

Для изменения вида формы, отчета или отдельных элементов может быть использован мастер, вызываемый кнопкой [Автоформат].

Мастер подстановок создает в поле таблицы раскрывающийся список значений из другой таблицы для выбора и ввода нужного значения. Для создания такого поля со списком достаточно в режиме конструктора таблицы выбрать тип данных этого поля — *Мастер подстановок*. Мастер подстановок можно вызвать в режиме таблицы командой меню *Вставка|Столбец подстановок*. Созданный в данном поле таблицы список наследуется при включении этого поля в форму.

Мастера по импорту/экспорту позволяют просматривать данные при импорте/экспорте текста или электронных таблиц, а также при экспорте данных Microsoft Access в текстовые файлы.

Мастер защиты при необходимости эвакуирует данные, для чего создает новую базу данных, копирует в нее все объекты из исходной базы данных, снимает все права, присвоенные членам группы пользователей, и шифрует новую базу данных. После завершения работы мастера администратор может присвоить новые права доступа пользователям и группам пользователей.

Мастер по разделению базы данных позволяет разделить ее на два файла, в первый из которых помещаются таблицы, а во второй — запросы, формы, отчеты, макросы и модули. При этом пользователи, работающие в сети, имея общий источник данных, смогут устраивать формы, отчеты и другие объекты, используемые для обработки данных, по своему усмотрению.

Использование технологии Windows в среде Access

Microsoft Access как средство создания реляционных БД использует все достоинства технологии Windows.

Среди достоинств средств Access выделим следующие.

1. СУБД Access полностью совместима с такими компонентами пакета Microsoft Office, как электронные таблицы Excel и текстовый процессор Word.

2. Access обеспечивает возможность динамического обмена данными DDE (Dynamic Data Exchange) с любым приложением Windows, поддерживающим DDE.

3. Access поддерживает также механизм OLE, обеспечивающий связь и внедрение объектов различных приложений, т.е. установление связи с объектами другого приложения и внедрение объекта в данное приложение БД. Причем достоинством внедренного объекта является то, что при его активизации открывается программа, которая его создала, поэтому новый пользователь имеет возможность изменить объект по своему усмотрению. При использовании механизма OLE как связи с объектом для другого приложения, объект по-прежнему сохраняется в файле приложения-источника. Следовательно, такой объект может обновляться независимо от приложения-потребителя, вызвавшего его, а в базе данных при этом можно всегда иметь последнюю версию объекта.

Внедряемыми или связываемыми объектами могут быть документы различных приложений Windows — рисунки, графики, электронные таблицы или звуковые файлы. Например, в таблице наряду с обычными реквизитами, характеризующими информационный объект, может храниться любая графическая информация о нем — схемы, чертежи, диаграммы и т.п. Таким образом в Access расширяется традиционное понятие данных, хранимых в базе.

4. Access распространил широко используемый в Windows метод drag-and-drop (перетащить и отпустить) на работу с формами и отчетами. Например, для создания подчиненных формы и отчета можно заранее перетащить подготовленные форму и отчет из окна базы данных. Также можно перетащить таблицу и запрос, из которых автоматически создаются подчиненная форма и запрос.

5. Access может использовать данные других СУБД, т.е. в ней непосредственно могут обрабатываться файлы систем Paradox, dBase, FoxPro, Btrieve.

6. Access может использовать все файлы СУБД, поддерживающие стандарт открытого доступа к данным ODBC (Open Database Connectivity) — Oracle, Microsoft SQL Server, Sybase SQL Server. Так, ODBC определяет язык и набор протоколов для обмена между пользовательским приложением и самими данными, хранящимися в сервере, т.е. используется как средство коммуника-

ции между настольным персональным компьютером (клиентом) и сервером.

2.2. Основные компоненты СУБД Access

Основными компонентами (объектами) базы данных являются таблицы, запросы, формы, отчеты, макросы и модули.

Таблица — фундаментальная структура системы управления реляционными базами данных. В Microsoft Access таблица — это объект, предназначенный для хранения данных в виде записей (строк) и полей (столбцов). При этом каждое поле содержит отдельную часть записи (например, фамилию, должность или инвентарный номер). Обычно каждая таблица используется для хранения сведений по одному конкретному вопросу (например, о сотрудниках или заказах).

Запрос — вопрос о данных, хранящихся в таблицах, или инструкция на отбор записей, подлежащих изменению.

Перечислим типы запросов, которые могут быть созданы с помощью Microsoft Access:

- *запрос-выборка*, задающий вопрос о данных, хранящихся в таблицах, и представляющий полученный динамический набор в режиме формы или таблицы без изменения данных. Изменения, внесенные в динамический набор, отражаются в базовых таблицах;
- *запрос-изменение*, изменяющий или перемещающий данные. К этому типу относятся запрос на добавление записей, запрос на удаление записей, запрос на создание таблицы или запрос на ее обновление;
- *перекрестные запросы*, предназначенные для группирования данных и представления их в компактном виде;
- *запрос с параметрами*, позволяющий определить одно или несколько условий отбора во время выполнения запроса;
- *запросы SQL*, которые могут быть созданы только с помощью инструкций SQL в режиме SQL: запрос-объединение, запрос к серверу и управляющий запрос. Язык SQL (Structured Query Language) — это язык запросов, который часто используется при анализе, обновлении и обработке реляционных баз данных (например, Microsoft Access).

Форма — это объект Microsoft Access, в котором можно разместить элементы управления, предназначенные для ввода, изображения и изменения данных в полях таблиц.

Отчет — это объект Microsoft Access, который позволяет представлять определенную пользователем информацию в определенном виде, просматривать и распечатывать ее.

Макрос — одна или несколько макрокоманд, которые можно использовать для автоматизации конкретной задачи.

Макрокоманда — основной строительный блок макроса; самостоятельная инструкция, которая может быть объединена с другими макрокомандами для автоматизации выполнения задачи.

Модуль — набор описаний, инструкций и процедур, сохраненных под одним именем. В Microsoft Access имеется три типа модулей: формы, отчета и общий. Модули форм и отчетов содержат локальную программу для форм или отчетов. Если процедуры общего модуля явным образом не объявлены личными для модуля, в котором они появляются, значит, они распознаются и могут вызываться процедурами из других модулей этой базы данных.

База данных может содержать несколько модулей, в том числе общие модули, модули форм и модули отчетов.

2.3. Типы данных СУБД Access

Для каждого поля таблиц базы данных необходимо указывать тип данных. Тип данных определяет вид и диапазон допустимых значений, которые могут быть введены в поле, а также объем памяти, выделяющийся для этого поля. Перечень типов данных полей и описание значений, сохраняемых в таких полях, приведены в табл. 2.1.

Таблица 2.1

Типы данных базы данных Microsoft Access

Тип данных	Содержимое типа данных
Текстовый	Текст и числа, например, имена и адреса, номера телефонов и почтовые индексы. Текстовое поле может содержать до 255 символов
Поле Memo	Длинный текст и числа, например комментарии и пояснения. Поле Memo может содержать до 64 000 символов
Числовой	Числовые данные, допускающие проведение математических расчетов, за исключением денежных. Свойство *Размер поля* (FieldSize) позволяет указывать различные типы числовых данных
Дата/время	Значения даты и времени. Пользователь имеет возможность выбора одного из многочисленных стандартных форматов или создания специального формата
Денежный	Денежные значения (не рекомендуется использовать для проведения денежных расчетов значения, принадлежащие к числовому типу данных, так как последние могут округляться при расчетах), которые всегда выводятся с указанным числом десятичных знаков после запятой

Окончание табл. 2.1

Тип данных	Содержимое типа данных
Счетчик	Автоматически вставляющиеся последовательные номера. Нумерация начинается с единицы. Поле счетчика, удобное для создания ключа, является совместимым с полем числового типа, для которого в свойстве *Размер поля* (FieldSize) указано значение *Длинное целое*
Логический	Значения *Да/Нет*, *Истина/Ложь*, *Вкл./Выкл.*
Поле объекта OLE	Объекты, созданные в других программах, поддерживающих протокол OLE, которые связываются или внедряются в базу данных Microsoft Access через элемент управления в форме или отчете

2.4. Создание новой базы данных

Создание новой базы данных Access осуществляется в соответствии с ее структурой, полученной в результате внемашинного проектирования, заключающегося в создании информационно-логической модели предметной области. Структура реляционной базы данных определяется составом таблиц и их взаимосвязями. Создание реляционной базы данных с помощью СУБД Access на компьютере начинается с формирования структуры таблиц. При этом формируется состав полей и задается их описание. После формирования структуры таблиц создается схема данных, в которой устанавливаются связи между таблицами. Access запоминает и использует эти связи при заполнении таблиц и обработке данных. Завершается создание базы данных процедурой заполнения таблиц конкретной информацией.

После запуска MS Access одновременно с окном базы данных открывается первое диалоговое окно, позволяющее начать создание БД или открыть уже существующую. На закладках (кнопках) окна базы данных представлены основные типы ее объектов: *Таблицы*, *Запросы*, *Формы*, *Отчеты*, *Макросы*, *Модули*. Рабочее поле окна базы данных предназначено для отображения списка объектов Access выбранного типа (рис. 2.1).

В составе окна базы данных находятся три управляющие кнопки:

первая кнопка [Открыть] выполняет три функции: непосредственно *Открыть*, если выбрана таблица, форма или запрос; *Просмотреть*, если выбран отчет; *Запустить*, если выбран макрос;

кнопка [Конструктор] позволяет перейти в режим доработки любого ранее созданного объекта;

кнопка [Создать] позволяет приступить к созданию нового объекта любого выбранного типа.

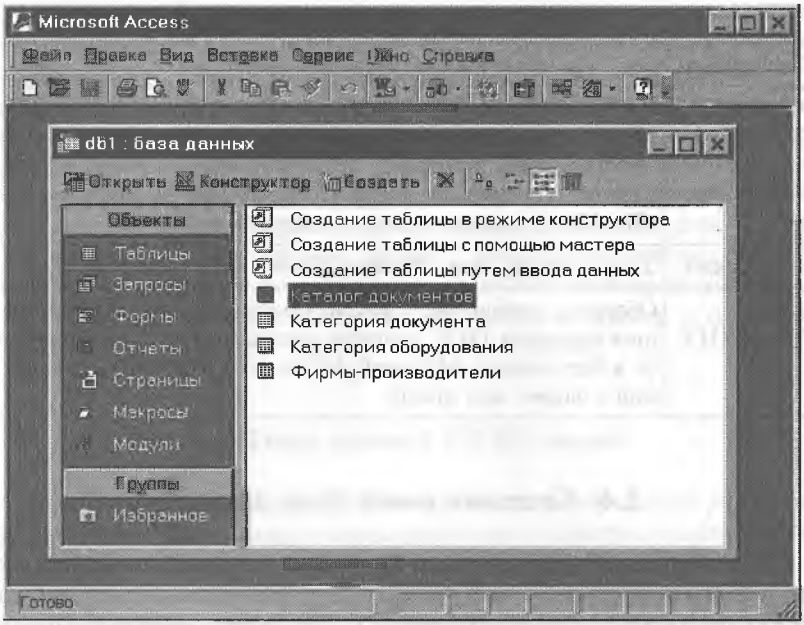


Рис. 2.1. Окно базы данных Microsoft Access

При создании объекта предоставляется возможность выбора режима его разработки. Это могут быть мастер, конструктор или какой-либо другой режим, зависящий от выбранного типа объекта.

2.5. Создание таблиц в СУБД Access

Таблицы создаются пользователем для хранения данных. Каждому объекту концептуальной модели предметной области соответствует одна таблица, которая состоит из полей (столбцов) и записей (строк). Каждое поле содержит одну характеристику (один атрибут) объекта предметной области. В записи собраны сведения об одном экземпляре этого объекта.

Работа по созданию базы данных на персональном компьютере (ПК) начинается с создания таблиц. После нажатия кнопки [Создать] в окне *База данных* разработчику предоставляется возможность выбора одного из пяти способов создания таблицы (табл. 2.2).

Если для создания таблицы выбран режим конструктора, то появляется окно *Таблица1 : таблица*, в котором определяется структура создаваемой таблицы базы данных (рис. 2.2).

Для определения поля в открывшемся окне задаются *Имя поля*, *Тип данных*, *Описание* (в виде краткого комментария), а также в

Таблица 2.2

Способы создания таблиц в СУБД Access

Способ	Описание
Режим таблицы	Для ввода данных предоставляется таблица с 30 полями. После ее сохранения Access сама решает, какой тип данных присвоить каждому полю. Недостатком способа является невозможность создания поля примечаний
Конструктор таблиц	Предоставляет возможность самостоятельного создания полей, выбора типа данных для них, размеров и установки свойства
Мастер таблиц	Предоставляет набор таблиц, из которых можно создавать таблицы по своему вкусу. При этом некоторые таблицы из этого набора могут полностью подойти для создаваемого приложения. Тип данных и другие свойства полей здесь уже определены
Импорт таблиц	Используется для создания копий таблиц приложений — источников данных. Иногда после импорта в таблице требуется изменить размер поля и некоторые другие свойства. Новой таблице присваивается имя, в ней определяется ключевое поле или Access делает это автоматически
Связь с таблицами	Устанавливается автоматическая непосредственная связь создаваемого приложения с данными таблиц других приложений, причем таблица остается в приложении-источнике и может использоваться несколькими приложениями. При этом экономятся емкость памяти, поскольку хранятся данные только одной таблицы, и время, так как информация обновляется только в таблице-источнике

разделе *Свойства поля* задаются общие свойства — на закладке *Общие* и тип элемента управления — на закладке *Подстановка*.

Каждое поле в таблице должно иметь уникальное имя, удовлетворяющее соглашениям об именах объектов в Access и являющееся комбинацией из букв, цифр, пробелов и специальных символов (за исключением знаков . ! «»). Максимальная длина имени — 64 символа.

Тип данных определяется значениями, которые предполагается вводить в поле, и операциями, которые будут выполняться с этими значениями. В Access допускается использование восьми типов данных. Список возможных типов данных каждого поля вызывается нажатием соответствующей кнопки.

Общие свойства поля задаются на закладке *Общие* для каждого поля и зависят от выбранного типа данных.

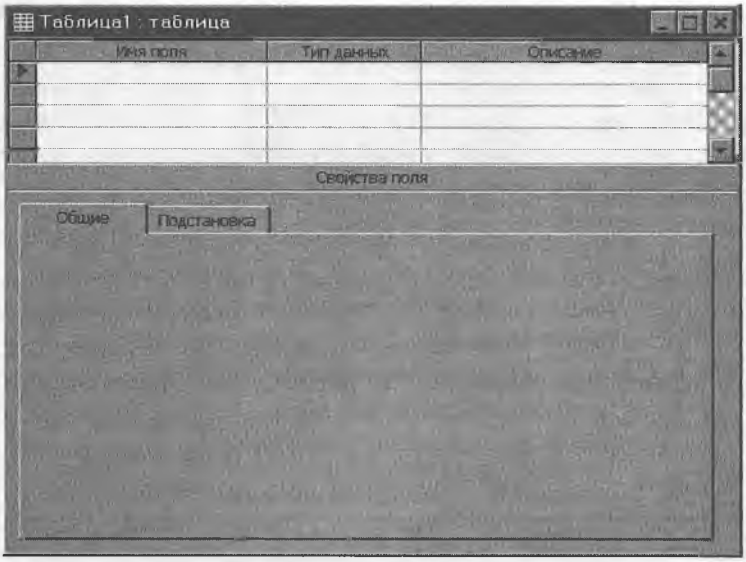


Рис. 2.2. Окно таблицы в режиме конструктора

Наиболее важные свойства полей:

Размер поля — определяет максимальный размер данных, сохраняемых в поле. Рекомендуется задавать минимально допустимый размер поля, так как сохранение таких полей требует меньше памяти и обработка выполняется быстрее;

Формат поля — является форматом отображения заданного типа данных и задает правила представления этих данных при выводе их на экран или печать. Конкретный формат выбирается в раскрывающемся списке значений свойства *Формат поля*. Для числового и денежного типов данных задается число знаков после запятой (от 0 до 15);

Подпись поля — задает текст, который выводится в таблицах, формах, отчетах;

Условие на значение — позволяет осуществлять контроль ввода данных, задает ограничения на вводимые значения, при нарушении условий запрещает ввод и выводит текст, заданный свойством *Сообщение об ошибке*;

Сообщение об ошибке — задает текст сообщения, выводимый на экран при нарушении ограничений, заданных свойством *Условие на значение*.

Тип элемента управления — это свойство, которое задается на закладке *Подстановка* в окне конструктора таблиц и определяет, будет ли отображаться поле в таблице и в какой форме (в виде поля, списка или поля со списком). Таким образом определяется тип элемента управления, используемого по умолчанию для ото-

бражения поля. Если для отображения поля выбран тип элемента управления *Список* или *Поле со списком*, то на закладке *Подстановка* появляются дополнительные свойства, которые определяют источник данных для строк списка и ряд других его характеристик.

Если при определении типа поля был выбран мастер подстановок, то им и будут заполнены значения свойств на закладке *Подстановка*.

Определение первичного ключа

Уникальный (первичный) ключ таблицы может быть простым или составным, включающим в себя несколько полей. Для определения ключа выделяются поля, составляющие его, и на панели инструментов нажимается кнопка [Ключевое поле] или выполняется команда *Правка|Ключевое поле*.

Если для установки ключевого поля используется мастер таблиц, то необходимо задать тип используемых в ключевом поле данных. Некоторые из них приведены в табл. 2.3.

Для ключевого поля автоматически строится индекс. Убедиться в этом можно, просмотрев информацию об индексах таблицы.

Окно *Индексы* вызывается щелчком мыши на кнопке [Индексы] просмотра и редактирования индексов или выполнением команды *Вид|Индексы*. В этом окне индекс первичного ключа имеет имя *Primary Key*. В столбце *Имя поля* этого окна перечисляются имена полей, составляющие индекс.

Индекс ключевого поля всегда уникален и не допускает пустых полей в записях.

Таблица 2.3

Типы данных ключевого поля

Тип данных	Описание	Тип поля
Порядковый номер, автоматически присваиваемый каждой новой записи	При вводе каждой новой записи Access автоматически присваивает ей порядковый номер. Вводить или редактировать данные в поле *Счетчик* (AutoNum) нельзя	Счетчик (AutoNum)
Номер, вводимый пользователем при добавлении каждой новой записи	При вводе записи в одно из ее полей заносится уникальное числовое значение, например номер документа. В это поле нельзя вводить буквы	Числовой (Number)
Сочетание букв и цифр, вводимое пользователем при добавлении каждой новой записи	При вводе записи в одно из ее полей заносится уникальное сочетание цифр и букв. Этот тип данных выбирают, если поле содержит и буквы и цифры	Текстовый (Text)

Сохранение таблицы

После определения структуры таблицы ее надо сохранить. Для этого используется команда *Файл|Сохранить* или кнопка панели инструментов [Сохранить]. В окне *Сохранение* вводится имя таблицы.

После сохранения таблицы ставший доступным режим таблицы позволяет перейти ко второму этапу ее создания — созданию записей. Переход осуществляется нажатием кнопки [Представление таблицы] на панели инструментов таблиц. В режиме таблицы можно вводить в создаваемую таблицу новые записи, заполняя ее поля.

При заполнении таблиц со связями и вводе записей в подчиненную таблицу необходимо отслеживать наличие записей с вводимыми значениями ключевых полей в главной таблице. После ввода значения в ячейку поля и попытки перейти к другой ячейке Access проверяет, являются ли введенные данные допустимыми для этого поля. Если данные не являются допустимыми и их преобразование невозможно, то появляется предупреждающее сообщение. Для того чтобы выйти из ячейки, следует ввести правильное значение.

2.6. Схема данных в Access

Структура реляционной базы данных в Access задается схемой данных, которая имеет иерархическую структуру и называется канонической реляционной моделью предметной области.

Схема данных графически отображается в отдельном окне, в котором таблицы представлены списками полей, а связи — линиями между полями разных таблиц.

При построении схемы данных Access автоматически определяет по выбранному полю тип связи между таблицами. Если поле, по которому нужно установить связь, является уникальным ключом как в главной таблице, так и в подчиненной, Access устанавливает связь типа один к одному. Если поле связи является уникальным ключом в главной таблице, а в подчиненной таблице является не ключевым или входит в составной ключ, Access устанавливает связь типа один ко многим от главной таблицы к подчиненной.

Кроме указанных типов связей в Access существуют связи-объединения, обеспечивающие объединение записей таблиц не по ключевому полю, а в следующих случаях:

- связываемые записи в обеих таблицах совпадают (связи устанавливаются по умолчанию);
- для всех записей первой таблицы, для которых отсутствуют связи со второй таблицей, устанавливаются связи с пустой записью второй таблицы;

• для всех записей второй таблицы, для которых отсутствуют связи с первой таблицей, устанавливаются связи с пустой записью первой таблицы.

Обеспечение целостности данных

При создании схемы данных пользователь включает в нее таблицы и устанавливает связи между ними. Причем для связей типов один к одному и один ко многим можно задать параметр, обеспечивающий целостность данных, а также автоматическое каскадное обновление или удаление связанных записей.

Обеспечение целостности данных означает выполнение для взаимосвязанных таблиц следующих условий корректировки базы данных:

• в подчиненную таблицу не может быть добавлена запись, для которой не существует в главной таблице ключа связи;

• в главной таблице нельзя удалить запись, если не удалены связанные с ней записи в подчиненной таблице;

• изменение значений ключа связи главной таблицы должно приводить к изменению соответствующих значений в записях подчиненной таблицы,

В случае если пользователь нарушил эти условия в операциях обновления или удаления данных в связанных таблицах, Access выводит соответствующее сообщение и не допускает выполнения операции. Access автоматически отслеживает целостность данных, если между таблицами в схеме данных установлена связь с параметрами обеспечения целостности. При вводе некорректных данных в связанные таблицы выводится соответствующее сообщение. Access не позволяет создавать связи с параметрами обеспечения целостности в схеме данных, если ранее введенные в таблицы данные не отвечают требованиям целостности.

Отметим, что установление между двумя таблицами связи типа один к одному или один ко многим и задание параметров целостности данных возможно только при следующих условиях:

• связываемые поля имеют одинаковый тип данных, причем имена полей могут быть различными;

• обе таблицы сохраняются в одной базе данных Access;

• главная таблица связывается с подчиненной по первичному простому или составному ключу (уникальному индексу) главной таблицы.

Если для выбранной связи обеспечивается поддержание целостности, то можно задать режимы каскадного обновления и удаления связанных записей.

В режиме каскадного обновления связанных записей при изменении значения в поле связи главной таблицы Access автоматически изменит значения в соответствующем поле в подчиненных записях.

В режиме каскадного удаления связанных записей при удалении записи из главной таблицы Access выполняет каскадное удаление подчиненных записей на всех уровнях.

Создание схемы данных и включение в нее таблиц

Создание схемы данных начинается в окне базы данных с выполнения команды *Сервис*|*Схема данных* или нажатия кнопки [Схема данных] на панели инструментов.

После нажатия кнопки [Схема данных] открывается окно *Добавление таблицы*, где можно выбрать таблицы и запросы, которые нужно включить в схему данных (рис. 2.3). Для размещения таблицы в окне *Схема данных* надо выделить ее в окне *Добавление таблицы* и нажать кнопку [Добавить]. Для выделения нескольких таблиц надо, держа нажатой клавишу [Ctrl], щелкнуть мышью на всех таблицах, переносимых в схему. После включения в схему данных всех нужных таблиц надо нажать кнопку [Закрыть]. В результате в окне *Схема данных* будут представлены все включенные в эту схему таблицы со списком своих полей.

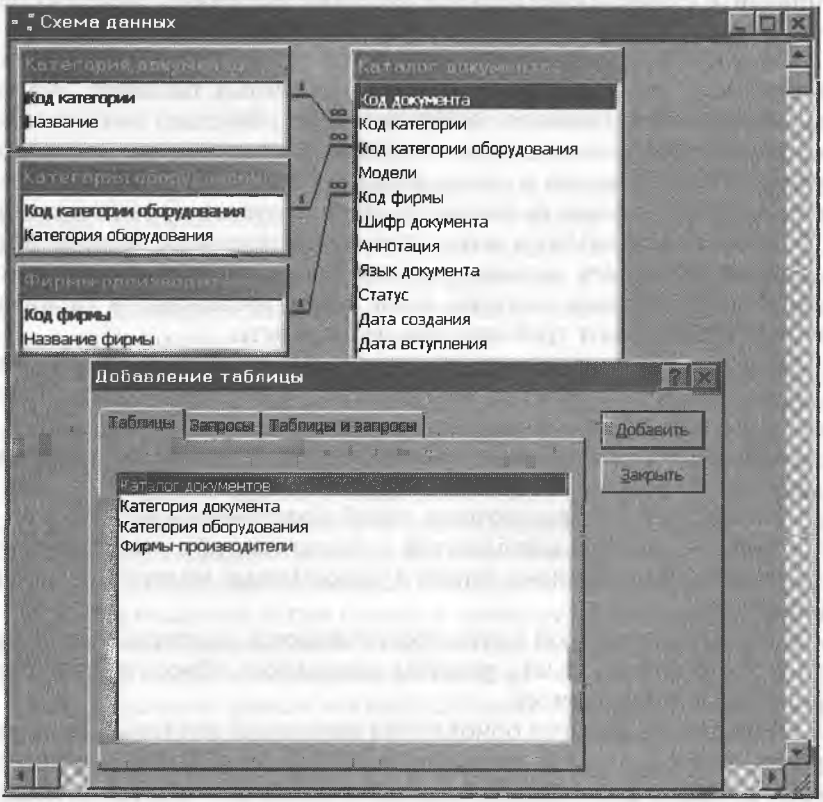


Рис. 2.3. Схема данных БД

Для установления связи между парой таблиц в схеме данных надо выделить в главной таблице уникальное ключевое поле, по которому устанавливается связь, а затем при нажатой кнопке мыши перетащить курсор в соответствующее поле подчиненной таблицы.

При установлении связи по соответствующему ключу необходимо выделить все поля, входящие в ключ главной таблицы, и перетащить их на одно из полей связи в подчиненной таблице. Для выделения всех полей, входящих в составной уникальный ключ, необходимо отмечать их при нажатой клавише [Ctrl]. При установлении связи откроется окно *Связи*. При этом в строке *Тип отношения* автоматически установится тип связи один ко многим.

При составном ключе связи в окне *Связи* необходимо для каждого ключевого поля *Таблица|Запрос* главной таблицы выбрать соответствующее поле подчиненной таблицы *Связанная таблица|Запрос*.

В этом же окне можно задать параметр *Обеспечение целостности данных* для выбранной связи. Если же таблицы уже содержат данные, не отвечающие требованиям целостности, то связь типа один ко многим не устанавливается и появляется соответствующее сообщение с соответствующей инструкцией. После задания параметра целостности можно в окне *Связи* отметить каскадное обновление связанных полей или каскадное удаление связанных записей.

После установления связей между таблицами получают схему данных в окне *Схема данных*. Перемещение и изменение размеров таблиц осуществляются принятыми в операционной системе Windows способами. Проверка работоспособности схемы данных осуществляется при конструировании форм, запросов, отчетов и при их использовании, а также при непосредственной работе с таблицами.

2.7. Модификация структуры базы данных

Модификация структуры базы данных определяется изменением структуры отдельных таблиц и схемы данных. При этом все изменения структуры таблиц производятся в режиме конструктора таблиц.

Изменения полей, которые не являются ключами или полями связи. Состав и последовательность, а также тип данных, свойства или имена этих полей можно изменять независимо от наличия связей таблицы с другими таблицами базы данных. Однако, если преобразования недопустимы, попытка изменения типа данных может привести к потере данных.

Изменение или удаление ключевого поля. При попытке изменить свойства или удалить ключевое поле загруженной и несвязанной

таблицы система предупреждает о возможности потери данных при удалении ключа. Если, например, удаляется поле в составном ключе, с других полей этого ключа снимается признак ключа.

Для отказа от определения первичного ключа в таблице достаточно нажать кнопку [Ключевое поле] или удалить индекс ключа в окне *Индексы* после нажатия соответствующей кнопки на панели инструментов. Если требуется изменить ключ таблицы, имеющей связи с другими таблицами, необходимо предварительно разорвать эти связи. Изменения, сделанные в структуре таблиц, автоматически не переносятся системой в использующие их формы, запросы и отчеты.

Изменение схемы данных. При модификации схемы данных осуществляется изменение состава ее таблиц, т. е. удаление, добавление таблиц и изменение их связей.

Необходимость изменения связей возникает, в частности, при изменении ключей в таблицах. Изменение ключа по составу, типу и размеру его полей не может производиться до тех пор, пока не удалены связи таблицы в схеме данных.

При изменении типа данных для неключевых полей, задействованных в связях таблицы, также предварительно необходимо удалить эти связи в схеме данных.

Для внесения изменений в схему данных необходимо закрыть все таблицы и выполнить команду *Сервис|Схема данных* или нажать кнопку [Схема данных] на панели инструментов.

Добавление таблиц выполняется нажатием кнопки [Добавить таблицу] после выделения в окне *Добавление таблицы* нужной таблицы.

Удаление таблицы из схемы данных осуществляется после перехода в окно *Схема данных*, в котором сначала удаляются ее связи, затем она выделяется, после чего подается команда *Правка|Удалить* или нажимается клавиша [Del].

Для удаления связи ее отмечают щелчком мыши, затем нажимают правую кнопку мыши, вызывающую контекстное меню, и подают команду *Удалить связь*. Помеченную связь можно также удалить с помощью команды *Правка|Удалить* или клавиши [Del].

Изменение параметров связи выполняется посредством команды *Связи|Изменить связь* или соответствующей команды контекстного меню.

2.8. Обработка данных в базе

2.8.1. Запросы в СУБД Access

Запросы создаются пользователем для выборки необходимых ему данных из одной или нескольких связанных таблиц и пред-

ставления выбранных данных также в виде таблицы. Запрос может формироваться двумя способами:

с помощью запросов по образцу — QBE (Query By Example);

с помощью инструкций языка структурированных запросов SQL (Structured Query Language), т.е. специализированного языка, предназначенного для организации запросов, а также для обновления и управления реляционными базами данных.

Практически все типы запросов в Access можно создать визуально. Исключение составляют сквозные запросы (SQL-pass-through), т.е. запросы из других приложений, запросы на изменение структуры данных и запросы объединения.

Визуально можно построить запросы добавления, удаления, обновления и создания таблиц.

Отметим также, что одной из наиболее сильных сторон Access являются фильтры, которые строятся с помощью запросов или посредством установки критериев. Для облегчения этой задачи используют параметрические запросы.

В Access имеется несколько видов запросов:

- запрос на выборку, т.е. выбирающий данные из взаимосвязанных таблиц и других запросов. В результате получают таблицу, существующую до закрытия запроса. Таблицу с результатами запроса можно использовать для работы с данными таблиц, на которых построен запрос;
- запрос на создание таблицы, основанный на запросе на выборку, но в отличие от последнего результат этого запроса сохраняется в новой таблице;
- запросы на обновление, добавление, удаление, являющиеся запросами действия, в результате выполнения которых изменяются данные в таблицах.

2.8.2. Основы конструирования запросов

Основные принципы конструирования запроса заложены в технике конструирования запроса на выборку, являющегося основой всех видов запросов.

Запрос на выборку позволяет достаточно просто выбрать данные из одной или нескольких взаимосвязанных таблиц. Результаты запроса отображаются в виде таблицы.

При конструировании запроса достаточно выделить и перетащить необходимые поля из таблиц, представленных в схеме данных запроса, в бланк запроса и ввести условия отбора записей.

Результаты выполнения запроса выводятся в режиме таблицы. Несмотря на то, что поля результирующей таблицы принадлежат, как правило, нескольким таблицам базы данных, с ними можно работать так, как если бы они принадлежали одной таблице. Можно также менять данные в таблице результатов запроса на

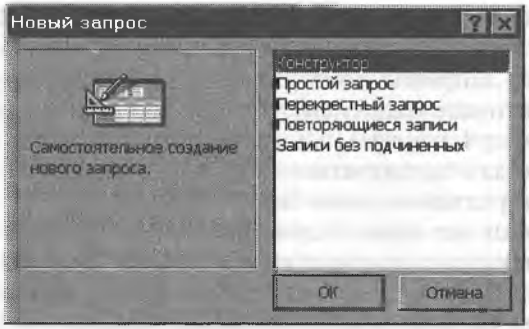


Рис. 2.4. Выбор вариантов построения запроса

выборку, при этом сделанные изменения будут внесены в базовые таблицы.

Для создания запроса в окне базы данных надо выбрать закладку *Запрос* и нажать кнопку [Создать]. В открывшемся окне *Новый запрос* из предложенных типов запросов (*Конструктор*, *Простой запрос*, *Перекрестный запрос*, *Повторяющиеся записи*, *Записи без подчиненных*) следует выбрать *Конструктор* (рис. 2.4).

В окне *Добавление таблицы* (рис. 2.5) выбрать используемые в запросе таблицы и нажать кнопку [Добавить]. Затем, нажав кнопку [Закрыть], выйти из окна *Добавление таблицы*.

В результате появится окно конструктора запросов *Имя запроса*:*запрос на выборку*.

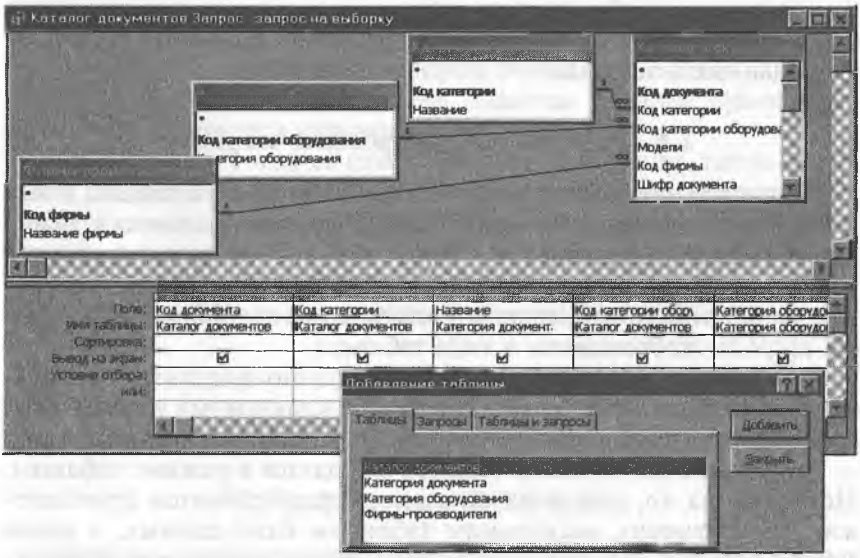


Рис. 2.5. Создание запроса с помощью конструктора

Окно конструктора запросов разделено на две панели. Верхняя панель содержит схему данных запроса, включающую в себя выбранные для данного запроса таблицы, которые представлены списками полей. Нижняя панель является бланком запроса по образцу (QBE), который нужно заполнить.

Схема данных запроса. В окне запроса отображаются выбранные таблицы и связи между ними, имеющиеся в логической схеме (схеме данных) БД. Кроме того, Access автоматически устанавливает между таблицами дополнительные связи, которых не было в логической модели, в том случае, если таблицы имеют поля с одинаковыми именами и типами данных (атрибутами). Логические связи между таблицами, которые Access не может установить автоматически, может создать пользователь, перетащив задействованные в связи поля из одного списка полей в другой.

При использовании в запросе других запросов или таблиц, не представленных в логической схеме базы данных, с ними также могут быть установлены связи-объединения, т. е. связи без ключевого слова.

Бланк запроса по образцу. Бланк запроса по образцу представлен в нижней панели окна запроса в виде таблицы, которая имеет для заполнения следующие строки: *Поле:*, *Имя таблицы:*, *Сортировка:*, *Вывод на экран:*, *Условие отбора:*, *или:*. До формирования запроса эта таблица не заполнена.

Каждый столбец бланка является одним полем запроса. Эти поля могут использоваться для включения их в таблицу результата выполнения запроса, задания сортировки по ним, а также задания условий отбора записей.

При заполнении бланка запроса необходимо:

в строку *Поле* включить имена полей, используемых в запросе;

в строке *Вывод на экран* отметить поля, которые должны быть включены в результирующую таблицу;

в строке *Условия отбора* задать условия отбора записей;

в строке *Сортировка* выбрать порядок сортировки записей результата.

Для включения нужных полей из таблиц БД в соответствующие столбцы запроса можно воспользоваться следующими приемами:

в первой строке бланка запроса *Поле* щелчком мыши вызвать появление кнопки списка полей и, воспользовавшись ею, выбрать из списка нужное поле. Список содержит все поля таблиц, представленных в бланке запроса;

перетащить нужное поле из списка полей таблицы в схеме данных запроса в первую строку бланка запроса.

В списке полей каждой таблицы на первом месте стоит символ звездочка (*), имеющий значение «Все поля таблицы», который выбирается, если в запрос включаются все поля.

Модификация запроса. Для добавления поля в бланк запроса надо перетащить его с помощью мыши из таблицы в схеме данных в нужное место бланка. При этом все столбцы полей справа от него передвинутся на один столбец вправо.

Для удаления поля в бланке запроса надо выделить удаляемый столбец, щелкнув кнопкой мыши а области маркировки столбца, и нажать клавишу [Del] или выполнить пункт меню *Правка|Удалить столбец.*

Для перемещения поля в бланке надо выделить соответствующий столбец и перетащить его в новую позицию с помощью мыши. При этом столбец, на место которого перемещен новый столбец, и все столбцы справа от него будут сдвинуты вправо.

2.8.3. Условия отбора записей, сортировка и фильтрация данных

Условия отбора записей могут задаваться для одного или нескольких полей в соответствующей строке бланка запроса.

Условием отбора является выражение, которое состоит из операторов сравнения и сравниваемых операторов. В качестве операторов сравнения и логических операторов могут использоваться следующие: =, <, >, < >, Between, In, Like, And, Or, Not, которые определяют операцию над одним или несколькими операндами.

Если условие отбора не содержит оператора, то по умолчанию используется оператор =.

В качестве операндов могут использоваться литералы, константы и идентификаторы (ссылки).

Литералом является значение, воспринимаемое буквально, а не как значение переменной или результат вычисления (например, число, строка, дата).

Константами являются не изменяющиеся значения (например, True, Falls, Да, Нет, Null).

Идентификатор представляет собой ссылку на значение поля, элемент управления или свойство. Идентификаторами могут быть, например, имена полей, таблиц, запросов, форм, отчетов, которые должны заключаться в квадратные скобки.

Если необходимо указать ссылку на поле в конкретных таблице, форме, отчете, то перед именем поля ставится имя таблицы, также заключенное в квадратные скобки и отделенное от имени поля восклицательным знаком. Например:

[Имя таблицы]! [Имя поля]

Условия отбора, заданные в одной строке, связываются с помощью логической операции И, а заданные в разных строках — с помощью логической операции ИЛИ. Эти операции могут быть

заданы явно в условии отбора с помощью операторов AND и OR соответственно.

Сформировать условие отбора можно с помощью построителя выражения. Перейти в окно *Построитель выражений* можно, нажав кнопку [Построитель] на панели инструментов или выбрав команду *Построить* в контекстно-зависимом меню. При этом курсор мыши должен быть установлен в ячейке ввода условия отбора.

После ввода выражения в бланк и нажатия клавиши [Enter] Access выполняет синтаксический анализ выражения и отображает его в соответствии с результатами этого анализа.

Для выполнения запроса необходимо на панели инструментов конструктора запросов нажать кнопку [Запуск (!)] или [Представление запроса].

Примеры выражений, используемых в качестве условий отбора, приведены в табл. 2.4.

Таблица 2.4

Примеры выражений, используемых в качестве условий отбора

Поле	Выражение	Описание
ПунктНазначения	“Москва”	Отображает заказы на доставку товаров в Москву
ПунктНазначения	“Москва” Or “Санкт-Петербургу”	Оператор Or используется для отображения заказов на доставку товаров в Москву или Санкт-Петербург
ДатаОтгрузки	Between #05.01.03# And #10.01.03#	Оператор Between ... And используется для отображения заказов на отгрузку товаров не ранее 5 января 2003 г. и не позднее 10 января 2003 г.
ДатаОтгрузки	#2/2/03#	Отображает заказы на отгрузку товаров 2 февраля 2003 г.
СтранаДоставки	In(“Россия”, “США”)	Оператор In используется для отображения заказов на доставку товаров в Россию или США
СтранаДоставки	Not “США”	Оператор Not используется для отображения заказов на доставку товаров во все страны, за исключением США
ИмяКлиента	Like “С*”	Отображает заказы на доставку товаров клиентам, имена которых начинаются с буквы С

Окончание табл. 2.4

Поле	Выражение	Описание
Название	>=«Н»	Отображает заказы на доставку товаров в фирмы, названия которых начинаются с букв, находящихся в диапазоне от Н до Я
ДатаЗаказа	< Date( )- 30	Функция Date используется для отображения заказов, сделанных более чем за 30 дней
ДатаЗаказа	Year([ДатаЗаказа])=2003	Функция Year используется для отображения заказов, сделанных в 2003 г.
ДатаЗаказа	Year([ДатаЗаказа])= Year(Now()) And Month([ДатаЗаказа])= Month(Now())	Функции Year и Month, а также оператор And используются для отображения заказов текущего года и месяца
ОбластьДоставки	Is Null	Отображает заказы для клиентов, у которых поле *ОбластьДоставки* является пустым
ОбластьДоставки	Is Not Null	Отображает заказы для клиентов, у которых поле *Область Доставки* содержит какое-либо значение
Факс	“ ”	Отображает заказы для клиентов, у которых нет факсимильного аппарата, т.е. для тех клиентов, у которых поле *Факс* содержит пустую строку, а не значение Null

Сортировка данных. Для удобства просмотра можно сортировать записи в таблице в определенной последовательности. Кнопки сортировки на панели инструментов (или команды меню *Записи*|*Сортировка*, *Сортировка по возрастанию* (*Сортировка по убыванию*) позволяют сортировать столбцы по возрастанию или убыванию. Прежде чем щелкнуть по кнопке сортировки, следует выбрать поля, используемые для сортировки. Современные СУБД (такие, как Access) никогда не сортируют таблицы физически, как это делалось раньше. Средства сортировки данных (а также фильтрации, поиска и замены) реализованы в Access как автоматически создаваемые запросы. Записи таблицы всегда располагаются в

файле базы данных в том порядке, в котором они были добавлены в таблицу.

Отбор данных с помощью фильтра. Фильтр — это набор условий, применяемых для отбора подмножества записей. В Access существуют фильтры четырех типов: фильтр по выделенному фрагменту, обычный фильтр, расширенный фильтр и фильтр по вводу.

Фильтрование данных в Access производится с помощью кнопок [Фильтр по выделенному] или [Изменить фильтр] либо команды меню *Записи|Фильтр*, *Изменить фильтр*. После нажатия второй кнопки от таблицы остается одна запись. Каждое поле становится полем со списком (когда в нем находится курсор), в котором можно выбрать из списка все значения для данного поля. После щелчка мышью по кнопке [Изменить фильтр] выбираются записи, соответствующие измененному фильтру. Еще более сложные условия фильтрации можно задать командой меню *Записи|Фильтр*, *Расширенный фильтр*.

2.8.4. Изменение данных в БД средствами запроса

Запрос на обновление может быть использован для замены данных в таблицах БД. Отбор заменяемых записей (полей) производится с помощью запроса на выборку, который затем в окне конструктора запросов с помощью кнопки [Обновление] на панели инструментов или команды меню *Запрос|Обновление* превращается в запрос на обновление.

Для обновления поля надо в строку *Обновление* ввести значение или выражение, определяющее новое значение поля. Такое выражение можно создать при помощи построителя выражений. После выполнения команды *Запрос|Запуск* или нажатия соответствующей кнопки на панели инструментов открывается диалоговое окно с сообщением о числе обновляемых записей и вопросом о продолжении операции обновления.

Результат выполнения запроса можно проверить переключением в режим таблицы с помощью команды *Вид|Режим таблица* или нажатием кнопки [Представление запроса] на панели инструментов. Если переключиться в режим таблицы до выполнения запроса, то можно просмотреть старое содержимое обновляемых полей.

Запрос на добавление используется для добавления записей из таблицы запроса в другую таблицу. Добавляемые записи выбираются из одной или нескольких взаимосвязанных таблиц с помощью запроса на выборку, который затем в окне конструктора запросов превращается в запрос на добавление кнопкой [Добавление] на панели инструментов или командой меню *Запрос|Добавление*. В открывшемся при этом окне *Добавление* в поле *Имя таблицы* вводится имя таблицы, в которую надо добавить записи.

После выполнения последней команды в бланке запроса появляется строка *Добавление*. Для формирования добавляемых записей надо включить в бланк запроса поля, соответствующие определенным полям таблицы, в которые будет производиться добавление, и там, где необходимо, записать условия отбора полей в ячейки строки *Условие отбора*. Если в таблице, в которую добавляются записи, есть ключ, то ключевые поля обязательно должны быть включены в бланк запроса.

Для указания в строке *Добавление* имен полей таблицы-получателя надо в каждой ячейке открыть список и выбрать нужное имя. Если выбранные поля имеют одни и те же имена в обеих таблицах, эти имена в строку *Добавление* вносятся автоматически.

Предварительный просмотр добавляемых записей производится кнопкой [Представление запроса] на панели инструментов. Возврат в режим конструктора запросов производится этой же кнопкой.

Для добавления записей нажимается кнопка [Запуск] на панели инструментов. В открывшемся диалоговом окне сообщается о числе обновляемых записей и задается вопрос о продолжении операции добавления.

Если таблица, в которую добавляются записи, содержит ключевое поле, то добавляемые записи должны содержать такое же поле.

Запрос на удаление позволяет удалить записи из одной таблицы или из нескольких взаимосвязанных таблиц. Удаляемые записи выбираются с помощью запроса на выборку, который в окне конструктора превращается в запрос на удаление при помощи кнопки [Удаление] на панели инструментов или команды меню *Запрос|Удаление*.

После выполнения этой команды в бланке запроса появляется строка *Удаление*. Символ звездочка (*) из списка полей таблицы, записи которой требуется удалить, перемещается с помощью мыши в бланк запроса. Для полей (для которых это необходимо) заполняется строка *Условие отбора*.

Для предварительного просмотра удаляемых записей можно нажать кнопку [Представление запроса] на панели инструментов, которая используется и для возврата в режим конструктора запроса. Для удаления записей нажимается кнопка [Запуск] на панели инструментов.

Результаты запроса зависят от установленных в схеме БД отношений между таблицами и параметров целостности. Если для связи установлен параметр целостности *Каскадное удаление связанных записей*, то в результате выполнения этого запроса будут удалены все связанные записи. Если параметр целостности *Каскадное удаление связанных записей* не установлен, то записи удаляются только в указанных в бланке запроса таблицах вне зависимости от их логических связей.

2.8.5. Элементы языка SQL и запросы в форме SQL

SQL (Structured Query Language) — это язык запросов, который используется при работе с реляционными базами данных в современных СУБД (ORACLE, dBASE IY, dBASE Y, Paradoxe, Access и др.).

Язык SQL стал стандартным языком запросов при работе с реляционными базами данных для архитектуры как файл-сервер, так и клиент-сервер, а также в условиях применения системы управления распределенными БД.

Язык SQL использует ограниченный набор команд, но в то же время — это реляционно полный язык, предназначенный для работы с базами данных, создания запросов выборки данных, выполнения вычислений, обеспечения их целостности. Синтаксис версий языка SQL может в определенной степени различаться для отдельных СУБД. Рассмотрим наиболее общие операторы языка SQL.

Операторы языка SQL для работы с реляционной базой данных

Создание реляционных таблиц. Создание реляционной базы данных означает спецификацию состава полей: указание имени, типа и длины каждого поля (если это необходимо). При этом каждая таблица должна иметь уникальное имя.

Синтаксис оператора создания новой таблицы:

```
CREATE TABLE таблица (поле1 тип [(размер)] [индекс1]
[, поле2 тип [(размер)] [индекс2] [, …]] [, составной
индекс[, …]])
```

Здесь таблица — имя создаваемой таблицы; поле1, поле2 — имена полей таблицы; тип — тип поля; размер — размер текстового поля; индекс1, индекс2 — директивы создания простых индексов; составной индекс — директива создания составного индекса.

Каждый индекс имеет уникальное в пределах данной таблицы имя. Для создания простого индекса используется следующая фраза (размещаемая за именем поля):

```
CONSTRAINT имя индекса {PRIMARY KEY|UNIQUE|
REFERENCES внешняя таблица [(внешнее поле)]}
```

Директива создания составного индекса (размещаемая в любом месте после определения его элементов) имеет следующий вид:

```
CONSTRAINT имя {PRIMARY KEY (ключевое1[, ключевое2
[, …]]) |UNIQUE (уникальное1[, …]]) | FOREIGN KEY (ссыл-
```

```
ка1[, ссылка2[, …]])REFERENCES внешняя таблица [(внеш-
нее поле1[, внешнее поле2[, …]])]}
```

Значения служебных слов:

UNIQUE — уникальный индекс (в таблице не может быть двух записей, имеющих одно и то же значение полей, входящих в него);

PRIMARY KEY — первичный ключ таблицы, который может состоять из нескольких полей (упорядочивает записи таблицы);

FOREIGN KEY — внешний ключ для связи с другими таблицами (может состоять из нескольких полей);

REFERENCES — ссылка на внешнюю таблицу.

Пример 2.1. Создание таблицы:

```
CREATE TABLE Студент
([Имя] TEXT,
[Фамилия] TEXT,
[Дата рождения] DATETIME,
CONSTRAINT Адр UNIQUE ([Имя]), [Фамилия], [Дата рож-
дения]))
```

В результате выполнения данного запроса будет создана таблица СТУДЕНТ, имеющая в своем составе:

- два текстовых поля — *Имя*, *Фамилия*;
- одно поле типа дата/время — *Дата рождения*.

Будет также создан составной индекс с именем *Адр* по значениям указанных полей, который будет иметь уникальное значение, так как в таблице не может быть двух записей с одинаковыми значениями полей, образующих его.

Изменение структуры таблиц. При необходимости можно изменить структуру таблицы:

- удалить существующие поля;
- добавить новые поля;
- создать или удалить индексы.

При этом все указанные действия затрагивают только одно поле или один индекс:

```
ALTER TABLE таблица
ADD{[COLUMN]поле тип[(размер)][CONSTRAINT индекс]
CONSTRAINT составной индекс}|
DROP {[COLUMN] поле i CONSTRAINT имя индекса}}
```

Опция ADD обеспечивает добавление поля таблицы, а опция DROP — удаление. Добавление опции CONSTRAINT означает подобные действия для индексов таблицы.

Пример 2.2. Изменение структуры таблицы:

```
ALTER TABLE Студент ADD COLUMN [Группа] TEXT(5)
```

Для создания нового индекса к существующей таблице можно также использовать следующую команду:

```
CREATE [UNIQUE] INDEX индекс
ON таблица (поле[, …])
[WITH {PRIMARY|DISALLOW NULL|IGNORE NULL}]
```

Фраза WITH обеспечивает наложение условий на значения полей, включенных в индекс:

DISALLOW NULL — запретить пустые значения в индексированных полях новых записей;

IGNORE NULL — включать в индекс записи, имеющие пустые значения в индексированных полях.

Пример 2.3. Создание индекса таблицы:

```
CREATE INDEX Гр ON Студент([группа]) WITH DISALLOW
NULL
```

Удаление таблицы. Для удаления таблицы (одновременно и структуры, и данных) используется следующая команда:

```
DROP TABLE имя таблицы
```

Для удаления только индекса таблицы (сами данные при этом не разрушаются) необходимо выполнить следующую команду:

```
DROP INDEX имя индекса ON имя таблицы
```

Пример 2.4. Удаление только индекса *Адр* таблицы:

```
DROP INDEX Адр ON Студент
```

Удаление всей таблицы:

```
DROP TABLE Студент
```

Ввод данных в таблицу. Формирование новой записи в таблице выполняется следующей командой:

```
INSERT INTO таблица [(поле1[, поле2[, …]])]
VALUES (значение1[, значение2[, …])
```

Здесь указываются имя таблицы, в которую добавляют запись, и состав полей, для которых вводятся значения.

Пример 2.5. Ввод данных в таблицу:

```
INSERT INTO Студент ([Фамилия], [Имя], [Дата рождения]) VALUES ("Петров", "Иван", 23/3/80)
```

Возможен также групповой ввод записей (пакетный режим), являющихся результатом выборки (запроса) из других таблиц:

```
INSERT INTO таблица [IN внешняя база данных]
SELECT [источник.]поле1[, поле2[, …]
FROM выражение
WHERE условие
```

В этом случае сначала выполняется оператор подзапроса SELECT, который и формирует выборку для добавления данных.

Фраза SELECT определяет структуру данных источника передаваемых записей — имена таблицы и полей, содержащих исходные данные для загрузки в таблицу.

Оператор FROM позволяет указать имена исходных таблиц, участвующих в формировании выборки, а фраза WHERE задает условия выполнения подзапроса. При этом структура данных выборки должна соответствовать структуре данных таблицы, в которую производится добавление.

Добавление (перезагрузка) записей возможно и во внешнюю базу данных, для которой указывается полностью специфицированное имя (диск, каталог, имя, расширение). При этом структуры таблиц должны совпадать.

Пример 2.6. Ввод данных в таблицу:

```
INSERT INTO Студент SELECT [Студент-заочник].* FROM [Студент-заочник]
```

Этой фразой все записи таблицы СТУДЕНТ-ЗАОЧНИК будут добавлены в таблицу СТУДЕНТ.

Пример 2.7. Ввод данных в таблицу:

```
INSERT INTO Студент SELECT [Студент-заочник].* FROM [Студент-заочник] WHERE [Дата рождения] > = # 01/01/80 #
```

В этом случае записи таблицы СТУДЕНТ-ЗАОЧНИК добавляются в таблицу СТУДЕНТ, если дата рождения студента больше или равна указанной.

Операции соединения таблиц. Операцию INNER JOIN можно использовать в любом предложении FROM. Она создает симмет-

ричное объединение — наиболее частую разновидность внутреннего объединения.

Записи из двух таблиц объединяются, если связующие их поля содержат одинаковые значения:

```
FROM таблица1 INNER JOIN таблица2 ON таблица1.поле1=таблица2.поле2
```

Данный оператор описывает симметричное соединение двух таблиц по ключам связи (поле1; поле2). Новая запись формируется в том случае, если в таблицах содержатся одинаковые значения ключей связи.

Возможны следующие варианты операции соединения таблиц:

LEFT JOIN (левостороннее) соединение, когда выбираются все записи левой таблицы и только те записи правой таблицы, которые содержат соответствующие ключи связи;

RIGHT JOIN (правостороннее) соединение, когда выбираются все записи правой таблицы и только те записи левой таблицы, которые содержат соответствующие ключи связи.

Пример 2.8. Соединение таблиц:

```
1. SELECT Студент.*, Оценка.* FROM Студенты INNER JOIN Оценка ON Студент.[№ зач.книжки] = Оценка.[№ зач.книжки]
2. SELECT Студент.*, Оценка.* FROM Студенты LEFT JOIN Оценка ON Студент.[№ зач.книжки] = Оценка.[№ зач.книжки]
3. SELECT Студент.*, Оценка.* FROM Студенты RIGHT JOIN Оценка ON Студент.[№ зач.книжки] = Оценка.[№ зач.книжки]
```

В первом случае создается симметричное соединение двух таблиц по полю [№ зач.книжки]. При этом записи не выводятся, если значение их ключей связи (указанное поле) не представлено в двух таблицах.

Во втором случае выводятся все записи таблицы СТУДЕНТ и соответствующие им записи таблицы ОЦЕНКА.

В третьем случае, наоборот, выводятся все записи таблицы ОЦЕНКА и соответствующие им записи таблицы СТУДЕНТ.

Операции JOIN могут быть вложенными для последовательного соединения нескольких таблиц.

Пример 2.9. Соединение таблиц:

```
SELECT Студент.*, Оценка.*, Дисциплина.[Наименование дисциплины] FROM (Студент INNER JOIN (Оценка INNER JOIN (Дисциплина ON Оценка.[Код дисциплины] = Дисцип-
```

```
лина.[Код дисциплины]) ON Студент.[№ зач.книжки] =
Оценка.[№ зач.книжки])
```

Здесь сначала происходит соединение таблиц ОЦЕНКА и ДИСЦИПЛИНА по ключу связи [Код дисциплины]. Соединение симметричное, т.е. если коды дисциплины не совпадают, записи этих таблиц не соединяются. Затем происходит соединение таблиц СТУДЕНТ и ОЦЕНКА по ключу связи [№ зач.книжки].

Таким образом, при условии совпадения ключей связи на выходе запроса получается результат соединения трех таблиц.

Удаление записей в таблице. В исходной таблице можно удалить отдельные записи или все записи, сохранив при этом ее структуру и индексы. При удалении записей в индексированной таблице автоматически корректируются ее индексы:

```
DELETE [таблица.*] FROM выражение WHERE условия от-
бора
```

Полная чистка таблицы от записей и очистка индексов выполняются следующей операцией:

```
DELETE * FROM таблица
```

Пример 2.10. Удаление всех записей в таблице:

```
DELETE * FROM Студент
```

Удаление только тех записей, в которых поле [Дата рождения] больше указанной даты:

```
DELETE * FROM Студент WHERE [Дата рождения]> #1.1.81 #
```

Удаление в таблице записей, связанных с другой таблицей (условия удаления записей могут относиться к полям связанных таблиц):

```
DELETE таблица.* FROM таблица INNERJOIN другая таб-
лица ON таблица.[поле N] = [другая таблица].[поле M]
WHERE условие
```

Пример 2.11. Удаление записей в таблице СТУДЕНТ, для которых имеются связанные записи в таблице СТУДЕНТ-ЗАОЧНИК:

```
DELETE Студент.* FROM Студент INNER JOIN [Студент-
заочник] ON Студент.[Группа] = [Студент-заочник]. [Группа]
```

Обновление (замена) значений полей записи. Изменение значений нескольких полей одной записи или группы записей табли-

цы, удовлетворяющих условиям отбора производится следующей фразой:

```
UPDATE таблица SET новое значение WHERE условия от-
бора Новое значение указывается как имя поля=новое
значение
```

Пример 2.12. Отбор студентов, чьи фамилии начинаются на букву В и дата рождения не превышает указанной, для перевода в группу 1212:

```
UPDATE Студент SET [Группа] = "1212"
WHERE [Фамилия] LIKE 'В*' AND [Дата рождения] < =
#01/01/81#
```

Изменение в таблице СТУДЕНТ номеров групп обучения путем добавления к ним буквы «а», если эти группы встречаются в таблице СТУДЕНТ-ЗАОЧНИК:

```
UPDATE Студент INNER JOIN [Студент-заочник] ON Сту-
дент.[Группа] = [Студент-заочник].[Группа] SET [Груп-
па] = [Группа]&"а"
```

Организация запросов в форме SQL

Синтаксис оператора SELECT. Выборка с помощью оператора SELECT — это наиболее частая команда при работе с реляционной базой данных. Данный оператор обладает большими возможностями по заданию структуры выходной информации, указанию источников входной информации, способа упорядочения выходной информации, формированию новых значений и т. п. (табл. 2.5).

При выполнении выборки могут формироваться и новые данные, так называемые вычисляемые поля, являющиеся результатом обработки исходных данных. Возможно упорядочение выводимых данных, формирование групп записей, подсчет групповых итогов, формирование подмножеств данных (записей), являющихся основой для формирования условий по обработке следующего этапа — вложенных запросов.

Универсальный оператор SELECT имеет следующую конструкцию:

```
SELECT [предикат] {*|таблица.* | [таблица.]поле1[,
[таблица.] поле2[, …]]}
    [AS псевдоним1[, псевдоним2[, …]]]
    FROM выражение[, …] [IN внешняя база данных]
    [WHERE …]
    [GROUP BY …]
```

```
[HAVING …]
[ORDER BY …]
[WITH OWNERACCESS OPTION]
```

Синтаксис оператора SELECT реализует сложные алгоритмы запросов.

Слово SELECT определяет структуру выводимой информации (это могут быть поля таблиц и вычисляемые выражения).

Вычисляемое выражение включает в себя:

- поля таблиц;
- константы;
- знаки операций;
- встроенные функции;
- групповые функции SQL;

Пример 2.13. Использование оператора SELECT:

```
1. SELECT [Имя], [Фамилия] FROM Студент
2. SELECT TOP5 [Фамилия] FROM Студент
3. SELECT TOP5 [Фамилия] FROM Студент ORDER BY [Группа]
```

В первом случае выбираются все записи таблицы СТУДЕНТ в составе указанных полей. Причем, если все поля отбираются в том же самом порядке, что и в структуре таблицы, можно использовать символ точки.

Во втором случае отбирается пять первых фамилий студентов.

В третьем случае отбирается пять первых фамилий студентов, а упорядочение записей осуществляется по учебным группам.

Если используются одноименные поля из нескольких таблиц, включенных в предложение FROM, следует указывать перед именем этих полей имя таблицы через точку (.), т. е. [Студент-заочник].[Группа] и [Студент].[Группа] — два одноименных поля из разных таблиц.

Для изменения заголовка столбца с результатами выборки используется служебное слово AS.

Пример 2.14. Использование служебного слова AS:

```
1. SELECT DISTINCT [Дата рождения] AS Юбилей FROM
Студент
2. SELECT [Фамилия]&" "&[Имя] AS ФИО, [Дата рожде-
ния] AS Год FROM Студент
```

В первом случае будут выведены неповторяющиеся даты рождения студентов, которые имеют новое наименование — *Юбилей*.

Во втором случае в результирующей таблице будут присутствовать все записи, но вместо [Дата рождения] будет указан *Год*, а вместо *Фамилия* и *Имя*, соединенных вместе через пробел, — *ФИО*.

Таблица 2.5

Аргументы оператора SELECT

Аргумент	Назначение
Предикат	Используются для ограничения числа возвращаемых записей: ALL — всех записей; DISTINCT — записей, различающихся в указанных для вывода полях; DISTINCTROW — записей, полностью различающихся по всем полям; TOP — заданного числа записей или процента записей в диапазоне, соответствующем фразе ORDER BY
Таблица	Определяет имя таблицы, поля которой формируют выходные данные
Поле1, Поле2	Определяют имена полей, используемых при отборе (порядок следования полей определяет выходную структуру выборки данных)
Псевдоним1, Псевдоним2	Определяют новые заголовки столбцов результата выборки данных
FROM	Определяет выражение, используемое для задания источника формирования выборки (обязательно присутствует в каждом операторе)
Внешняя база данных	Определяет имя внешней базы данных — источника данных для выборки
[WHERE ...]	Необязательный. Определяет условия отбора записей
[GROUP BY...]	Необязательный. Указывает поля (максимум 10) для формирования групп, по которым возможно вычисление групповых итогов; порядок следования полей определяет виды итогов — старший, промежуточный и т. п.
[HAVING ...]	Необязательный. Определяет условия отбора записей для сгруппированных данных (задан способ группирования GROUP BY ...)
[ORDER BY ...]	Определяет поля для выполнения упорядочения выходных записей, порядок следования которых соответствует старшинству ключей сортировки. Упорядочение возможно как по возрастанию (ASC), так и по убыванию (DESC) значения выбранного поля
[WITH OWNERACCESS OPTION]	Служит при работе в сети в составе защищенной рабочей группы для указания пользователям, не обладающим достаточными правами, возможности просмотра результата запроса или выполнения запроса

Наиболее часто слово AS применяется для наименований вычисляемых полей.

Задание условий выборки. Предложение WHERE может содержать выражения, связанные логическими операторами, с помощью которых задаются условия выборки (табл. 2.6).

Таблица 2.6

Логические условия для построения условий выборки

Оператор	Назначение	Оператор	Назначение
And	Логическое И — конъюнкция (логическое умножение)	Not	Отрицание
Eqv	Проверка логической эквивалентности выражений	Or	Логическое ИЛИ — дизъюнкция (включающее Or)
Imp	Логическая импликация выражений	Xor	Логическое ИЛИ (исключающее Or)

Кроме того, могут использоваться операторы для построения условий:

LIKE — выполняет сравнение строковых значений;

BETWEEN...AND — выполняет проверку на диапазон значений;

IN — выполняет проверку выражения на совпадение с любым из элементов списка;

IS — выполняет проверку значения на Null.

Заданные условия обеспечивают горизонтальную выборку данных, т.е. результатом запроса будут только те записи, которые удовлетворяют сформулированным условиям.

Пример 2.15. Задание условий выборки:

```
1. SELECT Студент.* FROM Студент WHERE [Дата рождения]>=#01.01.79#
2. SELECT Студент.* FROM Студент WHERE [Дата рождения]>=#01.01.79# AND [Группа] IN ("1212", "1213")
3. SELECT Студент.* FROM Студент WHERE [Дата рождения] BETWEEN #01.01.79# AND #01.01.81# AND [Группа] IN ("1212", "1213")
4. SELECT Студент.* FROM Студент INNER JOIN [Студент-заочник] On Студент.[Группа]=[Студент-заочник].[Группа] WHERE Студент.[Дата рождения]>=#01.01.79#
```

В первом случае выбираются студенты, родившиеся после 01.01.79.

Во втором случае отбираются все студенты, обучающиеся в группах 1212 или 1213, родившиеся после 01.01.79.

В третьем случае выбираются студенты, дата рождения которых находится в заданном диапазоне, обучающиеся в любой из указанных групп.

В четвертом случае выбираются студенты, которые обучаются в тех же группах, что и студенты-заочники, родившиеся после 01.01.79.

Групповые функции SQL. Групповые функции необходимы для определения статистических данных на основе наборов числовых значений:

- Avg — вычисляет среднее арифметическое набора чисел, содержащихся в указанном поле запроса;
- Count — вычисляет число выделенных записей в запросе;
- Min, Max — соответственно возвращают минимальное и максимальное значения из набора в указанном поле запроса;
- StDev, StDevPs — возвращают среднеквадратическое отклонение соответственно генеральной совокупности и выборки для указанного поля в запросе;
- Sum — возвращает сумму значений в заданном поле запроса;
- Var, VarPs — возвращают дисперсию распределения соответственно генеральной совокупности и выборки для указанного поля в запросе.

Для определения полей группирования указывается ключевое слово HAVING для заданного условия по группе при вычислении групповых значений.

Пример 2.16. Использование групповых функций SQL:

```
1. SELECT Фамилия, Avg(Результат) AS Средний балл
FROM Результаты GROUP BY [№ зач.книжки]
2. SELECT [Код дисциплины], Avg(Результаты) AS Сред-
ний балл FROM Результаты GROUP BY [Код дисциплины]
```

В первом случае создается список фамилий студентов с указанием среднего балла для каждого из них.

Во втором случае составляется список кодов дисциплин с указанием среднего балла по дисциплине.

Пример 2.17. Использование групповых функций SQL:

```
1. SELECT Фамилия, Avg(Результат) AS Средний балл
FROM Результаты GROUP BY [№ зач.книжки] HAVING Avg
(Результат)>4.5
2. SELECT [Код дисциплины], Avg(Результат)AS Сред-
ний балл FROM Результаты GROUP BY [Код дисциплины]
HAVING Avg(Результат)<4
```

В первом случае создается список фамилий студентов с указанием среднего балла для каждого из них и выводятся фамилии тех студентов, которые имеют средний балл выше 4,5.

Во втором случае выводится список кодов дисциплин со средним баллом при условии, что он ниже 4.

Подчиненный запрос. В инструкцию SELECT может быть вложена другая инструкция SELECT, SELECT... INTO, INSERT... INTO, DELETE или UPDATE.

Различают основной и подчиненные запросы, которые являются вложенными в основной запрос.

Подчиненный запрос можно использовать вместо выражения в списке полей инструкции SELECT или в предложениях WHERE и HAVING. Существуют три типа подчиненных запросов:

- сравнение (ANY|ALL|SOME) (инструкция);
- выражение [NOT] IN (инструкция);
- [NOT] EXISTS (инструкция).

Запрос первого типа служит для сравнения выражения с результатом подчиненного запроса.

Ключевые слова: ANY — каждый; ALL — все; SOME — некоторые.

Пример 2.18. Использование подчиненного запроса:

```
SELECT * FROM Оценка WHERE [Результат] > ANY (SELECT
[результат] FROM Оценка WHERE Результат.[№ зач.книжки]
="123124")
```

В этом случае отбираются только те записи из таблицы ОЦЕНКА, в которых значение результата больше каждой оценки студента с номером зачетной книжки 123124.

Запрос второго типа представляет собой выражение, которое должно быть найдено в наборе записей, являющихся результатом выполнения подчиненного запроса.

Пример 2.19. Использование подчиненных запросов:

```
1. SELECT * FROM Студент WHERE [№ зач. книжки] IN
(SELECT [№ зач.книжки] FROM Оценка WHERE [Результат]>=4)
2. SELECT * FROM Дисциплина WHERE [Код дисциплины]
NOT IN (SELECT [Код дисциплины] FROM Оценка)
```

В первом случае отбираются студенты, которые в таблице ОЦЕНКА имеют результат 4 или выше.

Во втором случае отбираются дисциплины, которые не встречаются в таблице ОЦЕНКА.

Запросом третьего типа является инструкция SELECT , заключенная в круглые скобки, с предикатом EXISTS в логическом выражении для определения, должен ли подчиненный запрос возвращать какие-либо записи.

Пример 2.20. Использование подчиненного запроса:

```
SELECT * FROM Студент WHERE EXISTS (SELECT * FROM
Оценка WHERE Сотрудник.[№ зач.книжки] = Оценка.[№ зач.
книжки])
```

В этом случае отбираются студенты, которые имеют хотя бы одну оценку.

2.9. Формы — диалоговый графический интерфейс для работы пользователя с базой данных

Формы предназначены для ввода и просмотра взаимосвязанных данных БД на экране в удобном для пользователя виде. Формы можно распечатывать, а также применять для создания панелей управления в приложении.

Любая форма, с помощью которой хотят просматривать, вводить или редактировать записи таблиц БД, должна быть предварительно сконструирована. В процессе подготовительной работы по разработке формы необходимо определить, из каких таблиц нужно отображать данные, какие именно поля должны быть представлены в форме, нужны ли вычисляемые поля, какие графические элементы (линии, поясняющие текст рисунки) будут использоваться для оформления.

2.9.1. Основы создания формы

Однотабличная форма может быть создана пользователем в режиме конструктора форм или с помощью мастера. В первом случае создание начинается с пустой формы и конструирование полностью возлагается на пользователя. Для создания однотабличной формы целесообразно использовать мастер форм или команды автоформы.

Чтобы начать создание формы, надо в окне базы данных выбрать закладку *Формы* и нажать кнопку [Создать]. Открывшееся диалоговое окно *Новая форма* (рис. 2.6) предоставляет возможность выбрать один из режимов создания формы: *Конструктор*, *Мастер форм*, *Автоформа*: *в столбец*, *Автоформа*: *ленточная*, *Автоформа*: *табличная*, *Диаграмма*, *Сводная таблица*.

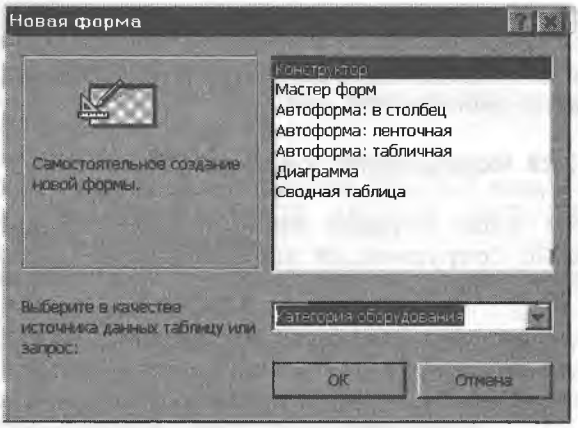


Рис. 2.6. Создание новой формы

Формы, которые удовлетворяют любому, даже самому требовательному вкусу, можно создать с помощью конструктора. Эффективно быстрое создание форм с помощью мастера и дальнейшее их совершенствование с помощью конструктора.

Мастер форм может создать форму для одной таблицы или для нескольких взаимосвязанных таблиц. При выборе только одной таблицы могут быть созданы формы: *В один столбец*, *Ленточная* или *Табличная*.

Форма *В один столбец* выводит в виде колонок для просмотра данные только одной записи, поля которой расположены в нужном порядке (рис. 2.7).

Ленточная форма выводит одну и более записей в зависимости от того, сколько можно уместить их на экране (рис. 2.8).

Табличная форма выводит данные обычным табличным способом, но в отличие от таблиц может выбирать поля для вывода (рис. 2.9).

Мастер форм позволяет пользователю определить, какие поля таблицы включаются в форму, и выбрать стиль ее оформления. Выбор таблицы для создания формы может быть произведен как в

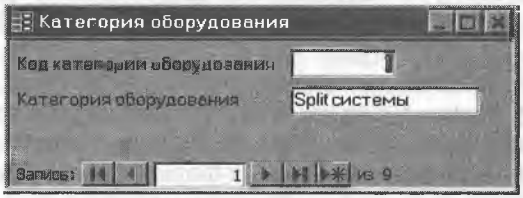


Рис. 2.7. Форма *В один столбец*

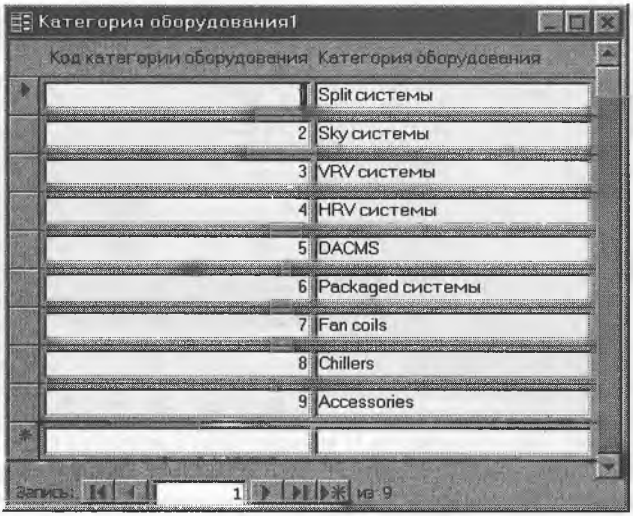


Рис. 2.8. Ленточная форма

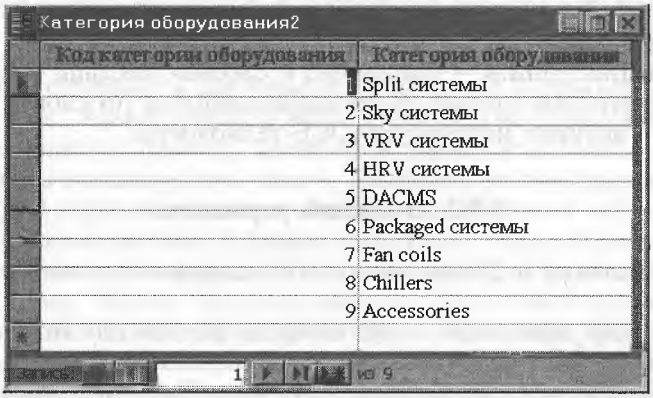


Рис. 2.9. Табличная форма

окне *Новая форма*, так и в первом диалоговом окне мастера *Создание форм*.

С помощью команд *Автоформа*: *в столбец*, *Автоформа*: *ленточная* и *Автоформа*: *табличная* для заданной таблицы создаются формы, которые отличаются от форм, создаваемых мастером, тем, что включают в себя все поля таблицы и не предоставляют возможности выбора стиля оформления. Эти команды, не вступая в диалог с пользователем и не отображая формы в режиме конструктора, выводят ее на экран в режиме формы, т.е. заполненную значениями из таблицы. Заметим, что таблица, для которой строится форма, выбирается в окне *Новая форма*. Форма, созданная

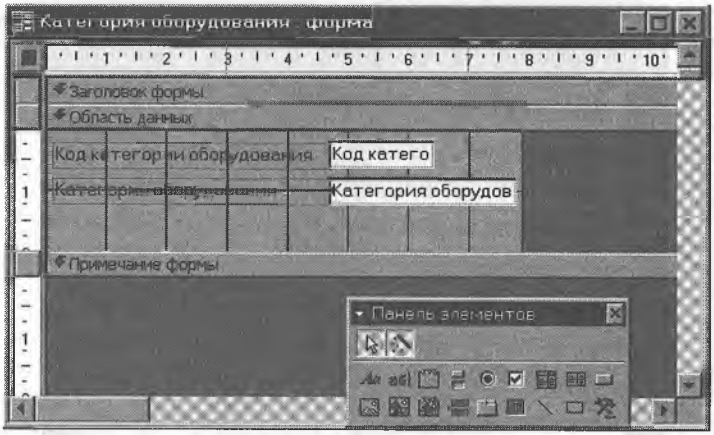


Рис. 2.10. Редактирование формы в режиме конструктора

мастером, так же, как и форма, созданная любой командой *Автоформа*, может быть отредактирована в соответствии с требованиями пользователя. Редактирование выполняется в режиме конструктора форм (рис. 2.10).

Последние опции — *Диаграмма* и *Сводная таблица* — позволяют создавать достаточно специализированные по своим задачам формы и активно используют OLE-технологии.

2.9.2. Элементы управления

Все сведения в форме или отчете содержатся в элементах управления.

Элементы управления — это объекты формы или отчета, которые служат для вывода данных на экран, выполнения макрокоманд или оформления формы или отчета. Например, поле можно использовать для вывода данных на экран в форме или отчете, кнопку — для открытия другой формы или отчета, а линию или прямоугольник — для разделения и группировки элементов управления с тем, чтобы они лучше воспринимались пользователем.

В Microsoft Access на панели элементов в режимах конструктора формы и конструктора запроса имеются следующие типы элементов управления (см. рис. 2.10): надпись, поле, группа, выключатель, переключатель, флажок, поле со списком, список, кнопка, рисунок, свободная рамка объекта, присоединенная рамка объекта, разрыв страницы, набор вкладок, подчиненная форма/отчет, линия, прямоугольник и дополнительные элементы HTML и ActiveX.

Элементы управления могут быть связанными, свободными или вычисляемыми. Связанные элементы управления присоединены к полю базовой таблицы или полю запроса и используются для отображения, ввода или обновления значений из полей базы данных. Для вычисляемого элемента управления в качестве источника данных используется выражение, в котором могут быть данные из поля базовой таблицы или поля запроса для формы или отчета, а также данные другого элемента управления формы или отчета. Для свободных элементов управления источников данных не существует. Используются они для вывода на экран данных, линий, прямоугольников и рисунков.

Надписи предназначены для отображения в форме или отчете описательных текстов: заголовков, подписей или кратких инструкций. В надписях не выводятся значения полей или выражений; они всегда являются свободными и не меняются при переходе от записи к записи.

Надпись может быть присоединена к другому элементу управления (такую надпись называют подписью). Например, если поле создается с присоединенной надписью, которая содержит подпись этого поля, эта надпись появляется как заголовок столбца в форме в режиме таблицы.

Надпись, созданная с помощью инструмента «Надпись», размещается отдельно и не присоединяется ни к какому элементу управления. Такие надписи используются для отображения разных сведений (например, заголовков формы или отчета), а также для вывода поясняющего текста. Надписи, не присоединенные к элементам управления, не отображаются в режиме таблицы.

Поля, используемые в форме или отчете для отображения данных из таблицы, запроса или инструкции SQL, называют присоединенными, потому что они связаны с данными в поле в источнике данных. Кроме того, существуют свободные поля. Например, можно создать свободное поле для отображения результатов вычислений или приема данных, вводимых пользователем. Содержимое свободного поля нигде не сохраняется.

Группа используется в форме или отчете для вывода ограниченного набора параметров. Группа делает выбор параметров простым и наглядным. В каждый момент времени в группе может быть выбран только один параметр. Группа состоит из рамки, набора флажков, переключателей или выключателей.

К полю присоединяется только рамка группы, а не находящиеся в ней флажки, выключатели или переключатели. Пользователь не должен определять свойство *Данные* (ControlSource) для каждого элемента управления в группе. Вместо этого следует задать в свойстве *Значение параметра* (OptionValue) каждого флажка, выключателя или переключателя число, являющееся допустимым для поля, к которому присоединена рамка группы. При выборе пара-

метра в группе Microsoft Access вводит в поле значение, равное значению свойства *Значение параметра* (OptionValue) выбранного элемента.

В свойстве *Значение параметра* (OptionValue) требуется задавать число, так как значение группы может быть только числовым. Microsoft Access сохраняет это число в базовой таблице.

Группа может быть также связана с выражением или быть свободной. Свободные группы применяются в специальных диалоговых окнах для принятия данных, вводимых пользователем, и для выполнения действий на основе этих данных.

Выключатель может быть использован в форме или отчете как отдельный элемент управления, в котором отображаются значения логического поля из базовой таблицы, запроса или инструкции SQL.

Когда пользователь нажимает кнопку выключателя, присоединенного к логическому полю, Microsoft Access отображает значение в базовой таблице в формате, который определяется значением свойства поля *Формат поля* (Format) (*Да/Нет*, *Истина/Ложь* или *Вкл./Выкл.*).

Выключатели особенно удобно использовать в группах, тогда легко видеть, какой из них нажат.

Переключатель может быть использован в форме или отчете как элемент управления, в котором отображаются значения логического поля из базовой таблицы, запроса или инструкции SQL.

Когда пользователь выбирает переключатель, присоединенный к логическому полю, Microsoft Access отображает значение в базовой таблице в формате, который определяется значением свойства поля *Формат поля* (Format) (*Да/Нет*, *Истина/Ложь* или *Вкл./Выкл.*).

Переключатели обычно используются в группе для отображения набора параметров, из которых необходимо выбрать один.

Флажок может быть использован в форме или отчете как отдельный элемент управления, в котором отображаются значения логического поля из базовой таблицы, запроса или инструкции SQL.

При установке или снятии флажка, присоединенного к логическому полю, Microsoft Access отображает значение в базовой таблице в формате, который определяется значением свойства поля *Формат поля* (Format) (*Да/Нет*, *Истина/Ложь* или *Вкл./Выкл.*). Кроме того, флажки включаются в группу для отображения набора выбираемых значений.

Во многих случаях удобнее выбрать нужное значение из списка, чем вводить его с клавиатуры по памяти.

Поле со списком позволяет выбрать любой из способов ввода значения, не требуя при этом значительного места в форме. Поле

со списком является комбинацией двух элементов: поля и раскрывающегося списка. Значение, выбранное или введенное в присоединенное поле со списком, вставляется и в поле, к которому присоединено поле со списком.

В поле со списком список состоит из строк с данными. Строки содержат один или несколько столбцов с заголовками или без заголовков. Если поле со списком, содержащим нескольких столбцов, является присоединенным, то сохраняется значение одного из столбцов.

Свободное поле со списком позволяет сохранять значение, используемое в другом элементе управления. Например, с помощью свободного поля со списком можно ограничить значения, отбираемые в другом поле со списком или специальном диалоговом окне. Свободное поле применяется также для поиска записи с помощью значения, выбранного или введенного в поле со списком.

Поля со списком имеют свойство *Ограничиться списком* (LimitToList), которое определяет, допускается ввод в поле любых значений или только значений, совпадающих с одним из значений списка.

Если в форме достаточно свободного места и требуется, чтобы список постоянно находился на экране, а также если требуется ограничить вводимые данные имеющимся списком, вместо поля со списком можно использовать список.

Список состоит из строк с данными. Строки содержат один или несколько столбцов, которые могут быть снабжены заголовками. Если список из нескольких столбцов является присоединенным, то сохраняются значения одного из столбцов.

Свободный список позволяет хранить значение, используемое в другом элементе управления. Например, с помощью свободного списка можно ограничить значения, отбираемые в другом списке или специальном диалоговом окне. Свободный список применяется также для поиска записи с помощью значения, выбранного в этом списке.

Во многих случаях удобнее выбрать нужное значение из списка, чем вводить конкретное значение по памяти. Кроме того, выбор из списка позволяет быть уверенным, что введенное значение является допустимым.

В тех случаях, если в форме недостаточно места для отображения списка или если наряду с выбором значений из списка требуется вводить новые значения с клавиатуры, вместо списка следует использовать поле со списком.

Кнопки используются в формах для выполнения определенного действия или ряда действий. Например, можно создать в форме кнопку, открывающую другую форму. Чтобы кнопка выполняла какое-либо действие, следует создать макрос или процедуру об-

работки события и связать их со свойством кнопки *Нажатие кнопки* (OnClick).

Мастер кнопок позволяет создать более 30 типов кнопок, при этом для созданной кнопки определяется процедура обработки события. Текст надписи на кнопке задается в качестве значения свойства *Подпись* (Caption). Чтобы поместить на кнопку рисунок, следует указать его в свойстве кнопки *Рисунок* (Picture).

Рисунки, свободные и присоединенные рамки объекта также являются элементами управления. В форму или отчет Microsoft Access можно добавлять объекты или части объектов, созданные в других приложениях, например рисунок, созданный в Microsoft Paint, электронную таблицу, созданную в Microsoft Excel, или текстовый документ, созданный в Microsoft Word. Причем можно вставлять все содержимое файла или только некоторую выделенную его часть.

Способ вставки рисунка или объекта зависит от того, какой объект предполагается создать: присоединенный или свободный. Присоединенный объект хранится в таблице. При переходе к новой записи в форме или отчете отображается другой объект. Например, таким способом удобно хранить фотографии всех сотрудников фирмы. Свободный объект является частью структуры формы или отчета. При переходе к новой записи объект не изменяется.

Подчиненная форма — это форма, находящаяся внутри другой формы, т. е. первичная форма называется главной формой, а форма внутри главной формы — подчиненной. Комбинацию форма/подчиненная форма часто называют также иерархической формой, или комбинацией родительской и дочерней форм.

Подчиненная форма удобна для вывода данных из таблиц или запросов, связанных отношением типа один ко многим. Главная и подчиненная формы в этом типе форм связаны таким образом, что в подчиненной форме выводятся только те записи, которые связаны с текущей записью в главной форме. Например, когда главная форма отображает тип *Напитки*, подчиненная форма отображает только те товары, которые входят в него.

При использовании формы, содержащей подчиненную форму для ввода новых записей, текущая запись в главной форме сохраняется при входе в подчиненную форму. Это гарантирует, что записи из таблицы на стороне «многие» будут иметь связанную запись в таблице на стороне «один», а также автоматически сохранится каждая запись, добавляемая в подчиненную форму.

Подчиненная форма может быть выведена в режиме таблицы и как простая или ленточная форма. Главная форма может быть выведена только как простая форма.

Главная форма может содержать любое число подчиненных форм, если каждая из них помещается в главную форму. Возможно также создание подчиненных форм двух уровней вложенности.

Это означает, что можно иметь одну подчиненную форму внутри главной формы, а другую подчиненную форму внутри первой подчиненной формы. Например, можно иметь главную форму, в которой выводятся данные о клиентах, одну подчиненную форму с выведенными данными о заказах и другую подчиненную форму, в которой отображается то, что заказано.

Подчиненным называют отчет, вставленный в другой отчет. При комбинировании один из отчетов, являющийся главным, может быть как присоединенным, так и свободным, т.е. не базирующимся на таблице, запросе или инструкции SQL.

Свободный главный отчет может служить контейнером нескольких не связанных между собой отчетов, которые требуется объединить.

Главный отчет связывают с таблицей, запросом или инструкцией SQL в тех случаях, когда в него требуется вставить подчиненные отчеты, в которых выводятся данные, связанные с данными в главном отчете. Например, в главном отчете могут быть выведены все записи о продажах за год, а в подчиненном отчете — итоговые суммы продаж за каждый квартал.

В главном отчете могут также содержаться данные, являющиеся общими для двух или нескольких подчиненных отчетов. В этом случае области данных выводятся в подчиненных отчетах.

В главный отчет наряду с подчиненными отчетами включают также подчиненные формы, причем число таких подчиненных форм не ограничивается. Более того, главный отчет может содержать подчиненные формы или отчеты двух уровней вложенности. Например, в главном отчете может содержаться подчиненный отчет, который, в свою очередь, содержит подчиненную форму или подчиненный отчет.

Разрывы страниц, ***Линии***, ***Прямоугольники*** — это элементы управления, используемые для оформления форм и отчетов.

Набор вкладок используется для представления нескольких страниц данных в одном наборе. Это особенно удобно при работе со многими элементами управления, которые могут быть распределены на две или более категорий. Например, элемент управления *Набор вкладок* может быть использован в форме *Сотрудники*, чтобы отделить общие сведения от личных.

Элементы HTML и ActiveX позволяют добавить формам и отчетам еще некоторые функциональные возможности.

2.9.3. Технология загрузки, просмотра и корректировки данных базы с использованием форм

Следует отметить, что технология создания целостной базы, в которой между таблицами установлены связи, предполагает упорядочение загрузки взаимосвязанных таблиц с целью обеспече-

ния пользователя удобным интерфейсом. Такая технология может строиться на использовании соответствующих экранных форм ввода/вывода, обеспечивающих корректный ввод взаимосвязанных данных. Эти формы, как правило, в значительной степени соответствуют формам первичных документов — источников данных для загрузки справочной информации и оперативных учетных данных. При этом реализуется важнейший аспект технологии работы с базой данных — их однократный ввод.

Таким образом, для получения рационально сконструированных форм, обеспечивающих удобную работу пользователя и корректный ввод взаимосвязанных данных при создании и корректировке целостности БД, целесообразно провести подготовительную работу по определению последовательности ее загрузки.

Требования к последовательности загрузки таблиц базы данных определяются схемой данных и формулируются следующим образом:

- независимо могут загружаться таблицы, которые не подчинены каким-либо другим таблицам;
- таблицы, подчиненные каким-либо другим таблицам, могут загружаться либо одновременно с ними, либо после их загрузки;
- в базу данных сначала загружаются с соответствующих документов справочные данные, а затем учетные.

В соответствии с этими требованиями можно рекомендовать следующий порядок загрузки целостной базы данных:

1. Определить документы для загрузки таблиц БД.
2. Определить таблицы в БД, предназначенные для загрузки каждого документа-источника.
3. Определить последовательность этапов загрузки таблиц.
4. Определить подсхему данных (фрагмент схемы данных) для каждого этапа загрузки БД, в которую могут входить:

таблица, являющаяся объектом загрузки;

таблица, связанная с таблицей, являющейся объектом загрузки;

таблица, главная относительно загружаемой.

5. Определить общую структуру экранной формы, т. е. ее макет, согласованный со структурой входного документа и подсхемой данных.
6. Определить состав размещаемых данных для каждой из частей составной формы.
7. Ввести ключевые поля основной части таблицы — источника данных в основную часть формы.
8. Предусмотреть в подчиненной форме поля для ключевых полей таблицы — источника данных, которых нет в основной части.

После выполнения перечисленных пунктов загрузки осуществляется конструирование экранной формы средствами Access.

2.9.4. Разработка многотабличных форм

Многотабличная форма создается на основе нескольких взаимосвязанных таблиц и может состоять из одной формы или из основной и одной или нескольких подчиненных форм. Подчиненная форма может быть построена на основе как подчиненной, так и главной таблиц.

Многотабличная форма может быть создана в режиме конструктора или с помощью мастера форм. Однако в Access наиболее технологичным является первоначальное создание форм с помощью мастера и доработка их в режиме конструктора.

При создании многотабличной формы с помощью мастера Access создает для нее базовую инструкцию SQL, в которую включаются сведения об используемых таблицах и полях.

Назовем способы создания многотабличной формы с помощью мастера.

1. *Явное включение подчиненной формы.* Подчиненная форма строится только на основе подчиненной таблицы по отношению к таблице, на основе которой построена основная часть формы.

2. *Вызов связанной формы по кнопке.* Созданные мастером связанные подчиненные формы могут не включаться непосредственно в основную форму, а вызываться при необходимости включенной в нее кнопкой. При этом открывающееся содержимое связанной формы синхронизировано с текущей записью формы. Этот способ построения удобен для сложных многотабличных форм, перегруженных большим числом элементов управления, а также в случае, когда пользователю не требуется постоянно видеть связанные данные.

3. *Без использования подчиненных и связанных форм.* Такая многотабличная форма создается, если необходимо отображать записи подчиненной таблицы, дополненные полями из одной или нескольких главных таблиц. В этом случае записеобразующим источником данных, выводимых в форму, является запись подчиненной таблицы. При этом форма отображает поля из записи подчиненной таблицы и поля из единственной связанной с ней записи главной таблицы.

4. *На основе запроса.* Для запроса, в котором записи уже созданы за счет объединения полей связанных записей главной и каждой из подчиненной таблиц, мастер строит форму так же, как если бы ему были заданы исходные таблицы. Благодаря этому создается форма, обеспечивающая однократное отображение данных, так как она базируется на исходных нормализованных таблицах.

Полученная с помощью мастера составная форма при необходимости может быть отредактирована, в том числе дополнена другими включаемыми формами.

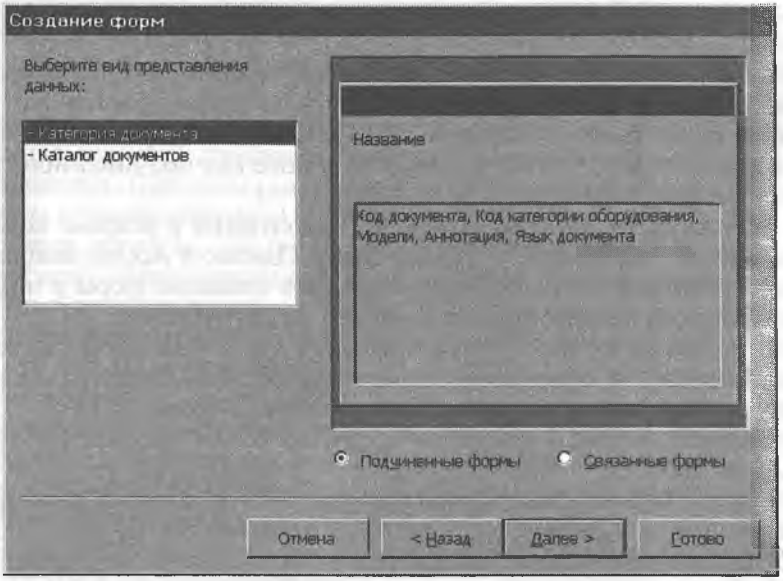


Рис. 2.11. Создание многотабличной формы с подчиненной формой

Выбор необходимых таблиц и полей производится в следующем порядке:

• в окне базы данных выбрать закладку *Форма* и нажать клавишу [Создать];

• в окне *Новая форма* выбрать режим создания *Мастер форм*, а в качестве источника данных основной части формы выбрать из списка таблицу или запрос;

• в первом открывшемся диалоговом окне *Создание форм* последовательно выбрать таблицы и из них поля, включаемые в форму, после чего нажать кнопку [Далее];

• во втором открывшемся диалоговом окне *Создание форм* (рис. 2.11) выбрать вариант создания многотабличной формы, для чего в рамке *Выберите вид представления данных* выделить таблицу, которая является источником основной части формы. (При этом если таблица была выбрана в окне *Новая форма*, она уже выделена.)

Если таблица — источник основной части формы — является главной по отношению к другой таблице, тоже выбранной для формы, то в окне *Создание форм* выбирают один из двух возможных типов подключения подчиненной формы:

для непосредственного включения — *Подчиненные формы* (см. рис. 2.11);

для включения кнопки, вызывающей связанную форму, — *Связанные формы* (рис. 2.12).

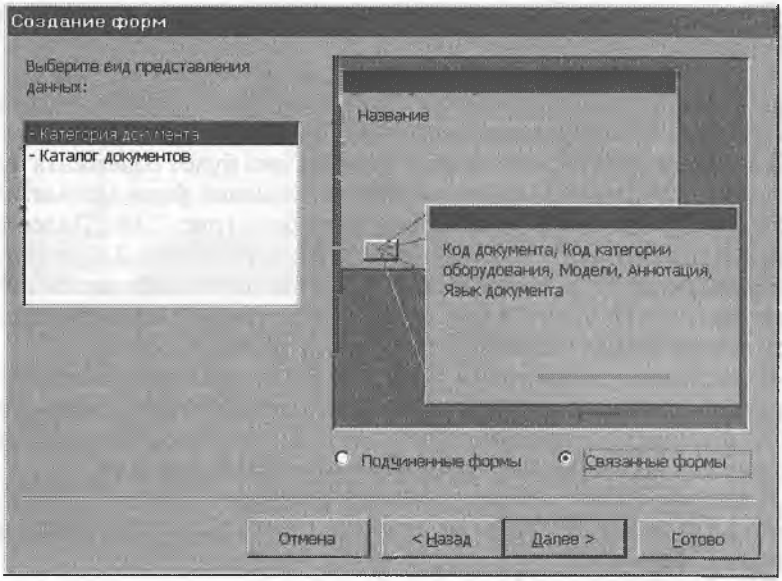


Рис. 2.12. Создание многотабличной формы со связанной формой

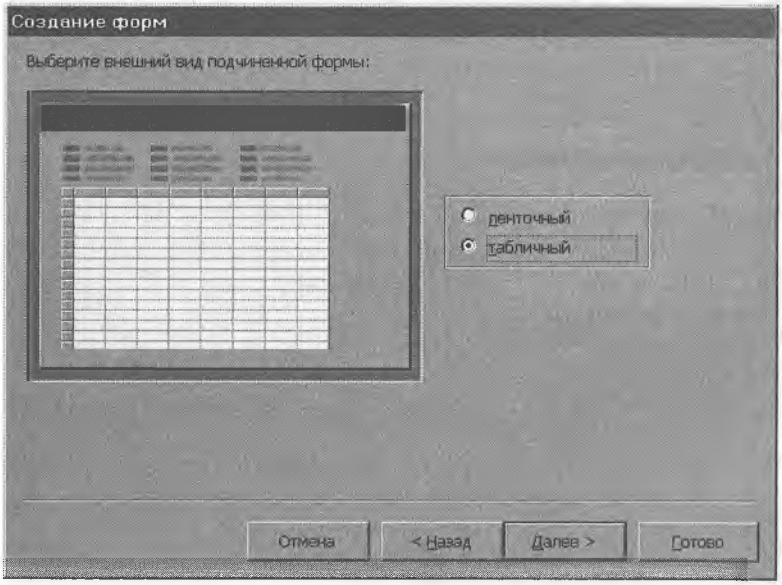


Рис. 2.13. Окно выбора вида формы

Далее можно выбрать вид подчиненной или связанной формы *Ленточный* или *Табличный* (рис. 2.13)

Если таблица — источник основной части формы — является подчиненной по отношению к другой таблице, тоже выбранной для формы, то создаваемая многотабличная форма не будет включать в себя подчиненную форму. Однако она будет содержать поля из главной таблицы. При этом в окне *Создание форм* автоматически установится тип формы *Одиночная форма* (рис. 2.14). Далее выбирают вид формы: *В один столбец*, *Ленточный* или *Табличный*.

В следующем диалоговом окне *Создание форм* выбирается стиль оформления (*Обычный* или какой-либо другой), который определяет отображение надписей и значений полей в форме.

В последнем диалоговом окне *Создание форм* завершается создание формы мастером, т.е. можно отредактировать заголовки форм и выбрать дальнейшие действия: *Открытие формы для просмотра или ввода данных* либо *Изменение макета формы*.

В первом случае автоматически выводится форма с данными (рис. 2.15) и после нажатия кнопки [Готово] мастер завершает создание формы.

Во втором случае форма выводится в режиме конструктора, позволяющем произвести нужную доработку.

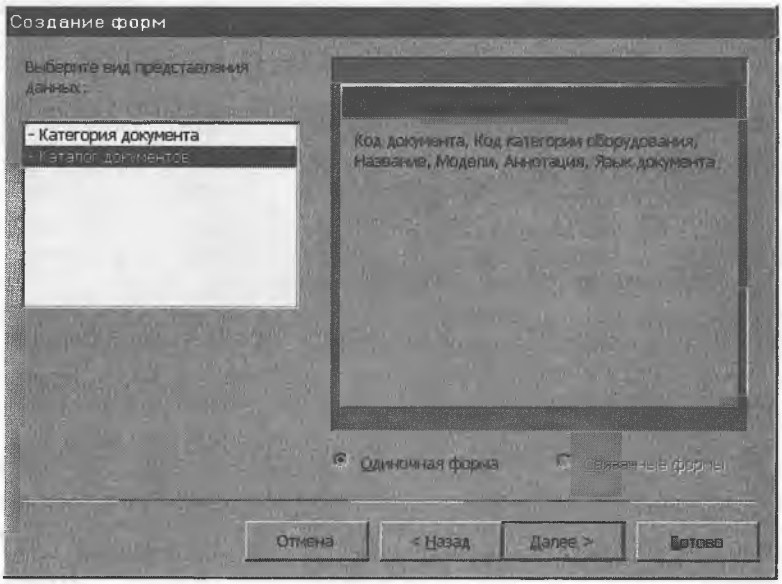


Рис. 2.14. Таблица — источник основной части формы, являющаяся подчиненной по отношению к другой таблице, тоже выбранной для формы

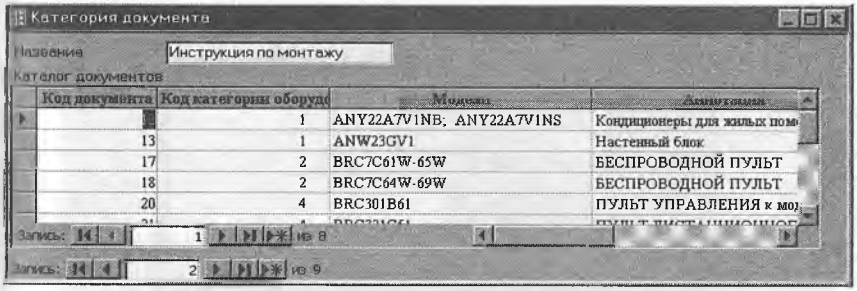


Рис. 2.15. Многотабличная форма, включающая в себя подчиненную форму

Используя технику редактирования формы, можно перемещать поля, менять их свойства, в том числе шрифт и его размеры, подпись поля, текст в заголовке формы основной части.

Переход к редактированию подчиненной формы осуществляется двойным нажатием кнопки мыши на поле подчиненной формы.

2.10. Разработка отчетов

Средства Access по разработке отчетов предназначены для создания макета отчета, по которому может быть осуществлен вывод данных из таблиц в виде выходного печатного документа. Эти средства позволяют конструировать отчет сложной структуры, обеспечивающей вывод взаимосвязанных данных из многих таблиц. При этом возможно выполнение самых высоких требований к оформлению документа.

Access имеет следующие способы создания отчетов: *Конструктор*, *Мастер отчетов*, *Автоотчет*: *в столбец*, *Автоотчет*: *ленточный*, *Мастер диаграмм*, *Почтовые наклейки* (рис. 2.16).

Во многих случаях удобно использовать мастер отчетов и созданный им отчет затем дорабатывать в режиме конструктора.

При необходимости вывода данных из многих таблиц в качестве основы для отчета можно использовать многотабличный запрос. С помощью запроса можно выполнять наиболее сложные виды выборки и предварительной обработки данных. Разнообразные возможности конструктора отчетов позволяют полученные в запросе данные успешно структурировать и оформлять.

Отметим, что в режиме автоотчета, задаваемого с помощью меню или кнопки [Новый объект] на панели инструментов *База данных*, создается отчет, данные в котором выведены в столбец. Режим ленточного автоотчета, который выводит данные из всех полей таблицы в колонку, можно вызвать нажатием кнопки [Создать] на вкладке *Отчеты*.

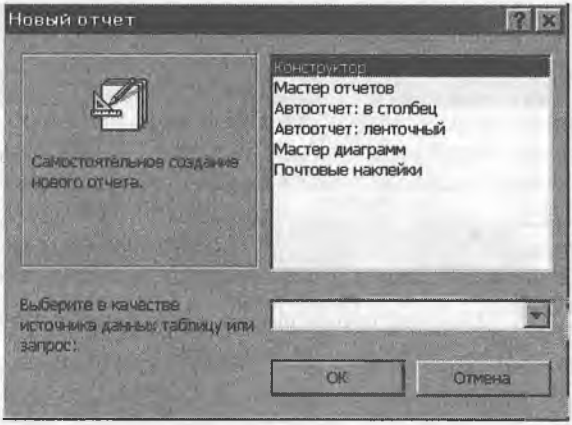


Рис. 2.16. Окно выбора способа создания отчета

Мастер отчетов отличается от автоотчетов тем, что позволяет выбирать поля для отчета, группировать данные по какому-либо полю, устанавливать интервал группировки, порядок сортировки, а также выбирать макет отчета и его стиль. При этом главным преимуществом отчетов перед формами является возможность группировки в них данных, что значительно улучшает внешний вид документа и его удобочитаемость.

Для создания отчета с помощью мастера отчетов следует нажать кнопку [Создать] на вкладке *Отчеты* в окне *База данных* и, указав в окне *Новый отчет* таблицу, на базе которой создается отчет, щелкнуть мышью на строке *Мастер отчетов*. В результате на экране появится первое диалогового окно мастера отчетов *Создание отчетов*. В этом окне (рис. 2.17) поля списка *Доступные поля* перемещаются в список *Выбранные поля* нажатием кнопки со стрелкой (>). Все поля из одного списка в другой можно переместить кнопками с двойными стрелками (»), после чего следует щелкнуть мышью по кнопке [Далее].

В следующем открывшемся диалоговом окне мастера (рис. 2.18) определяется способ группировки, для чего необходимые поля из левого списка переносятся в правый. Заметим, что данные в отчете можно группировать не более чем по трем полям. Выбор уровня группировки осуществляется кнопкой [Уровень], после чего нажимают кнопку [Группировка].

Открывшееся при этом окно *Интервал группировки* служит для изменения интервалов группирования данных в отчете. При использовании стандартного значения *Обычный* в одну группу объединяются записи с одинаковыми значениями в заданном поле. Переход в следующее окно для определения способа сортировки данных осуществляется нажатием кнопки [OK].

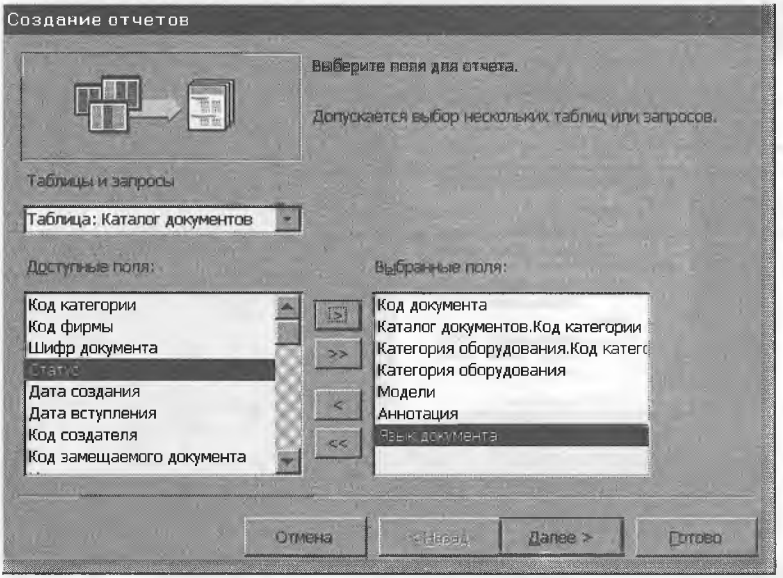


Рис. 2.17. Выбор таблиц и полей для формирования отчета

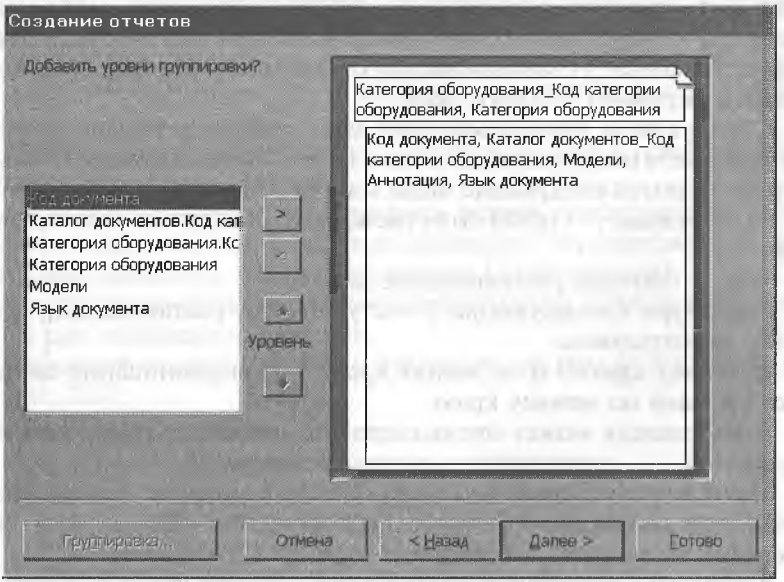


Рис. 2.18. Выбор уровня группировки

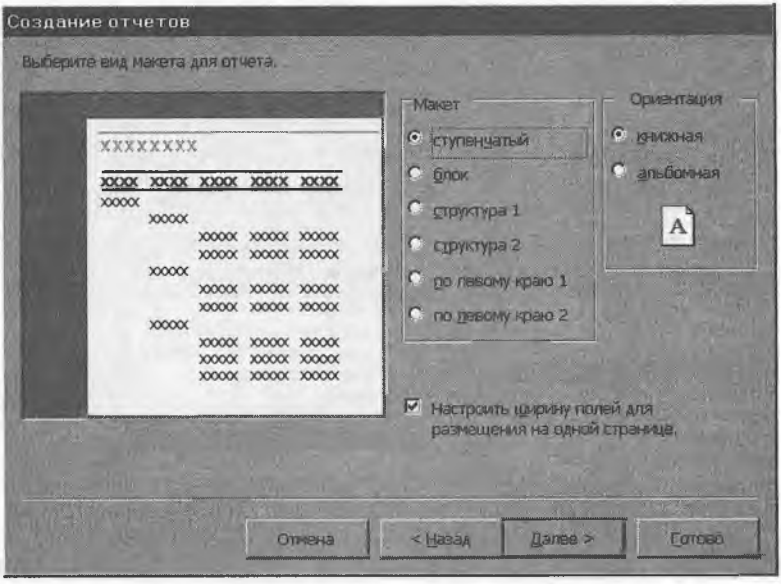


Рис. 2.19. Выбор вида макета отчета

Access автоматически сортирует данные по тем же полям, по которым выполняется их группировка. Если записи в группе должны быть рассортированы и по другим полям, их необходимо указать в четырех специально отведенных для этого полях. Порядок сортировки устанавливается с помощью кнопки, расположенной справа от данного поля.

В следующем диалоговом окне мастер отчетов предлагает указать варианты отображения данных в отчете. В распоряжении пользователя имеются следующие виды макета (рис. 2.19):

ступенчатый — ступенчатое расположение данных разных уровней;

блок — блочное расположение данных;

структура 1 и *структура 2* — ступенчатое расположение уровней с перекрытием;

по левому краю 1 и *по левому краю 2* — выравнивание данных всех уровней по левому краю.

Пользователь может активизировать любой из предложенных вариантов.

Мастер предоставляет возможность просмотреть, как выглядит отчет при выбранном способе расположения данных, для чего отведена левая часть этого окна.

В области *Ориентация* этого же окна можно выбрать подходящую ориентацию листа.

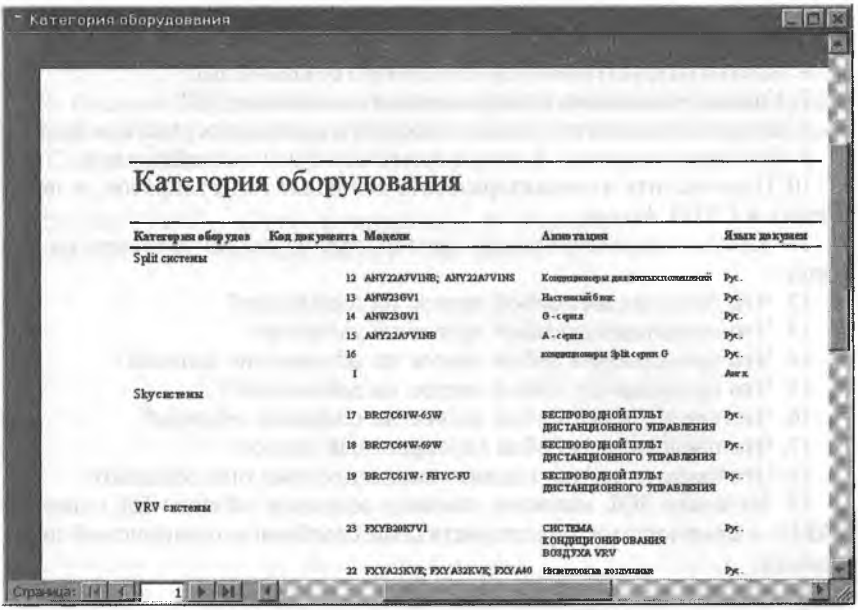


Рис. 2.20. Результат создания отчета с помощью мастера отчетов

Чтобы полнее использовать площадь страницы, следует установить опцию *Настроить ширину полей для размещения на одной странице*, с помощью которой подбирается оптимальная ширина полей. После этого нажать кнопку [Далее].

Следующее окно мастера предназначено для выбора стиля оформления отчета и в нем также имеется поле для просмотра образцов.

Выбрав стиль, следует нажать кнопку [Далее] в последний раз и присвоить отчету имя.

Из этого же окна можно открыть режим конструктора, чтобы внести исправления в отчет или улучшить его оформление, для чего надо щелкнуть мышью по кнопке [Конструктор].

Отчет можно сохранить с помощью команды *Сохранить* из меню *Файл* под подходящим именем.

Результат создания отчета с помощью мастера отчетов показан на рис. 2.20.

Контрольные вопросы и упражнения

1. Дать краткую характеристику СУБД Access.
2. Что такое реляционная СУБД?
3. Перечислить (кратко) сервисные возможности Access.
4. Перечислить типы данных, допустимых для использования в Access.

5. Что представляют собой и как осуществляются сортировка и фильтрация данных?

6. Кратко охарактеризовать технологию создания БД.

7. Какими способами осуществляется заполнение БД?

8. Описать технологию ввода и просмотра данных посредством формы.

9. Что такое запросы? Какими возможностями они обладают?

10. Перечислить и охарактеризовать основные типы запросов, используемых в СУБД Access.

11. Кратко охарактеризовать технологию создания запросов на выборку.

12. Что представляет собой запрос на изменение?

13. Что представляет собой запрос на удаление?

14. Что представляет собой запрос на обновление записей?

15. Что представляет собой запрос на добавление?

16. Что представляет собой запрос на создание таблицы?

17. Что представляет собой перекрестный запрос?

18. Что такое отчеты и какими возможностями они обладают?

19. На языке SQL написать команду создания таблицы БД с именем TABL1 и следующими характеристиками столбцов и ограничений целостности:

A — целый;

B — символьный (25 символов);

C — символьный (3 символа);

ограничение на уникальность: столбец *A* — первичный ключ.

20. На языке SQL написать команду удаления таблицы БД с именем TABL1.

21. На языке SQL написать команду добавления в таблицу TABL1 столбца со следующими характеристиками: имя столбца — *D*; тип данных — строковый, 10 символов.

22. На языке SQL написать команду добавления в таблицу со схемой

```
СТУДЕНТ(ФИО, Дата рождения, № группы)
```

строки со следующими значениями столбцов:

```
<Иванов И.И., 20 августа 1980, 2>.
```

23. На языке SQL написать команду выборки данных о студентах учебной группы №2. Схема таблицы:

```
СТУДЕНТ(ФИО, Дата рождения, № группы).
```

24. На языке SQL написать команду выборки данных о студентах учебной группы №2, рожденных в августе 1980 г. Схема таблицы:

```
СТУДЕНТ(ФИО, Дата рождения, № группы).
```

25. На языке SQL написать команду выборки данных о студентах учебной группы №2, обучающихся по дисциплине Д1. Схемы таблиц:

```
СТУДЕНТ(ФИО, Дата рождения, № группы);
ОЦЕНКА(ФИО, Дисциплина, Оценка).
```

26. На языке SQL написать команду выборки данных о студентах учебной группы №2, получивших неудовлетворительные оценки на экзаменах. Схемы таблиц:

```
СТУДЕНТ(ФИО, Дата рождения, № группы);
ОЦЕНКА(ФИО, Дисциплина, Оценка).
```

ГЛАВА 3
РАЗРАБОТКА ПРИЛОЖЕНИЙ ПОЛЬЗОВАТЕЛЯ

3.1. Макросы и их создание

Несмотря на наличие в Access таких объектов, как запросы, формы и отчеты, для реализации практических задач пользователи высокой квалификации могут использовать средства программирования: язык макросов и язык Visual Basic for Applications (VBA). Макросы и модули на языке VBA оперируют указанными выше объектами Access и объединяют разрозненные действия с ними в единую программу, направленную на решение задачи пользователя, уменьшая при этом его вмешательство или полностью отстраняя от решения.

Макрос — это программа, состоящая из последовательности макрокоманд. *Макрокоманда* — это инструкция, ориентированная на выполнение определенного действия.

Например, макрокомандой можно открыть форму, отчет, напечатать отчет, запустить на выполнение запрос, применить фильтр, присвоить значение, создать свое меню для формы или отчета. Макрокоманда *ЗадатьКомандуМеню* позволяет выполнить любую заданную команду меню. Имеющийся в Access набор макрокоманд (около 50) реализует практически любые действия, которые необходимы для решения различных задач.

Язык макросов обеспечивает возможность выполнения большинства задач без использования программирования на Visual Basic. Макросы, являясь надстройкой над Visual Basic, обеспечивают пользователя средствами решения задач, не требующими детального знания программирования. Язык макросов является языком более высокого уровня, чем Visual Basic.

Набор макросов, имеющийся в Access, определяет набор методов обработки его объектов. В Access имеются также средства, обеспечивающие взаимодействие макросов с объектами при выполнении пользователем определенных действий. Это позволяет управлять выполнением программы извне, т.е. пользователем. Выполняя различные действия для решения своих задач, пользователь инициирует выполнение макросов, автоматизирующих решение связанных с его действиями подзадач. Такой подход существенно отличает программирование задач пользователя на язы-

ке макросов от их программирования в традиционном понимании, при котором только программа управляет процессом решения.

Отметим, что наличие средств запуска программ пользователем не исключает возможности написания программ на языке макросов, которые без вмешательства пользователя могут решать нужные задачи полностью. В этом случае задача решается рядом взаимосвязанных макросов. При этом связи между макросами могут иметь сложную ветвистую логическую структуру, для организации которой в макросе определяются условия выполнения макрокоманд. Это позволяет пользователю для решения своих задач запускать только главную программу, а далее все управление ее выполнением осуществляется изнутри. После запуска программа сама открывает необходимые объекты, выбирает и обрабатывает данные, вызывает другие макросы, следуя алгоритму решения задачи пользователя. При необходимости программа инициирует диалог с пользователем.

Формирование макроса осуществляется в диалоговом режиме и сводится к записи в окне макроса последовательности макрокоманд, в соответствии с которой они и выполняются.

Создание макроса начинается в окне базы данных, где надо выбрать закладку *Макросы* и нажать кнопку [Создать], которая открывает соответствующее окно. В этом окне макрокоманды, составляющие макрос, можно ввести в столбец *Макрокоманда*, для чего достаточно нажать кнопку раскрытия списка макрокоманд в этом столбце и выбрать нужную макрокоманду (рис. 3.1). Можно ввести имя макрокоманды и с клавиатуры. Макрокоманда по умолчанию создается со значениями аргументов, соответствующими выбранному объекту. Например, при перетаскивании таблицы создается макрокоманда *ОткрытьТаблицу*.

В нижней части окна можно сформировать *Аргументы макрокоманды*. Значения аргументов задаются путем выбора их из списка, открывшегося в данной строке. В поле справа от строк аргументов выводится сообщение с пояснениями для выбранного аргумента. В строки столбца *Примечание* вводится необязательный комментарий, описывающий результат выполнения макрокоманды. Каждая новая макрокоманда макроса добавляется к уже существующим макрокомандам записью ее в ближайшую незанятую строку бланка. При этом порядок размещения макрокоманд в бланке определяет последовательность их выполнения. После ввода всех макрокоманд в макрос его надо сохранить, воспользовавшись командой меню *Файл|Сохранить* или кнопкой на панели инструментов макроса.

Для выполнения макроса можно нажать кнопку на панели инструментов [Запуск]. Если макрос уже закрыт, то его надо выбрать в окне базы данных и нажать кнопку [Запуск] в этом окне. Таким

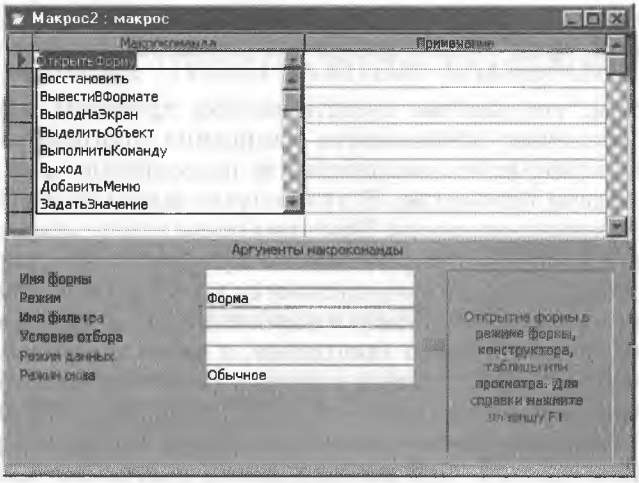


Рис. 3.1. Окно создания макроса

образом, по инициативе пользователя задача может решаться многократно. Для просмотра и редактирования существующего макроса надо выбрать его в окне базы данных и нажать кнопку [Конструктор].

Группа макросов создается как один макрос, в котором содержатся несколько макросов (например, связанных с решением одной задачи или используемых при работе с одной формой). Работать с группой часто оказывается удобнее, чем с несколькими отдельными макросами.

Для создания группы макросов нужно вызвать столбец *Имя макроса*, нажав кнопку [Имена макросов] на панели инструментов. В ячейку этого столбца надо ввести имя первого макроса, входящего в группу, затем записать макрокоманды, выполняемые в первом макросе. Аналогичным образом надо ввести имена других макросов и их макрокоманды. Все макросы, созданные в одном окне, будут составлять одну группу макросов. Имя, указанное при сохранении такой группы макросов, будет именем группы, которое выводится в списке макросов в окне базы данных.

Для ссылок на макросы, которые вошли в группу макросов, используется следующий синтаксис:

```
<Имя Группы Макросов>.<Имя Макроса>
```

Использование условий в макросе. Условия позволяют задать порядок передачи управления между макрокомандами в макросе и обеспечивают выполнение определенных ветвей алгоритма. Например, если в макросе проверяется значение поля в форме на соответствие заданным условиям, то для одних значений может

потребоваться вывод сообщения, а для других значений — вывод отчета.

Условие вводится в строку макрокоманды в столбец *Условие*, который вызывается в окно макроса нажатием соответствующей кнопки на панели инструментов. Задается условие с помощью логического выражения.

Ветвления в программе. В программе, состоящей из макрокоманд, можно организовать ветвления.

Для организации ветвлений в программе нужно наряду с условиями использовать макрокоманды *ОстановитьМакрос* и *ЗапускМакроса*, последняя из которых позволяет создавать также циклы в программах.

Организация выполнения макросов. При запуске макроса выполнение начинается с первой макрокоманды и следует по алгоритму, реализуемому макросом. В процессе выполнения проверяются условия и в зависимости от результата выполняются те или иные макрокоманды или макросы. При вызове другого макроса управление передается вызванному макросу. Вызванный макрос может выполняться несколько раз. После выполнения вызванного макроса управление возвращается к вызывающему макросу и продолжается выполнение его макрокоманд. При этом, следуя алгоритму, макрос выполняется по одному из заранее определенных путей из множества возможных. Таким образом, макрос сам выбирает этот путь в зависимости от условий.

В Access имеется возможность организации выполнения макросов с использованием механизма расширенной обработки событий (т.е. результатов действий пользователя). Access распознает определенные события, к которым может привязываться запуск макроса. Событиями, например, являются открытие отчета, ввод новых данных, перевод фокуса на другую запись или поле в форме, щелчок мышью. Существуют события формы, события элемента управления, события записи и раздела формы, события отчета и раздела отчета. Различные события вызывают различную реакцию системы, поэтому порядок выполнения макросов зависит от порядка возникновения событий и в значительной степени определяется действиями пользователя в формах. При этом управление программой в основном осуществляется пользователем, который выполняет различные действия, а программа реагирует на них.

3.2. Программирование на языке VBA

Язык VBA стал общим инструментом для всех приложений Microsoft Office, позволяющим решать любые задачи программирования, начиная от автоматизации действий конкретного пользо-

вателя и кончая разработкой полномасштабных приложений, использующих Microsoft Office как среду разработки.

Модель программирования в Access является событийно-управляемой, т. е. в процессе работы приложения возникают события, которые запускают специальные программы — обработчики событий. Большое количество разнообразных событий определено в таких объектах Access, как формы, отчеты и элементы управления в них.

Программный доступ к постоянным (хранимым в приложении Access) и временным объектам осуществляется с помощью объектных моделей VBA.

3.2.1. Объекты и семейства VBA

Язык VBA является объектно-ориентированным языком программирования. Стандартные объекты VBA представляют собой основные средства манипуляции с данными Microsoft Access и других приложений семейства Microsoft Office. Знание технологии объектно-ориентированного программирования и состава объектных моделей VBA позволяет разрабатывать профессиональные приложения, выполняющие всю необходимую обработку данных.

Объект (object) — абстракция, которой оперируют в объектно-ориентированных языках программирования. Объект обладает собственными характерными признаками, отличающими его от других объектов; кроме того, объект имеет свое поведение.

Класс (class) — описание совокупности однотипных объектов. Класс можно сравнить с типом данных, где переменной является объект. В этом случае говорят, что объект представляет собой экземпляр определенного класса.

Свойство (property) — отдельная характеристика объекта или класса. Свойство объекта может принимать определенное значение.

Метод (method) — процедура (или функция) объекта или класса. У объекта или класса может быть определенное количество методов и свойств. Методы определяют поведение объекта. В объектно-ориентированных языках программирования поведение приложения определяется поведением созданных в нем объектов.

Объект может реагировать на определенное ***событие*** (event), происходящее в процессе работы приложения и влияющее на объект. Совокупность событий, на которые объект способен реагировать, определяется создателем класса, экземпляром которого является данный объект. Реакцией объекта на произошедшее событие может быть выполнение им некоторых заданных действий — специальной процедуры, которая называется процедурой обработки события. Любому событию объекта может быть назначена некоторая процедура его обработки.

Семейство (collection) — упорядоченный набор однотипных объектов, т.е. экземпляров одного класса. Семейство тоже является объектом, и одним из методов этого объекта является процедура, возвращающая ссылку на конкретный объект в семействе. Одним из свойств семейства является число объектов, хранящихся в нем.

Объектная модель (object model) — совокупность взаимосвязанных объектов, описывающих программную систему.

В VBA определены специальные объектные модели для каждого компонента семейства Microsoft Office и объектные модели, общие для всех компонентов Microsoft Office. С помощью объектных моделей, определенных в VBA, осуществляется управление приложениями Microsoft Office.

В базе данных Microsoft Access могут храниться такие объекты, как таблицы, запросы, формы, отчеты, макросы и модули, а также ссылки на объекты — страницы доступа к данным. В проекте Microsoft Access могут храниться такие объекты, как формы, отчеты, макросы и модули, ссылки на страницы доступа к данным, а также ссылки на объекты, хранящиеся в базе данных на SQL-сервере (таблицы, представления, диаграммы базы данных и хранимые процедуры). Страница доступа к данным представляет собой Web-страницу, хранящуюся отдельно от БД или проекта Microsoft Access . В базе данных или проекте хранится только ссылка на страницу доступа к данным в виде ярлыка (подобного ярлыку файла Windows). Доступ к объектам, хранящимся в приложении Microsoft Access, осуществляется с помощью окна базы данных или проекта.

К постоянным объектам (содержащимся в базе данных или проекте Access) относятся подчиненные объекты, например элементы управления в форме, отчете, на странице доступа к данным. Кроме постоянных объектов бывают временные объекты, т.е. объекты VBA, которые существуют только в период времени выполнения приложения.

Создание объектов в Microsoft Access осуществляется интерактивно или программно.

Чтобы изменить свойства объекта, достаточно щелкнуть правой кнопкой мыши по нему (например, по таблице в окне базы данных или проекта Access) и выбрать в контекстном меню команду *Свойства* (Properties). Появится окно свойств объекта, представленное на рис. 3.2.

Окно свойств объекта элемента управления (рис. 3.3) тоже открывается с помощью команды контекстного меню *Свойства* (Properties). Чтобы отобразить контекстное меню элемента управления, достаточно открыть форму (отчет или страницу доступа к данным) в режиме конструктора и щелкнуть правой кнопкой мыши по элементу управления.

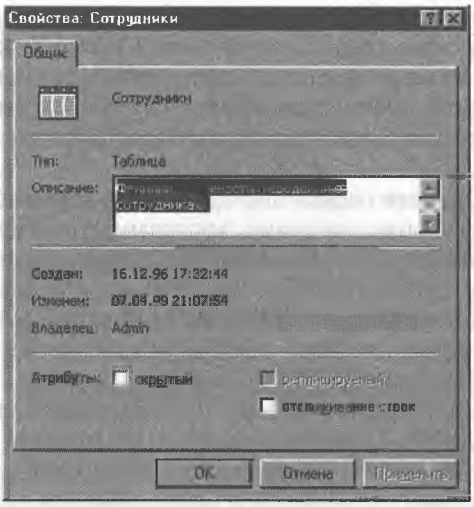


Рис. 3.2. Окно свойств объекта приложения Access

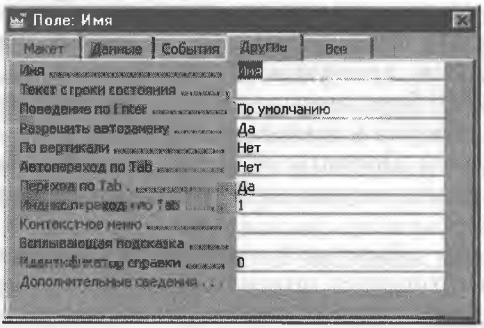


Рис. 3.3. Окно свойств объекта элементов управления

Другие свойства объектов можно изменить только программно, т.е. с помощью макроса или процедуры VBA.

Программный доступ к постоянным (хранимым в приложении Access) и временным объектам осуществляется с помощью объектных моделей VBA.

Объектная модель Microsoft Access реализована в виде набора объектов, собранных в библиотеке Access (рис. 3.4). Основным элементом в иерархии объектов библиотеки Access является объект Application, который содержит ссылки на все объекты и семейства объектов Microsoft Access. Каждый объект из библиотеки Access имеет в качестве свойства объект Application, который ссылается на активное приложение Microsoft Access.

Каждый объект может содержать набор свойств, часть из которых может являться ссылками на другие объекты.

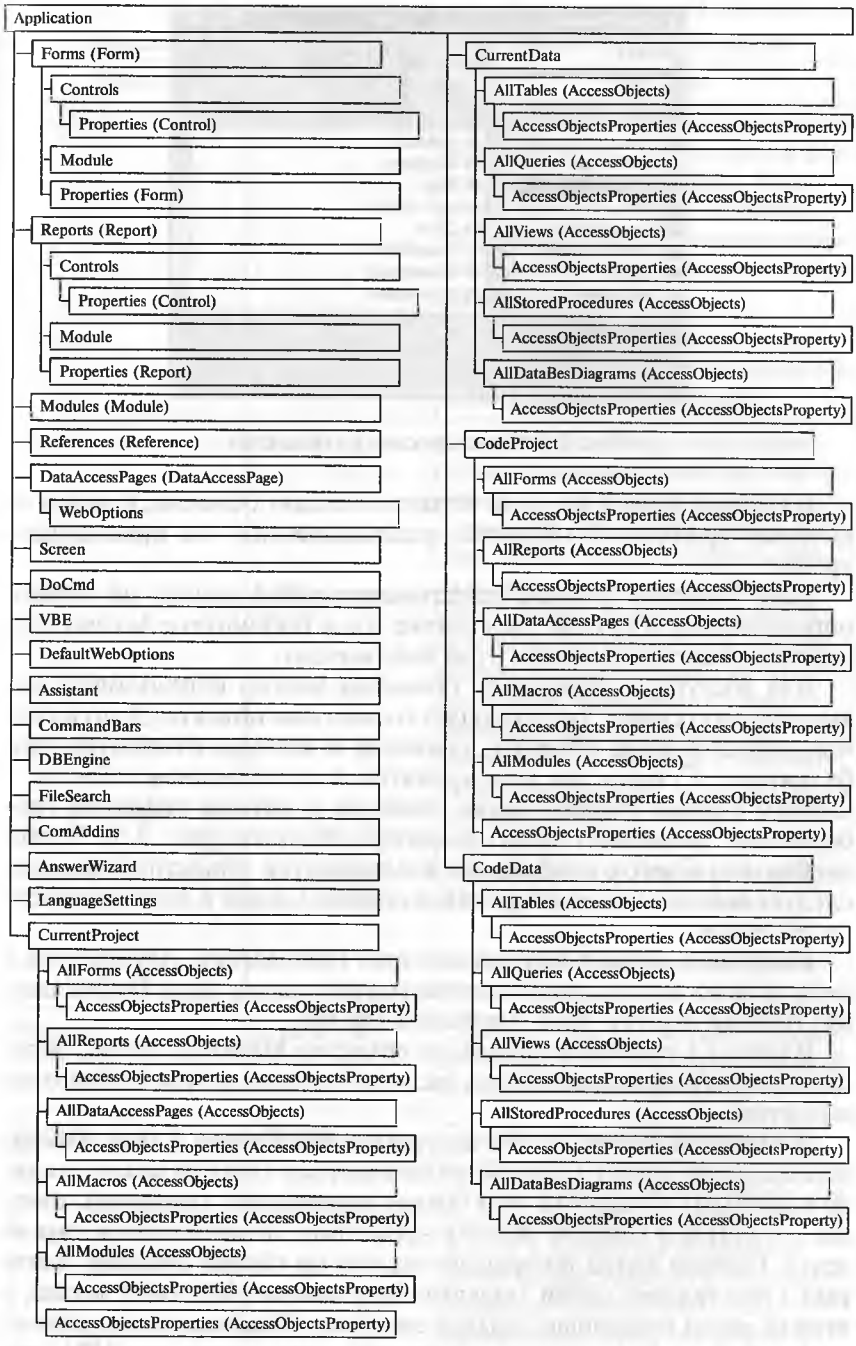


Рис. 3.4. Структура объектов MS Access

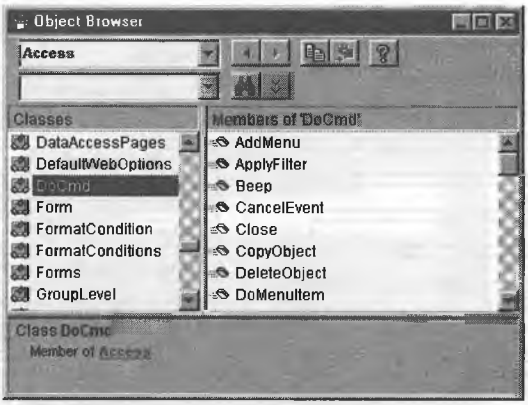


Рис. 3.5. Окно просмотра объектов

В каждый новый уровень иерархии входят объекты, ссылки на которые хранятся в объектах, расположенных на предыдущем уровне.

Если свойство объекта представляет собой ссылку на объект, определенный в другой библиотеке (не в библиотеке Access), для него приводится название этой библиотеки.

Для доступа к некоторым объектам можно использовать сокращенную ссылку, содержащую только имя объекта. Дело в том, что определенные объекты, свойства и методы считаются глобальными. Ссылки на них хранятся в специальном объекте с именем Global. Узнать, какие свойства и методы являются глобальными, позволяет окно просмотра объектов (рис. 3.5). Чтобы отобразить список глобальных компонентов объектной модели, следует выбрать элемент globals в списке *Classes* в окне просмотра объектов.

Например, объект DoCmd является глобальным. Для доступа к нему можно использовать сокращенную ссылку вида *DoCmd* вместо полной ссылки вида *Application.DoCmd*

В табл. 3.1 приведено описание объектов Microsoft Access, определенных в библиотеке Access (если не указана другая библиотека объектов).

В Microsoft Access 97 для программного доступа к базе данных используется объект Database из библиотеки DAO (объекты доступа к данным). Ссылку на этот объект возвращают, например, методы CurrentDb и CodeDb объекта Application, отличающиеся друг от друга. Первый метод возвращает ссылку на объект Database, который представляет собой текущую базу данных Microsoft Access, а второй метод возвращает ссылку на объект Database, представляющий собой ту базу данных, в которой выполняется код VBA, содержащий вызов метода CodeDb. Метод CodeDb целесообразно ис-

Таблица 3.1

Описание составляющих объектной модели Microsoft Access

Объект	Тип	Описание
Application	Объект	Ссылается на активное приложение Microsoft Access. Используется для управления приложением. Этот объект является COM-компонентом и может быть использован другим приложением, которое поддерживает Automation
Forms	Семейство	Содержит объекты Form, соответствующие всем открытым в данный момент формам в БД Access
Form	Объект	Ссылается на конкретную форму Access, описывает свойства формы, элементы управления и модуль, содержащий процедуры формы. Каждый объект Form содержит в качестве свойства по умолчанию семейство Controls, представляющее собой элементы управления данной формы. Следовательно, доступ к элементам управления формы может осуществляться двумя способами — явно (как к элементу семейства Controls) и неявно (как к элементу объекта Form): 1. Forms!Заказы.Controls!НомерЗаказа 2. Forms!Заказы!НомерЗаказа Последний способ доступа выполняется быстрее
Reports	Семейство	Содержит объекты Report, соответствующие всем открытым в данный момент отчетам в БД Access
Report	Объект	Ссылается на конкретный отчет Access, описывает свойства отчета, элементы управления и модуль, содержащий процедуры отчета. Доступ к элементам управления отчета осуществляется с помощью семейства Controls (свойства по умолчанию) двумя способами — явно и неявно

Продолжение табл. 3.1

Объект	Тип	Описание
Modules	Семейство	Содержит объекты Module, соответствующие всем открытым в данный момент стандартным модулям и модулям объектов в БД Access
Module	Объект	Ссылается на конкретный стандартный модуль или модуль класса Access, описывает содержимое модуля в строках кода. Приложение Microsoft Access кроме стандартных модулей, не привязанных к объектам, модулей форм и отчетов, привязанных соответственно к форме или отчету, может иметь модули классов, не зависящие от других объектов приложения и определяющие новый класс. Тип модуля можно узнать с помощью свойства Type
References	Семейство	Содержит объекты Reference, представляющие собой установленные ссылки в приложении Access. С помощью данного семейства в приложении можно динамически устанавливать или удалять ссылки на внешние библиотеки
Reference	Объект	Соответствует ссылке, установленной на библиотеку объектов или типов, содержит информацию о ссылке, включающую в себя имя библиотеки и путь к соответствующему файлу
DataAccessPages	Новое семейство	Содержит объекты DataAccessPages, соответствующие открытым в данный момент страницам доступа к данным в БД или проекте Access
DataAccessPage	Новый объект	Ссылается на конкретные страницу доступа к данным Access. Описывает свойства страницы
Screen	Объект	Ссылается на конкретные форму, отчет или элемент управления, которые в данный момент имеют фокус

Продолжение табл. 3.1

Объект	Тип	Описание
DoCmd	Объект	Позволяет выполнить макрос или встроенную инструкцию Access с помощью процедуры на VBA
VBE	Новый объект из библиотеки VBIDE	Предоставляет доступ к функциональным средствам нового редактора Visual Basic для Microsoft Access
DefaultWebOptions	Новый объект	Предоставляет доступ к атрибутам приложения, используемым в Access по умолчанию при открытии и сохранении Web-страниц
WebOptions	Новый объект	Предоставляет доступ к атрибутам конкретной страницы доступа к данным, используемым в Access при открытии и сохранении Web-страниц. Эти атрибуты имеют более высокий приоритет, чем соответствующие атрибуты приложения. Если установлены атрибуты страницы доступа к данным, то соответствующие атрибуты приложения для работы с Web-страницами игнорируются. Если атрибуты страницы доступа к данным изменяются, то автоматически такие же значения получают атрибуты приложения. Следовательно, полезно сохранять текущие атрибуты приложения, чтобы восстанавливать их после изменения атрибутов страницы доступа к данным
DBEngine	Объект из библиотеки DAO	Является объектом самого верхнего уровня в объектной модели DAO. Позволяет использовать объекты доступа к данным в приложении Access
CurrentProject	Новый объект	Ссылается на программный проект (представляющий собой совокупность всех программных модулей, включая стандартные модули и модули классов) текущей базы данных или проекта Microsoft Access.

Объект	Тип	Описание
CurrentProject	Новый объект	Этот объект содержит семейства объектов AccessObjects, соответствующих реальным объектам базы данных или проекта: AllForms, AllReports, AllMacros, AllModules, AllDataAccessPages. Перечисленные семейства содержат все реальные объекты БД независимо от того, открыты они или закрыты в данный момент
CurrentData	Новый объект	Ссылается на объекты, сохраненные приложением — источником данных (ядром Jet или SQL-сервером) в текущей базе данных. Содержит семейства объектов AccessObject: AllTables, AllQueries, AllViews, AllSto-redProcedures, AllDatabaseDiagrams. Перечисленные семейства содержат все реальные объекты БД, независимо от того, открыты они или закрыты в данный момент
CodeProject	Новый объект	Ссылается на программный проект кода той базы объекта (или проекта Microsoft Access), в которой выполняется (и содержится) код VBA, имеющий данную ссылку. Этот объект содержит семейства объектов AccessObjects, соответствующих реальным объектам базы данных или проекта: AllForms, AllReports, AllMacros, AllModules, AllData-AccessPages. Перечисленные семейства содержат все реальные объекты БД независимо от того, открыты они или закрыты в данный момент
CodeData	Новый объект	Ссылается на объекты, сохраненные приложением — источником данных (ядром Jet или SQL-сервером) в той БД, в которой выполняется (и содержится) код VBA, имеющий данную ссылку. Содержит семейства объектов AccessObject:

Продолжение табл. 3.1

Объект	Тип	Описание
CodeData	Новый объект	AllTables, AllQueries, AllViews, AllStoredProcedures, AllDatabase-Diagrams. Перечисленные семейства содержат все реальные объекты базы данных независимо от того, открыты они или закрыты в данный момент
Controls	Семейство	Содержит объекты Control, представляющие собой все элементы управления в конкретных форме, отчете или секции. Это семейство является свойством объектов Form, Report, Section и Control
Control	Объект	Представляет собой любой конкретный элемент управления в форме, отчете или секции. Может ссылаться на один из объектов, характеризующих элемент управления определенного типа: CheckBox, TextBox, ComboBox, CommandButton, CustomControl, BoundObjectFrame, Image, ListBox, ObjectFrame, OptionButton, Option-Group, Page, Section, SubForm, TogleButton
Format Conditions	Новое семейство	Содержит объекты FormatCondition и представляет собой набор форматов по условию для объектов Text-Box и ComboBox
Format Condition	Новый объект	Представляет собой форматирование по условию, определенное для элемента управления типа поле ввода (объект TextBox) или список с полем ввода (объект ComboBox)
AllForms	Новое семейство	Содержит объекты AccessObject и представляет собой все формы в объектах CurrentProject и CodeProject
AllReports	Новое семейство	Содержит объекты AccessObject и представляет собой все отчеты в объектах CurrentProject и CodeProject
AllDataAccessPages	Новое семейство	Содержит объекты AccessObject и представляет собой все страницы

Продолжение табл. 3.

Объект	Тип	Описание
AllDataAccessPages	Новое семейство	доступа к данным в объектах CurrentProject и CodeProject
AllMacros	Новое семейство	Содержит объекты AccessObject и представляет собой все макросы в объектах CurrentProject и CodeProject
AllModules	Новое семейство	Содержит объекты AccessObject и представляет собой все программные модули в объектах CurrentProject и CodeProject
AllTables	Новое семейство	Содержит объекты AccessObject и представляет собой все таблицы в объектах CurrentData и CodeData
AllQueries	Новое семейство	Содержит объекты AccessObject и представляет собой все запросы в объектах CurrentData и CodeData. При этом количество запросов в проекте Access должно равняться нулю
AllViews	Новое семейство	Содержит объекты AccessObject и характеризует все представления в объектах CurrentData и CodeData. При этом количество представлений в базе данных Access должно равняться нулю
AllStoredProcedures	Новое семейство	Содержит объекты AccessObject и представляет собой все хранимые процедуры в объектах CurrentData и CodeData. При этом количество хранимых процедур в базе данных Access должно равняться нулю
AllDatabaseDiagrams	Новое семейство	Содержит объекты AccessObject и представляет собой все схемы базы данных в объектах CurrentData и CodeData. При этом количество схем базы данных в БД Access должно равняться нулю
AccessObject	Новый объект	Ссылается на реальный объект Microsoft Access в любом из следующих семейств: AllForms, AllReports, AllMacros, AllModules, AllDataAccessPages, AllTables, AllQueries,

Окончание табл. 3.1

Объект	Тип	Описание
AccessObject	Новый объект	AllViews, AllStoredProcedures, AllDatabaseDiagrams. В зависимости от того, к какому семейству он принадлежит, представляет собой любой из объектов Access — таблицу (Table), запрос (Query), отчет (Report), форму (From), модуль (Module), макрос (Macro), страницу доступа к данным (Dataaccess page), представление (View), хранимую процедуру (Stored procedure) или схему базы данных (Database diagram). Объект AccessObject ссылается на существующий объект базы данных. При этом нельзя создать новый или удалить существующий объект AccessObject
AccessObjectProperty	Новое семейство	Содержит настраиваемые объекты семейства AccessObjectProperty, описывающие свойства и однозначно характеризующие конкретные объекты: AccessObject, CodeData, CodeProject, CurrentData или CurrentProject
AccessObjectProperty	Новый объект	Представляет собой встроенные или определенные пользователем характеристики (свойства) любого из следующих объектов: AccessObject, CodeData, CodeProject, CurrentData или CurrentProject

пользовать в процедурах библиотек Access, так как вызовом такой процедуры можно получить доступ к библиотечной базе данных.

В Microsoft Access 2000 у объекта Application появились новые свойства: CurrentProject, CurrentData, CodeProject и CodeData, позволяющие манипулировать объектами базы данных. Теперь вместо объекта Database из библиотеки DAO для работы с базой данных или проектом Access можно использовать объекты CurrentProject, CurrentData, CodeProject и CodeData из библиотеки Access.

Пары объектов CurrentProject, CodeProject и CurrentData, CodeData имеют одинаковые свойства и методы. Отличие объектов

CurrentProject и CurrentData от CodeProject и CodeData подобно отличию друг от друга объектов, возвращаемых методами CurrentDb и CodeDb. Объекты CurrentProject и CurrentData соответствуют текущей базе данных, а объекты CodeProject и CodeData — базе данных, в которой выполняется (и содержится) код VBA, содержащий ссылки на эти объекты.

3.2.2. Процедуры и функции VBA

Основными компонентами программы на VBA являются процедуры и функции, которые представляют собой фрагменты программного кода, заключенные между операторами Sub и End Sub или между Function и End Function. Например:

```
Sub <имяПроцедуры> (<аргумент1 >, <аргумент2>,…)
<операторVisualBasic1>
<операторVisualBasic2>
End Sub
```

Функция отличается от процедуры тем, что ее имя выступает также в качестве переменной и используется для возвращения значения в точку вызова функции. Например:

```
Function <имяФункции> (<аргумент1>, <аргумент2>,…)
<операторVisualBasic1>
<операторVisualBasic2>
…
<имяФункции> = <возвращаемоеЗначение>
End Function
```

Для того чтобы использовать процедуру или функцию, необходимо их вызвать. Процедуру с непустым списком аргументов можно вызвать только из другой процедуры или функции, использовав ее имя со списком фактических значений аргументов в качестве одного из операторов VBA. Функцию же можно вызвать не только с помощью отдельного оператора VBA, но и поместив ее имя со списком фактических значений аргументов прямо в формулу или выражение в программе на VBA, или, например, прямо в формулу в вычисляемых полях запросов, форм и отчетов Access. Процедура с пустым списком аргументов (так называемый командный макрос) может быть вызвана не только из другой процедуры или функции, но и с помощью комбинации клавиш быстрого вызова, команд раскрывающихся меню или кнопок панелей инструментов. Можно также связать такую процедуру с выполнением самых различных событий, например, с открытием формы или отчета, с щелчком мышью по кнопке в форме, с воздействи-

ем на элементы управления форм, в частности на элементы управления ActiveX. Такие процедуры называют процедурами обработки событий. Функции или процедуры, нуждающиеся в передаче им аргументов, таким способом вызвать нельзя. Если вызываемая процедура имеет уникальное имя и находится в том же модуле, что и вызывающая процедура, то для ее вызова достаточно указать это имя и затем задать список фактических значений аргументов, не заключая его в скобки. Второй способ вызова процедуры состоит в использовании оператора Call. В этом случае сначала указывают оператор Call, потом имя процедуры, а затем список параметров, обязательно заключенный в скобки. Функцию можно вызывать так же, как и процедуру, но гораздо чаше применяется другой, специфический способ ее вызова: использование ее имени с заключенным в скобки списком параметров в правой части оператора присваивания.

Пример 3.1. Вызов процедуры под именем myF1 с передачей ей двух аргументов (константы и выражения):

```
myF1 7, i + 2
```

или

```
Call myF1(7, i + 2).
```

Пример 3.2. Вызов функций Left и Mid и использование возвращаемого ими значения в выражении:

```
yStr = Left (y, 1) & Mid (y, 2, 1).
```

Допустимы два различных способа передачи переменных процедуре или функции: ***по ссылке*** и ***по значению***.

Если переменная передается по ссылке, то это означает, что процедуре или функции будет передан адрес этой переменной в памяти. При этом вызываемая процедура может изменить значение фактического параметра.

Если же фактический параметр передается по значению, то вызываемая процедура или функция получает только значение фактического параметра, но не саму переменную, используемую в качестве этого параметра. Все изменения полученного значения фактического параметра (если они выполняются вызываемой процедурой) не сказываются на значении его переменной.

Способ передачи параметров переменных процедуре или функции указывается при описании ее аргументов (формальных параметров). Имени аргумента может предшествовать явный описатель способа передачи (ByRef — задает передачу по ссылке, а ByVal — по значению). Если же явное указание способа передачи парамет-

ра отсутствует, то по умолчанию подразумевается передача по ссылке.

Пример 3.3. Передача параметров переменных процедуре:

```
Sub Main ()
a = 10
b = 20
c = 30
Call Example1 (a, b, c)
Call MsgBox(a)
Call MsgBox(b)
Call MsgBox(c)
End Sub
Sub Example1 (x, ByVal y, ByRef z)
x = x + 1
y = y + 1
z = z + 1
Call MsgBox(x)
Call MsgBox(y)
Call MsgBox(z)
End Sub
```

Вспомогательная процедура Example1 использует в качестве формальных аргументов три переменные, описанные по-разному. Далее в теле этой процедуры каждый из аргументов увеличивается на единицу, а затем их значения выводятся на экран с помощью функции MsgBox. Основная процедура Main устанавливает значения переменных a, b и c, а затем передает их в качестве фактических аргументов процедуре Example1. При этом первый аргумент передается по ссылке (действует умолчание), второй по значению, а третий снова по ссылке. После возврата из процедуры Example1 основная процедура также выводит на экран значения трех переменных, передававшихся в качестве аргументов.

Всего на экран выводится шесть значений: сначала числа 11, 21 и 31 (все значения увеличены на единицу и выводятся процедурой Example1); затем числа 11, 20 и 31 (которые выводятся процедурой Main, причем переменные, переданные по ссылке, увеличены, а переменная, переданная по значению, нет).

Программа может состоять (и обычно состоит) из многих процедур и функций, которые могут располагаться в одном или нескольких ***модулях***, группирующихся в ***проекты***. При этом в одном проекте могут мирно сосуществовать несколько различных программ, использующих общие модули или процедуры.

Каждая из процедур, находящихся в одном модуле, должна иметь уникальное имя, однако в проекте может содержаться несколько различных модулей. Обычно рекомендуется использовать только уникальные имена процедур в одном проекте, но допус-

тимы и исключения. В том случае, если в проекте содержится несколько различных процедур с одним и тем же именем, следует для уточнения имени использовать при вызове процедуры следующий синтаксис:

```
<имяМодуля>.<имяПроцедуры>
```

Если при этом имя модуля состоит из нескольких слов, следует заключить это имя в квадратные скобки. Например, если модуль называется *Графические процедуры*, а процедура — *График*, вызов может выглядеть следующим образом:

```
[Графические процедуры].График
```

Допускается также использование процедур, расположенных и в других проектах. При этом может потребоваться еще один уровень уточнения имени:

```
<имяПроекта>.<имяМодуля>.<имяПроцедуры>
```

3.2.3. Переменные, константы и типы данных

Как и в других языках программирования, в VBA для хранения временных значений, передачи параметров и проведения вычислений используются переменные.

Обычно перед использованием переменной производится ее объявление (определение имен переменных, которые будут использоваться в программе) и объявление типа данных, для хранения которых предназначена эта переменная.

В VBA, как и в обычном языке Basic, для этого используется оператор Dim. Синтаксис этого оператора имеет следующий вид:

```
Dim <имяПеременной> [As <типДанных>]
```

В VBA действуют следующие правила именования переменных.

Имя не может быть длиннее 255 символов.

Имя должно начинаться с буквы, за которой могут следовать буквы, цифры или нижняя черта.

Имя не должно содержать пробелов, знаков препинания или специальных символов, за исключением самого последнего знака.

В конце к имени переменной может быть добавлен один из следующих шести специальных символов — описателей типа данных: !, #, $, %, &, @. Эти символы не являются частью имени переменной, т.е. если в программе используются одновременно имена string1$ и string1, то они относятся к одной и той же строковой переменной. Нельзя использовать одно и то же имя переменной с разными символами определения типа данных или од-

новременно явное описание типа данных и не соответствующий этому типу данных специальный символ.

Кроме того, не допускается использование в качестве имен переменных ключевых слов VBA и имен стандартных объектов.

Допускается использование в именах переменных букв не только латинского алфавита, но и кириллицы, что может оказаться удобным для русских пользователей: при желании можно давать переменным имена на русском языке.

Пример 3.4. Описание переменных:

```
Dim i As Integer, j As Integer
Dim x As Double
```

В VBA имеется оператор Option Explicit. Если начать модуль с этого оператора (он должен быть расположен в самом начале модуля до начала первой процедуры), то VBA будет требовать обязательного объявления переменных в этом модуле и генерировать сообщения об ошибке всякий раз, как встретит необъявленную переменную.

Краткий перечень используемых типов данных VBA дан в табл. 3.2.

Таблица 3.2

Типы данных VBA

Тип данных	Описание
Array	Массив переменных; для ссылки на конкретный элемент массива используется индекс. Требуемая память зависит от размеров массива
Boolean	Принимает одно из двух логических значений: True (*Истина*) и False (*Ложь*). Требуемая память 2 байт
Byte	Число без знака от 0 до 255. Требуемая память 1 байт
Currency	Используется для денежных вычислений с фиксированным числом знаков после десятичной запятой в тех случаях, когда важно избежать возможных ошибок округления. Диапазон возможных значений: от –922 337 203 685 477,5808 до 922 337 203 685 477,5807. Требуемая память 8 байт. Символ определения по умолчанию @
Date	Используется для хранения дат. Диапазон возможных значений от 1 января 0100 г. до 31 декабря 9999 г. Требуемая память 8 байт

Окончание табл. 3.2

Тип данных	Описание
Double	Числовые значения с плавающей точкой двойной точности. Диапазон возможных значений для отрицательных чисел от –1,79769313486232E–308 до –4,94065645841247E–324. Диапазон возможных значений для положительных чисел от 4,94065645841247E–324 до 1,79769313486232–E308. Требуемая память 8 байт. Символ определения типа по умолчанию #
Integer	Короткие целые числовые значения. Диапазон возможных значений от –32 768 до 32 767. Требуемая память 2 байт. Символ определения типа по умолчанию %
Long	Длинные целые числовые значения. Диапазон возможных значений от –2 147 483 648 до 2 147 483 647. Требуемая память 4 байт. Символ определения типа по умолчанию &
Object	Используется только для хранения ссылок на объекты. Требуемая память 4 байт
Single	Числовые значения с плавающей точкой обычной точности. Диапазон возможных значений для отрицательных чисел от –3,402823E38 до –1,401298E–45. Диапазон возможных значений для положительных чисел от 1,401298E–45 до 3,402823E38. Требуемая память 4 байт. Символ определения типа по умолчанию !
String	Используется для хранения строковых значений. Длина строки от 0 до 64 Кбайт. Требуемая память 1 байт на символ. Символ определения типа по умолчанию $
Variant	Может использоваться для хранения различных типов данных: даты/времени, чисел с плавающей точкой, целых чисел, строк, объектов. Требуемая память 16 байт плюс 1 байт на каждый символ строковых значений. Символ определения типа по умолчанию отсутствует
Определяемый пользователем тип	Определяемые пользователем типы данных, назначение и размер выделяемой памяти зависят от определения. Используется для описания структур данных. Позволяет хранить в переменной такого типа множество значений различного типа

При описании переменной указание типа данных может быть опущено. Тип переменной может в таком случае определяться последним символом ее имени: @, #, %, &, !, $ (соответственно Currency, Double, Integer, Long, Single, String). Например, поскольку символ $ является символом определения типа для строковых данных, то переменная под именем text$ автоматически становится переменной типа «строка символов». В дальнейшем этот специальный символ указания типа данных может быть опущен, однако постоянное присутствие в имени переменной символа определения типа будет напоминать о том, к какому типу данных относится эта переменная, что поможет избежать ошибок использования несовместимых типов данных.

Если же последний символ не является ни одним из перечисленных ранее и явное указание типа тоже не используется, то в этом случае переменной назначается по умолчанию тип данных variant, который позволяет хранить в ней данные любого типа.

Для определения типа данных аргументов процедуры или функции используется описание типа данных непосредственно в их заглавной строке. Например, заглавная строка процедуры, описывающей параметры как переменные строкового типа, имеет вид

```
Sub SplitStr(str1 As String, str2 As String, str3 As
String)
```

Определение типа данных возвращаемого функцией значения завершает заглавную строку функции, например, выражение

```
Function FindSplitSpace (str1 As String) As Integer
```

описывает возвращаемое функцией значение как переменную короткого целого типа.

Для описания констант используется оператор Const, схожий с оператором описания переменных Dim. Синтаксис этого оператора имеет вид

```
Const <имяКонстанты> [As <типДанных>] = <выражение>
```

Здесь <выражение> — это любое значение или формула, возвращающие значение, которое должно использоваться в качестве константы. Например, оператор, определяющий целую константу maxLen, имеет вид

```
Const maxLen% = 30
```

Как и переменные, константы могут содержать значения различных типов данных, но при этом они не меняют своих значений во время выполнения программы.

Кроме описываемых пользователем констант существуют еще предопределенные встроенные константы, которые используют в тексте программ без предварительного описания. При именовании встроенных констант используется стандартное соглашение, позволяющее определить, к объектам какого приложения они относятся: имена встроенных констант, относящихся к объектам Access, начинаются с префикса ac, относящихся к объектам Excel — с префикса xl, относящихся к объектам Word — с префикса wd, а относящихся к объектам VBA — с префикса vb.

Например, в команде

```
DoCmd.OpenForm "Orders", acNormal,, stLinkCriteria
```

используется встроенная константа Access acNormal.

Кроме обычных переменных в Visual Basic часто используются переменные, представляющие собой ссылку на объект. Оказывается, что зачастую использование переменных для ссылок на объекты позволяет не только сократить и упростить текст программы, но и существенно ускорить ее работу.

Использование переменной объекта немного отличается от использования обычных переменных: нужно не только объявить такую переменную, но и перед ее использованием назначить ей соответствующий объект с помощью специального оператора Set. Синтаксис такого объявления и назначения имеет вид

```
Dim <имяПеременной> As Object
Set <имяПеременной> = <ссылкаНаОбъект>
```

Иногда при объявлении такой переменной удобно заранее указать конкретный тип объекта, для чего можно использовать любой конкретный объект из объектной модели Office. Например:

```
Dim MyBase As Database
Set MyBase = DBEngine.Workspaces(0).Databases(0)
```

После такого объявления и назначения можно использовать переменную MyBase для обращения к текущей открытой базе данных.

Массив — это переменная, в которой хранится одновременно несколько значений одинакового типа. Формально массив представляет собой совокупность однотипных индексированных переменных.

Число индексов массива также может быть различным. Чаще всего используются массивы с одним или двумя индексами, реже — с тремя; большее число индексов встречается крайне редко. В VBA допускается использовать до 60 индексов. О числе индексов мас-

сива обычно говорят как о его размерности. Массивы с одним индексом называют одномерными, с двумя — двухмерными и т.д. Массивы с большим количеством измерений могут занимать очень большие объемы памяти, поэтому следует быть осторожным при их применении.

Прежде чем использовать, массив нужно обязательно объявить его с помощью оператора Dim и указать при этом тип хранящихся в нем значений. Все значения в массиве обязаны принадлежать к одному типу данных. Это ограничение на практике можно обойти, используя при объявлении массива тип variant, тогда элементы массива смогут принимать значения различных типов.

Синтаксис оператора объявления массива имеет вид

```
Dim <имяМассива> (<размер1>, <размер2>,…) As <тип Данных>
```

Здесь (<размер1>, <размер2>, ...) — размеры массива, т.е. число индексов и максимально допустимое значение для каждого конкретного индекса.

При этом индексирование элементов массива по умолчанию начинается с нуля. Так, объявление

```
Dim Array1 (9) As Integer
```

определяет одномерный массив из 10 элементов, являющихся переменными целого типа, а объявление

```
Dim Array2 (4, 9) As Variant
```

определяет двухмерный массив из пятидесяти (5 × 10) элементов, являющихся переменными универсального типа Variant.

При объявлении массива можно указать не только верхнюю границу индекса, но и нижнюю, т.е. явно задать диапазон изменения конкретного индекса массива, причем нижняя граница может быть любым целым числом. Синтаксис такого определения имеет вид

```
Dim <имяМассива> (<мин1> To <макс1>, …) As <типДанных>
```

В приведенных примерах речь все время шла о массивах фиксированного размера, число элементов в которых явно указано во время описания массива в операторе Dim. Такие массивы называются *статическими*.

В VBA допускается использование и *динамических* массивов, размеры которых при описании не фиксируются. Определение размера динамического массива может быть сделано непосред-

ственно во время выполнения программы. При определении динамического массива в операторе Dim после имени массива стоят лишь пустые скобки и описание типа переменных. Число индексов и диапазон их изменения не задаются. Однако перед тем как использовать массив, нужно выполнить оператор ReDim, который задаст размерность и диапазоны изменения индексов динамического массива. Синтаксис объявления и определения размеров динамического массива имеет вид

```
Dim <имяМассива> () As <типДанных>
ReDim <имяМассива> (<размер1>, <размер2>,…)
```

3.2.4. Область действия переменных и процедур

Все процедуры, функции, переменные и константы в VBA имеют свою область действия. Это означает, что все они могут использоваться только в определенном месте программного кода, т.е. там, где они описаны. Например, если переменная A описана с помощью оператора Dim в теле процедуры с именем Proc1, то именно эта процедура и является ее областью действия. Таким образом, если имеется другая процедура Proc2, то использовать в ней эту же переменную нельзя. Если попытаться сделать это, можно либо получить сообщение об ошибке из-за использования неописанной переменной (в случае, если используется упоминавшийся выше оператор Option Explicit), либо другую переменную с тем же самым именем, но никак не связанную с одноименной переменной из первой процедуры.

В каком месте программы и как именно описана переменная, определяет область ее действия, а также длительность жизни в памяти и сохранения присвоенного ей значения.

Имеется три различных уровня при определении области действия переменных: уровень процедуры, уровень модуля и уровень проекта.

Чтобы определить переменную на уровне процедуры, ее описание помещается в тело этой процедуры.

Чтобы определить процедуру на уровне модуля и сделать ее тем самым доступной для совместного использования во всех процедурах этого модуля, следует поместить ее описание в секции объявлений модуля (перед текстом каких-либо процедур или функций). При этом можно использовать и явное описание области действия, т.е. вместо ключевого слова Dim использовать в этом случае ключевое слово Private.

Наконец, чтобы описать переменную на уровне проекта, необходимо расположить ее описание в секции объявлений одного из модулей проекта и при этом обязательно использовать ключевое слово Public. Описанные таким образом переменные могут

использоваться в любом модуле проекта. Все сказанное относится к описанию и определению области действия констант и массивов.

Для переменных имеется еще один способ описания, не изменяющий их уровня, но позволяющий сохранить значение переменной, описанной на уровне процедуры, после завершения работы этой процедуры. Для этого используют описатель Static, т.е. определяют такую переменную как статическую. Эта переменная сохраняет выделенное ей место в памяти и свое значение даже после завершения процедуры, в которой она была описана и использована.

Тем не менее статическая переменная не может быть использована в других процедурах. Изменяется лишь время ее жизни, но не область действия. Если произойдет повторный вызов той же самой процедуры, в которой была описана статическая переменная, то эта переменная сохранит свое прежнее значение, которое она имела в момент завершения работы этой процедуры при предыдущем вызове. Обыкновенные (не статические) переменные всякий раз инициализируются заново и получают при входе в процедуру пустые значения.

Процедуры и функции имеют только два уровня областей действия: уровень модуля и уровень проекта. По умолчанию используется уровень проекта. Таким образом процедура или функция может быть вызвана любой другой процедурой или функцией в этом проекте. При описании процедур и функций на уровне проекта может также использоваться необязательное ключевое слово Public, но никакого воздействия на процедуру наличие или отсутствие этого слова не оказывает.

Если требуется описать процедуру, используемую только на уровне модуля, применяется ключевое слово Private. Такое описание не только сужает область действия для процедуры, но и запрещает ее использование как самостоятельной, т.е. эту процедуру можно вызвать только из другой процедуры.

Наконец, при описании процедур или функций может использоваться и ключевое слово Static. Оно никак не влияет на область действия процедуры, но воздействует на все переменные, описанные внутри этой процедуры или функции. В этом случае все локальные переменные получают статус Static и тем самым сохраняются в памяти после завершения такой процедуры и при повторном ее вызове сохраняют свои прежние значения.

Пример 3.5. Описание модуля:

```
Public A1 As String
Private A2 As Integer
```

```
Dim A3 As Single
Sub Proc1 ()
   Dim A4 As Integer
   Static A5 As Integer
   A1 = "Текстовая строка 1"
   A2 = 2
   A3 = 3.14
   A4 = A4+4
   A5 = A5+5
   MsgBox A4
   MsgBox A5
   End Sub
Sub Proc2() Proc1
   MsgBox A1
   MsgBox A2
   MsgBox A3
   MsgBox A4
   MsgBox A5
   Proc1
EndSub
```

В этом примере переменная A1 определена на уровне всего проекта (использовано ключевое слово Public), переменные A2 и A1 — на уровне модуля, переменная A4 — только на уровне процедуры Proc1, а переменная A5 хотя и определена в теле процедуры Proc1, но описана как статическая переменная.

При вызове процедуры Proc2 происходит следующее: из этой процедуры, в свою очередь, вызывается процедура Proc1, которая присваивает значения всем пяти переменным A1...A5, а затем показывает текущие значения переменных A4 и A5 в диалоговом окне.

После завершения этой процедуры выводятся текущие значения переменных A1... A5 из процедуры Proc2. При этом оказывается, что переменные A1... A3 сохранили свои значения, поскольку они описаны на уровне модуля, а переменные A4 и A5 принимают пустые значения, поскольку областью их действия являются процедуры, в которых они используются. Никакие изменения этих переменных внутри одной из процедур не имеют отношения к аналогичным переменным из другой процедуры, так как на самом деле это разные переменные, просто для них используются совпадающие имена.

Затем происходит еще один вызов процедуры Proc1, и она вновь начинает изменять и выводить на экран значения переменных A4 и A5. При этом переменная A4 вновь получает значение 4, поскольку при новом вызове процедуры для этой переменной заново выделяется память, и она будет инициализирована пустым значением. В отличие от A4 переменная A5, описанная как статическая переменная, сохраняет свое прежнее значение от предыдущего вызова этой процедуры, и в результате ее значение при повторном вызове оказывается равным 10.

3.2.5. Модули VBA

Код VBA в приложении Access собран в модули. Модули являются такими же объектами Access, как таблицы, запросы, формы, отчеты, страницы и макросы, о чем свидетельствует ярлык на панели объектов в окне *База данных* (Database). Основное содержание модулей — это процедуры на языке VBA. Существует два типа модулей: стандартные и модули класса.

Стандартные модули содержат общие процедуры, которые могут использоваться в разных местах приложения: при обработке событий в разных объектах, для вычисления значений в разных запросах или формах, а также вызываться из других модулей и т.д. Эти процедуры не связаны с конкретным объектом: формой или отчетом. Они могут использоваться для объявления глобальных (т.е. доступных из всех окон приложения) переменных, констант, типов, внешних процедур.

Если в процедурах модуля нет ссылок на конкретные объекты данного приложения (формы, отчеты, элементы управления), то такой модуль может с успехом использоваться другими приложениями Access.

Список стандартных модулей приложения всегда можно увидеть, нажав кнопку [Модули] (Modules) в окне *База данных* (Database), в котором обычно и выполняется работа с этими модулями. Если выделить в окне базы данных Борей (Northwind) модуль *Служебные функции* (Utility Functions) и нажать кнопку [Конструктор] (Design), то откроется окно редактора кода VBA, в котором можно увидеть содержание этого модуля, состоящего из двух строк описания и одной процедуры — функции IsLoaded (рис. 3.6). Это пример универсальной функции, проверяющей, заг-

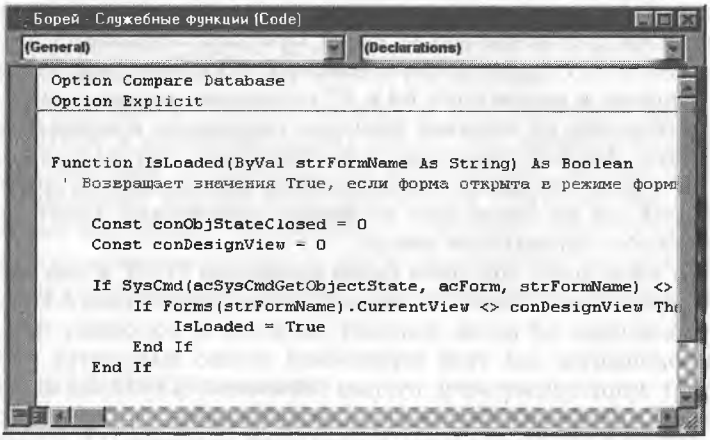


Рис 3.6. Функция IsLoaded

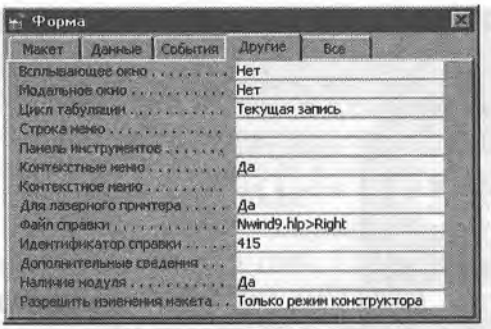


Рис. 3.7. Диалоговое окно свойств формы

ружена ли форма, имя которой передается ей в качестве аргумента. Приведенная функция никак не связана не только с объектом, но и с данным приложением и может использоваться как в нем самом, так и в любом другом приложении.

Модуль класса отличается от стандартного модуля тем, что кроме процедур он содержит описание объекта и используется для создания объектов. Процедуры, определенные в этом модуле, являются методами и свойствами объекта. Примерами модулей класса являются модули форм и отчетов.

Модули форм и отчетов связаны с конкретными формой и отчетом и содержат процедуры обработки событий для этих формы и отчета. Модуль формы не создается сразу при создании новой формы. Он создается и связывается с формой, как только производится попытка создать первую процедуру обработки событий для этой формы или одного из элементов управления формы, либо нажимается кнопка [Программа] (Code) в окне конструктора формы. Чтобы убедиться в этом, можно открыть любую форму приложения Борей (Northwind) в режиме конструктора и посмотреть ее свойства. На вкладке *Другие* (Others) есть свойство *Наличие модуля* (Has Module), которое должно иметь значение *Нет* (No). После нажатия кнопки [Программа] (Code), которая служит для открывания редактора кода VBA, это свойство будет иметь значение *Да* (Yes) (рис. 3.7).

В окне редактора кода VBA отображается объект *Форма* (Form), а справа от него — поле со списком событий, с которыми могут быть связаны процедуры VBA (рис. 3.8).

Если в модуле для некоторого события существует процедура, то это событие выделяется в списке жирным шрифтом. В данном случае таких событий нет, так как обработка событий в форме *Клиенты* (Customers), которую мы открыли, выполняется с помощью макросов. При попытке открытия редактора VBA автоматически создается модуль формы, который будет иметь название Form_Клиенты, и в нем процедура обработки события Load.

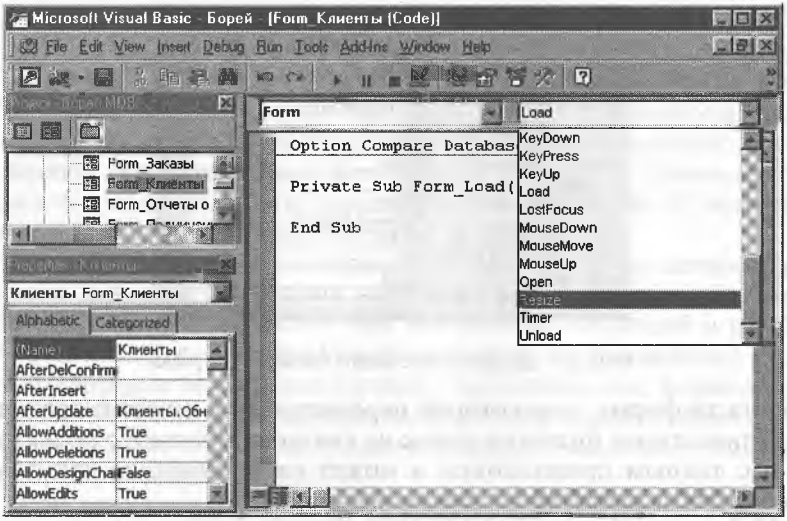


Рис. 3.8. Окно редактора кода VBA

Открыть форму из программы VBA и изменить какие-то свойства этой формы или свойства ее элементов управления можно двумя методами.

1. Использование макрокоманды OpenForm, как метода объекта DoCmd, например:

```
DoCmd.OpenForm "Товары"
Forms!Товары.RecordSource = "Товары клиента"
```

2. Использование ссылки на соответствующий модуль класса, например:

```
Form_Товары.Visible = True
Form_Товары.RecordSource = "Товары клиента"
```

В обоих случаях при выполнении программы открывается стандартный экземпляр формы *Товары* (Products) и подменяется источник записей для этой формы.

Формы и отчеты являются стандартными классами объектов в Access, однако можно использовать модули класса и для создания специальных объектов. В этом случае имя, под которым сохраняется модуль класса, становится именем специального объекта. Процедуры типа Sub и Function, определенные в модуле класса, при этом станут методами объекта, а процедуры типа Property Let, Property Get и Property Set — свойствами объекта. Для описания метода, не возвращающего никакое значение, используется про-

цедура Sub, а для описания метода, возвращающего некоторое значение, — процедура Function. Процедура Property Get возвращает значение свойства объекта. Процедура Property Set присваивает значение свойству объекта. Процедура Property Let устанавливает значение свойства, не принадлежащего объекту.

Для создания стандартного модуля или модуля класса необходимо выполнить следующие действия:

- выбрать команду *Модуль* (Module) или *Модуль класса* (Class Module) в меню *Вставка* (Insert) (рис. 3.9). При этом откроется редактор кода VBA с пустым окном модуля;
- создать необходимые процедуры и описания;
- сохранить модуль, нажав кнопку [Сохранить] (Save) на панели инструментов. При этом откроется диалоговое окно *Сохранение* (Save), в которое нужно ввести имя нового модуля и нажать кнопку [OK].

После этого новый модуль появится в списке модулей окна базы данных. Чтобы его открыть, можно нажать кнопку [Конструктор] (Design) окна базы данных или в режиме конструктора формы или отчета нажать кнопку [Программа] (Code) на панели инструментов.

Окно Object Browser позволяет просматривать все объекты, их свойства и методы, доступные для текущего проекта. Эти объекты могут быть встроенными объектами Access или VBA, а также объектами, которые созданы в приложении или входят во внешние биб-

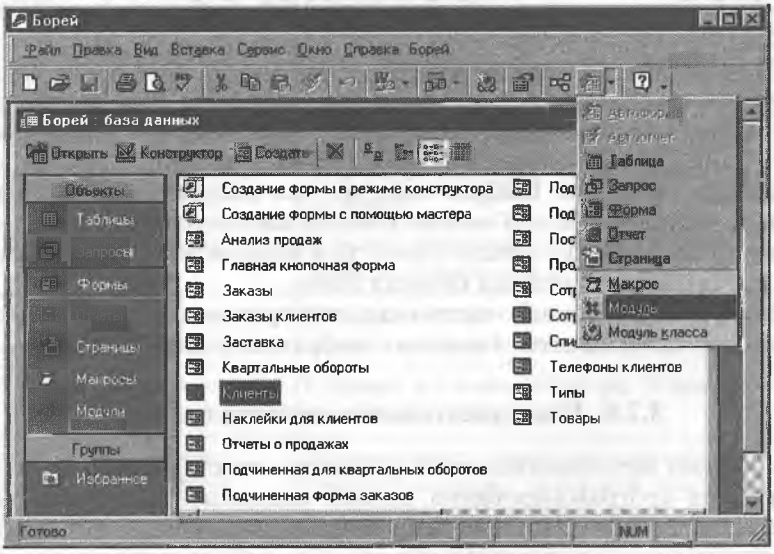


Рис. 3.9. Создание нового модуля

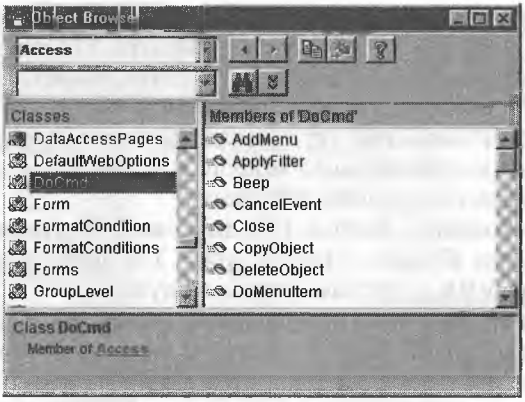


Рис. 3.10. Окно просмотра объектов

лиотеки, на которые имеются ссылки в текущем проекте. Вызвать окно просмотра объектов можно еще тремя способами:

- нажать клавишу [F2];
- нажать кнопку [Object Browser] на панели инструментов;
- выбрать команду View, Object Browser.

Окно Object Browser состоит из нескольких списков (рис. 3.10), которые обеспечивают трехуровневое представление информации.

Список Проект/библиотека (Project/Library) в левом верхнем углу окна содержит перечень всех библиотек и проектов, на которые имеются ссылки в данном проекте. Как минимум он включает в себя библиотеку Access, библиотеку VBA, библиотеку текущего проекта.

При выборе из списка одной из библиотек в нижнем левом поле Classes отображается список следующего уровня — перечень всех объектов, входящих в эту библиотеку. Например, если выбрать библиотеку Access, то в списке Classes можно увидеть много знакомых объектов. Выбрав один из них (например, DoCmd), в правом поле Members of можно увидеть все методы этого объекта. Если бы мы выбрали объект Form, то в правом поле отобразились бы все свойства и методы объекта Form.

При этом в нижней части окна, которая называется *Область описания*, отображается описание выбранного элемента.

3.2.6. Инструментальные средства отладки

Помимо интеллектуального редактора текста профессиональная среда программирования должна содержать инструментальные средства отладки, которые призваны дать разработчику максимально ясное представление о том, как работает его программа. И уже искусство разработчика позволит, используя все имеющи-

еся в его распоряжении средства, быстро выявить ошибки. Набор средств отладки в Access широк. Это и специальное меню *Откладка* (Debug), и во многом дублирующие его кнопки на панели инструментов, и специальные окна отладки.

В табл. 3.3 представлено описание команд отладки, а на рис. 3.11 показаны меню Отладка (Debug) и специальная панель инструментов Debug.

Таблица 3.3

Назначение команд отладки

Команда	Назначение
Компиляция (Compile)	Компилирует все модули в текущей базе данных
Шаг с заходом (Step Into)	Исполняет очередную строку кода с заходом в процедуры
Шаг с обходом (Step Over)	Выполняет остаток текущей процедуры и останавливается в вызывающей программе на следующей строке после вызова этой процедуры
Шаг с выходом (Step Out)	Выполняет остаток текущей процедуры и останавливается в вызывающей программе на следующей строке после вызова этой процедуры
Запуск до курсора (Run to Cursor)	Выполняет все строки кода от текущей строки до строки, в которой установлен курсор, и останавливает выполнение перед этой строкой
Добавление контрольного значения (Add Watch)	Открывает окно *Добавление контрольного значения*
Изменение контрольного значения (Edit Watch)	Открывает окно *Изменение контрольного значения*
Быстрый просмотр (Quick Watch)	Выводит в специальном окне текущее значение выражения в точке останова
Установка/сброс точек останова (Toggle Breakpoint)	Устанавливает/снимает точку останова на строку, в которой находится курсор
Сброс всех точек останова (Clear All Breakpoints)	Снимает все точки останова, установленные в данном модуле
Установка следующего предложения (Set Next Statement)	Устанавливает очередную выполняемую команду на строку, в которой находится курсор
Показ следующего предложения (Show Next Statement)	Отображает в окне редактора очередную команду для выполнения

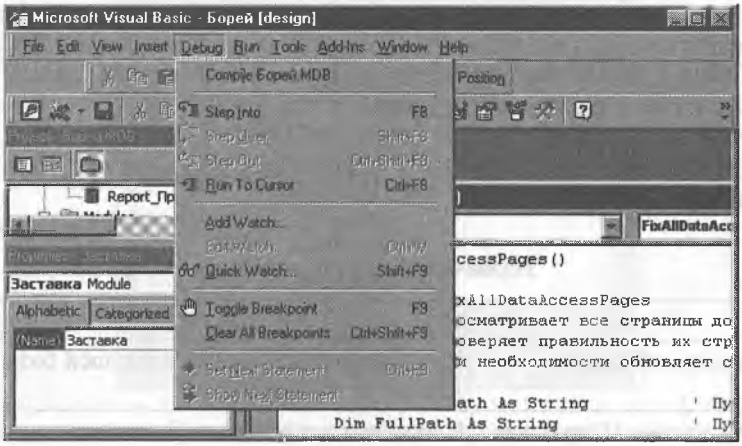


Рис. 3.11. Меню и панель инструментов Debug

3.2.7. Управляющие конструкции языка VBA

Как и во всех других языках программирования, в VBA имеются различные управляющие конструкции, позволяющие изменять порядок выполнения программы. Без использования управляющих конструкций будет происходить последовательное выполнение операторов языка программирования от первого до последнего. Хотя в некоторых самых простых случаях этого и бывает достаточно, однако обычно все-таки требуется изменять порядок выполнения операторов, либо пропуская выполнение некоторых из них, либо, наоборот, многократно повторяя. Оказывается, для реализации любых алгоритмов достаточно иметь только два вида инструкций управления: ветвления и циклы.

Ветвления. Управляющие конструкции ветвления позволяют проверить некоторое условие, а затем в зависимости от результатов этой проверки выполнить ту или иную группу операторов. Для организации ветвлений в VBA используются различные формы оператора ветвления If и оператор выбора Select Case.

Простейшая, краткая форма оператора If используется сначала для проверки одного условия, а затем в зависимости от результата этой проверки либо для выполнения, либо для пропуска одного оператора или блока из нескольких операторов. Краткая форма оператора ветвления If может иметь как однострочную, так и блочную форму. Запись в одну строку краткой формы If имеет вид

```
If <условие> Then <оператор>
```

В блочной форме краткое ветвление выглядит следующим образом:

```
If <условие> Then
   <оператор1>
   <оператор2>
   …
End If
```

В качестве условия можно использовать логическое выражение, возвращающее значения True (*Истина*) или False (*Ложь*), или любое арифметическое выражение. Если используется арифметическое выражение, то нулевое значение этого выражения эквивалентно логическому значению False, а любое ненулевое выражение — значению True. В том случае, когда условие возвращает значение False, оператор или блок операторов, заключенных между ключевыми словами Then и End If и составляющих тело краткого оператора ветвления, не будет выполняться.

Полная форма оператора If используется в тех случаях, когда имеются два различных блока операторов и по результатам проверки условия нужно выполнить один из них. Такая форма If не может записываться в одну строку и всегда имеет блочную форму записи:

```
If <условие> Then
   <блокОператоров1>
Else
   <блокОператоров2>
End If
```

Если условие истинно, выполняется первый блок операторов, заключенный между ключевыми словами Then и Else; в противном случае выполняется второй блок, заключенный между ключевыми словами Else и End If.

Иногда приходится делать выбор одного действия из целой группы альтернативных действий на основе проверки нескольких различных условий. Для этого можно использовать цепочку операторов ветвления If... Then... ElseIf:

```
If <условие1> Then
   <блокОператоров1>
ElseIf <условие2> Then
   <блокОператоров2>
ElseIf <условие3> Then
   <блокОператоров3>
   …
ElseIf <условиеN> Then
   <блокОператоровN>
Else
   <блокОператоров_Else>
End If
```

Цепочки операторов If... Then... ElseIf обладают большой гибкостью и позволяют решить все проблемы, однако если выбор одной из нескольких возможностей все время основывается на различных значениях одного и того же выражения, гораздо удобнее использовать специально предназначенный для этого оператор выбора Select Case, имеющий следующий синтаксис:

```
Select Case <проверяемоеВыражение>
  Case <списокЗначений1>
       <блокОператоров1>
  Case <списокЗначений2>
       <блокОператоров2>
  Case <списокЗначений3>
       <блокОператоров3>
  …
  Case Else
       <блокОператоров_Else>
End Select
```

Проверяемое выражение вычисляется в начале работы оператора Select Case и может возвращать значение любого типа, например логическое, числовое или строковое.

Список выражений содержит одно или несколько выражений, разделенных запятой. При выполнении оператора проверяется, соответствует ли хотя бы один из элементов этого списка проверяемому выражению. Элементы списка выражений могут иметь одну из следующих форм:

```
<выражение>
```

в этом случае проверяется, совпадает ли значение проверяемого выражения с этим выражением;

```
<выражение1> To <выражение2>
```

в этом случае проверяется, находится ли значение проверяемого выражения в указанном диапазоне значений;

```
Is <логическийОператор> <выражение>
```

в этом случае проверяемое выражение сравнивается с указанным значением с помощью заданного логического оператора (например, условие Is > = 10 считается выполненным, если проверяемое значение не меньше 10).

Если хотя бы один из элементов списка соответствует проверяемому выражению, то выполняется соответствующая группа операторов и на этом выполнение оператора Select Case заканчивается, а остальные списки выражений не проверяются, т. е. в этом

случае отыскивается только первый подходящий элемент списков выражений. Если же ни один из элементов всех этих списков не соответствует значению проверяемого выражения, то выполняются операторы группы Else (если таковая присутствует).

Циклы. В VBA имеется богатый выбор средств организации циклов, которые можно разделить на две основные группы: циклы с условием Do...Loop и циклы с перечислением For...Next.

Циклы типа Do...Loop используются в тех случаях, когда заранее не известно, сколько раз должно повториться выполнение блока операторов, составляющего тело цикла. Такой цикл продолжает свою работу до тех пор, пока не будет выполнено определенное условие. Существуют четыре вида циклов Do...Loop, которые различаются типом проверяемого условия и временем выполнения этой проверки. Синтаксисы этих четырех конструкций приведены в табл. 3.4.

Таблица 3.4

Синтаксисы управляющих конструкций

Синтаксис конструкции	Описание
Do While <условие> <блокОператоров> Loop	Условие проверяется до выполнения группы операторов, образующих тело цикла. Цикл продолжает свою работу, пока это условие выполняется (т. е. имеет значение True), иными словами, в этой конструкции указывается условие продолжения работы цикла
Do <блокОператоров> Loop While <условие>	Условие проверяется после выполнения хотя бы один раз операторов, составляющих тело цикла. Цикл продолжает свою работу, пока это условие остается истинным, иными словами, в этой конструкции указывается условие продолжения работы цикла
Do Until <условие> <блокОператоров> Loop	Условие проверяется до выполнения группы операторов, образующих тело цикла. Цикл продолжает свою работу, если это условие еще не выполнено, и прекращает работу, когда оно станет истинным, иными словами, в этой конструкции указывается условие прекращения работы цикла
Do <блокОператоров> Loop Until <условие>	Условие проверяется после выполнения хотя бы один раз операторов, составляющих тело цикла. Цикл продолжает свою работу, если это условие еще не выполнено, а когда оно станет истинным, цикл прекращает работу, иными словами, в этой конструкции указывается условие прекращения работы цикла

В VBA имеется также два вида операторов цикла с перечислением For... Next. Очень часто при обработке массивов, а также в тех случаях, когда требуется повторить выполнение некоторой группы операторов заданное число раз, используется цикл For... Next со счетчиком. В отличие от циклов Do... Loop данный цикл использует специальную переменную, называемую счетчиком, значение которой увеличивается или уменьшается при каждом выполнении тела цикла на заданную величину. Когда значение этой переменной достигает заданного значения, выполнение цикла заканчивается.

Синтаксис такого цикла имеет следующий вид (в квадратные скобки заключены необязательные элементы синтаксической конструкции):

```
For <счетчик> = <начальноеЗначение> To <конечное
Значение>
   [Step <приращение>]
   <блокОператоров>
Next [<счетчик>]
```

Рассмотрим еще один вид цикла For... Next, часто используемый в VBA при обработке объектов, составляющих массив или семейство однородных объектов. В цикле For Each... Next счетчик отсутствует, а тело цикла выполняется для каждого элемента массива или семейства объектов. Синтаксис такого цикла имеет следующий вид:

```
For Each <элемент> In <совокупность>
   <блокОператоров>
Next [<элемент>]
```

Здесь <элемент> — это переменная, используемая для ссылки на элементы семейства объектов; <совокупность> — это имя массива или семейства.

Пример 3.6. Выдача на печать списка всех полей для всех таблиц текущей открытой базы данных:

```
Public Sub EnumerateAllFields()
Dim MyBase As Database
Dim t As TableDef, f As Field
   Set MyBase = DBEngine.Workspaces(0).Databases(0)
   For Each t In MyBase.TableDefs
        Debug.Print "Таблица: " & t.Name
        For Each f In t.Fields
             Debug.Print "Поле: " & f.Name
        Next f
   Next t
   Set MyBase =Nothing
End Sub
```

В операторах Dim переменная MyBase объявлена как объект «База данных DAO», а переменные t и f — как определение таблицы и поле таблицы соответственно. Оператор Set назначает переменной MyBase текущую открытую базу данных. Далее для каждого определения выполняется вывод на печать названия таблицы, а затем вложенный цикл такого же типа печатает названия всех ее полей.

Пример 3.7. Имеется трехмерный числовой массив из 1000 элементов (размером $10 \times 10 \times 10$), который следует заполнить случайными вещественными числами в диапазоне от 0 до 1. Если использовать обычные циклы For... Next со счетчиками (применяя счетчики в качестве индексов элементов массива), то для решения этой задачи потребуется написать три вложенных цикла For... Next:

```
Dim tArray(9, 9, 9) As Single
Dim i%, j%, k%
Randomize
For i=0 To 9
   For j=0 To 9
        For k=0 To 9
            tArray(i, j, k) = Rnd()
        Next k
   Next j
Next i
```

Однако если вместо циклов со счетчиками воспользоваться циклом For Each ... Next, для решения задачи достаточно записать всего один цикл:

```
Dim tArray(9, 9, 9) As Single
Dim elem As Variant
Randomize
For Each elem In tArray
   elem = Rnd()
Next
```

Выход из циклов и процедур. Обычно выполнение процедуры заканчивается после выполнения ее последнего оператора, а выполнение цикла — после нескольких выполнений тела цикла, когда достигнуто условие завершения его работы. Однако в некоторых случаях бывает нужно прекратить выполнение процедуры или цикла досрочно, избежав выполнения лишних операторов процедуры или лишних повторений цикла.

Например, если при выполнении процедуры произошла ошибка, которая делает продолжение ее работы бессмысленным, можно выполнить команду немедленного выхода из процедуры.

Другой пример: если цикл For... Next используется для поиска нужного значения в массиве, то после того как нужный элемент

найден, нет смысла продолжать дальнейший перебор элементов массива.

Досрочный выход из управляющей конструкции можно осуществить с помощью одного из операторов Exit. Для досрочного выхода из циклов Do...Loop используется оператор Exit Do, а для выхода из циклов For — оператор Exit For. Для досрочного выхода из процедур и функций используются соответственно операторы Exit Sub и Exit Function.

3.2.8. Работа с формами, отчетами, запросами, таблицами

Программирование в формах и отчетах, как правило, составляет большую часть кода приложения, так как именно формы и отчеты являются основой интерфейса пользователя, и с помощью программирования этот интерфейс гибко настраивается нужным образом. Основой для программирования в формах и отчетах является множество событий, которые обрабатываются процедурами обработки этих событий.

Помимо процедур обработки событий программирование может использоваться для динамического (т. е. в процессе работы приложения) изменения свойств форм, отчетов и элементов управления.

Обычно события инициируются действиями пользователя. В зависимости от этих действий их можно подразделить на несколько типов: события данных, события фокуса, события клавиатуры, события мыши, события печати, события фильтра, события окна, события ошибок и события таймера.

Перечень основных событий Microsoft Access приведен в Приложении 1.

Для работы с формами, отчетами, запросами и таблицами можно использовать объект AccessObject, который представляет собой конкретный объект MS Access, содержащийся в одной из коллекций, представленных в табл. 3.5.

Свойство IsLoaded показывает, является ли загруженным в данный момент объект AccessObject. Оно может принимать одно из двух значений: True (объект загружен) или False (объект не загружен). Значение этого свойства доступно только для чтения.

Свойство Name задает строковое выражение, которое представляет собой имя объекта AccessObject, элемента управления или объекта Reference. Для таких объектов MS Access, как таблицы, запросы, формы, число символов в имени не должно превышать 64. Для элементов управления длина имени может составлять 255 символов.

Свойство Properties возвращает ссылку на коллекцию AccessObject Properties объектов AccessObject, CurrentProject или

CodeProject. Эта коллекция содержит все свойства перечисленных объектов.

Свойство Type объекта AccessObject возвращает тип объекта MS Access. Константы данного свойства приведены в табл. 3.6.

Таблица 3.5

Коллекции, содержащие объект AccessObject соответствующего типа

Коллекция	Тип объекта	Объект MS Access
AllForms	Form	Формы
AllReports	Report	Отчеты
AllMacros	Macro	Макросы
AllModules	Module	Модули
AllDataAccessPages	Data access page	Страницы доступа к данным
AllTables	Table	Таблицы
AllQueries	Query	Запросы
AllViews	View	Представления
AllStoredProcedures	Stored procedure	Хранимые процедуры
AllDatabaseDiagrams	Database diagram	Схемы данных

Таблица 3.6

Константы, определяющие тип объекта AccessObject

Константа	Описание
acDataAccessPage	Страница доступа к данным
acForm	Форма
acMacro	Макрос
acModule	Модуль
acQuery	Запрос
acReport	Отчет
acServerView	Представление
acStoredProcedure	Хранимая процедура
acTable	Таблица

В структуре объектов MS Access присутствуют объекты Form и Control, которые позволяют работать соответственно с формами и элементами управления (рис. 3.12).

Формы и элементы управления обладают рядом специфичных свойств. Например, формы MS Access имеют оригинальное свойство *Источник записей* (RecordSource), которое определяет, записи какой таблицы или запроса отображаются в форме.

В Access имеется две коллекции, в названии которых содержится слово Form (форма): коллекция Forms и коллекция AllForms. Эти коллекции имеют принципиальное различие: элементами первой из них являются объекты Form, каждый из которых представляет собой открытую форму, а элементами второй — объекты AccessObject. Следовательно, для разных целей используют разные коллекции и содержащиеся в них объекты.

Например, чтобы узнать, сколько всего форм содержится в базе данных, можно воспользоваться свойством Count коллекции AllForms, а для определения числа открытых в данный момент форм можно воспользоваться тем же свойством коллекции Forms.

Аналогично можно поступить в случае, когда требуется узнать, открыта ли какая-либо форма в данный момент, т.е. перебрать все элементы коллекции Forms (с помощью цикла For Each) или воспользоваться свойством IsLoaded объекта AccessObject, являющегося элементом коллекции AllForms.

Для работы с формой или элементом управления, которые в данный момент имеют фокус, удобно использовать объект Screen, т.е., используя свойства объекта Screen, можно сослаться на активную форму, отчет или элемент управления, которые в данный момент имеют фокус.

Получить доступ к элементу управления, который имеет фокус, позволяет свойство ActiveControl.

Пример 3.8. Присвоение подписи кнопке, которая имеет фокус:

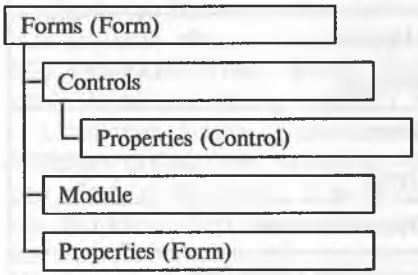


Рис. 3.12. Структура объектов, встроенных в объект Form

```
  Dim ctlCurrControl As
Control
  Set ctlCurrControl =
= Screen.ActiveControl
  ctlCurrControl.Caption =
= "Active Control"
```

Использование свойства ActiveControl в случае, когда ни один из элементов управления не имеет фокуса, приведет к ошибке.

Свойство ActiveForm дает ссылку на объект Form, представляющий собой форму, которая в данный момент имеет фокус. Если фокус имеет подчиненная форма, то данное свойство возвращает ссылку на главную форму.

Пример 3.9. Задание подписи форме, которая имеет фокус:

```
Dim frmCurrForm As Form
Set frmCurrForm = Screen.ActiveForm
frmCurrForm.Caption = "Active Form"
```

Коллекция Forms содержит все формы, открытые в данный момент. Чтобы сослаться на конкретную открытую форму, можно использовать ее имя или индекс в коллекции.

Существуют четыре варианта синтаксиса ссылки на форму (табл. 3.7).

Таким же образом можно ссылаться и на отчеты (естественно, используя вместо коллекции Forms коллекцию Reports).

Объект Form ссылается на конкретную открытую форму. Каждый объект Form является членом коллекции Forms, в которой содержатся все открытые формы.

Формы имеют достаточно много свойств, поэтому рассмотреть все из них не представляется возможным. Для более подробного ознакомления со свойствами (и не только со свойствами) можно воспользоваться справочной системой MS Access.

Для получения справки по тому или иному свойству достаточно перейти в режим конструктора форм, установить курсор в соответствующее поле и нажать клавишу [F1].

Существуют свойства, которые можно изменять на этапе конструирования формы в окне свойств. К некоторым свойствам доступ можно получить только программным способом. Приведенные в работе два варианта написания названий свойств (русский и английский, указанный в скобках) свидетельствуют о том, что

Таблица 3.7

Варианты синтаксиса ссылки на форму

Синтаксис	Описание
Forms!Заказы	Ссылка на форму *Заказы* с помощью оператора !
Forms![Заказы клиентов]	Ссылка на форму *Заказы клиентов*. Квадратные скобки используются в случае наличия в имени формы пробелов
Forms("Заказы клиентов")	Ссылка на элемент коллекции Forms по имени формы
Forms(1)	Ссылка на элемент коллекции Forms по порядковому номеру

это свойство можно изменять как в программе VBA, так и в режиме конструктора формы. Значения этих свойств также имеют два варианта написания: русское в конструкторе форм (в окне свойств) и оригинальное — в программах VBA (указано в скобках).

Несмотря на то что формы имеют большое количество свойств, на практике обычно используется лишь незначительная их часть. В табл. 3.8 представлены свойства, определяющие внешний вид формы, а в табл. 3.9 — ряд наиболее часто применяемых свойств формы.

Таблица 3.8

Свойства, определяющие внешний вид формы

Свойство	Описание
Подпись (Caption)	Определяет название окна. Представляет собой строковое выражение, содержащее не более 2048 символов
Кнопка закрытия (CloseButton)	Определяет, доступна ли кнопка закрытия окна формы: *Да* (True) и *Нет* (False)
Кнопка контекстной справки (WhatsThisButton)	Определяет, доступна ли кнопка контекстной справки: *Да* (True) и *Нет* (False)
Кнопка оконного меню (ControlBox)	Определяет присутствие на форме кнопки, вызывающей оконное меню: *Да* (True) и *Нет* (False)
Кнопки размеров окна (MinMaxBufton)	Определяет, доступны ли кнопки управления окном [Свернуть] (Minimize) и [Развернуть] (Maximize): 0 — *Отсутствуют* (None); 1 — *Свертывание* (Min Enabled); 2 — *Развертывание* (Max Enabled); 3 — *Все* (Both Enabled)
Модальное окно (Modal)	Определяет, является ли окно модальным: *Да* (True) и *Нет* (False)
Всплывающее окно (Popup)	Определяет, открывается ли форма, как всплывающее окно: *Да* (True) и *Нет* (False). Примером всплывающего окна может служить окно свойств, которое всегда находится поверх остальных окон приложения, даже если оно не является активным
Мозаичное заполнение (PictureTiling)	Определяет мозаичное заполнение фоновым рисунком элемента управления или окна формы: *Да* (True) и *Нет* (False). Выравнивание мозаики задается свойством *Выравнивание рисунка* (PictureAlignment)
Разделительные линии (DividingLines)	Определяет вывод на экран линии, разделяющей области формы или записи: *Да* (True) и *Нет* (False)

Окончание табл. 3.8

Свойство	Описание
Рисунок (Picture)	Определяет рисунок, размещенный на форме или элементе управления. Значением данного свойства является строка, представляющая собой путь к файлу
Тип границы (BorderStyle)	Определяет тип границы окна формы путем установки одного из следующих вариантов: 0 — *Отсутствует* (None); 1 — *Тонкая* (Thin); 2 — *Изменяемая* (Sizable); 3 — *Окно диалога* (Dialog). Все перечисленные варианты, кроме второго, не позволяют пользователю изменять размер окна формы
Выравнивание рисунка (PictureAlignment)	Определяет один из вариантов расположения фонового рисунка (или элемента управления *Рисунок* (Image)) на форме: 0 — *Сверху слева* (Top Left); 1 — *Сверху справа* (Top Right); 2 — *По центру* (Center); 3 — *Снизу слева* (Bottom Left); 4 — *Снизу справа* (Bottom Right); 5 — *По центру формы* (Form Center)
Тип рисунка (PictureType)	Определяет тип рисунка: *Внедренный* (Embedded) или *Связанный* (Linked). Внедренный рисунок является частью файла базы данных и не требует наличия на диске файла с изображением. Связанный рисунок требует наличия файла на диске, путь к которому определяет значение свойства *Рисунок* (Picture)
Высота (Height)	Высота формы
Ширина (Width)	Ширина формы

Таблица 3.9

Свойства формы

Свойство	Описание
Режим по умолчанию (DefaultView)	Определяет вид формы при открытии. Значения данного свойства могут быть следующими: 0 — *Простая форма* (Single Form); 1 — *Ленточная форма* (Continuous Forms); 2 — *Таблица* (Datasheet)

Продолжение табл.3.9

Свойство	Описание
Допустимые режимы (ViewsAllowed)	Определяет возможность переключения между режимами формы и таблицы путем выбора одного из значений; 0 — *Все* (All); 1 — *Форма* (Form); 2 — *Таблица* (Table). Указанные варианты могут также быть заданы пользователем с помощью команд *Вид|Режим формы* и *Вид|Режим таблицы*
Ввод данных (DataEntry)	Определяет режим открытия формы, присоединенной к источнику данных только для ввода данных. Если свойство имеет значение *Да* (True), то при открытии формы выводится пустая запись, а в случае значения *Нет* (False) — существующие записи
Вывод на экран (Visible)	Определяет возможность отображения формы на экране: *Да* (True) и *Нет* (False)
Имя (Name)	Определяют имя формы, которое используется для идентификации формы в программах VBA, макрокомандах и т. д. Значение этого свойства не может быть задано в конструкторе форм, но может быть изменено в окне базы данных или в программе VBA (здесь имя формы задается строковым выражением)
Источник записей (RecordSource)	Определяет источник данных формы: таблицу или запрос. В конструкторе формы можно задать значение этого свойства в окне свойств путем выбора из списка. В программе VBA значением данного свойства является строковая переменная, содержащая имя таблицы, запроса или инструкцию SQL
Наличие модуля (HasModule)	Определяет наличие у данной формы модуля путем установки значений: *Да* (True) и *Нет* (False)
Область выделения (RecordSelectors)	Задает возможность вывода на экран области выделения путем присвоения значений *Да* (True) и *Нет* (False)
Панель инструментов (Toolbar)	Определяет панель инструментов, которая будет выводиться при открытии формы или

Продолжение табл.3.9

Свойство	Описание
Панель инструментов (Toolbar)	отчета. Значение данного свойства можно установить в режиме конструктора форм, выбрав панель инструментов из списка. В программе VBA значением данного свойства является строковое выражение, содержащее имя требуемой панели управления
Перехват нажатия клавиш (KeyPreview)	Позволяет реализовать перехват нажатия клавиш клавиатуры путем присвоения значений *Да* (True) и *Нет* (False)
Поле номера записи (NavigationButtons)	Определяет вывод на экран поля номера записи: *Да* (True) или *Нет* (False)
Полосы прокрутки (ScrollBars)	Определяет вид выводимых на экран полос прокрутки формы (или поля): 0 — *Отсутствуют*; 1 — *Только по горизонтали*; 2 — *Только по вертикали*; 3 — *Все*
Применение фильтров (AllowFilters)	Задает возможность применения фильтров в форме: *Да* (True) и *Нет* (False)
Разрешить добавление (AllowAdditions)	Определяет, может ли пользователь добавлять записи в форме: *Да* (True) и *Нет* (False)
Разрешить изменение (AllowEdits)	Устанавливает возможность изменения пользователем записи в форме: *Да* (True) и *Нет* (False)
Разрешить удаление (AllowDeletions)	Задает возможность пользователю удалять записи в форме: *Да* (True) и *Нет* (False)
Строка меню (MenuBar)	Определяет строку меню, используемую в MS Access при открытии данной формы. Значением данного свойства является строка, определяющая имя используемой строки меню
Контекстное меню (SnortcutMenu)	Определяет возможность отображения контекстного меню при щелчке на форме правой кнопкой мыши. Если значение данного свойства равно *Да* (True), то контекстное меню выводится, а если *Нет* (False) — не выводится
Контекстное меню (ShortcutMenuBar)	Определяет контекстное меню, открываемое при щелчке правой кнопкой мыши на форме или элементе управления. Значени-

Окончание табл.3.9

Свойство	Описание
Контекстное меню (ShortcutMenuBar)	ем данного свойства является строка, представляющая собой имя контекстного меню
Тип набора записей (RecordSetType)	Определяет тип набора записей формы: 0 — *Динамический набор* (Dynaset); 1 — *Динамический набор* ((Dynaset) (Inconsistent Updates)); 2 — *Статический набор* (Snapshot). Первые два типа набора записей позволяют изменять данные в полях, присоединенных к элементам управления, а последний не позволяет
Фильтр (Filter)	Определяет фильтр, используемый в форме. Его значением является строка, представляющая собой фильтр
Цикл табуляции (Cycle)	Определяет действия, выполняемые при нажатии клавиши [Tab]: 0 — *Все записи* (All Records); 1 — *Текущая запись* (Current Record); 2 — *Текущая страница* (Current Page)

Методы форм

Формы имеют несколько методов. Приведем их названия и рассмотрим назначение.

1. Метод DefaultControl возвращает объект, предназначенный для установки свойства по умолчанию заданному типу элементов управления на форме или отчете. Например, перед созданием текстового поля на форме этому полю можно установить свойства, задаваемые по умолчанию.

2. Метод Recalc немедленно обновляет все вычисляемые элементы управления формы. Данный метод применяют, если были изменены значения в полях, от которых зависят значения вычисляемых элементов управления.

3. Метод Refresh немедленно обновляет записи источника данных формы, чтобы отобразить сделанные пользователями изменения в многопользовательской среде. Однако этот метод не выполняет отбор записей заново, т.е. не формирует повторный запрос к базе данных. При этом отображаются только изменения, внесенные в текущий набор записей и не отображаются данные, добавленные или удаленные из базы данных после последнего обновления набора записей. Для выполнения нового отбора записей следует использовать метод Requery.

4. Метод Repaint завершает все отложенные обновления экрана для указанной формы. Одновременно он завершает отложенные операции перерасчета элементов управления формы. Данный метод применяется при изменении значений нескольких полей в том случае, когда эти значения влияют на значения других полей, что важно, так как в окне MS Access изменения могут отображаться не сразу.

Следует обратить внимание на разницу между методами Repaint и Refresh. Метод Repaint не обновляет данные, отображаемые на экране, он обновляет только изображение на экране.

5. Метод Requery обновляет данные, служащие источником данных формы или элемента управления на активной форме. Данные обновляются путем повторного запроса к источнику данных формы или элемента управления. Таким образом, этот метод позволяет отображать текущие данные в форме или элементе управления.

Использование метода Requery предполагает выполнение одного из следующих действий:

повторное выполнение запроса формы или элемента управления;

вывод всех добавленных или измененных записей и уборка записей, удаленных из базовой таблицы формы или элемента управления;

обновление выводимых в форме или элементе управления записей в соответствии с изменением свойства *Фильтр* (Filter) формы.

Если фокус имеет подчиненная форма, то обновляется только ее источник данных, но не источник данных основной формы.

6. Метод SetFocus устанавливает фокус на указанные форму или элемент управления активной формы. Данный метод используется, например, в том случае, если необходимо узнать значение свойства Text элемента управления *Поле* (TextBox), так как поле должно иметь фокус.

Фокус можно перевести только на видимый и доступный элемент управления или видимую форму.

7. Метод Undo восстанавливает измененные значения элемента управления или формы. Этот метод можно использовать, например, в случае, когда запись имеет поля, в которые введены недопустимые значения:

```
object.Undo
```

Применение данного метода к элементу управления приводит к потере изменений в этом элементе. В случае применения этого метода к форме будут потеряны изменения в текущей записи.

Применение метода Undo имеет смысл только до обновления элемента управления или формы. Обычно его используют в про-

цедурах обработки событий *До обновления* (BeforeUpdate) для формы и *Изменение* (Change) для элемента управления.

Пример 3.10. Вариант создания новой формы и определение ее свойств:

```
Sub NewForm()
   Dim frm As Form
   'Создает новую форму
   Set frm = CreateForm
   'Задает значения свойств формы
   With frm
        .RecordSource = "Товары"
        .Caption = "Форма Товары"
        .ScrollBars = 0
        .NavigationButtons = True
   End With
   'Восстанавливает окно формы
   DoCmd.Restore
End Sub
```

Коллекция AllForms содержит объекты AccessObject для каждой формы в объектах CurrentProject или CodeProject. Таким образом данная коллекция содержит все формы, содержащиеся в базе данных.

Работа с элементами управления

Элементы управления в MS Access имеют некоторые характерные особенности. У большинства элементов управления таких, как *Поле* (TextBox), *Поле со списком* (ComboBox), *Переключатель* (RadioButton), *Флажок* (CheckBox), есть свойство *Данные* (ControlSource), которое определяет, какие данные выводятся в элементе управления, т.е. при изменении или введении данных в поле изменятся и данные в соответствующей таблице базы данных.

Для работы с элементами управления следует использовать коллекцию Controls и объект Control

Коллекция Controls содержит все элементы управления формы, отчета, раздела формы или отчета, а также элементы управления, расположенные на другом элементе управления (например, элемент управления *Набор страниц* (MultiPages) может иметь такую коллекцию) или присоединенные к другому элементу управления. Таким образом, коллекция Controls встроена в объекты Form, Report, Section и Control.

Для работы с элементами управления раздела формы или отчета можно использовать свойство Section, которое позволяет получить ссылку на нужный объект Section, а потом уже сослать-

ся на конкретный элемент управления, содержащийся в коллекции Controls данного раздела.

Чтобы сослаться на конкретный элемент управления, можно использовать следующую инструкцию:

```
Forms!Form.Controls!Control
```

Здесь Control — имя элемента управления, на который необходимо сослаться.

Имена элементов управления в формах принято начинать с последовательности символов ctl.

Коллекция Controls не имеет методов, позволяющих добавить элемент управления или удалить его. Чтобы добавить элемент управления, можно воспользоваться функцией CreateControl, которая создает его в указанной форме.

Объект Control представляет собой элемент управления в форме или разделе формы, присоединенный к другому элементу управления или находящийся внутри него. Все объекты Control являются элементами коллекции Controls соответственно объектов Form и Report.

Элементы управления раздела являются элементами коллекции Controls этого раздела. Элементы управления, находящиеся внутри группы переключателей, принадлежат коллекции Controls этой группы. Элемент управления *Надпись* (Label), присоединенный к элементу управления, принадлежит коллекции Controls этого элемента управления.

Каждый тип объекта Control обозначается одной из встроенных констант. Например, кнопка обозначается константой acCommandButton. Константы, определяющие тип элемента управления, представлены в табл. 3.10.

Свойство ControlType определяет тип элемента управления, представляемого объектом Control. Значениями данного свойства являются константы, представленные в табл. 3.10.

Свойство Column определяет конкретный столбец или комбинацию столбца и строки элемента управления (поля со списком или списка). Синтаксис свойства Column имеет следующий вид:

```
Control.Column(column, row)
```

Элементы синтаксиса свойства Column представлены в табл. 3.11.

Это свойство может использоваться, например, для присвоения значения указанного элемента списка значению поля или другого элемента управления.

Свойство Controls возвращает ссылку на коллекцию Controls формы, подчиненной формы или раздела и обычно используется, когда необходимо сослаться на один из их элементов управле-

Таблица 3.10

Константы, определяющие тип элемента управления

Константа	Элемент управления
acBoundObjectFrame	*Присоединенная рамка объекта* (Bound Object Frame)
acCheckBox	*Флажок* (Check Box)
acComboBox	*Поле со списком* (Combo Box)
acCommandButton	*Кнопка* (Command Button)
acCustomControl	*Элемент управления ActiveX* (Custom Control)
acImage	*Рисунок* (Image)
acLabel	*Надпись* (Label)
acLine	*Линия* (Line)
acListBox	*Список* (ListBox)
acObjectFrame	*Свободная рамка объекта или диаграмма* (Object Frame)
acOptionButton	*Переключатель* (Option Button)
acOptionGroup	*Группа переключателей* (Option Group)
acPage	*Страница* (Page)
acPageBreak	*Разрыв страницы* (Page Break)
acRectangle	*Прямоугольник* (Rectangle)
acSubform	*Подчиненная форма или отчет* (Subform)
acTabCtl	*Набор вкладок* (Tab)
acTextBox	*Текстовое поле* (Text Box)
acToggleButton	*Выключатель* (Toggle Button)

Таблица 3.11

Элементы синтаксиса свойства Column

Элемент	Описание
Control	Обязательный элемент. Объект, представляющий собой *Активный список* или *Поле со списком*
Column	Обязательный элемент. Целое число, указывающее столбец в диапазоне от 0 до значения свойства минус 1
Row	Необязательный элемент. Целое число, указывающее строку в диапазоне от 0 до значения свойства ListCount минус 1

ния. Например, чтобы узнать число элементов управления формы *Клиенты*, можно использовать следующую запись:

```
Forms("Клиенты").Controls.Count
```

Свойство Form возвращает ссылку на форму или форму, ассоциированную с подчиненной формой. Обычно это свойство используется, чтобы сослаться на форму, содержащую подчиненную форму. Приведем пример ссылки на элемент управления *Код товара* подчиненной формы *Заказы* формы *Клиенты*:

```
Forms!Клиенты!Заказы. Form![Код товара]
```

Свойство ItemData возвращает значение, содержащееся в присоединенном столбце указанной строки элемента управления *Список* (ListBox) или *Поле со списком* (ComboBox). Синтаксис свойства ItemData имеет следующий вид:

```
control.ItemData(rowindex)
```

Элементы синтаксиса свойства ItemData представлены в табл. 3.12.

Свойство ItemsSelected возвращает ссылку на семейство ItemsSelected, которое содержит в отличие от других семейств не объекты, а значения типа Variant. Эти значения представляют собой целочисленные индексы, указывающие положение выделенной строки в списке или в поле со списком.

Данное свойство можно использовать вместе со свойствами Column или ItemData для получения данных из выделенных строк списка.

Таблица 3.12

Элементы синтаксиса свойства ItemData

Элемент	Описание
Control	Обязательный элемент. Объект, представляющий собой *Список* (ListBox) или *Поле со списком* (ComboBox)
Rowindex	Обязательный элемент. Целое число в диапазоне от 0 до значения свойства ListCount минус 1, определяющего строку, из которой вы хотите получить значение

Свойство OldValue содержит неизмененные данные, которые имел присоединенный элемент управления до начала редактирования.

Пример 3.11. Выполнение отмены всех изменений для всех элементов управления в форме после нажатия кнопки [Отмена]:

```
Sub Отмена_Click()
Dim ctl As Control
   For Each ctl in Me.Controls
        ctl.Value = ctl.OldValue
   Next ctl
End Sub
```

После перемещения на следующую запись происходит обновление источника записей, после чего текущее значение поля и значение свойства OldValue становятся одинаковыми.

Свойство Properties возвращает ссылку на форму или коллекцию Properties элемента управления. Коллекция Properties содержит все свойства, связанные с формой или элементом управления. Для ссылки на конкретный элемент коллекции можно использовать индекс или имя объекта.

Элементы управления так же, как и формы, имеют большое количество свойств. Рассмотрим только наиболее распространенные из них.

Существует набор свойств, которыми обладают практически все элементы управления (табл. 3.13).

Таблица 3.13

Свойства, общие для всех элементов управления

Свойство	Описание
Имя (Nome)	Имя элемента управления, которое используется для его идентификации в программах VBA, макрокомандах и т. д.
От левого края (Left)	Определяет положение левого края элемента управления относительно раздела формы или отчета. Значения данного свойства задаются в режиме конструктора форм в окне свойств или в программе VBA с помощью числовых выражений
От верхнего края (Left)	Определяет положение верхнего края элемента управления относительно раздела формы или отчета. Значения данного свойства задаются в режиме конструктора форм в окне свойств или в программе VBA с помощью числовых выражений
Ширина границы (*BorderWidth*)	Определяет ширину границы элемента управления. Данное свойство может принимать следующие значения: 0 — *Сверхтонкая* (Hairline); 1 — *1 пункт* (1 pt); 2 — *2 пункта* (2 pt); 3 — *3 пункта* (3 pt);

Продолжение табл. 3.13

Свойство	Описание
Ширина границы (*BorderWidth*)	4 — *4 пункта* (4 pt); 5 — *5 пунктов* (5 pt); 6 — *6 пунктов* (6 pt)
Тип границы (BorderStyle)	Определяет тип границы элемента управления. Данное свойство может принимать следующие значения: 0 — *Отсутствует* (Transparent); 1 — *Сплошная* (Solid); 2 — *Штриховая* (Dashes); 3 — *Пунктирная* (Short dashes); 4 — *Точечная* (Dots); 5 — *Редкоточечная* (Sparse dots), *Штрих-пунктирная* (Dash dot); 6 — *Штрихточечная* (Dash dot dot)
Оформление (SpecialEffect)	Определяет способ объемного представления элемента управления путем установки одного из следующих значений: 0 — *Обычное* (Flat); 1 — *Приподнятое* (Raised); 2 — *Утопленное* (Sunken); 3 — *Вдавленное* (Etched); 4 — *С тенью* (Shadowed); 5 — *Рельефное* (Chiseled). Внешний вид различных вариантов оформления можно посмотреть в конструкторе форм с помощью кнопки [Оформление] (Special Effect) на панели инструментов *Формат* (Formatting)
Тип фона (BackStyle)	Задает цвет фона элемента управления видимым или прозрачным путем установки следующих значений: 0 — *Обычный* (Normal); 1 — *Прозрачный* (Transparent). Видимым является цвет формы, на которой расположен элемент управления
Цвет фона (BackColor)	Определяет цвет фона элемента управления. Значение данного свойства задается в конструкторе форм с помощью кнопки [Цвет заливки/фона] (Fill/Back Color) на панели инструментов *Форматирование* (Formatting)
Всплывающая подсказка (ControlTipText)	Определяет текст всплывающей подсказки, выводимой на экран, когда указатель мыши задерживается над элементом управления. Значение данного свойства представляет собой строку длиной не более 255 символов

Окончание табл. 3.13

Свойство	Описание
Текст строки состояния (StatusBarText)	Определяет текст, выводимый в строке состояния, когда элемент управления имеет фокус. Значение данного свойства представляет собой строку длиной не более 255 символов
Доступ (Enabled)	Определяет возможность получения фокуса элементом управления путем установки атрибутов *Да* (True) и *Нет* (False)
Вывод на экран (Visible)	Определяет возможность отображения на экране элемента управления с помощью установки значений *Да* (True) или *Нет* (False)
Блокировка (Locked)	Определяет возможность изменения данных в элементе управления путем установки атрибутов *Да* (True) и *Нет* (False)
Переход по Tab (TabStop)	Определяет, можно ли перевести фокус на элемент управления с помощью клавиши [Tab]: *Да* (True) и *Нет* (False)
Подпись (Caption)	Определяет текст, выводимый на кнопке, и надписи. Значением является строка длиной не более 2048 символов
Индекс перехода по Tab (TabIndex)	Определяет позицию элемента управления в последовательности перехода фокуса в форме при нажатии клавиши [Tab]. Значением данного свойства являются целые числа, лежащие в диапазоне от 0 (для первого элемента управления) до общего числа элементов управления формы минус 1

Кроме общих для всех элементов управления свойств существуют некоторые характерные свойства полей (табл. 3.14).

Таблица 3.14

Характерные свойства полей

Свойство	Описание
Формат поля (Format)	Определяет формат, в котором выводятся данные в поле в режиме формы и на печать. Допустимые значения (форматы) зависят от типа присоединенного поля таблицы, используемой в качестве источника данных
Значение по умолчанию (DefaultValue)	Этому свойству может задаваться значение, которое будет вводиться в поле при создании новой записи и должно находиться в диапазоне допустимых зна-

Окончание табл. 3.14

Свойство	Описание
Значение по умолчанию (DefaultValue)	чений для каждого типа данных. Может использоваться, например, для ввода в поле текущей даты с помощью функции Now. Диапазон допустимых значений зависит от типа присоединенного поля таблицы
Число десятичных знаков (DecimalPlaces)	Определяет число десятичных знаков, отображаемых в поле. Значение данного свойства может быть равно *Авто* (Auto), т. е. 255, или находиться в диапазоне от 1 до 15. Если значение равно *Авто* (Auto), то число знаков после запятой определяется значением свойства *Формат поля* (Format). В противном случае число десятичных знаков находится в диапазоне от 1 до 15

Свойства, общие для всех элементов управления, описанные в табл. 3.13, присущи также и полям со списком. Кроме того, поля со списком обладают набором дополнительных свойств, представленных в табл. 3.15.

Таблица 3.15

Свойства полей со списком

Свойство	Описание
Выделение (Selected)	Определяет, выделен ли конкретный элемент списка путем назначения данному свойству значений *Да* (True) или *Нет* (False). Все значения для каждого элемента списка хранятся в массиве с индексами, отсчитываемыми от 0. Если элемент списка выделен, то значение свойства Selected равняется *Да* (True). Данное свойство доступно для чтения и записи. Если свойство *Несвязное выделение* (MultiSelect) имеет значение *Простой* (Simple), то выделенным может быть только один элемент списка. При установленном значении свойства *Несвязное выделение* (MultiSelect) выделенными могут быть все элементы списка. Например, пятый элемент списка *Выбор товара* активной формы выделяется фразой Me![Выбор товара].Selected(4) = True
Несвязное выделение (MultiSelect)	Определяет способ одновременного выделения нескольких элементов списка. Данное свойство может принимать сделующее значения: 0 — *Отсутствует* (None); 1 — *Простой* (Simple); 2 — *Со связным выбором* (Extended).

Продолжение табл. 3.15

Свойство	Описание
Несвязное выделение (MultiSelect)	При значении данного свойства, равном нулю, одновременное выделение нескольких элементов списка невозможно. При значении, равном единице, выделение нескольких элементов производится при выборе элементов списка с помощью мыши или нажатием клавиши [Пробел]. При значении, равном двум, допускается одновременное выделение нескольких элементов списка, расположенных друг за другом, нажатием кнопки мыши или клавиш перемещения курсора при удерживаемой клавише [Shift]. Отдельный элемент списка выделяется, а также выделение снимается нажатием кнопки мыши при нажатой клавише [Ctrl], т.е. выделение в списке производится аналогично выделению файлов и папок в правой части *Проводника*
Количество столбцов (ColumnCount)	Определяет число столбцов списка, отображаемых на экране. Если источником строк для списка являются таблица или запрос, то поля источника данных отображаются в направлении справа налево. Максимальное число столбцов определяется числом полей в таблице или запросе
Заглавия столбцов (ColumnHeads)	Определяет, выводятся ли заголовки столбцов списка (поля со списком) путем задания значений *Да* (True) или *Нет* (False). Заглавия столбцов выводятся как их первые строчки. Содержимое заглавия зависит от значения свойства *Тип* (Type) источника строк. Если в качестве источника строк используются таблица или запрос, то заголовки столбцов определяются как имена соответствующих столбцов
Ширина списка (ListWidth)	Определяет ширину списка. Значение задается в сантиметрах
Количество строк (LislRows)	Определяет число строк списка. Значение данного свойства находится в диапазоне от 1 до 255
Цвет текста (ForeColor)	Определяет цвет шрифта списка. Значение данного свойства представляет собой числовое выражение, которое можно задать в окне свойств с помощью диалогового окна *Цвет* (Color Builder), вызываемого кнопкой [Построить] (Build)
Цвет фона (BackColor)	Определяет цвет фона списка. Значение данного свойства представляет собой числовое выражение

Окончание табл. 3.15

Свойство	Описание
Ограничиться списком (LimitToList)	Определяет вывод в поле со списком только значений, содержащихся в списке, путем установки значений *Да* (True) или *Нет* (False)
Шрифт (FontName)	Определяет шрифт текста в списке. Значением данного свойства является строка, представляющая собой имя шрифта
Размер шрифта (FontSize)	Определяет размер шрифта в списке. Значения данного свойства представляют собой целые числа в диапазоне от 1 до 127
Курсив (FontItalic)	Позволяет задать наклонный стиль шрифта элемента управления путем установки значений *Да* (True) или *Нет* (False)
Подчеркнутый (FontUnderline)	Позволяет задать подчеркнутый стиль шрифта элемента управления путем установки значений *Да* (True) или *Нет* (False)
Насыщенность (FontWeight)	Определяет ширину линии, используемой для отображения символов элемента управления на экране и при печати, посредством задания следующих вариантов свойства: 100 — *Тонкий* (Thin); 200 — *Сверхсветлый* (ExtraLight); 300 — *Светлый* (Light); 400 — *Обычный* (Normal); 500 — *Средний* (Medium); 600 — *Плотный* (Semi-Bold); 700 — *Полужирный* (Bold); 800 — *Жирный* (ExtraBold); 900 — *Сверхжирный* (Heavy)

Кнопка (Button) помимо свойств, присущих всем элементам управления, имеет и некоторые специфичные свойства, представленные в табл. 3.16.

Таблица 3.16

Свойства кнопки

Свойство	Описание
Прозрачный (Transparent)	Определяет, является кнопка видимой *Нет* (False) или прозрачной *Да* (True)
Автоматический повтор (AutoRepeat)	Определяет путем установки значений *Да* (True) или *Нет* (False), выполняются ли повторно процедура обработки события или макрос, пока кнопка остается нажатой

Объект Control имеет пять методов: Dropdown, Requery, SetFocus, SizeToFit, Undo.

Некоторые из них были рассмотрены при описании методов форм, теперь рассмотрим остальные.

Метод Dropdown раскрывает содержание поля со списком. Например, этот метод можно использовать для раскрытия списка в момент получения фокуса элементом управления.

Если при вызове данного метода элемент управления не имеет фокуса, произойдет ошибка.

Метод SizeToFit задает размер элемента управления в соответствии с размерами текста или изображения, которые содержатся в этом элементе управления. Данный метод может применяться только в режиме конструктора форм. Его также можно использовать в процессе создания в программе кнопок для определения их размеров.

Этот метод воздействует не на все элементы управления. Присоединенные элементы управления могут изменять свой размер при переходе от записи к записи. Это справедливо для полей, списков, полей со списками и присоединенных рамок объектов.

Пример 3.12. Создадим форму *Товары* (рис. 3.13) с отбором данных на основе процедур обработки событий изменения полей со списками.

При вводе данных пользователь будет сначала выбирать категорию товара из одного списка, а потом собственно товар из другого списка. Причем можно выбрать режим для ввода новых товаров или для изменения информации о уже существующих товарах.

Сначала создадим базу данных, состоящую из двух таблиц, связанных через схему данных (рис. 3.14), а затем с помощью мастера, выбрав в качестве источника записей таблицу ТОВАРЫ (*Код товара*, Категория, Наименование, Цена) и все поля этой таблицы, создадим форму.

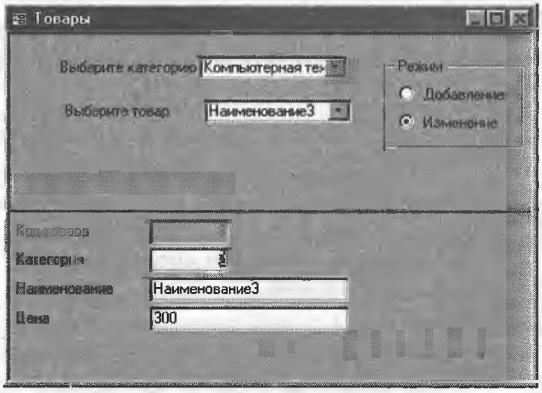


Рис. 3.13. Форма *Товары*

Рис. 3.14. Схема данных базы

После этого будем работать с формой в режиме конструктора форм и в первую очередь зададим в окне свойств этой формы значения некоторых свойств в соответствии с табл. 3.17.

Таблица 3.17

Значения свойств формы Товары

Свойство	Значение
Область выделения (Record Selectors)	*Нет* (False)
Кнопки перехода (Navigation Button)	*Нет* (False)
Режим по умолчанию (Default View)	*Простая форма* (Single Form)
Полосы прокрутки (Scroll Bars)	*Отсутствуют* (Neither)
Фильтр	Код Товара = Forms!Товары![Выбор Товара]
Применение фильтров	*Да* (True)

В заголовок формы добавим два поля со списком: *Выбор категории товара* и *Выбор товара*, а также преобразуем поля *Код товара* и *Код категории* товара в соответствующие поля со списком.

В данном случае удобно создать поле со списком *Выбор категории товара* с помощью мастера. При этом в качестве источника строк необходимо выбрать таблицу *Категории*, которая содержит поля *Код категории* типа счетчик и *Наименование категории* текстового типа. В данную таблицу должны быть добавлены записи, как показано на рис. 3.15.

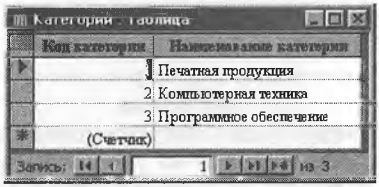


Рис. 3.15. Таблица *Категории*

В качестве значения свойства *Тип источника строк* выберем *Таблица* или *Запрос*, а в качестве значения свойства *Источник строк* зададим запрос вида, показанного на рис. 3.16.

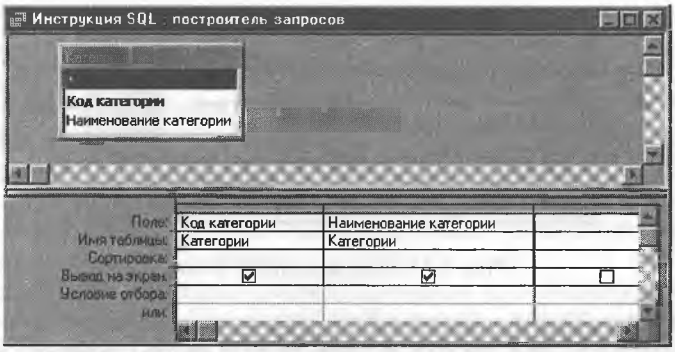


Рис. 3.16. Запрос для поля со списком *Выбор категории товара*

Зададим значения еще нескольким свойствам данного элемента управления (табл. 3.18).

Таблица 3.18

Значения некоторых свойств поля со списком *Выбор категории товара*

Свойство	Значение
Присоединенный столбец (Bound Column)	1. В этом случае присоединенным будет первый столбец запроса, приведенного на рис. 3.16
Количество столбцов (Column Count)	2
Ширина столбцов (Column Widths)	0 см. В данном случае задается ширина только первого столбца; ширина второго столбца не задается, так как она будет равна ширине списка
Ширина списка (List Width)	Авто (Auto)

Остальные свойства используют значения по умолчанию.

Теперь необходимо создать процедуру обработки события *После обновления* (AfterUpdate):

```
Private Sub Выбор_категории_товара_AfterUpdate()
Dim ctl As Control
   Set ctl = Me![Выбор Товара]
   ctl.Enabled = True
   ctl.Requery
   ctl.SetFocus
End Sub
```

В данной процедуре объявляется переменная ctl типа Control и затем ей присваивается ссылка на поле со списком *Выбор товара*. Далее эле-

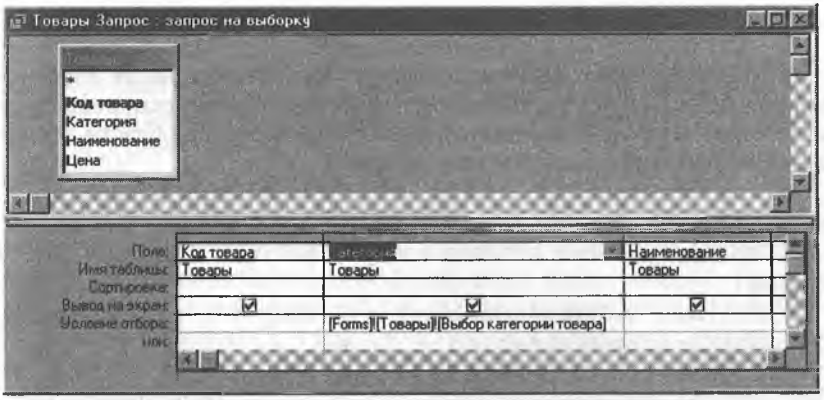


Рис. 3.17. Запрос для поля со списком *Выбор товара*

мент управления делается доступным с помощью присваивания свойству Enabled значения True. Затем используется метод Requery для выполнения повторного запроса к источнику данных этого списка. С помощью метода SetFocus устанавливается фокус на данный элемент управления.

Теперь создадим поле со списком *Выбор товара*. В качестве источника строк для данного списка зададим запрос, показанный на рис. 3.17.

Зададим значения еще нескольким свойствам данного элемента управления (табл. 3.19).

Таблица 3.19

Значения некоторых свойств поля со списком *Выбор товара*

Свойство	Значение
Присоединенный столбец (BoundColumn)	1. В этом случае присоединенным будет первый столбец запроса, приведенного на рис. 3.17
Количество столбцов (ColumnCount)	3
Ширина столбцов (ColumnWidth)	0 см; 0 см. В данном случае задается ширина первого и второго столбцов; ширина третьего столбца не задается, так как она будет равна ширине списка
Ширина списка (ListWidth)	Авто (Auto)

В данном запросе отбираются только записи, содержащие информацию о товарах категории, которая выбрана в поле со списком *Выбор категории товара*. Чтобы обеспечить это в качестве условия отбора по полю *Категория*, зададим следующее выражение:

```
[Forms]![Товары]![Выбор категории товара].
```

Теперь необходимо создать процедуре обработки события *После обновления* (AfterUpdate) поля со списком *Выбор товара*, основное предназначение которой — применить фильтр для вывода в области данных только информации о выбранном товаре:

```
Private Sub Выбор_товара_AfterUpdate()
DoCmd.ApplyFilter , "[Код товара] = Forms! Товары! [Выбор
товара]"
   Me![Код товара].Enabled = False
   Me!Категория.Enabled = True
   Me!Категория.SetFocus
   Me!Наименование.Enabled = True
   Me!Цена.Enabled = True
End Sub
```

В данной процедуре фильтр применяется с помощью метода ApplyFilter объекта DoCmd. Другие инструкции делают доступными соответствующие элементы управления.

Также необходимо создать две процедуры обработки формы: для событий *Открытие* (Activate) и *После обновления* (AfterUpdate). Первая процедура определяет режим изменения данных для формы после ее открытия, а вторая — выполняет повторный запрос к источнику данных поля со списком *Выбор товара* каждый раз, когда добавляется новая запись в форме.

Процедура обработки события *После обновления* (AfterUpdate) формы *Товары* имеет следующий вид:

```
Private Sub Form_AfterUpdate()
   Me![Выбор товара].Requery
End Sub
```

В данной процедуре используется только метод Requery для обновления запроса к источнику данных элемента управления *Выбор товара* активной формы.

Приведем процедуру обработки события *Открытие* (Activate):

```
Private Sub Form_Activate()
'Задаем режим изменения данных
   Me.AllowEdits = True
   Me.AllowAdditions = False
'Включаем поле со списком Выбор категории товара и
'переходим на него
   Me![Выбор категории товара].Enabled = True
   Me![Выбор категории товара].SetFocus
   Me![Выбор товара].Enabled = False
```

```
'Отключаем элементы управления в области данных
   Me![Код товара].Enabled = False
   Me![Категория].Enabled = False
   Me![Наименование].Enabled = False
   Me![Цена].Enabled = False
End Sub
```

Теперь создадим группу из двух переключателей *Изменение* и *Добавление*, которые позволят переходить из режима изменения информации о товарах в режим добавления информации о новых товарах. Значения свойства *Значение параметра* (OptionVaLue) для переключателя *Изменение* установим равным 2, а для переключателя *Добавление* — равным 1. Для группы необходимо также задать значение свойства *Значение по умолчанию* (DefaultValue), равное 2. При выборе одного из этих переключателей будет включаться соответствующий режим работы с формой. Процедура обработки события *После обновления* (AfterUpdate) будет иметь следующий вид:

```
Private Sub Режим_AfterUpdate()
   'Если выбран переключатель Изменение, то
   If Forms!Товары!Режим.Value = 2 Then
   'Задает для свойства AllowEdits значение True,
   'чтобы перейти в режим изменения
   frm.AllowEdits = True
   'Включаем поле со списком Выбор категории товара
   'и переходим на него
   Me![Выбор категории товара].Enabled = True
   Me![Выбор категории товара].SetFocus
   Me![Выбор товара].Enabled = False
   'Отключаем элементы управления в области данных
   Me![Категория].Enabled = False
   Me![Наименование].Enabled = False
   Me![Цена].Enabled = False
   'Если выбран переключатель Добавление, то
   Else
   Me.AllowEdits = False
   Me.AllowAdditions = True
   'Включаем элементы управления в области данных,
   'кроме поля Код товара
   Me![Код товара].Enabled = False
   Me![Категория].Enabled = True
   Me![Наименование].Enabled = True
   Me![Цена].Enabled = True
   'Переходим на новую запись и переводим фокус
   'на поле Категория
   DoCmd.GoToRecord, , acNewRec
   Me![Категория].SetFocus
```

```
'Отключаем поля со списками в заголовке формы
   Me![Выбор категории товара].Enabled = False
   Me![Выбор товара].Enabled = False
End If
End Sub
```

Созданная для примера форма позволяет работать в режимах изменения данных и ввода новых данных.

Работа с отчетами

Для работы с отчетами MS Access предоставляет ряд объектов, структура которых изображена на рис. 3.18.

Коллекция Reports содержит все отрытые отчеты базы данных, каждый из которых представляется объектом Report.

Каждый отчет имеет коллекцию Controls, членами которой являются элементы управления, содержащиеся в нем. В каждый объект Report также встроены объект Module и коллекция Properties. Объект Module представляет собой модуль отчета, а коллекция Properties содержит все встроенные свойства отчета.

Структура объектов, встроенных в объект Report, практически идентична структуре объектов, встроенных в объект Form. Сам объект Report имеет большое количество тех же свойств, что и объект Form.

Для работы с конкретными разделами отчета можно использовать свойство Section объекта Report. Например, чтобы сделать видимым верхний колонтитул отчета, следует добавить в программу следующие инструкции:

```
   Reports("Отчет1").Section(acPageHeader).Visible =
= True
```

Коллекция Reports содержит все открытые в данный момент отчеты. Для ссылки на конкретный отчет используется такой же синтаксис, как и для ссылки на форму.

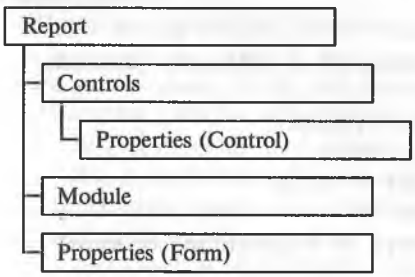


Рис. 3.18. Иерархия объектов, встроенных в объект Report

Коллекция AllReports содержит объекты AccessObject для каждого отчета в объекте CurrentProject или CodeProject.

Объект Report ссылается на конкретный отчет MS Access. Каждый объект Report является элементом коллекции Reports.

Свойство *Группировка по датам* (DateGrouping) определяет способ выполнения группи-

ровки по дате в отчете. Данное свойство может иметь два значения, причем:

если значение равно 0 — Американский стандарт (US Defaults), то используются настройки календаря США, т.е. первый день недели — Воскресенье (Sunday) и начало первой недели года — с 1 января;

если значение равно 1 — Параметры настройки (Use System Settings), то используются настройки, основанные на значениях, выбранных в диалоговом окне *Свойства*: *Язык и стандарты* (Regional Settings Property) на панели управления Windows (Windows Control Panel).

Значение свойства *Группировка по датам* (DateGrouping) можно задать только в окне свойств или процедуре события *Открытие* (Open) объекта Report.

Вид группировки определяется значением свойства GroupOn. Чтобы задать значение данного свойства, в режиме конструктора форм следует нажать кнопку [Сортировка и группировка] на панели инструментов *Конструктор отчетов* и в появившемся диалоговом окне *Сортировка и группировка* задать необходимые параметры.

Свойство GroupLevel задает уровни группировки или позволяет выполнить в отчете сортировку. Данное свойство представляет собой массив, в котором каждый элемент задает уровень группировки. Для ссылки на конкретный уровень группировки используют следующий синтаксис:

```
GroupLevel(n)
```

Здесь параметр n задает уровень группировки (начиная с 0). Число уровней группировки может достигать 10.

В случае использования варианта записи GroupLevel (0) будет произведена группировка или сортировка по первому полю или выражению отчета или формы.

При использовании свойства GroupLevel должны быть также заданы соответствующие значения свойств SortOrder, GroupOn, GroupInterval, KeepTogether и ControlSource. Эти значения можно задать в процедуре обработки события *Открытие* (Open).

Свойство *Фильтр включен* (FilterOn) определяет, будет ли применен (значение True) фильтр, задаваемый свойством *Фильтр* (Filter). В случае установки данному свойству значения False заданный ранее фильтр будет удален.

Для автоматического применения фильтра при открытии отчета или формы следует задать значение True свойству *Фильтр включен* (FilterOn) в процедуре обработки события *Открытие* (Open).

Свойство *Порядок сортировки* (OrderBy) определяет, по каким полям производится сортировка записей в форме, отчете, запро-

Таблица 3.20

Значения свойства *Данные* (ControlSource)

Значение	Описание
Page	Номер текущей страницы 1, 2, 3 и т.д.
“Страница” & Page & “из” & Pages	Позволяет вывести номер текущей страницы и общее количество страниц. Например, страница 1 из 10
Pages	Общее количество страниц

се или таблице. Значением данного свойства должна являться строка, которая определяет имя поля или нескольких полей, разделенных запятой. Сортировка производится по возрастанию. Если требуется задать сортировку по убыванию, то после имени поля следует добавить ключевое слово Desc.

Свойство *Сортировка включена* (OrderByOn) определяет, будет (значение True) выполнена сортировка в соответствии со значением свойства *Порядок сортировки* (OrderBy) или нет (значение False).

Свойства Page и Pages возвращают информацию, которую можно использовать для печати количества страниц в отчете или форме, т.е. свойство Page возвращает номер текущей страницы, а свойство Pages — количество страниц, содержащихся в отчете или форме.

Свойство Page доступно для записи только в режиме предварительного просмотра или во время печати, а свойство Pages всегда доступно только для чтения.

Чтобы сослаться на данные свойства в программе, требуется разместить в отчете или форме (обычно в нижнем колонтитуле) поле, значение свойства *Данные* (ControlSource) которого будет равно одному из значений, представленных в табл. 3.20.

Свойства *Верхний колонтитул* (PageHeader) и *Нижний колонтитул* (PageFooter) определяют способ печати верхнего и нижнего колонтитулов. Возможные значения данного свойства приведены в табл. 3.21.

Свойство *Источник записей* (RecordSource) задает источник данных для формы или отчета. В качестве источника данных могут выступать таблица, запрос или оператор языка SQL. Значением данного свойства должна быть строка, которая представляет собой имя таблицы, запроса или оператор SQL.

Свойство Section позволяет сослаться на раздел отчета или формы и получить доступ к свойствам данного раздела или его элементам управления. Значения данного свойства приведены в табл. 3.22 (в скобках указаны значения констант).

Таблица 3.21

Возможные значения свойств *Верхний колонтитул* (PageHeader) и *Нижний колонтитул* (PageFooter)

Значение	Описание
0 — *Все страницы* (All Pages)	Задает печать колонтитулов на всех страницах отчета
1 — *Без заголовка* (Not With Rpt Hdr)	Отменяет печать верхнего или нижнего колонтитула на странице, где выводится заголовок отчета
2 — *Без примечания* (Not With Rpt Ftr)	Верхний или нижний колонтитул не печатаются на странице, где находится примечание отчета
3 — *Без заголовка/примечания* (Not With Rpt Hdr/Ftr)	Отменяет печать верхнего или нижнего колонтитула на странице, где находится заголовок или примечание отчета

Таблица 3.22

Константы, определяющие раздел формы или отчета

Константа	Раздел
acDetail (0)	Область данных формы или отчета
acHeader (1)	Заголовок формы или отчета
acFooter (2)	Примечание формы или отчета
acPageHeader (3)	Верхний колонтитул формы или отчета
acPageFooter (4)	Нижний колонтитул формы или отчета
acGroupLevel1Header (5)	Заголовок группы первого уровня группировки (только для отчетов)
acGroupLevel1 Footer (6)	Примечание группы первого уровня группировки
acGroupLevel2Header (7)	Заголовок группы второго уровня группировки (только для отчетов)
acGroupLevel2Footer (8)	Примечание группы второго уровня группировки

Если отчет имеет дополнительные уровни группировки, то заголовкам и примечаниям этих групп присваиваются значения, начиная с 9.

Работа с запросами

Все запросы, содержащиеся в файле базы данных, являются элементами коллекции AllQueries. Каждый запрос представляет

собой объект AccessObject типа Query. Коллекция AllQueries встроена в объекты CodeData и CurrentData.

Коллекция AllQueries содержит объекты AccessObject для каждого запроса.

Объект CodeData ссылается на объекты, сохраненные приложением — источником данных в базе данных, в которой выполняется код VBA.

Объект CurrentProject ссылается на проект (совокупность всех стандартных модулей и модулей классов) для текущей базы данных или проекта MS Access.

Объект CodeProject ссылается на программу на языке VBA, содержащуюся в базе данных. В данный объект встроены коллекции AllForms, AllReports, AllMacros, AllModules и AllDataAccessPages.

Объект CurrentData ссылается на объекты, которые сохранены приложением — источником данных (Jet- или SQL-сервером) в текущей базе данных.

Этот объект включает в себя:

коллекцию AllTables, содержащую все таблицы;

коллекцию AllQueries, содержащую все запросы (в файлах проектов MS Access, имеющих расширение ADP, запросов нет);

коллекцию AllViews, содержащую все представления (в файлах проектов MS Access с расширением .mdb представления отсутствуют);

коллекцию AllStoredProcedures, содержащую все хранимые процедуры (в файлах баз данных MS Access с расширением MDB хранимые процедуры отсутствуют);

коллекцию AllDatabaseDiagrams, содержащую все схемы базы данных (в файлах баз данных MS Access с расширением MDB схемы базы данных отсутствуют).

3.2.9. Создание процедур обработки событий

Рассмотрим создание процедуры обработки событий. Для большинства элементов управления формы, а также для самой формы и отчета стандартный набор действий следующий:

- открыть форму в режиме конструктора. Если при этом окно свойств на экране отсутствует, то следует щелкнуть мышью по кнопке [Свойства] (Properties) на панели инструментов;
- выбрать нужный элемент управления (или щелкнуть мышью на маленьком черном квадрате в верхнем левом углу формы, тогда выберется вся форма), после чего в окне свойств отобразятся свойства выбранного элемента;
- открыть вкладку *События* (Events);
- выбрать событие, для которого будет создаваться процедура обработки, и щелкнуть по нему правой кнопкой мыши (рис. 3.19);

Рис. 3.19. Диалоговое окно событий формы

• выбрать из контекстного меню пункт *Построить* (Build). В открывшемся диалоговом окне *Построитель* (Choose Builder) выбрать из списка элемент *Программы* (Code Builder) и нажать кнопку [OK]. Откроется окно редактора VBA, в котором появятся первая и последняя строки процедуры (рис. 3.20).

Если процедура обработки выбранного события имеет аргументы, они будут также присутствовать в заголовке процедуры (рис. 3.21).

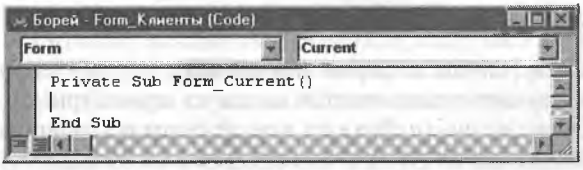


Рис. 3.20. Заготовка процедуры обработки событий

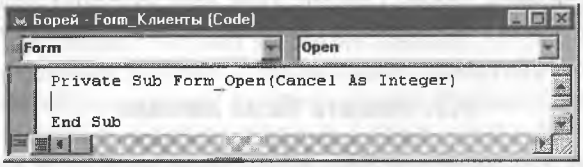


Рис. 3.21. Процедура обработки событий с аргументом Cancel

Теперь нужно ввести код процедуры между этими двумя строками.

Пример 3.13. Синхронизация данных в двух связанных формах с помощью процедуры обработки события *Текущая запись* (Current) в модуле формы *Поставщики* (Supplier).

В форме есть кнопка [Просмотр товаров] (Review Products), при нажатии которой выводится форма *Список товаров* (Product List), показывающая товары данного поставщика. Естественно, что при переходе к новой записи в форме *Поставщики*, записи в форме *Список товаров* тоже должны быть заменены.

Процедура обработки имеет следующий вид:

```
Private Sub Form_Current()
On Error GoTo Err_Form_Current
'Отображение товаров текущего поставщика при открытии формы Список товаров.
Dim strDocName As String
Dim strLinkCriteria As String
   strDocName = "Список товаров"
   strLinkCriteria = "[КодПоставщика] =
                  Forms![Поставщики]![КодПоставщика]"
If IsNull(Me![Название]) Then
   Exit Sub
ElseIf IsLoaded("Список товаров") Then
      DoCmd.OpenForm strDocName,,, StrLinkCriteria
   End If
Exit_Form_Current:
   Exit Sub
Err_Form_Current:
   MsgBox Err.Description
   Resume Exit Form_Current
End Sub
```

Если текущая запись в форме *Поставщики* пустая, т.е. добавляется новая запись, то сразу выполняется выход из процедуры.

Если текущая запись отображает конкретного поставщика, то проверяется, загружена ли форма *Список товаров*.

Если форма загружена, то меняется набор записей в ней. Делается это с помощью макрокоманды *ОткрытьФорму* (OpenForm) с соответствующим условием отбора записей. При этом на самом деле форма не открывается, просто повторно запрашивается источник данных.

3.3. Защита базы данных

Microsoft Access обеспечивает два традиционных способа защиты базы данных: установку пароля, требуемого при ее откры-

тии, и защиту на уровне пользователя, позволяющую ограничивать ту часть БД, к которой пользователь будет иметь доступ или сможет изменять. Кроме того, можно удалить изменяемую программу Visual Basic из базы данных, чтобы предотвратить изменения структуры форм, отчетов и модулей, сохранив БД как файл MDE.

Простейшим способом защиты БД является установка пароля. В этом случае при каждом открытии базы данных будет появляться диалоговое окно, в которое требуется ввести пароль, и только те пользователи, которые введут правильный пароль, смогут открыть эту базу данных. Данный способ достаточно надежен для БД, которая совместно используется небольшой группой пользователей или на автономном компьютере, установка пароля обычно оказывается достаточной.

Наиболее гибким и распространенным является способ защиты базы данных на уровне пользователей, который подобен способам защиты, используемым в большинстве сетевых систем. В этом случае от пользователей требуется идентифицировать себя и ввести пароль, когда они запускают Microsoft Access.

С помощью средств защиты можно указать, какие операции по обработке объектов базы данных разрешается выполнять конкретному пользователю или группе пользователей. При этом о каждом пользователе или группе пользователей ведутся учетные записи с указанием прав доступа к тем или иным объектам.

Рабочей называется группа пользователей, работающих с одной базой данных и имеющих общий файл.

Файл рабочей группы — это системный файл с информацией о группе пользователей, работающих с БД коллективного доступа. В файлах рабочих групп хранятся учетные записи, пароли, а также данные о правах доступа к объектам БД.

Изменение стандартного или создание нового файла рабочей группы выполняет программа Администратор рабочих групп, находящаяся в папке System папки Windows.

После установки Access пользователь получает право доступа ко всем объектам БД, т.е. становится членом группы Admins с именем Admin. Члены группы Admins (администраторы) имеют право на модификацию БД.

Чтобы устранить произвольный доступ в систему всех членов группы Admins, следует установить пароль для каждого администратора в ее регистрационной записи. В противном случае при каждом запуске Access администратор будет регистрироваться как пользователь Admin, т.е. без указания пароля.

После создания рабочей группы можно приступить к созданию учетных (регистрационных) записей. По умолчанию создается учетная запись Admin, а также учетные записи групп Admins (Администраторы) и Users (Пользователи) и предоставляются права доступа ко всем объектам.

Учетная запись администратора включается в рабочую группу Admins. Администратор имеет право доступа ко всем объектам, созданным в этой группе.

Кроме администратора в рабочей группе может быть указан владелец базы данных (в системе обеспечения безопасности Access владельцы объектов имеют особый статус). По умолчанию пользователь, создавший объект, становится владельцем этого объекта и имеет право на работу с ним.

Администраторы и владельцы имеют следующие права:

администратор БД всегда может получить право доступа ко всем объектам, созданным членами данной рабочей группы;

владелец БД всегда может открыть базу данных;

владелец объекта наделен полными правами доступа к этому объекту.

Пользователь Admin является владельцем любой базы данных и всех объектов. Поскольку для него не устанавливается пароль, то для защиты БД от несанкционированного доступа необходимо изменить право владения базой и ее объектами.

Существует несколько способов смены владельца объектов базы данных:

импортирование всех объектов БД в новый файл;

использование вкладки *Смена владельца* диалогового окна *Разрешения.*

Администратор БД предоставляет всем или некоторым членам рабочей группы права доступа к различным объектам базы данных. Права доступа хранятся в файле базы данных и характеризуют ее объекты.

К разграничению прав доступа пользователей и групп приступают после создания рабочей группы, определения администратора и владельца базы данных, а также создания учетных записей пользователей и групп. Пользователь наследуют права той группы, к которой принадлежит.

Перечень прав доступа, определенных в Access, приведен в табл. 3.23.

Из таблицы видно, что некоторые права доступа обуславливают наличие других прав. Так, таблица, в которой разрешено обновление данных, доступна для чтения данных и макета. При наличии прав администратора пользователю доступны все перечисленные выше права.

Определить права доступа к некоторому объекту может владелец этого объекта, администратор в рабочей группе Admins или пользователь, которому присвоены права администратора именно для этого объекта.

Все права доступа к объекту сохраняются при его изменении только в том случае, если не применялся буфер обмена или не выполнялся импорт/экспорт объекта. Однако все связанные с

Таблица 3.23

Перечень прав доступа, определенных в Access

Право доступа	Действие	Объекты доступа
Открытие/Запуск	Открытие БД, формы, отчета или запуск макроса	БД, формы, отчеты и макросы
Чтение макета	Просмотр объектов в режиме конструктора	Таблицы, запросы, формы, отчеты, макросы и модули
Изменение макета	Просмотр, изменение и удаление объектов в режиме конструктора	То же
Администратора	Полный доступ к объектам и данным, включая возможность присвоения прав доступа	Базы данных, таблицы, запросы, формы, отчеты, макросы и модули
Чтение данных	Просмотр данных	Таблицы и запросы
Обновление данных	Просмотр и изменение данных без вставки и удаления	То же
Вставка данных	Просмотр и вставка данных без изменения и удаления	»
Удаление данных	Просмотр и удаление данных	»
Монопольный доступ	Открытие БД в монопольном режиме	Базы данных

объектом права доступа могут быть утеряны при сохранении объекта под новым именем посредством команды *Сохранить как|Экспорт*.

Если база данных содержит программы Visual Basic, то ее сохранение как MDE-файла скомпилирует все модули, удалит все изменяемые исходные программы и выполнит сжатие базы данных. Программы Visual Basic будут по-прежнему выполняться, но их нельзя будет просмотреть или изменить, благодаря чему уменьшится размер базы данных. Кроме того, будет оптимизировано использование памяти, а следовательно, повысится быстродействие.

Сохранение базы данных как MDE-файла сделает невозможным выполнение следующих действий:

просмотр, изменение или создание форм, отчетов или модулей в режиме конструктора;

добавление, удаление или изменение ссылок на библиотеки объектов или базы данных;

изменение программы с помощью свойств или методов Microsoft Access или модели объектов VBA, так как MDE-файл не содержит текстов исходных программ;

изменение названия проекта VBA базы данных в диалоговом окне *Параметры*;

импорт и экспорт форм, отчетов и модулей. Однако таблицы, запросы и макросы можно будет импортировать и экспортировать в базы данных, не являющиеся MDE-файлами.

Еще одним способом защиты является шифрование — это защита БД от несанкционированного доступа с помощью текстового редактора или средств работы с файлами, например, входящих в состав Windows или Norton Utilities. Информация в зашифрованной базе данных недоступна для чтения. Шифрование несколько замедляет работу Access, так как на шифрование и дешифрование файлов расходуется время.

Шифрование и дешифрование базы данных могут производить только члены группы Admins.

Контрольные вопросы и упражнения

1. Дать определение макроса. Какими возможностями он обладает?
2. Дать определение модуля. Какими возможностями он обладает?
3. Охарактеризовать объектную модель MS Access.
4. Описать технологию создания процедур на VBA.
5. Описать технологию создания баз данных на VBA.
6. Описать технологию создания таблиц на VBA.
7. Описать технологию создания процедуры обработки событий на VBA.
8. Написать процедуру создания формы на VBA.
9. Написать процедуру создания отчета на VBA.
10. Каковы основные свойства форм, доступные при программировании на VBA?
11. Каковы способы защиты базы данных?
12. Пояснить технологию защиты базы данных на уровне пользователей.
13. Каковы преимущества использования MDE-файлов?
14. Что представляет собой защита базы данных с помощью шифрования?

ГЛАВА 4
АРХИТЕКТУРА СИСТЕМЫ БАЗ ДАННЫХ

4.1. Развитие архитектуры СУБД

Первоначально СУБД имели централизованную архитектуру, в которой сама СУБД, базы данных и прикладные программы, работающие с базами данных, функционировали на центральном компьютере. Все процессы, связанные с обработкой данных, производились на центральном компьютере, что определяло жесткие требования к его производительности.

Развитие и распространение компьютерных сетей привело к развитию новых архитектурных принципов в организации баз данных.

В основе широкого распространения локальных сетей компьютеров лежит идея разделения ресурсов. Высокая пропускная способность локальных сетей обеспечивает эффективный доступ из одного узла локальной сети к ресурсам, находящимся в других узлах. Однако целесообразно иметь не только доступ к ресурсами удаленного компьютера, но также получать от этого компьютера некоторый сервис, который специфичен для ресурсов данного рода. Сервис может обеспечиваться программными средствами, которые должны располагаться только на этом компьютере и которые нецелесообразно дублировать в нескольких узлах. Таким образом, в сети различают рабочие станции и серверы локальной сети.

Рабочая станция предназначена для непосредственной работы пользователя или категории пользователей и обладает ресурсами, соответствующими локальным потребностям данного пользователя.

Сервер локальной сети должен обладать ресурсами, соответствующими его функциональному назначению и потребностям сети.

Примерами серверов могут служить:

сервер телекоммуникаций, обеспечивающий услуги по связи данной локальной сети с внешним миром;

вычислительный сервер, дающий возможность производить вычисления, которые невозможно выполнить на рабочих станциях;

дисковый сервер, обладающий расширенными ресурсами внешней памяти и предоставляющий их в использование рабочим станциям и другим серверам;

файловый сервер, поддерживающий общее хранилище файлов для всех рабочих станций;

сервер баз данных (фактически обычная СУБД, принимающая запросы по локальной сети и возвращающая результаты).

Сервер локальной сети предоставляет ресурсы (услуги) рабочим станциям и/или другим серверам.

Компьютер, запрашивающий услуги у некоторого сервера, принято называть *клиентом* локальной сети, а компонент локальной сети, оказывающий услуги некоторым клиентам — *сервером*.

4.2. Архитектура файлового сервера

Архитектура файлового сервера служит основой для расширения возможностей СУБД централизованной архитектуры в направлении поддержки многопользовательского режима. В таких системах СУБД может располагаться и работать на нескольких персональных компьютерах, а базы данных располагаются в разделяемых файлах, которые находятся на файловом сервере. Пользователь, работающий на персональном компьютере, имеет возможность через СУБД обратиться к базе данных на файловом сервере. В ответ на запрос СУБД файловый сервер направляет по сети требуемый блок данных.

К недостаткам такой архитектуры относятся высокий сетевой трафик (по сети передаются целые файлы базы данных) и низкий уровень безопасности доступа к данным.

В Microsoft Access имеется два основных варианта совместного использования баз данных по технологии файлового сервера.

1. Совместное использование целой базы данных Access. Особенно широкое распространение получили сети, поддерживающие концепцию файлового сервера. База данных Access в такой сети может размещаться на компьютере, выделенном в качестве файлового сервера. При этом СУБД Access может быть установлена или на файловом сервере, или на каждой рабочей станции. Обработка данных базы в обоих случаях осуществляется на рабочих станциях пользователей. При использовании в локальной вычислительной сети (ЛВС) средств Access работа с БД в сети для пользователей практически не зависит от ее конфигурации и способа размещения в ней СУБД. При этом все пользователи работают с одними и теми же данными, т.е. используют одни и те же формы, отчеты, запросы, макросы и модули. Это удобно, когда все пользователи должны использовать базу данных одинаково.

Концепция файлового сервера в локальной сети обеспечивается рядом сетевых операционных систем. Наиболее популярными являются Microsoft Windows NT и NetWare Novell. Window NT имеет версию Window NT Server, предназначенную для управления фай-

ловым и другими серверами сети, и версию Windows NT Workstation, которая устанавливается на рабочей станции. Windows NT Workstation является полностью 32-разрядной операционной системой, под управлением которой могут выполняться различные приложения, в том числе и Microsoft Access. Отметим, что Windows NT Workstation может работать не только на процессорах Intel, но и на ряде RISC-процессоров.

2. Совместное использование только таблиц базы данных Access. Можно поместить на сетевой сервер только таблицы и хранить остальные объекты базы данных на компьютерах пользователей. В этом случае работа с базой данных Access происходит быстрее, так как по сети передаются только данные. При этом пользователи могут изменять формы, отчеты и другие объекты в соответствии со своими конкретными требованиями, не влияя на работу других пользователей.

Для отделения таблиц от других объектов базы данных применяется мастер разделения баз данных.

4.3. Репликация баз данных

Для пользователей, которые совместно работают с одним приложением, но не всегда имеют возможность подключиться к ЛВС, Access предлагает использование репликации базы данных.

Репликацией называется создание специальных копий — реплик общей базы данных Access, с которыми пользователи могут одновременно работать на разных компьютерах. Например, при работе в командировке или дома, когда невозможно подключиться к сети, или когда необходимо уменьшить загрузку сети. Отличие реплики от обычной копии файлов баз данных заключается в том, что для реплики базы данных возможна синхронизация изменений.

Преобразование БД в реплицированную базу данных выполняется командой меню *Сервис|Репликация|Создать дополнительную реплику*. При этом Access присваивает базе данных статус основной реплики и создает одну новую реплику.

После внесения изменений в реплики возможна их синхронизация, которая выполняется с помощью команды *Синхронизация*.

При проведении сеанса синхронизации изменения, сделанные одним пользователем, могут автоматически вноситься в общую реплику и реплики других пользователей, и наоборот.

В процессе синхронизации производится обмен обновленными записями и объектами между репликами.

Если пользователи двух разных реплик по-разному изменили одну и ту же запись, то при синхронизации реплик создается конфликтная таблица. Для того чтобы просмотреть и исправить конф-

ликтующие записи, следует выполнить команду *Устранить конфликты*.

Отметим, что в БД реплицироваться могут не все объекты. Часть объектов может использоваться локально. Некоторые объекты реплицируются группами пользователей. При проведении сеанса синхронизации работа с базой данных может продолжаться. Для создания реплик базы данных можно использовать также портфель Windows.

4.4. Системная архитектура клиент—сервер

Чтобы прикладная программа, выполняющаяся на рабочей станции, могла запросить услугу у некоторого сервера, как минимум, требуется некоторое интерфейсное программное обеспечение, поддерживающее такого рода взаимодействие. Отсюда следует основные принципы системной архитектуры *клиент—сервер*.

Система разбивается на две части, которые могут выполняться в разных узлах сети, — клиентскую и серверную. При этом прикладная программа или конечный пользователь взаимодействуют с клиентской частью системы, которая в простейшем случае обеспечивает надсетевой интерфейс, а клиентская часть системы при необходимости обращается по сети к серверной части (рис. 4.1).

Доступ к базе данных от прикладной программы или пользователя производится путем обращения к клиентской части системы. В качестве основного интерфейса между клиентской и серверной частями выступает язык баз данных SQL. На стороне клиента СУБД работает только такое программное обеспечение, которое не имеет непосредственного доступа к базам данных, а обращается для этого к серверу с использованием языка SQL.

СУБД Access может функционировать в локальной сети, поддерживающей концепцию клиент—сервер. В такой сети используется сервер баз данных SQL, который располагается на мощной машине — сервере — и называется SQL-сервер. SQL-сервер выполняет обработку данных, размещенных на сервере, и отвечает за их целостность и сохранность. Язык структурированных запросов SQL используется для управления базой данных на сервере.

Рис. 4.1. Архитектура клиент—сервер

На рабочих станциях — клиентах — работает Access. Пользователи Access могут взаимодействовать не только со своими локальными базами, но и с данными, расположенными на сервере. Клиент может посылать на сервер запросы SQL, получать с него необходимые данные, а также посылать обратно на сервер обновленные данные.

Большинство существующих серверов баз данных используют реляционный язык структурированных запросов SQL. Широко известны следующие серверы баз данных: SQL Server фирмы Microsoft, Oracle Server фирмы Oracle и NetWare SQL фирмы Novell. SQL-серверы баз данных являются самым мощным приложением для сетевой обработки данных.

Серверы SQL устанавливаются в среде сетевой операционной системы. Например, Microsoft SQL Server может быть установлен на компьютере с Windows NT Server.

Подключение из Microsoft Access к серверам баз данных SQL может осуществляться с помощью драйверов ODBC (Open Database Connectivity), которые обеспечивают доступ клиентам к базам данных, т.е. поддерживают стандартные протоколы обмена для серверов баз данных SQL. Каждому серверу БД соответствует свой драйвер ODBC. После подключения данные из базы сервера можно обрабатывать, импортировать, экспортировать и связывать (присоединять) средствами Access. В комплект поставки MS Access включен драйвер ODBC для MS SQL Server.

Использование унифицированного языка запросов SQL позволяет работать с одной и той же базы данных пользователям из различных приложений. Данные из БД могут получать Access, Excel, FoxPro и многие другие приложения, использующие протокол ODBC, посылая запросы SQL для общения с сервером БД.

В Microsoft Access существует возможность создания приложения типа клиент—север и работы с ним.

Проект Microsoft Access (.adp) является новым типом файлов Access, обеспечивающим эффективный и естественный доступ к базам данных Microsoft SQL Server с помощью архитектуры компонентов OLE DB. Эта архитектура компонентов базы данных реализует эффективный доступ по сети и через Интернет к источникам данных многих типов, в том числе к реляционным источникам данных, почтовым файлам, неформатированным текстовым файлам и электронным таблицам. Используя проекты Access, можно легко создавать приложения в архитектуре клиент—сервер.

Проект Access назван проектом, так как содержит только программные или HTML-объекты базы данных: формы, отчеты, страницы доступа к данным, макросы и модули, которые используются для создания приложений. В отличие от базы данных Microsoft Access проект Access не содержит объектов, основанных на дан-

ных или определениях данных: таблиц, представлений, схем баз данных или сохраненных процедур. Вместо этого перечисленные объекты базы данных хранятся в базе данных SQL Server.

Представление — это виртуальная таблица, создаваемая запросом, описание которой сохраняется в базе данных. Например, можно определить представление, содержащее лишь некоторые из столбцов таблицы, чтобы ограничить доступ к определенным сведениям. В большинстве операций базы данных представления могут рассматриваться как таблицы (в том числе в запросах на выборку). Любая операция, выполненная над представлением, затрагивает данные в таблице или таблицах, по которым создается представление.

Сохраненная процедура (хранимая процедура) — это заранее откомпилированная последовательность инструкций SQL и необязательных управляющих инструкций, сохраненных под общим именем, которые выполняются как одна программная единица. Хранимые процедуры располагаются в базе данных SQL (такой как Microsoft SQL Server) обычно на сервере и могут выполняться с помощью одного вызова из приложения, а также могут принимать описанные пользователем переменные, условия выполнения или другие управляющие инструкции.

Для создания приложения и доступа к данным проект Access следует подключить к базе данных SQL Server с помощью мастера

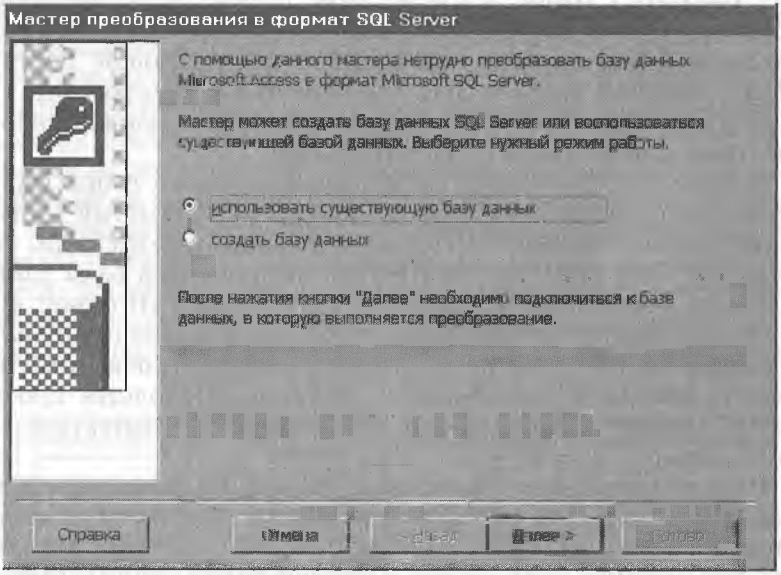


Рис. 4.2. Мастер преобразования в формат SQL Server

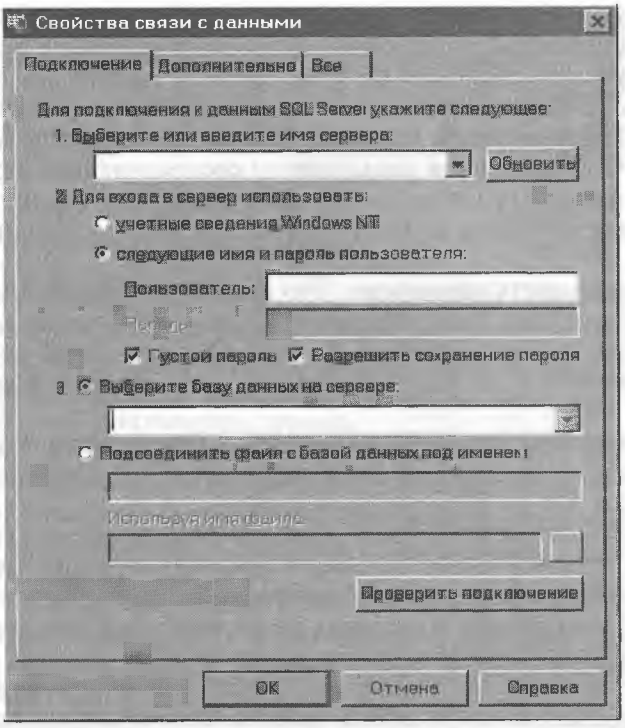


Рис. 4.3. Подключение проекта Access к базе данных SQL Server

баз данных (рис. 4.2) или с помощью команды *Создать* из меню *Файл* (рис. 4.3).

Допускается подключение проекта Access к базам данных Microsoft SQL Server 6.5 или 7.0 в операционных системах Microsoft Windows NT версии 4.0 или более поздней.

Работа с проектом Access очень похожа на работу с базой данных Access. Процесс создания форм, отчетов, страниц доступа к данным, макросов и модулей практически не отличается от процесса, используемого при создании базы данных Access.

После подключения к базе данных SQL Server можно просматривать, создавать, изменять и удалять таблицы, представления, сохраненные процедуры и схемы баз данных с помощью средств разработки Microsoft SQL Server Design Tools.

4.5. Распределенные системы баз данных

Существуют системы баз данных, в которых клиент может получать доступ к любому числу серверов одновременно (т.е. за один

запрос можно получить комбинированные данные двух и более серверов). В этом случае серверы рассматриваются клиентом как один сервер (с логической точки зрения), и пользователь может не знать, на каком именно компьютере какая часть данных содержится. Такие системы называют *распределенными системами баз данных*. С точки зрения пользователя распределенная база данных должна выглядеть точно так же, как нераспределенная. При этом распределенная база данных должна отвечать следующим требованиям.

1. Локальность автономии. Это означает, что функционирование данного узла сети управляется этим узлом и не зависит от функционирования другого узла сети. Под локальной автономией подразумевается также, что все узлы сети рассматриваются как равные.

2. Непрерывность функционирования. Подразумевается, что даже в случае неисправности отдельного узла работа системы продолжается, хотя и на более низком уровне.

3. Независимость от расположения. Пользователям не следует знать, в каком физическом месте хранятся данные, наоборот, с логической точки зрения пользователям следует обеспечить такой режим, при котором создается впечатление, что все данные хранятся на их собственном локальном узле.

4. Независимость от аппаратного обеспечения и операционной системы. Данные должны интегрироваться на компьютерах с различными техническими характеристиками, архитектурами и операционными системами, чтобы для пользователя создавалось представление единой системы.

5. Независимость от сети. Система должна поддерживать не только узлы с разным аппаратным обеспечением и разными операционными системами, но и разные типы сетей.

6. Независимость от СУБД. Различные СУБД на различных узлах сети должны быть интегрируемы друг с другом.

4.6. Интеграция базы данных с глобальной сетью Интернет

При обеспечении WWW-доступа к существующим БД возможен ряд альтернативных путей — комплексов технологических и организационных решений. Практика использования WWW-технологии для доступа к существующим БД предоставляет в настоящее время широкий спектр технологических решений, по-разному связанных между собой: перекрывающих, взаимодействующих и т.д. Выбор конкретных решений при обеспечении доступа зависит от специфики конкретной СУБД.

WWW-доступ к существующим базам данных может осуществляться двумя основными способами.

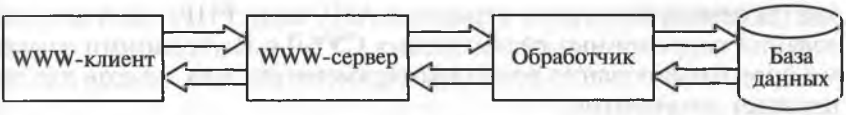


Рис. 4.4. Структура динамического создания гипертекстовых документов на основе содержимого БД

1. Однократное или периодическое преобразование содержимого БД в статические документы.

В этом варианте содержимое БД просматривает специальная программа (преобразователь), создающая множество файлов — связных HTML-документов. Полученные файлы копируются на WWW-сервер. Доступ к ним осуществляется, как к статическим гипертекстовым документам сервера.

Этот вариант характеризуется минимальными начальными расходами. Он эффективен при небольших массивах данных простой структуры с редким обновлением, а также при пониженных требованиях к актуальности данных, предоставляемых через WWW. Кроме того, для него характерны полное отсутствие механизма поиска и наличие возможности использования индексирования.

В качестве преобразователя в этом случае может выступать программное обеспечение, автоматически или полуавтоматически генерирующее статические документы. Программа-преобразователь может являться самостоятельно разработанной программой либо быть интегрированным средством из числа существующих на рынке разнообразных программ типа генераторов отчетов.

2. Динамическое создание гипертекстовых документов на основе содержимого БД.

В этом варианте доступ к БД осуществляется с помощью специальной программы, запускаемой WWW-сервером в ответ на запрос WWW-клиента. Такая программа, обрабатывая запрос, просматривает содержимое БД и создает выходной HTML-документ, возвращаемый клиенту (рис. 4.4).

Данный вариант эффективен для больших баз данных со сложной структурой и при необходимости поддержки операций поиска, а также при частом обновлении и невозможности синхронизации преобразования БД в статические документы с обновлением содержимого. В этом варианте возможно осуществление изменения БД из WWW-интерфейсов.

Для реализации такой технологии необходимо использовать взаимодействие WWW-сервера с запускаемыми программами CGI (Common Gateway Interface). Выбор программных средств для этого в настоящее время достаточно широк — это и универсальные языки программирования (C, Perl), и интегрированные средства типа генераторов отчетов. Кроме того, могут использоваться средства создания сценариев на страницах гипертекстовых докумен-

тов (активные серверные страницы ASP, язык PHP). При использовании современных реляционных СУБД с внутренними языками программирования возможно применение этих языков для генерации документов.

Контрольные вопросы и упражнения

1. Каковы тенденции развития архитектуры баз данных?
2. Описать архитектуру файлового сервера.
3. Какова технология репликации баз данных?
4. Описать архитектуру клиент—сервер.
5. Описать архитектуру распределенных баз данных.
6. Каковы способы применения баз данных MS Access в сети?
7. Рассказать об интеграции баз данных с глобальной сетью Интернет.

Перечень основных событий Microsoft Access

1. Событие *Текущая запись* (Current) происходит, когда очередная запись получает фокус или выполняется повторный запрос к источнику данных формы — таблице или запросу, т. е. оно возникает как при открытии формы, так и при переходе от одной записи к другой.

Чаще всего это событие используется для синхронизации записей в связанных формах.

2. Событие *Удаление* (Delete) происходит, когда пользователь пытается удалить запись из формы. Оно происходит до того, как запись реально удаляется из базы данных. Процедура обработки этого события имеет логический параметр Cancel. Если установить значение этого параметра как True, то удаление записи будет предотвращено.

3. Событие *До подтверждения Del* (BeforeDelConfirm) возникает до появления соответствующего окна. Процедура обработки этого события имеет два параметра: Cancel и Response. Присвоив в процедуре значение True параметру Cancel, можно отменить удаление; при этом окно подтверждения выдаваться на экран не будет. Следовательно, это еще одна возможность отменить удаление программно (третья возможность отмены будет предоставлена пользователю в диалоговом окне подтверждения удаления). Если же параметру Cancel присвоить значение False, то параметр Response можно использовать, чтобы определить, нужно ли выдавать окно подтверждения. Если параметр Response принимает значение 1, то запись будет удалена без подтверждения; если же значение равно 0, то Access выдаст окно, запрашивающее у пользователя подтверждение удаления записи.

4. Событие *После подтверждения Del* (AfterDelConfirm) происходит как после подтверждения удаления записи, так и при отмене удаления. Процедура обработки данного события имеет один параметр Status, который может принимать значения 0; 1; 2 и определять, была ли удалена запись: 0 — запись была успешно удалена; 1 — удаление отменено программой обработки события; 2 — удаление было отменено пользователем в окне подтвержде-

ния удаления. Это событие может быть использовано в программе для проверки удаления записи.

5. Событие *До вставки* (BeforeInsert) происходит, как только пользователь вводит первый символ в новую запись (в одно из полей, не обязательно первое), но до того, как запись фактически будет создана. Процедура обработки этого события может быть использована для проверки разрешения вставки. Процедура имеет один параметр Cancel. Если установить значение этого параметра равным True, то вставка записи будет запрещена. После этого события отменить вставку уже нельзя, можно только удалить вставленную запись.

6. Событие *После вставки* происходит после того, как в таблицу добавлена новая запись, обычно при переходе к следующей записи в форме. Процедура обработки этого события обычно используется для того, чтобы сделать повторный запрос к источнику данных с целью вывода новой записи.

7. Событие *До обновления* (BeforeUpdate) возникает при любом изменении данных в записи или элементе управления. Это событие может относиться как к элементу управления, так и к записи в целом. Процедура обработки этого события имеет один параметр Cancel, используемый для отмены введенных изменений, для чего ему нужно присвоить значение True. Это событие обычно используется для проверки условий на значение в поле таблицы или записи в целом, если эти условия сложные (простые условия обычно задаются в свойстве *Условие на значение* (ValidationRule) элемента управления). Условия проверяются сразу для нескольких значений, причем в них используются ссылки на элементы управления в других формах. При разных значениях введенных данных выдаются разные сообщения об ошибках.

8. Событие *После обновления* (AfterUpdate) происходит после обновления данных в записи или элементе управления. И хотя обновление данных уже произошло, можно еще восстановить старые значения, воспользовавшись свойством OldValue элемента управления, сохраняющим его старое значение, которое сменится только после события *После обновления*.

9. Событие *Изменение* (Change) возникает в следующих случаях:

при изменении содержимого текстового поля или поля со списком; причем изменением может быть любой непосредственно введенный или удаляемый символ;

при изменении значения свойства *Текст* (Text) элемента управления с помощью макроса или процедуры VBA;

в элементе управления *Набор вкладок* (Tab Control) при переходе с одной вкладки на другую.

10. Событие *Отсутствие в списке* (NotInList) возникает в поле со списком, когда пользователь вводит вручную значение в текстовую часть поля, которое отсутствует в списке, и после этого

пытается перейти в другое поле или сохранить запись. Для того чтобы это событие происходило, нужно присвоить свойству *Ограничиться списком* (LimitToList) значение Yes. Если это свойство имеет значение No, разрешается ввод в поле данных, не совпадающих ни с одним значением из списка.

Процедура обработки данного события имеет два параметра: NewData и Response. Параметр NewData содержит введенные данные, а параметр Response управляет обработкой события и может иметь значения 0; 1; 2. Значение 0 позволяет вывести на экран стандартное сообщение о том, что введенные данные отсутствуют в списке, и запретить ввод. Значение 1 позволяет вместо стандартного сообщения вывести специальное сообщение (например, запрашивающее) о том, следует ли сохранить введенное значение. Новые данные при этом не добавляются в список. Значение 2 разрешает добавить новое значение в список. При этом в процедуре обработки данного события нужно добавить значение к источнику строк для поля со списком, после чего поле обновляется, так как Access повторно запрашивает источник строк. Однако если источником строк для поля со списком является таблица-справочник, то простого добавления значения может оказаться недостаточно. Скорее всего придется вывести специальную форму, в которой пользователь должен будет заполнить все необходимые поля. После сохранения записи в этой форме новые данные добавляются в список.

Типичная ситуация, когда требуются такие действия, — это добавление нового клиента при выписке ему стандартного документа (счета, накладной и т.д.).

11. Событие *Внесены изменения* (Dirty) возникает в следующих ситуациях:

при изменении содержимого текстового поля или поля со списком, причем изменением может быть любой непосредственно введенный или удаляемый символ;

при изменении значения свойства *Текст* (Text) элемента управления с помощью макроса или процедуры VBA;

в элементе управления *Набор вкладок* (Tab Control) при переходе с одной вкладки на другую.

В отличие от события *Изменение* данное событие относится к форме. Процедура его обработки имеет один параметр Cancel. Если установить значение этого параметра равным True, то событие будет отменено. Отмена события будет вызывать откат всех изменений в записи, что эквивалентно нажатию клавиши [Esc]. Это новое событие в Access, которое удобно использовать для проверки наличия изменений в записи.

12. Событие *При обновлении* (Updated) возникает при изменении объекта OLE и применяется только к свободным и присоединенным рамкам объекта. Процедура обработки данного события

используется для проверки, изменились ли данные в объекте OLE после последнего сохранения. Процедура имеет один параметр Code, указывающий, каким образом обновлялся объект, который может принимать значения 0; 1; 2; 3. Значение 0 указывает, что данные объекта изменены. Значение 1 указывает, что данные объекта сохраняются приложением, создавшим объект. Значение 2 указывает, что файл объекта OLE закрывается приложением, которое его создало. Значение 3 указывает, что файл объекта OLE переименован создавшим его приложением.

13. События фокуса происходят, когда форма, отчет или элемент управления в форме получают или теряют фокус, а также когда форма или отчет становятся активными или, наоборот, — неактивными.

14. Событие *Вход* происходит перед тем, как элемент управления в форме получает фокус от другого элемента управления в той же форме, или когда при открытии формы получает фокус первый элемент управления. Его удобно использовать для вывода на экран каких-либо сведений об этом элементе. Данное событие происходит до события *Получение фокуса* (GetFocus), но после события *Текущая запись* (Current).

15. Событие *Выход* (Exit) происходит перед тем, как данный элемент управления передаст фокус другому элементу управления той же формы, но до события *Потеря фокуса* (LostFocus).

16. Событие *Получение фокуса* (GetFocus) происходит, когда форма или элемент управления формы получают фокус. Элемент управления может получить фокус только, если он видим и доступен (т.е. его свойства *Вывод на экран* (Visible) и *Доступ* (Enabled) имеют значения *Да*). При этом событие *Получение фокуса* происходит после события *Вход*. Форма может получить фокус, только если все поля в ней заблокированы, в противном случае событие *Получение фокуса* для формы не возникает.

17. Событие *Потеря фокуса* (LostFocus) происходит каждый раз, когда форма или элемент управления в форме теряют фокус. Данное событие происходит после события *Выход* (Exit).

18. Событие *Включение* (Activate) возникает, когда форма или отчет получают фокус, становясь активными. Это происходит, когда форма или отчет открываются при щелчке мышью на одном из элементов управления, т.е. при переносе таким образом фокуса, и когда в программе VBA выполняется метод SetFocus объекта. Форма при этом обязательно должна быть видима. Событие *Включение* возникает до события *Получение фокуса*, и его удобно использовать для выведения на экран панели инструментов, связанной с формой.

19. Событие *Отключение* (Deactivate) происходит, когда фокус из формы или отчета переносится на другое окно (таблицы, запроса, формы, отчета, макроса, модуля или базы данных). Однако

оно не возникает, когда фокус переходит в диалоговое окно или другое приложение. Событие *Отключение* возникает после события *Потеря фокуса*.

20. События клавиатуры происходят в форме и элементе формы, когда пользователь нажимает клавиши на клавиатуре или же выполняется макрокоманда SendKeys. Все события клавиатуры связываются с тем объектом в форме, который имеет в данный момент фокус. Обычно это один из элементов управления. Форма может получить фокус (а значит, и события клавиатуры могут относиться к форме) только в случае, если все ее элементы управления заблокированы или невидимы. Если нужно привязать эти события именно к форме, а не к элементу формы, то можно присвоить свойству *Перехват нажатия клавиш* (KeyPreview) для формы значение *Да* (Yes). Тогда все события клавиатуры возникают сначала для формы, а потом уже для элемента управления, имеющего фокус. Это позволяет программировать реакцию формы на нажатие определенных клавиш вне зависимости от того, в каком элементе управления формы находится фокус.

21. События *Клавиша вниз* (KeyDown) и *Клавиша вверх* (KeyUp) возникают всякий раз, когда пользователь нажимает или отпускает клавишу на клавиатуре и при этом фокус находится на элементе управления или форме. Процедуры обработки этих событий используют, когда требуется определить, какую клавишу нажал пользователь: функциональную, клавишу управления курсором, клавишу цифровой панели или комбинацию клавиш с [Shift], [Ctrl] или [Alt].

Данные события имеют два параметра: KeyCode и Shift. Параметр KeyCode — это целое число, представляющее собой код нажатой клавиши. Параметр Shift позволяет определить, в каком сочетании нажимались клавиши: 1 — [Shift], 2 — [Ctrl], 4 — [Alt], 0 — не использовались никакие сочетания клавиш. Если же использовалась комбинация клавиш [Shift] + [Ctrl] + [Alt] в любом сочетании, то параметр Shift будет равен сумме значений каждой клавиши.

22. Событие *Нажатие клавиши* (KeyPress) происходит, если пользователь нажимает и отпускает любую комбинацию клавиш для элемента управления или формы, имеющей фокус. В отличие от событий *Клавиша вниз* и *Клавиша вверх* это событие не происходит, когда нажимаются функциональные клавиши, клавиши управления курсором и клавиши [Shift],[Ctrl], или [Alt]. Кроме того, эти события различны для верхнего и нижнего регистров. Процедура обработки этого события имеет один параметр KeyAscii — целое число, представляющее собой код нажатой клавиши.

Если пользователь нажимает и удерживает некоторую клавишу, то события *Клавиша вниз* и *Нажатие клавиши* повторяются до

тех пор, пока он не отпустит эту клавишу, после чего возникает событие *Клавиша вверх*. Если результатом нажатия клавиши является перевод фокуса с одного элемента на другой, то событие *Клавиша вниз* возникает для первого элемента, а события *Нажатие клавиши* и *Клавиша вверх* — для второго. Если в результате нажатия клавиши появляется диалоговое окно, то возникают события *Клавиша вниз* и *Нажатие клавиши*, а событие *Клавиша вверх* не возникает.

23. События мыши происходят, когда какое-либо действие в форме или ее элементе управления выполняется с помощью мыши. События мыши не определены для элементов управления в отчетах, а также для флажков и переключателей в группах, они определены только для группы в целом.

24. Событие *Нажатие кнопки* (Click) возникает как в самой форме, так и в элементах управления формы. В форме событие *Нажатие кнопки* возникает, когда пользователь щелкает мышью на пустой ее области или на области выделения записи в форме. Для элемента управления событие *Нажатие кнопки* возникает при щелчке мышью не только на самом элементе, но и на присоединенной к нему надписи, а также в следующих случаях:

при выборе элемента из списка независимо от того, был он выбран с помощью мыши или клавиш управления курсором с последующим нажатием клавиши [Enter];

при нажатии клавиши [Пробел], когда фокус установлен на флажке, переключателе или командной кнопке;

при нажатии клавиши [Enter] в форме, которая содержит кнопку свойства *По умолчанию* (Default) со значением *Да* (Yes) (тогда именно на эту кнопку по умолчанию устанавливается фокус);

при нажатии клавиши [Esc] в форме, которая содержит кнопку [Отмена] (Cancel) со значением свойства *Да* (Yes);

при нажатии клавиш доступа, если они связаны с кнопками на форме.

Таким образом, процедуры обработки событий *Нажатие кнопки* запускаются независимо от того, каким образом эта кнопка выбрана — щелчком мыши, нажатием клавиши [Enter] или нажатием клавиши доступа. Процедура обработки события запускается только один раз. Если требуется, чтобы она запускалась несколько раз (пока кнопка остается нажатой), нужно использовать свойство *Автоматический повтор* (AutoRepeat) для кнопки. Если нужно определить, какой кнопкой мыши выполнялся щелчок, следует использовать события *Кнопка вниз* (MouseDown) и *Кнопка вверх* (MouseUp).

25. Событие *Двойное нажатие кнопки* (DblClick) происходит после быстрого двойного щелчка мышью на форме или элементе управления, при этом интервал между щелчками не должен превышать предельного времени, заданного в панели управления

Windows. Событие *Двойное нажатие кнопки* для формы и элемента управления формы определено так же, как и событие *Нажатие кнопки*. Однако для элементов управления результат этого события зависит от типа элемента управления. По умолчанию двойной щелчок мышью в текстовом поле приводит к выделению слова, а в объекте OLE — запускает этот объект для редактирования. Вводя процедуру обработки для данного события, можно переопределить стандартные действия Access. При этом процедура обработки имеет один параметр Cancel, при присвоении которому значения True можно отменить это событие.

26. Событие *Перемещение указателя* (MouseMove) генерируется непрерывно, когда пользователь перемещает указатель мыши по объектам формы. Пока указатель движется в границах объекта, событие генерируется для данного объекта; когда указатель попадает на пустую область формы, область выделения записи или полосу прокрутки, событие генерируется для формы. Событие возникает также при перемещении формы или элемента управления, например с помощью процедуры VBA при неподвижном указателе мыши.

Процедура обработки события имеет четыре параметра:

Button — определяет состояние кнопок мыши в момент возникновения события (перемещение указателя может происходить при нескольких нажатых кнопках или не нажатых кнопках мыши);

Shift — определяет состояние клавиш [Shift], [Ctrl] и [Alt] в тот момент, когда нажата кнопка, определяемая параметром Button;

x и *y* — текущие координаты указателя мыши в твипах.

27. События *Кнопка вниз* (MouseDown) и *Кнопка вверх* (MouseUp) возникают, когда пользователь нажимает и соответственно отпускает кнопку мыши.

В отличие от событий *Нажатие кнопки* и *Двойное нажатие кнопки* данные события позволяют определить, какая кнопка нажата.

Процедуры обработки этих событий имеют четыре параметра: Button, shift, *x* и *y*. Указанные параметры аналогичны параметрам процедуры для события *Перемещение указателя* (за исключением параметра Button — так как в данном случае нажимается конкретная кнопка мыши, этот параметр определяет, какая это кнопка).

Если пользователь нажмет сразу две кнопки, то события возникнут отдельно для каждой из этих кнопок.

28. Событие *Форматирование* (Format) происходит после отбора данных для отчета, но перед фактическим форматированием каждого раздела. При этом в разделе данных это событие происходит для каждой записи в отчете, что позволяет при необходимости по-разному форматировать каждую из этих записей. В отчете данное событие возникает для заголовка каждой группы. Процедура обработки этого события имеет два параметра: Cancel и FormatCount. Присвоение значения True параметру Cancel позволяет отменить форматирование данного раздела, что дает возмож-

ность пропускать разделы отчета, не оставляя пустого места на странице. Параметр FormatCount — это счетчик, который считает, сколько раз произошло событие *Форматирование*.

29. Событие *Возврат* (Retreat) происходит, если при форматировании раздела требуется вернуться к разделу, который уже отформатирован. Оно происходит после события *Форматирование*, но до события *Печать* (Print). Процедура обработки этого события позволяет изменить любое уже выполненное форматирование и обеспечить таким образом нужное расположение элементов отчета на странице.

Событие *Возврат* (Retreat) возникает практически после каждого события *Форматирование* (кроме тех разделов, которые не будут печататься). Процедура обработки данного события имеет два параметра: Cancel и Printcount. Параметр Cancel позволяет отменить печать текущего раздела или текущей записи в отчете при присвоении ему значения True. Однако при этом остается пустое место на странице, поэтому эту процедуру можно использовать, когда изменения не касаются формата страницы отчета. Параметр Printcount — это счетчик, который считает, сколько раз произошло событие.

30. Событие *Страница* (Page) возникает после форматирования страницы отчета, но до вывода ее на печать, и позволяет с помощью процедуры обработки добавить на страницу некоторые элементы оформления, например рамку.

31. Событие *Отсутствие данных* (No Data) возникает после форматирования отчета, но до его вывода на печать (до первого события *Страница*), и позволяет обнаружить отсутствие записей для отчета; в этом случае печать можно отменить. Процедура обработки этого события имеет один параметр cancel, которому следует присвоить значение True, если нужно отменить печать отчета.

32. Событие *Применение фильтра* (ApplyFilter) возникает во всех случаях, когда пользователь выполняет фильтрацию записей в форме с помощью соответствующих команд меню, контекстного меню или кнопок панели инструментов [Применить фильтр] и [Удалить фильтр]. Программу обработки этого события обычно используют либо для проверки условия в фильтре, либо для изменения вида формы перед применением фильтра, если требуется скрыть лишние поля или, наоборот, показать скрытые. Программа обработки события имеет два параметра: Cancel и ApplyType. Параметр Cancel позволяет отменить операцию фильтрации, если, например, условие сформулировано неправильно, для чего нужно присвоить ему значение True. Параметр ApplyType определяет исполняемое действие и может принимать значения 0; 1; 2. Значение 0 указывает на удаление фильтра, 1 — на его применение, 2 — на закрытие его окна.

33. Событие *Фильтрация* (Filter) возникает перед открытием окна фильтра или расширенного фильтра, когда пользователь пытается выполнить команду *Изменить фильтр* (Filter by Form). Использовать это событие очень удобно, если требуется, например, ввести в фильтр некоторые условия по умолчанию или запретить включать в условия отбора некоторые поля. Чтобы запретить включить некоторое поле в условие отбора в окне фильтра, достаточно скрыть его в процедуре обработки события *Фильтрация* (Filter). Правда, это относится только к окну обычного фильтра, так как в окне расширенного фильтра выводятся все поля, в том числе и скрытые.

Можно даже заменить стандартное окно фильтра своим собственным, в котором пользователь и будет задавать условия отбора. Процедура обработки события имеет два параметра: Cancel и FilterType. Параметр Cancel позволяет отменить открытие стандартного окна фильтра, если вместо него будет выводиться специальная форма, присвоением ему значения True. Параметр FilterType определяет, какое окно открывается, и может принимать значения 0 или 1. Значение 0 соответствует обычному фильтру, а 1 — расширенному.

34. Событие *Открытие* (Open) происходит после выполнения запроса, лежащего в основе формы или отчета, но до отображения первой записи или печати отчета. Процедура обработки этого события имеет один параметр Cancel, при присвоении которому значения True отменяется открытие формы или отчета.

35. Событие *Закрытие* (Close) является последним перед удалением формы с экрана. Обычно его используют для открытия другой формы.

36. Событие *Загрузка* (Load) происходит сразу после события *Открытие* (Open), но в отличие от него не может быть отменено. Обычно его используют для динамического изменения свойств формы или элементов управления перед выведением формы на экран.

37. Событие *Выгрузка* (Unload) происходит при закрытии формы и может быть отменено. Обычно это событие используется для проверки различных условий, которые определяют, можно ли закрывать форму. Процедура обработки этого события имеет один параметр Cancel. При значении True этого параметра закрытие формы отменяется.

38. Событие *Изменение размера* (Resize) возникает при открытии формы и изменении ее размеров. Его обычно используют, если требуется подстроить размер элементов управления под изменяющиеся размеры формы или найти заново вычисляемые элементы. Если при каждом изменении размеров формы необходимо обновление экрана, следует использовать в процедуре обработки этого события метод Repaint.

39. Событие *Ошибка* (Error) возникает, когда в процессе обработки формы или отчета ядром Access возникает ошибка. В процедуре обработки этого события можно перехватить стандартное сообщение об ошибке, которое выдает Access, и выдать собственное сообщение. Процедура имеет два параметра: DataErr и Response. Параметр DataErr содержит код ошибки, а параметр Response может принимать два значения: 0 и 1. Значение 0 отменяет выдачу стандартного сообщения об ошибке, а значение 1 позволяет его отобразить.

40. Событие *Таймер* (Timer) возникает регулярно через интервал времени, который задается свойством *Интервал таймера* (TimerInterval) формы, и позволяет определять действия, которые должны выполняться периодически по сигналу таймера. Обычно оно используется для регулярных обновлений экрана в многопользовательском приложении. В этом случае в процедуре обработки этого события следует использовать метод Requery, который будет выполнять повторный запрос источника данных формы.

Функции VBA в алфавитном порядке

Abs (VBA). Возвращает значение, тип которого совпадает с типом переданного аргумента, равное абсолютному значению указанного числа.

Синтаксис:
Abs(число)

Обязательный аргумент *число* может представлять собой любое допустимое числовое выражение. Если число имеет значение Null, возвращается значение Null. Если аргумент представляет собой не инициализированную переменную, возвращается нулевое значение.

Array (VBA). Возвращает значение типа Variant, содержащее массив.

Синтаксис:
Array(списокАргументов)

Обязательный аргумент *списокАргументов* представляет собой разделенный запятыми список значений, присваиваемых элементам массива, содержащегося внутри значения типа Variant. Если аргументы не указываются, создается массив нулевой длины.

Для ссылки на элемент массива записывается имя переменной, за которым в скобках следуют номера индексов нужного элемента. В приведенном здесь примере первая инструкция создает переменную A типа Variant, вторая — присваивает массив переменной A, а последняя — показывает, как присвоить другой переменной значение второго элемента массива:

```
Dim A As Variant
A = Array(10,20,30)
B = A(2)
```

Нижняя граница индексов массива, созданного с помощью функции Array, всегда равняется нулю. В отличие от массивов дру-

гих типов она не определяется нижней границей, заданной в инструкции Option Base.

Значение типа Variant, не описанное как массив, все равно может содержать массив. Переменная типа Variant может содержать массив любого типа, за исключением строк фиксированной длины и определяемых пользователем типов. Хотя значение типа Variant, содержащее массив, концептуально отличается от массива, элементы которого имеют тип Variant, доступ к элементам массива осуществляется тем же способом.

Asc (VBA). Возвращает значение типа Integer, представляющее собой код символа для первого символа строки.

Синтаксис:
Asc(строка)

Обязательный аргумент *строка* является любым допустимым строковым выражением. Если *строка* не содержит символов, возникает ошибка выполнения.

Возвращаемые значения находятся в диапазоне 0...255 для однобайтовых и в диапазоне –32 768...32 767 для двухбайтовых наборов символов (DBCS).

Для работы с байтами данных, содержащихся в *строке*, предназначена другая функция AscB, которая вместо кода первого символа возвращает первый байт.

Функция AscW возвращает код символа, соответствующий кодировке Unicode (за исключением платформ, в которых этот код не поддерживается), и поведение этой функции аналогично поведению функции Asc.

Atn (VBA). Возвращает значение типа Double, содержащее арктангенс числа.

Синтаксис:
Atn(число)

Обязательный аргумент *число* представляет собой значение типа Double или любое допустимое числовое выражение.

Функция Atn определяет величину угла (в радианах) по указанному отношению двух сторон прямоугольного треугольника (противолежащего и прилегающего катетов), которое задается с помощью аргумента *число*.

Значение, возвращаемое данной функцией, находится в диапазоне от $-\pi/2$ до $\pi/2$ радиан.

Для преобразования градусов в радианы следует умножить градусы на $\pi/180$, а для преобразования радиан в градусы — радианы на $180/\pi$.

Функция Atn является обратной функции Tan, которая возвращает тангенс указанного угла.

Avg (DAO). Вычисляет среднее арифметическое набора чисел, содержащихся в указанном поле запроса.

Синтаксис:
Avg(выражение)

Аргумент *выражение* является строковым выражением, которое определяет поле, содержащее числовые данные для вычисления среднего значения, или выражение, выполняющее вычисления с данными из этого поля. Операнды аргумента *выражение* могут включать в себя имя поля таблицы, константу или функцию, которая может быть внутренней или определяться пользователем, но не может быть ни одной из других статистических функций SQL.

Среднее значение, вычисленное функцией Avg, является числовым значением (суммой этих значений, деленной на их количество). Например, возможно использование функции Avg для вычисления средней стоимости доставки.

Функция Avg не включает в вычисления поля со значениями Null.

Функция Avg используется в выражении запроса и в свойстве SQL объекта QueryDef или при создании объекта Recordset на основе запроса SQL.

CBool (VBA), CByte (VBA), CCur (VBA), CDate (VBA), CDbl (VBA), CDec (VBA), CInt (VBA), CLng (VBA), CSng (VBA), CVar (VBA), CStr (VBA). Каждая из этих функций преобразует выражение к соответствующему типу данных.

Синтаксисы:
CBool(выражение)
CByte(выражение)
CCur(выражение)
CDate(выражение)
CDbl(выражение)
CDec(выражение)
CInt(выражение)
CLng(выражение)
CSng(выражение)
CVar(выражение)
CStr(выражение)

Обязательный аргумент *выражение* является любым строковым или числовым выражением.

Имя функции определяет ее возвращаемый тип (табл. П2.1)

Если переданное в функцию значение аргумента *выражение* находится вне допустимого диапазона для соответствующего типа данных, возникает ошибка.

Таблица П

Возвращаемые типы функций

Функция	Возвращаемый тип	Диапазон аргумента *выражение*
CBool	Boolean	Любая допустимая строка или числовое выражение
CByte	Byte	От 0 до 255
CCur	Currency	От –922 337 203 685 477,5808 до 922 337 203 685 477,5807
CDate	Date	Любое допустимое выражение даты
CDbl	Double	От –1,79769313486232E308 до –4,94065645841247E–324 для отрицательных чисел; от 4,94065645841247E–324 до 1,79769313486232E308 для положительных чисел
CDec	Decimal	±79 228 162 514 264 337 593 543 950 335 без масштабирования, т. е. для чисел без дробной части. Для чисел с 28 знаками в дробной части ±7,9228162514264337593543950335. Минимальное ненулевое число 0,0000000000000000000000000001
CInt	Integer	От –32 768 до 32 767 с округлением дробной части
CLng	Long	От –2 147 483 648 до 2 147 483 647 с округлением дробной части
CSng	Single	От –3,402823E38 до –1,401298E–45 для отрицательных чисел; от 1,401298E–45 до 3,402823E38 для положительных чисел
CVar	Variant	Диапазон значений Double для числовых значений. Диапазон значений String для нечисловых значений
CStr	String	От аргумента *выражение* зависят возвращаемые значения данной функции

Функции преобразования типов данных используются для документирования программы. Это позволяет показать, что результат некоторой операции должен быть представлен определенным типом данных, а не типом данных, используемым по умолчанию. Например, функция CCur обеспечивает арифметические действия над числами в денежном формате в тех случаях, когда по умолчанию используются действия над числами с плавающей точкой обычной или двойной точности либо над целыми числами.

Функции преобразования типов данных следует использовать вместо функции Val для обеспечения совместимости с различными национальными настройками. Например, при использовании функции CCur разделители целой и дробной частей, а также разделители разрядов и параметры денежных единиц правильно распознаются в зависимости от национальной настройки компьютера.

Если дробная часть числа равна 0,5, то функции CInt и CLng всегда округляют число до ближайшего четного числа. Например, 0,5 округляется до нуля, а 1,5 до двух. Необходимо отличать функции CInt и CLng от функций Fix и Int, которые выполняют усечение, а не округление дробной части числа. Кроме того, функции Fix и Int всегда возвращают значение с тем же типом данных, который был передан в аргументе.

Чтобы определить, может ли аргумент *дата* быть преобразован к типу даты или времени, следует использовать функцию IsDate.

Функция CDate распознает литералы даты и литералы времени, а также числа, которые попадают в приемлемый диапазон дат. При преобразовании числа в дату переводится целая часть этого числа. Любая дробная часть числа преобразуется во время суток, отсчитываемое от полуночи.

Функция CDate распознает форматы дат в соответствии с национальной настройкой системы. Правильный порядок дней, месяцев и лет может не быть определен, если дата задается в формате, отличном от распознаваемых форматов дат. Кроме того, длинный формат даты также не распознается, если он содержит строку для дня недели.

Для обеспечения совместимости с предыдущими версиями Visual Basic поддерживается также функция CVDate, синтаксис которой совпадает с синтаксисом функции CDate, но CVDate возвращает значение с подтипом Date типа Variant, а не значение типа Date. Поскольку теперь определен внутренний тип данных Date, нет необходимости в использовании функции CVDate. Тот же результат получается при преобразовании аргумента *выражение* к типу Date, а затем присвоении его переменной типа Variant, что соответствует преобразованию всех других внутренних типов данных к их эквивалентным подтипам Variant.

Функция CDec не возвращает конкретный тип данных; вместо этого всегда возвращается значение типа Variant, преобразованное к подтипу Decimal.

Choose (VBA). Возвращает значение, выбранное из списка аргументов.

Синтаксис:
Choose(индекс, вариант1[, вариант2, ...[, вариантn]])

Синтаксис функции Choose содержит элементы, представленные в табл. П2.2.

Таблица П2.2

Синтаксис функции Choose

Элемент	Описание
Индекс	Обязательный. Представляет собой числовое выражение или поле, значением которого является число, лежащее между единицей и числом элементов в списке
Вариант	Обязательный. Представляет собой выражение типа Variant, содержащее один из элементов списка

Функция Choose возвращает значение из списка, выбранное на основании значения аргумента *индекс*. Если *индекс* равняется 1, возвращается первый элемент списка; если *индекс* равняется 2, возвращается второй элемент списка, и т.п.

Функцию Choose можно использовать для выбора одного из возможных значений, представленных в виде списка.

Функция Choose вычисляет каждый элемент списка, а возвращает только один из них. В некоторых случаях это приводит к нежелательным побочным эффектам. Например, если выражение, определяющее элементы списка, содержит функцию MsgBox, то соответствующие сообщения будут последовательно появляться на экране по мере вычисления каждого из этих элементов несмотря на то, что функцией Choose будет возвращен только один из них.

Функция Choose возвращает значение Null, если *индекс* меньше 1 или больше числа элементов в списке.

Если *индекс* не является целым числом, он округляется до ближайшего целого числа перед выполнением операции.

Chr (VBA). Возвращает значение типа String, содержащее символ, соответствующий указанному коду символа.

Синтаксис:
Chr(кодСимвола)

Обязательный аргумент кодСимвола является значением типа Long, определяющим символ.

Коды 0...31 соответствуют стандартным управляющим символам ASCII. Например, Chr(10) возвращает символ перевода строки. Обычным диапазоном значений аргумента *кодСимвола* является интервал 0...255, однако в системах DBCS допустимыми являются значения от –32 768 до 65 536.

Эквивалентной функцией побайтовой обработки значений типа String является функция ChrB. Эта функция всегда возвращает один байт, а не символ, который может занимать один или два байта.

Функция ChrW возвращает значение типа String, содержащее символ в основном формате Unicode, за исключением платформ, в которых Unicode не поддерживается и функция работает аналогично функции Chr.

CodeDb. Используется в модуле для определения имени объекта Database, описывающего открытую базу данных, в которой выполняется текущая программа, и для работы с объектами доступа к данным, являющимися частью библиотечной базы данных.

Например, функция CodeDb может быть использована в модуле библиотечной базы данных для создания объекта Database, описывающего библиотечную базу данных. После этого становится возможным открытие и изменение набора записей, выбираемых из таблицы в библиотечной базе данных.

Синтаксис:
Set базаДанных = CodeDb

Функция CodeDb использует аргумент *база Данных*, являющийся объектной переменной типа Database.

Функция CodeDb возвращает объект Database со значением свойства Name, являющимся полным именем (включающим в себя путь) файла базы данных, из которой была вызвана данная функция, и используется для работы с объектами доступа к данным в библиотечной базе данных. При вызове данной функции в библиотечной базе данных текущей остается база данных, из которой была вызвана функция (даже во время выполнения программы модуля библиотечной базы данных). Для ссылки на объекты доступа к данным в библиотечной базе данных необходимо знать имя объекта Database, представляющего собой библиотечную базу данных.

Предположим, например, что в библиотечной базе данных имеется таблица, содержащая список сообщений об ошибках. Для того чтобы выполнить из программы обработку данных в этой таблице, следует с помощью функции CodeDb определить имя объекта Database, содержащего ссылку на библиотечную базу данных, в которой находится данная таблица.

Если функция CodeDb вызывается в текущей базе данных, то она возвращает имя текущей базы данных аналогично функции CurrentDb.

Command. Возвращает параметры командной строки, указанные при запуске Microsoft Access.

Синтаксис:
Command

При запуске Microsoft Access из командной строки вся часть командной строки, расположенная после ключа /cmd, передается в программу как ее аргумент. Функция Command возвращает переданный аргумент командной строки.

Для того чтобы изменить аргумент командной строки после открытия базы данных, следует выбрать в меню *Сервис* команду *Параметры* и на вкладке *Другие* ввести новое значение аргумента в поле *Параметры* командной строки. После этого функция Command будет возвращать новое значение аргумента.

При вызове функции Command в любом месте (кроме программы в модуле Visual Basic) следует помещать пустые скобки после имени функции. Например, при вызове функции Command из поля в форме следует ввести в ячейку свойства *Данные* (ControlSource) следующее выражение:

```
=Command()
```

Cos (VBA). Возвращает значение типа Double, содержащее косинус угла.

Синтаксис:
Cos(число)

Обязательный аргумент *число* представляет собой значение типа Double или любое допустимое числовое выражение, задающее угол в радианах.

Функция Cos определяет отношение длин двух сторон прямоугольного треугольника (прилежащего катета и гипотенузы) по указанному углу (в радианах).

Значение, возвращаемое данной функцией, лежит в диапазоне от –1 до 1.

Для преобразования градусов в радианы следует умножить градусы на $\pi/180$, а для преобразования радиан в градусы — радианы на $180/\pi$.

Count (DAO). Вычисляет количество записей, возвращаемых запросом.

Синтаксис:
Count(выражение)

Аргумент *выражение* является строковым выражением, определяющим поле, содержащее данные для подсчета, или выражение, выполняющее вычисления с данными из этого поля. Операнды аргумента *выражение* могут включать в себя имя поля таблицы, константу или функцию. (Функция может быть внутренней или определяться пользователем, но не может быть ни одной из

других статистических функций SQL.) Предназначен для подсчета любых данных, включая текстовые.

Функцию Count используют для подсчета числа записей в базовом запросе. Например, возможно вычисление с помощью функции Count числа заказов, доставленных в конкретную страну.

Хотя аргумент *выражение* допускает выполнение вычислений над полем, функция Count просто возвращает число записей, независимо от того, какие данные содержатся в этих записях.

Функция Count не подсчитывает записи со значениями Null, если только аргумент *выражение* не содержит подстановочный знак — звездочку (*). Если звездочка используется, функция Count вычисляет общее число записей, включая те, которые содержат пустые поля. Функция Count(*) работает значительно быстрее функции Count([Имя столбца]), но не следует заключать звездочку в прямые кавычки (“ ”).

Пример вычисления числа записей в таблице ЗАКАЗЫ:

```
SELECT Count(*)
AS ЧислоЗаказов FROM Заказы
```

Если в аргументе *выражение* задано несколько полей, функция Count подсчитывает запись только в том случае, если хотя бы одно из этих полей не содержит значение Null. Если все указанные поля содержат значения Null, запись не подсчитывается. Для разделения имен полей используется символ (&).

Пример способа ограничения числа записей теми записями, для которых поля *ДатаИсполнения* или *СтоимостьДоставки* не содержат пустых значений:

```
SELECT Count('ДатаИсполнения & СтоимостьДоставки')
AS [Not Null] FROM Заказы
```

Функцию Count можно использовать в выражении запроса, а также в свойстве SQL объекта QueryDef или при создании объекта Recordset на основе запроса SQL.

CreateControl, CreateReportControl. Первая функция создает элемент управления в указанной открытой форме, а вторая — в указанном открытом отчете.

Предположим, что требуется разработать специальную программу мастера, создающего форму определенного типа. В этом случае функция CreateControl обеспечивает добавление мастером элементов управления в создаваемую форму.

Синтаксисы:

CreateControl(имяФормы, типЭлемента[, раздел[, главный[, имяПоля[, слева[, сверху[, ширина[, высота]]]]]]])

CreateReportControl(имяОтчета, типЭлемента[, раздел[, главный[, имяПоля[, слева[, сверху[, ширина[, высота]]]]]]])

Функции CreateControl и CreateReportControl используют аргументы, показанные в табл. П2.3.

Таблица П2.3

Аргументы функций CreateControl и CreateReportControl

Аргумент	Описание
имяФормы, имяОтчета	Строковое выражение, определяющее имя открытой формы или отчета, в которых создается элемент управления
типЭлемента	Одна из перечисленных (далее в таблице) встроенных констант, определяющих тип создаваемого элемента управления. Пользователь имеет возможность просматривать эти константы в окне просмотра объектов и вставлять их в собственные программы. Для этого следует нажать кнопку [Просмотр объектов] на панели инструментов Visual Basic и выбрать Access в поле со списком *Проект/библиотека* и Constants в списке *Классы*
раздел	Одна из перечисленных далее в таблице встроенных констант, определяющих раздел, в котором создается элемент управления
главный	Строковое выражение, определяющее имя главного элемента управления для присоединенного элемента управления. Для элементов управления, не являющихся присоединенными, данный аргумент должен быть пропущен либо его значением может быть пустая строка
имяПоля	Имя поля, с которым связывается данный элемент управления. Если создается элемент управления, не присоединенный к полю, значением данного аргумента должна быть пустая строка
слева, сверху	Числовые выражения, задающие координаты верхнего левого угла элемента управления в единицах твип
ширина, высота	Числовые выражения, задающие ширину и высоту элемента управления в единицах твип
Встроенная константа	*Определяемый элемент управления*
acLabel	Надпись
acRectangle	Прямоугольник
acLine	Линия

Окончание табл. П2.3

Аргумент	Описание
acImage	Рисунок
acCommandButton	Кнопка
acOptionButton	Переключатель
acCheckBox	Флажок
acOptionGroup	Группа параметров
acBoundObjectFrame	Присоединенная рамка объекта
acTextBox	Поле
acListBox	Список
acComboBox	Поле со списком
acSubform	Подчиненная форма
acObjectFrame	Свободная рамка объекта
acPage	Страница
acPageBreak	Конец страницы
acCustomControl	Элемент ActiveX
acToggleButton	Выключатель
acTabCtl	Набор вкладок
Встроенная константа	*Раздел*
acDetail	(По умолчанию) Область данных
acHeader	Заголовок формы или отчета
acFooter	Примечание формы или отчета
acPageHeader	Верхний колонтитул
acPageFooter	Нижний колонтитул
acGroupLevel1Header	Заголовок группы уровня 1 (только в отчетах)
acGroupLevel1Footer	Примечание группы уровня 1 (только в отчетах)
acGroupLevel2Header	Заголовок группы уровня 2 (только в отчетах)
acGroupLevel2Footer	Примечание группы уровня 2 (только в отчетах)

Функции CreateControl и CreateReportControl используются в специальных программах мастеров для создания элементов управления в форме или отчете. Обе эти функции возвращают объект

Control. Применяются они только в режимах конструктора формы и конструктора отчета соответственно.

Аргумент *главный* позволяет определить связь между главным и подчиненным элементами управления. Например, если поле имеет присоединенную подпись, то поле является главным (родительским) элементом управления, а подпись — подчиненным (дочерним). При создании подписи значением аргумента *главный* должна быть строка, указывающая имя главного элемента управления. При создании поля значением аргумента *главный* должна быть пустая строка.

Аргумент *главный* необходимо задавать также для флажков, переключателей или выключателей, помещаемых в группу, которая является главным элементом управления для них. Только подписи, флажки, переключатели и выключатели могут иметь главный элемент управления. Однако все эти элементы управления могут быть созданы и как самостоятельные, т.е. не имеющие главного элемента управления.

Значение аргумента *имяПоля* задается в соответствии с типом создаваемого элемента управления, а также с учетом того, является ли этот элемент присоединенным к полю в таблице. Присоединенными элементами могут быть поле, список, поле со списком, группа и присоединенная рамка объекта. Для переключателя, выключателя и флажка присоединение допускается, если они не входят в группу.

Если указать в аргументе *имяПоля* имя поля таблицы, создается присоединенный элемент управления, все свойства которого автоматически получают значения соответствующих свойств поля таблицы. Например, элемент управления автоматически получает значение свойства *Условие на значение* (ValidationRule).

Мастер, создающий элементы управления в новых или существующих формах или отчетах, должен предварительно открыть документ в режиме конструктора.

Удалить элемент управления из формы или отчета позволяют инструкции DeleteControl и DeleteReportControl.

CreateForm, CreateReport. Первая функция создает форму и возвращает объект Form, вторая — создает отчет и возвращает объект Report.

Предположим, например, что требуется разработать специальную программу мастера, создающего отчет о продажах. В этом случае функция CreateReport обеспечивает создание мастером нового отчета, базирующегося на указанном шаблоне отчета.

Синтаксисы:
CreateForm([базаДанных[, шаблонФормы]])
CreateReport([базаДанных[, шаблонОтчета]])

Таблица П2.4

Аргументы функций CreateForm и CreateReport

Аргумент	Описание
базаДанных	Строковое выражение, задающее имя базы данных, содержащей шаблон, используемый для создания формы или отчета. Если данный аргумент опущен, используется текущая база данных (значение, возвращаемое функцией CurrentDb). Если задана база данных отлична от текущей, то она должна быть открыта как библиотечная
шаблонФормы, шаблонОтчета	Строковые выражения, задающие соответственно имена формам и отчетам, используемым как шаблоны при создании новых форм или отчетов. Если данный аргумент опущен, используется шаблон, указанный на вкладке *Формы/отчеты* в диалоговом окне *Параметры*, вызываемом командой *Параметры* из меню *Сервис*

Функции CreateForm и CreateReport используют аргументы, приведенные в табл. П2.4.

Функции CreateForm и CreateReport используются в специальных программах мастеров для создания новых форм и отчетов.

При вызове функции CreateForm в режиме конструктора открывается новое свернутое в значок окно формы, а при вызове функции CreateReport — окно отчета.

В аргументах *шаблонФормы* и *шаблонОтчета* допускается указание имен форм и отчетов, специально созданных для использования в качестве шаблонов, а также любых других форм и отчетов из базы данных, указанной в аргументе *базаДанных*. Если в аргументе *базаДанных* задана не текущая база, то она должна быть открыта как библиотечная. (Получить дополнительные сведения по библиотечным базам данных можно в руководстве по разработке приложений для Microsoft Access 97.)

Если шаблон формы или отчета создан в другой базе данных, то вместо загрузки библиотечной базы данных этот шаблон может быть импортирован в текущую БД. При этом следует убедиться, что имя шаблона выводится в поле *Шаблон формы* или *Шаблон отчета* на вкладке *Формы/отчеты* в диалоговом окне *Параметры*, так как в дальнейшем формы и отчеты, создаваемые с помощью функций CreateForm или CreateReport, будут базироваться на этом шаблоне.

Если в аргументе *шаблонФормы* или *шаблонОтчета* указано неверное имя, то Visual Basic использует соответственно шаблон формы или отчета, указанный в поле *Шаблон формы* или *Шаблон отчета* на вкладке *Формы/отчеты* в диалоговом окне *Параметры*.

При создании формы и отчета соответственно с помощью функций CreateForm и CreateReport для свойства *Наличие модуля* (HasModule) устанавливается значение False (0). Если требуется, чтобы новая форма и отчет имели модуль класса, то для этого свойства следует установить значение True (–1).

Функции CreateForm и CreateReport создают формы и отчеты, свернутые в значки.

CreateGroupLevel. Задает поле или выражение, по которому выполняется группировка или сортировка данных в отчете.

Предположим, что требуется разработать специальную программу мастера, предоставляющего пользователю при разработке отчета возможность выбора полей, по которым производится группировка данных в отчете. В этом случае следует вызвать в программе мастера функцию CreateGroupLevel для создания групп в соответствии с выбором пользователя.

Синтаксис:
CreateGroupLevel(отчет, выражение, заголовок, примечание)

Функция CreateGroupLevel использует аргументы, приведенные в табл. П2.5.

Функцию CreateGroupLevel используют при разработке программы мастера, создающего отчет с областями групп или итого-

Таблица П2.5

Аргументы функции CreateGroupLevel

Аргумент	Описание
отчет	Строковое выражение, задающее имя отчета, в котором определяется новый уровень группировки
выражение	Строковое выражение, определяющее поле или выражение, по которому выполняется группировка или сортировка данных
заголовок, примечание	Представляют собой значения типа Integer, указывающие, будет ли группа, определяемая полем или выражением, иметь заголовок и(или) примечание группы. Если для аргументов *заголовок* и(или) *примечание* задано значение True (–1), то группа, определяемая полем или выражением, будет иметь присоединенную область заголовка и(или) примечания. При значении False (0) аргументов *заголовок* и(или) *примечание* группа не имеет соответствующих присоединенных областей. Допускается одновременное создание заголовка и примечаний. При этом оба аргумента должны принимать значение True

выми полями. Эта функция задает группировку или сортировку данных по указанному полю или выражению и создание областей заголовка и примечаний для данной группы.

Функция CreateGroupLevel доступна только в режиме конструктора отчета.

В Microsoft Access создаваемые в отчете группы регистрируются в массиве свойства *Уровень группировки* (GroupLevel). При вызове функции CreateGroupLevel в массив добавляется новый уровень группировки, определяемый аргументом *выражение*. Функция CreateGroupLevel возвращает значение, являющееся указателем на позицию нового уровня группировки в массиве. Первому полю или выражению, определяющему группировку, присваивается уровень 0, второму — уровень 1 и т.д. В отчетах поддерживается до десяти уровней группировки (0...9).

Если указать для аргументов *заголовок* и(или) *примечание* значения True, то свойства *Заголовок группы* (GroupHeader) и(или) *Примечание группы* (GroupFooter) получат значение *Да*, и в отчете для группы данного уровня создадутся области заголовка и(или) примечаний.

После создания областей заголовка и(или) примечаний пользователь имеет возможность определить другие свойства для данного уровня группировки: *Группировка* (GroupOn), *Интервал* (GroupInterval) и *Не разрывать* (KeepTogether). Значения этих свойств могут быть заданы в программе Visual Basic или интерактивно в окне *Сортировка и группировка*, которое выводится при нажатии кнопки [Сортировка и группировка] на панели инструментов конструктора отчетов.

Мастер, создающий новый уровень группировки в новом или существующем отчете, должен открыть этот отчет в режиме конструктора.

CreateObject (VBA). Создает и возвращает ссылку на объект ActiveX.

Синтаксис:
CreateObject(класс)

Аргумент *класс* использует синтаксис

имяПриложения.типОбъекта

Элемент *имяПриложения* является обязательным, имеет значение Variant (String) и указывает имя приложения — источника объекта. Элемент *типОбъекта*, является обязательным, имеет значение Variant (String) и представляет собой тип или класс объекта, который следует создать.

Каждое приложение, поддерживающее программирование объектов, может создавать объекты по крайней мере одного типа. Например, текстовым процессором могут быть созданы объект Application (приложение), объект Document (документ) и объект Toolbar (панель инструментов).

Чтобы создать объект ActiveX, следует присвоить объектной переменной объект, возвращенный функцией CreateObject:

```
'Описывает объектную переменную, содержащую ссылку
'на объект
'Предложение Dim as Object задает связывание на позд-
'ней стадии
Dim ExcelSheet As Object
Set ExcelSheet = CreateObject("Excel.Sheet")
```

CreateForm, CreateReport. Первая функция создает форму и возвращает объект Form, а вторая — создает отчет и возвращает объект Report.

Предположим, например, что требуется разработать специальную программу мастера, создающего отчет о продажах. В этом случае функция CreateReport обеспечивает создание мастером нового отчета, базирующегося на указанном шаблоне отчета.

Синтаксисы:
CreateForm([базаДанных[, шаблонФормы]])
CreateReport([базаДанных[, шаблонОтчета]])

Функции CreateForm и CreateReport используют аргументы, представленные в табл. П2.6.

Функции CreateForm и CreateReport используются в специальных программах мастеров для создания соответственно новых форм и отчетов. При вызове в режиме конструктора функции CreateForm открывается новое свернутое в значок окно формы, а при вызове функции CreateReport — окно отчета.

В аргументах *шаблонФормы* и *шаблонОтчета* допускается указывать соответственно имена форм и отчетов, специально созданных для использования в качестве шаблонов, и любых других форм и отчетов из базы данных, указанной в аргументе *базаДанных*. Если в аргументе *базаДанных* задана не текущая база, то она должна быть открыта как библиотечная.

Если шаблон формы или отчета создан в другой базе данных, то вместо загрузки библиотечной БД этот шаблон может быть импортирован в текущую базу. При этом следует убедиться, что имя шаблона выводится в поле *Шаблон формы* или *Шаблон отчета* на вкладке *Формы/отчеты* в диалоговом окне *Параметры*, так как в дальнейшем формы и отчеты, создаваемые с помощью функций

Таблица П2.6

Аргументы функций CreateForm и CreateReport

Аргумент	Описание
базаДанных	Строковое выражение, задающее имя базы данных, содержащей шаблон, используемый для создания формы или отчета. Если данный аргумент опущен, используется текущая база данных (значение, возвращаемое функцией CurrentDb). Если заданная база данных отлична от текущей, то она должна быть открыта как библиотечная
шаблонФормы, шаблонОтчета	Строковые выражения, задающие соответственно имена формы или отчета, используемых как шаблон при создании новой формы или отчета. Если данный аргумент опущен, используется шаблон, указанный на вкладке *Формы/отчеты* в диалоговом окне *Параметры*, открывающемся по команде *Параметры* из меню *Сервис*

CreateForm или CreateReport, будут базироваться на этом шаблоне.

Если в аргументе *шаблонФормы* или *шаблонОтчета* указано неверное имя, то Visual Basic использует соответственно шаблон формы или отчета, указанный в поле *Шаблон формы* или *Шаблон отчета* на вкладке *Формы/отчеты* в диалоговом окне *Параметры*.

При создании формы и отчета соответственно с помощью функций CreateForm и CreateReport для их свойства *Наличие модуля* (HasModule) устанавливается значение False (0). Если требуется, чтобы новая форма и отчет имели модуль класса, то для этого свойства следует установить значение True (–1).

CurDir (VBA). Возвращает значение типа Variant (String), представляющее собой текущий путь.

Синтаксис:
CurDir[(диск)]

Необязательный аргумент *диск* является строковым выражением, указывающим существующий диск. Если диск не указан или аргумент *диск* является пустой строкой (“ ”), функция CurDir возвращает путь к текущему диску.

CurrentDb. Возвращает объектную переменную типа Database, представляющую собой текущую базу данных, открытую в окне Microsoft Access.

Синтаксис:
CurrentDb

При необходимости прямого изменения структуры базы данных и обращения к данным из программы Visual Basic следует использовать объекты доступа к данным. Функция CurrentDb предоставляет способ доступа к текущей базе данных из программы Visual Basic, при котором не требуется знания имени конкретной БД. После создания переменной, указывающей на текущую базу данных, становится возможным доступ к объектам и семействам, входящим в иерархию объектов доступа к данным.

Функция CurrentDb позволяет создать несколько объектных переменных, представляющих собой одну и ту же базу данных.

Приведем пример, где переменные dbsA и dbsB указывают на текущую базу данных:

```
Dim dbsA As Database, dbsB As Database
Set dbsA = CurrentDb
Set dbsB = CurrentDb
```

В ранних версиях Microsoft Access для возвращения указателя текущей базы данных использовался следующий синтаксис:

```
DBEngine.Workspaces(0).Databases(0) или DBEngine(0)(0)
```

В Microsoft Access для Windows 95 вместо него следует использовать функцию CurrentDb, которая создает новый экземпляр текущей базы данных, тогда как конструкция DBEngine(0)(0) представляет собой ссылку на открытую копию текущей БД. С помощью функции CurrentDb пользователь имеет возможность создать несколько объектных переменных типа Database, представляющих собой текущую базу данных. Microsoft Access по-прежнему поддерживает синтаксис вида

```
DBEngine(0)(0)
```

Однако при этом рекомендуется вносить изменения в имеющиеся программы во избежание возможных конфликтов при работе с сетевой базой данных.

При необходимости работы с другой базой данных в то время, когда текущая база данных открыта в окне Microsoft Access, следует использовать метод OpenDatabase объекта Workspace, который не открывает вторую базу данных в окне Microsoft Access, а просто возвращает переменную типа Database, представляющую собой вторую базу данных. Приведем пример возвращения указателя на текущую базу данных и базу данных Contacts.mdb:

```
Dim dbsCurrent As Database, dbsContacts As Database
Set dbsCurrent = CurrentDb
Set dbsContacts = DBEngine.Workspaces(0).Open
Database("Contacts.mdb")
```

CurrentUser. Возвращает имя текущего пользователя базы данных.

Например, функцию CurrentUser используют в процедурах, регистрирующих пользователей, которые вносят изменения в базы данных.

Синтаксис:
CurrentUser

Функция CurrentUser возвращает строку, содержащую имя текущей учетной записи пользователя.

Если не создана защищенная рабочая группа, то функция CurrentUser возвращает имя используемой по умолчанию учетной записи пользователя Admin, которая дает пользователю полные права на все объекты базы данных.

Если включена защита рабочей группы, функция CurrentUser возвращает имя текущей учетной записи пользователя. Для всех учетных записей пользователя кроме Admin необходимо указывать разрешения на объекты базы данных.

Инструкция, которая определяет имя текущего пользователя и выводит его в окно диалога, имеет вид

```
MsgBox("Текущий пользователь: " & CurrentUser)
```

CVErr (VBA). Возвращает значение типа Variant с подтипом Error, содержащее код ошибки, указанный пользователем.

Синтаксис:
CVErr(кодОшибки)

Обязательный аргумент *кодОшибки* является любым допустимым кодом ошибки.

Функция CVErr применяется для создания ошибок, определяемых пользователем, в создаваемых им процедурах. Например, при создании функции, которая принимает несколько аргументов и обычно возвращает строку, имеется возможность проверки того, что введенные аргументы попадают в допустимый диапазон значений. Если же это не так, весьма вероятно, что созданная функция не вернет ожидаемое значение. В этом случае функция CVErr позволит вернуть код ошибки, чтобы определить действия, которые необходимо предпринять.

Следует отметить, что неявное преобразование значения Error недопустимо. Например, не допускается прямое присвоение возвращаемого значения CVErr переменной с типом, отличным от Variant. Однако имеется возможность выполнения явного преобразования (с помощью CInt, CDbl и т.п.) значения, возвращаемого CVErr, и присвоения этому значению переменной с соответствующим типом данных.

Date (VBA). Возвращает значение типа Variant (Date), содержащее текущую системную дату.

Синтаксис:
Date

Для установки системной даты используется инструкция Date.

DateAdd (VBA). Возвращает значение типа Variant (Date), содержащее дату, к которой добавлен указанный временной интервал.

Синтаксис:
DateAdd(interval, number, date)

Синтаксис функции DateAdd содержит аргументы, приведенные в табл. П2.7.

Таблица П2.7

Аргументы функции DateAdd

Аргумент	Описание
interval	Обязательный. Представляет собой строковое выражение, указывающее тип добавляемого временно́го интервала
number	Обязательный. Представляет собой числовое выражение, указывающее число временны́х интервалов, которое следует добавить. Это число может быть положительным (для получения более поздних дат) или отрицательным (для получения более ранних дат)
date	Обязательный. Имеет значение типа Variant (Date) или литерал даты, представляющий собой дату, к которой добавляется указанный временной интервал

Допустимые значения аргумента interval следующие:
уууу — год; Q — квартал; m — месяц; Y — день года; D — день месяца; w — день недели; ww — неделя; H — часы; N — минуты; S — секунды.

Функция DateAdd предназначена для добавления или вычитания указанного временно´го интервала из значения даты. Например, с помощью этой функции можно вычислить дату на 30 дней более позднюю, чем текущая, или время на 45 минут более позднее, чем текущее.

Для добавления дней к аргументу date можно задавать временной интервал как день года ("e"), день месяца ("d") или день недели ("w").

Функция DateAdd не возвращает неправильных дат. Один месяц к 31 января добавляет следующее выражение:

```
DateAdd("m", 1, "31-янв-95")
```

В этом случае будет возвращена дата 28-фев-95, а не 31-фев-95. Если же в качестве аргумента date указать 31-янв-96, то будет возвращена дата 29-фев-96, поскольку 1996 г. является високосным.

Если будет получена дата более ранняя, чем 100-й год (т.е. вычитаемый временной интервал будет содержать больше лет, чем исходное значение date), возникнет ошибка.

Если аргумент number не является значением типа Long, его значение округляется до ближайшего целого числа перед выполнением операции.

DateDiff (VBA). Возвращает значение типа Variant (Long), указывающее число временны́х интервалов между двумя датами.

Синтаксис:
DateDiff(interval, date1, date2[, firstdayofweek[, firstweekofyear]])

Синтаксис функции DateDiff содержит аргументы, приведенные в табл. П2.8.

Таблица П2.8

Аргументы функции DateDiff

Аргумент	Описание
interval	Обязательный. Представляет собой строковое выражение, указывающее тип временно́го интервала, который следует использовать при вычислении разности между датами date1 и date2
date1, date2	Обязательные. Имеют значения типа Variant (Date), т.е. две даты, разность между которыми следует вычислить
firstdayofweek	Необязательный. Представляет собой константу, указывающую первый день недели. Если этот аргумент опущен, считается, что неделя начинается с воскресенья

Аргумент	Описание
firstweekofyear	Необязательный. Представляет собой константу, указывающую первую неделю года. Если этот аргумент опущен, первой неделей считается неделя, содержащая 1 января

Допустимые значения аргумента interval следующие:

yyyy — год; q — квартал; m — месяц; y — день года; d — день месяца; w — день недели; ww — неделя; h — часы; n — минуты; s — секунды.

Аргумент firstdayofweek может определяться строковой константой или иметь числовое значение:

Константа	Значение	Описание
vbUseSystem	0	Используется значение NLS API
vbSunday	1	Воскресенье (по умолчанию)
vbMonday	2	Понедельник
vbTuesday	3	Вторник
vbWednesday	4	Среда
vbThursday	5	Четверг
vbFriday	6	Пятница
vbSaturday	7	Суббота

Аргумент firstweekofyear может определяться строковой константой или иметь числовое значение:

Константа	Значение	Описание
vbUseSystem	0	Используется значение NLS API
vbFirstJan1	1	Неделя, которая содержит 1 января (по умолчанию)
vbFirstFourDays	2	Первая неделя, которая содержит по крайней мере четыре дня нового года
vbFirstFullWeek	3	Первая полная неделя года

Функция DateDiff предназначена для определения числа указанных временных интервалов между двумя датами. Например, с помощью этой функции можно вычислить число дней между двумя датами или число недель между текущей датой и концом года.

Для вычисления числа дней между датами date1 и date2 можно использовать временные интервалы типа день года (“y”) или день месяца (“d”). Если interval задается как день недели (“w”), возвращается число недель между двумя датами. Если date1 соответствует понедельнику, подсчитывается число понедельников между date1 и date2. При этом date2 учитывается, а date1 нет. Если interval задается в неделях (“ww”), функция DateDiff возвращает число календарных недель между двумя датами, т.е. число воскресений между date1 и date2. При этом дата date2 учитывается (если ей соответствует воскресенье), а date1 нет (даже если ей соответствует воскресенье).

Если date1 определяет более позднюю дату, чем date2, возвращается отрицательное значение.

Аргумент firstdayofweek влияет на вычисления, использующие временные интервалы типа “w” и “ww”.

Если дата задается как литерал даты, указанный год становится постоянной ее частью. Однако если дата заключается в прямые кавычки (“ ”), а год опущен, то при каждом вычислении выражения даты в него будет подставляться текущий год. Это позволяет создать код, который может использоваться в течение нескольких лет.

При сравнении дат 31 декабря и 1 января следующего года функция DateDiff для интервала типа год (“yyyy”) возвращает значение 1, хотя разница между датами составляет всего один день.

DatePart (VBA). Возвращает значение типа Variant (Integer), содержащее указанный компонент даты.

Синтаксис:
DatePart(interval, date[,firstdayofweek[, firstweekofyear]])

Синтаксис функции DatePart содержит аргументы, приведенные в табл. П2.9.

Таблица П2.9

Аргументы функции DatePart

Аргумент	Описание
interval	Обязательный. Это строковое выражение, определяющее тип возвращаемого временного интервала
date	Обязательный. Это значение типа Variant (Date), представляющее собой дату, подлежащую обработке
Firstdayofweek	Необязательный. Это константа, указывающая первый день недели. Если этот аргумент опущен, считается, что неделя начинается с воскресенья

Окончание табл. П2.9

Аргумент	Описание
Firstweekofyear	Необязательный. Это константа, указывающая первую неделю года. Если этот аргумент опущен, первой неделей считается неделя, содержащая 1 января

DateSerial (VBA). Возвращает значение типа Variant (Date), соответствующее указанным году, месяцу и дню.

Синтаксис:
DateSerial(year, month, day)

Синтаксис функции DateSerial содержит аргументы, приведенные в табл. П2.10.

Таблица П2.10

Аргументы функции DateSerial

Аргумент	Описание
year	Обязательный. Имеет значение типа Integer, представляющее собой число от 100 до 9999 или числовое выражение
month	Обязательный. Имеет значение типа Integer, представляющее собой любое числовое выражение
day	Обязательный. Имеет значение типа Integer, представляющее собой любое числовое выражение

Значение каждого аргумента функции DateSerial должно находиться в соответствующем диапазоне: 1...31 — дни; 1...12 — месяцы. Кроме того, можно использовать числовые выражения для описания относительной даты, т.е. даты на определенное число дней, месяцев и лет более поздней или ранней, чем указанная.

Рассмотрим пример, аргументами функции DateSerial в котором являются не абсолютные значения, а числовые выражения, т.е. пример определения даты на 1 день 2 месяца и 10 лет более ранний, чем 1 августа 1990 г., т.е. 31 мая 1980 г.:

```
DateSerial(1990 - 10, 8 - 2, 1 - 1)
```

Значения аргумента year, лежащие в интервале от 0 и 99, соответствуют годам двадцатого столетия (1900 — 1999). При описании дат, не относящихся к двадцатому столетию, необходимо использовать все четыре цифры года (например, 1800).

Если значение какого-либо аргумента превышает максимально допустимое для него, то соответствующим образом увеличивается более старший компонент даты. Например, 35 дней означают

1 месяц и несколько оставшихся дней (в зависимости от числа дней в конкретном месяце). Однако если значение любого аргумента лежит вне диапазона 32 768 ... 32 767 или сочетание всех трех аргументов описывает дату, лежащую вне допустимого диапазона дат, возникает ошибка.

DateValue (VBA). Возвращает значение типа Variant (Date).

Синтаксис:
DateValue(дата)

Обязательный аргумент *дата* обычно является строковым выражением, представляющим собой дату в интервале от 1 января 100 года до 31 декабря 9999 года. Кроме того, в качестве этого аргумента можно использовать любое выражение, представляющее собой дату, время или сочетание даты и времени, лежащие в указанном диапазоне.

В строке, содержащей только числа и допустимые разделители компонентов даты, функция DateValue определяет порядок компонентов (месяц, день и год) согласно системному краткому ее формату. Кроме того, функция DateValue правильно обрабатывает допустимые даты, содержащие полные или краткие названия месяцев. Например, все следующие представления даты являются допустимыми: 30.12.1991, 30.12.91, 30 декабря 1991 и 30-дек-91.

Если дата не содержит сведений о годе, функция DateValue использует текущий год по системному календарю компьютера.

Если дата содержит сведения о времени, они не возвращаются функцией DateValue. Однако если дата содержит неправильные сведения о времени (например, «89:98»), возникает ошибка.

DAvg. Функция DAvg возвращает среднее значение для набора значений, принадлежащих указанному подмножеству записей.

Функцию DAvg используют в макросах или программах на Visual Basic, в выражениях для запросов, а также для определения вычисляемого элемента управления.

Например, включение функции DAvg в строку условий запроса на выборку позволяет отобрать записи, в которых значения конкретного поля превышают среднее значение для данного поля. В вычисляемом поле функция DAvg позволяет вывести рядом со значением нового заказа среднее значение по предыдущим заказам.

Синтаксис:
DAvg(выражение, набор[, условие])

Функция DAvg использует аргументы, приведенные в табл. П2.11.

Таблица П2.11

Аргументы функции DAvg

Аргумент	Описание
выражение	Выражение, определяющее поле, которое содержит усредняемые данные. Данный аргумент может задаваться строковым выражением, определяющим поле в таблице или запросе, либо представлять собой выражение, задающее выполнение вычислений над данными, содержащимися в поле. Допускается использование в аргументе *выражение* имени поля в таблице или элемента управления в форме, константы, а также встроенной или определяемой пользователем функции. Не допускается использование в аргументе *выражение* других статистических функций по подмножеству или статистических функций SQL
набор	Строковое выражение, определяющее набор записей, образующих подмножество. Может представлять собой имя таблицы или запроса
условие	Необязательное строковое выражение, ограничивающее диапазон данных, для которых рассчитывается среднее значение. Например, аргумент *условие* часто является эквивалентом предложения WHERE инструкции SQL, но без ключевого слова WHERE. Если аргумент *условие* опущен, DAvg выполняет расчеты над полем, заданным в аргументе *выражение*, для всего набора записей. Любое поле, указанное в аргументе *условие*, должно принадлежать подмножеству, заданному аргументом *набор;* в противном случае функция DAvg возвращает значение Null

Записи, содержащие в указанном поле пустые (Null) значения, в расчет среднего значения не включаются.

При любом использовании функции DAvg в макросе или модуле, в выражении для запроса или вычисляемом элементе управления необходимо обеспечить правильное составление аргумента *условие.*

Допускается ввод функции DAvg в строку *Условие отбора* бланка запроса. Например, предположим, что требуется просмотреть список всех товаров, заказы на которые превышают среднее значение. В этом случае следует создать запрос по таблицам ЗАКАЗЫ, ЗАКАЗАНО и ТОВАРЫ, включить в бланк запроса поля *Марка* и *Количество* и ввести следующее выражение в ячейку строки *Условие отбора* под полем *Количество*:

```
>DAvg("[Количество]", "Заказано")
```

Функция DAvg может быть включена в выражение для вычисляемого поля в запросе, а также в строку *Обновление* запроса на обновление.

В выражениях для вычисляемых полей итоговых запросов используют как функцию DAvg, так и функцию Avg. При использовании функции DAvg расчеты средних значений выполняются до группировки данных. При использовании функции Avg расчеты средних значений выполняются после группировки данных.

Функцию DAvg используют в вычисляемом элементе управления, если необходимо указать условия отбора, ограничивающие диапазон усредняемых данных. Например, для вывода средней стоимости доставки заказов в Крым следует ввести в ячейку свойства *Данные* (ControlSource) вычисляемого поля следующее выражение:

```
= DAvg("[СтоимостьДоставки]", "Заказы", "[Область-
Получателя] = 'Крым'")
```

Если требуется просто найти среднее значение для всех записей в наборе, определяемом аргументом *набор*, следует использовать функцию Avg.

Допускается использование функции DAvg в модуле или макросе либо в вычисляемом элементе управления формы в случае, когда поле, для которого проводится усреднение, не принадлежит к базовому источнику записей формы. Например, предположим, что базовой таблицей формы является таблица ЗАКАЗЫ и что требуется вывести в форме для сравнения среднее значение поля *Количество* из таблицы ЗАКАЗАНО для конкретного клиента. Функция DAvg позволяет выполнить такие расчеты и вывести результат в виде формы.

Day (VBA). Возвращает значение типа Variant (Integer), содержащее целое число (от 1 до 31), которое представляет собой день месяца.

Синтаксис:
Day(дата)

Обязательный аргумент *дата* может быть любым значением типа Variant, числовым выражением, строковым выражением или любым их сочетанием, которое представляет собой дату. Если дата содержит значение Null, возвращается значение Null.

DCount. Возвращает число записей в указанном наборе (подмножестве) записей.

Функцию DCount используют в программах на Visual Basic, макросах, выражениях для запросов, а также для определения вы-

числяемого элемента управления. Например, функция DCount в модуле позволяет подсчитать число записей в таблице ЗАКАЗЫ, сделанных в определенный день.

Синтаксис:
DCount(выражение, набор[, условие])

Функция DCount использует аргументы, приведенные в табл. П2.12.

Функцию DCount применяют для подсчета количества записей в подмножестве, когда не требуется использовать конкретные значения. Хотя в аргументе *выражение* можно указать любые расчеты, DCount всегда возвращает число записей. Значение аргумента *выражение* является недоступным.

При любом использовании функции DCount (в макросе или модуле, выражении для запроса или вычисляемом элементе уп-

Таблица П2.12

Аргументы функции DCount

Аргумент	Описание
выражение	Выражение, определяющее поле, для которого производится подсчет значений. Данный аргумент может задаваться строковым выражением, определяющим поле в таблице или запросе, или представлять собой выражение, задающее выполнение вычислений над данными, содержащимися в поле. Допускается использование в аргументе *выражение* имени поля в таблице или элемента управления в форме, константы, а также встроенной или определяемой пользователем функции. Не допускается использование в аргументе *выражение* других статистических функций по подмножеству или статистических функций SQL
набор	Строковое выражение, определяющее набор записей, образующих подмножество. Может представлять собой имя таблицы или запроса
условие	Необязательное строковое выражение, ограничивающее диапазон данных, для которых подсчитывается число значений. Например, аргумент *условие* часто является эквивалентом предложения WHERE инструкции SQL, но без ключевого слова WHERE. Если аргумент *условие* опущен, DCount выполняет расчеты над полем, заданным в аргументе *выражение*, для всего набора записей. Любое поле, указанное в аргументе *условие*, должно принадлежать подмножеству, заданному аргументом *набор*; в противном случае функция DCount возвращает значение Null

равления) необходимо обеспечить правильное составление аргумента *условие*.

Функцию DCount используют в вычисляемом элементе управления, если необходимо указать условия отбора, ограничивающие диапазон подсчитываемых записей. Например, для вывода числа заказов, отправляемых в Крым, следует ввести в ячейку свойства *Данные* (ControlSource) вычисляемого поля следующее выражение:

```
= DCount("[КодЗаказа]", "Заказы", "[ОбластьПолучателя] = 'Крым'")
```

Если требуется просто подсчитать число записей в наборе, определяемом аргументом *набор*, следует использовать функцию Count.

Функция Count оптимизирована для быстрого подсчета числа записей в запросах, т.е. в запросах вместо функции DCount лучше использовать функцию Count, задавая условия отбора в бланке.

Функцию DCount рекомендуется использовать для подсчета записей в программах модулей, макросах или вычисляемых полях.

Допускается использование функции DCount для подсчета записей, содержащих конкретное поле, в случае, когда поле, для которого проводится подсчет значений, не принадлежит к базовому источнику записей формы. Например, в форме, у которой базовой таблицей является таблица ТОВАРЫ, можно вывести число заказов, содержащихся в таблице ЗАКАЗЫ.

Функция DCount не подсчитывает записи, содержащие пустые (Null) значения в поле, задаваемом аргументом *выражение*, за исключением случая, когда в этом аргументе задан подстановочный знак звездочка (*). При указании данного подстановочного знака DCount подсчитывает полное число записей, в том числе содержащих значения типа Null.

Полное число записей в таблице ЗАКАЗЫ подсчитывается с помощью следующей конструкции:

```
intX = DCount("*", "Заказы")
```

Если в аргументе *набор* указана таблица с определенным ключевым полем, то для подсчета всех записей в этой таблице достаточно указать ключевое поле в аргументе *выражение*, поскольку оно не может содержать значения типа Null.

Если в аргументе *выражение* задаются несколько полей, то их имена следует объединять с помощью оператора конкатенации, представляемого символом амперсанд (&) или знаком плюс (+). Если имена полей разделяются амперсандом, DCount подсчитывает все записи в перечисленных полях. Если используется знак

плюс, то DCount возвращает число записей во всех перечисленных полях с непустыми значениями. Для примера приведем результат применения этих операторов для поля, содержащего данные во всех записях (*НазваниеПолучателя*), и поля, содержащего пустые значения (*ОбластьПолучателя*):

```
1. intW = DCount("[НазваниеПолучателя]", "Заказы")
   'Возвращает 831
2. intX = DCount("[ОбластьПолучателя]", "Заказы")
   'Возвращает 323
3. intY = DCount("[НазваниеПолучателя] + [Область
   Получателя]", "Заказы")
   'Возвращает 323
4. intZ = DCount("[НазваниеПолучателя] & [Область
   Получателя]", "Заказы")
   'Возвращает 831
```

DDB (VBA). Возвращает значение типа Double, указывающее величину амортизации имущества для заданного периода, с использованием метода двукратного учета амортизации или иного явно указанного метода.

Синтаксис:
DDB(cost, salvage, life, period[, factor])

Синтаксис функции DDB содержит аргументы, приведенные в табл. П2.13.

Таблица П2.13

Аргументы функции DDB

Аргумент	Описание
cost	Обязательный. Представляет собой выражение типа Double, указывающее начальную стоимость фондов
salvage	Обязательный. Представляет собой выражение типа Double, указывающее стоимость фондов к концу периода эксплуатации
life	Обязательный. Представляет собой выражение типа Double, указывающее длительность периода эксплуатации
period	Обязательный. Представляет собой выражение типа Double, указывающее период времени, для которого вычисляется снижение стоимости
factor	Необязательный. Представляет собой выражение типа Variant, указывающее фактор расчета снижения стоимости. Если этот аргумент опущен, используется значение по умолчанию 2 (двойное убывание)

Вычисление снижения стоимости по методу с ускоренной амортизацией обеспечивает учет наиболее быстрого снижения стоимости в начальный период эксплуатации.

Аргументы life и period должны задаваться в одних и тех же единицах. Например, если длительность периода эксплуатации life выражена в месяцах, period также следует указывать в месяцах. При этом все аргументы должны иметь положительные значения.

Расчет значения функции DDB производится по следующей формуле:

```
Снижение стоимости за period = ((cost—salvage) *
factor) / life
```

DDE. Позволяет открыть сеанс динамического обмена данными (DDE) с другим приложением, направить требование на прием данных из этого приложения и вывести полученные данные в элементе управления в форме или отчете.

Например, указание функции DDE в свойстве *Данные* (ControlSource) поля в форме позволяет вывести в этом поле содержимое конкретной ячейки электронной таблицы Microsoft Excel.

Синтаксис:
DDE(приложение, документ, раздел)

Функция DDE использует аргументы, приведенные в табл. П2.14.

Таблица П2.14

Аргументы функции DDE

Аргумент	Описание
приложение	Строковое выражение, которое определяет приложение, участвующее в сеансе DDE. Обычно для приложений, работающих в среде Microsoft Windows, аргумент *приложение* задает имя файла .exe (без расширения .exe). Например, для открытия канала связи DDE с Microsoft Excel следует указать "Excell"в аргументе *приложение*
документ	Строковое выражение, содержащее имя документа, принимаемое приложением. В аргументе *документ* обычно указывается документ или файл данных. За списком поддерживаемых приложением имен документов следует обращаться к документации данного приложения
раздел	Строковое выражение, содержащее имя раздела данных, принимаемое приложением. За списком поддерживаемых приложением имен разделов данных следует обращаться к документации данного приложения

При вызове функции DDE делается попытка открыть сеанс связи DDE с указанным приложением, передать соответствующее имя документа и требование на прием данных, указанных в аргументе *раздел*. При успешном выполнении функция DDE возвращает строку, содержащую затребованные данные.

В требовании на прием данных из Microsoft Excel аргумент *раздел* должен содержать идентификатор адреса ячейки (строки и столбца, например, "R1C1") или имя диапазона ячеек. Например, для передачи функцией DDE требования на прием содержимого ячейки, находящейся в первой строке и первом столбце электронной таблицы Microsoft Excel, в ячейку *Данные* (ControlSource) в окне свойств поля вводится следующее выражение:

```
=DDE("Excel", "Лист1", "R1C1")
```

Функция DDE используется только для указания свойства *Данные* (ControlSource) поля, группы, флажка или поля со списком. Не допускается вызов функции DDE в инструкциях Visual Basic.

После ввода функции DDE элемент управления становится нередактируемым в режимах формы и предварительного просмотра. Например, после ввода функции DDE в поле оно становится нередактируемым, т.е. содержимое этого поля следует редактировать в другом приложении. Поскольку свойство *Данные* (ControlSource) становится в режимах формы и предварительного просмотра доступным только для чтения, все изменения должны вноситься в элемент управления в режиме конструктора.

Максимальное число одновременно открываемых сеансов DDE определяется настройками Windows, а также памятью и ресурсами компьютера. Если попытка открыть сеанс оказалась неудачной из-за того, что приложение не запущено или не распознается документ, или превышено максимально допустимое число сеансов, функция DDE возвращает значение Null.

В конфигурации другого приложения может быть указано, что запросы на открытие сеанса DDE игнорируются. В этом случае функция DDE возвращает Null. Аналогично в настройках Microsoft Access можно указать игнорирование запросов на открытие сеансов из других приложений, т.е. надо выбрать в меню *Сервис* команду *Параметры* и вкладку *Другие*, а в группе *Операции DDE* установить флажок *Пропуск команд DDE*.

Совет. Чтобы работать с объектами другого приложения из Microsoft Access, следует использовать механизм программирования объектов.

В табл. П2.15 показано использование функции DDE с разными элементами управления.

Таблица П2.15

Использование функции DDE с разными элементами управления

Элемент управления	Описание
Поле	Аргумент *раздел* может представлять собой текст или числа. Если он представляет несколько элементов данных (например, имя диапазона, содержащего несколько ячеек электронной таблицы Microsoft Excel), то функция DDE возвращает содержимое первого элемента. Функция DDE позволяет также вывести в поле содержимое ячейки электронной таблицы
Поле со списком	Функция DDE заполняет список данными, указанными в аргументе *раздел*. Не допускается ввод этих данных в поле.Функция DDE позволяет также вывести в поле со списком список значений, сохраняемых в электронной таблице Microsoft Excel
Группа параметров	Свойство *Значение параметра* (OptionValue) каждого из переключателей в группе имеет числовое значение. Обычно первый переключатель имеет значение 1, второй — значение 2 и т.д.Числовое значение, возвращаемое функцией DDE, определяет, какой из переключателей будет выбран. Например, если функция DDE возвращает значение 2, — это значит, что будет включен второй переключатель; если же функция DDE возвращает значение, не соответствующее ни одному из свойств *Значение параметра* (OptionValue), — не будет выбран ни один из переключателей. Если аргумент *раздел* представляет собой несколько элементов данных (например, имя диапазона, содержащего несколько ячеек электронной таблицы Microsoft Excel), то функция DDE возвращает содержимое первого элемента
Флажок	Если функция DDE возвращает 0, флажок будет снят. Если функция DDE возвращает любое ненулевое значение (например, 1 или –1), флажок будет установлен. Если аргумент *раздел* представляет собой несколько элементов данных (например, имя диапазона, содержащего несколько ячеек электронной таблицы Microsoft Excel), состояние флажка становится неопределенным

DDEInitiate. Позволяет открыть сеанс динамического обмена данными (DDE) с другим приложением и открывает канал связи DDE, обеспечивающий передачу данных между сервером DDE и приложением-клиентом.

Например, для передачи данных из электронной таблицы Microsoft Excel в базу данных Microsoft Access следует открыть канал связи между двумя приложениями с помощью функции DDEInitiate. В этом случае Microsoft Access будет выполнять роль приложения-клиента, а Excel — приложения-сервера.

Синтаксис:
DDEInitiate(приложение, документ)

Функция DDEInitiate использует аргументы, приведенные в табл. П2.16.

Таблица П2.16

Аргументы функции DDEInitiate

Аргумент	Описание
приложение	Строковое выражение, которое определяет приложение, участвующее в сеансе DDE. Обычно для приложений, работающих в среде Microsoft Windows (например, Microsoft Excel), аргумент *приложение* задает имя файла .exe (без расширения .exe)
документ	Строковое выражение, содержащее имя документа, принимаемое приложением. За списком поддерживаемых приложением имен документов следует обращаться к документации данного приложения

При успешном выполнении функция DDEInitiate открывает сеанс связи DDE с указанными приложением и документом и возвращает значение типа Long. Данное значение представляет собой уникальный номер, определяющий канал, по которому будет проводиться обмен данными. Этот номер канала будет использоваться в аргументах других функций и инструкций DDE.

Если приложение еще не запущено, а также если запущенное приложение не принимает аргумент *документ* или не поддерживает протокол DDE, функция DDEInitiate возвращает ошибку при выполнении.

Допустимые значения аргумента *документ* определяются приложением. В приложениях, использующих документы или файлы данных, допустимыми значениями данного аргумента обычно являются имена этих файлов.

Максимально возможное число одновременно открытых каналов определяется настройками Windows, а также системной памятью и ресурсами компьютера. Если канал не используется, то

для экономии ресурсов следует закрыть его с помощью инструкций DDETerminate или DDETerminateAll.

DDERequest. Передает в приложение-сервер по открытому каналу динамического обмена данными (DDE) требование на прием данных из указанного раздела.

Например, если открыт канал связи DDE между Microsoft Access и Microsoft Excel, функция DDERequest позволяет передать текст из электронной таблицы Microsoft Excel в базу данных Microsoft Access. В этом случае Microsoft Access будет выполнять роль приложения-клиента, а Excel — приложения-сервера.

Синтаксис:
DDERequest(канал, раздел)

Функция DDERequest использует аргументы, приведенные в табл. П2.17.

Таблица П2.17

Аргументы функции DDERequest

Аргумент	Описание
канал	Номер канала. Это целое значение, возвращаемое функцией DDEInitiate
раздел	Строковое выражение, содержащее имя раздела данных, принимаемое приложением, определенным при вызове функции DDEInitiate.За списком поддерживаемых приложением имен разделов данных следует обращаться к документации данного приложения

Аргумент *канал* задает номер используемого канала связи DDE, а аргумент *раздел* определяет данные, загружаемые из приложения-сервера. Допустимые значения аргумента *раздел* определяются именами приложения и документа, указанными при открытии канала. Например, разделом может быть диапазон ячеек электронной таблицы Microsoft Excel.

При успешном выполнении функция DDERequest возвращает значение типа Variant в виде строки, содержащей затребованные данные.

Допускается прием данных только в обычном текстовом формате. Прием рисунков или текста в другом формате не поддерживается.

Аргумент *канал* должен представлять собой целое число, совпадающее с номером открытого канала. При указании другого значения, а также при невозможности передачи затребованных данных возникает ошибка при выполнении.

DDESend. Позволяет открыть сеанс динамического обмена данными (DDE) с другим приложением и передать в это приложение данные из элемента управления в форме или отчете.

Например, функция DDESend вводится в ячейку свойства *Данные* (ControlSource) поля, чтобы передать данные из этого поля в ячейку электронной таблицы Microsoft Excel.

Синтаксис:
DDESend(приложение, документ, раздел, данные)

Функция DDESend использует аргументы, приведенные в табл. П2.18.

Таблица П2.18

Аргументы функции DDESend

Аргумент	Описание
приложение	Строковое выражение, которое определяет приложение, участвующее в сеансе DDE. Обычно для приложений, работающих в среде Microsoft Windows, аргумент приложение задает имя файла .exe (без расширения .exe). Например, для открытия канала связи DDE с Microsoft Excel следует указать "Excel" в аргументе *приложение*
документ	Строковое выражение, содержащее имя документа, принимаемое приложением. В аргументе *документ* обычно указывается документ или файл данных. За списком поддерживаемых приложением имен документов следует обращаться к документации данного приложения
раздел	Строковое выражение, содержащее имя раздела данных, принимаемое приложением. За списком поддерживаемых приложением имен разделов данных следует обращаться к документации данного приложения
данные	Строка или выражение, содержащие данные, передаваемые в приложение

При вызове функции DDESend делается попытка открыть сеанс связи DDE с указанным приложением, передать соответствующее имя документа и указать раздел, который должен принять данные. Например, если аргумент *приложение* определяет Microsoft Excel, документ может иметь вид "Лист1", а раздел представлять собой идентификатор адреса ячейки (строки и столбца, например, "R1C1") или имя диапазона ячеек.

Аргумент *данные* определяет передаваемые данные и может содержать строку (например, "Отчет подготовил В.Сидоров") или выражение, включающее в себя функцию как часть результирующей строки (например, "Отчет подготовлен" & Date()). Если в

аргументе *раздел* указывается несколько адресов элементов (например, имя диапазона ячеек электронной таблицы Microsoft Excel), функция DDESend передает данные в первый элемент.

Приведем пример передачи функцией DDESend строки "Некий текст" в первую ячейку электронной таблицы Microsoft Excel. Данное выражение может быть введено в ячейку *Данные* (ControlSource) в окне свойств поля в следующем виде:

```
=DDESend("Excel", "Лист1", "R1C1", "Некий текст")
```

Предположим теперь, что требуется передать данные из присоединенного элемента управления, находящегося в форме Microsoft Access, в ячейку электронной таблицы Microsoft Excel. Если в свойстве *Данные* (ControlSource) связанного элемента управления уже указано имя поля или выражения, следует создать еще одно поле или поле со списком, задать для него в свойстве *Данные* выражение, включающее в себя функцию DDESend, и указать в аргументе *данные* имя связанного элемента управления.

Например, если связанным является поле *Фамилия*, то в ячейку *Данные* второго поля следует ввести следующее выражение:

```
=DDESend("Excel", "Лист1", "R1C1", [Фамилия])
```

В качестве промежуточного элемента управления необходимо использовать поле или поле со списком. Не допускается указывать имя связанного элемента управления в аргументе *данные* для флажка или группы переключателей.

Функция DDESend используется только для указания свойства *Данные* (ControlSource) поля, группы, флажка или поля со списком. Не допускается вызов функции DDESend в инструкциях Visual Basic.

После ввода функции DDESend элемент управления становится нередактируемым в режимах формы и предварительного просмотра. Поскольку свойство *Данные* (ControlSource) становится в режимах формы и предварительного просмотра доступным только для чтения, все изменения должны вноситься в элемент управления в режиме конструктора.

Максимальное число одновременно открываемых сеансов DDE определяется настройками Windows, а также памятью и ресурсами компьютера. Если попытка открыть сеанс оказалась неудачной из-за того, что приложение не запущено, не распознается документ или превышено максимально допустимое число сеансов, функция DDESend возвращает значение Null.

DFirst, DLast. Используют для возвращения значений из случайно выбранных записей определенного поля в таблице или запросе. Функции DFirst и DLast применяются в макросах, модулях,

выражениях для запросов и вычисляемых элементах управления формы или отчета.

Синтаксисы:
DFirst(выражение, набор[, условие])
DLast(выражение, набор[, условие])

Функции DFirst и DLast используют аргументы, приведенные в табл. П2.19.

Таблица П2.19

Аргументы функций DFirst и DLast

Аргумент	Описание
выражение	Выражение, определяющее поле, в котором производится поиск первого или последнего значения. Данный аргумент может задаваться строковым выражением, определяющим поле в таблице или запросе, или представлять собой выражение, задающее выполнение вычислений над данными, содержащимися в поле. Допускается использование в аргументе *выражение* имени поля в таблице или элемента управления в форме, константы, а также встроенной или определяемой пользователем функции. Не допускается использование в аргументе *выражение* других статистических функций по подмножеству или статистических функций SQL
набор	Строковое выражение, определяющее набор записей, образующих подмножество
условие	Необязательное строковое выражение, ограничивающее диапазон данных, для которых определяется значение первого или последнего поля. Например, аргумент *условие* часто является эквивалентом предложения WHERE инструкции SQL, но без ключевого слова WHERE. Если аргумент *условие* опущен, Dfirst и DLast выполняют действия над полем, заданным в аргументе *выражение*, для всего набора записей. Любое поле, указанное в аргументе *условие*, должно принадлежать подмножеству, заданному аргументом *набор*; в противном случае функции Dfirst и Dlast возвращают значение Null

Dir (VBA). Возвращает значение типа String, представляющее собой имя файла, каталога или папки, которое удовлетворяет указанному шаблону имени файла, набору атрибутов файла или метке тома на диске.

Синтаксис:
Dir[(путь[, атрибуты])]

Синтаксис функции Dir содержит аргументы, приведенные в табл. П2.20.

Таблица П2.20

Аргументы функции Dir

Аргумент	Описание
путь	Необязательный. Представляет собой строковое выражение, указывающее имя файла; может содержать имя каталога или папки и диска. Если аргумент *путь* не найден, возвращается пустая строка (“ ”)
атрибуты	Необязательный. Представляет собой константу или числовое выражение, описывающее атрибуты файла. Если этот аргумент опущен, возвращаются все файлы, имена которых удовлетворяют аргументу *путь*

Аргумент *атрибуты* может определяться строковой константой или иметь числовое значение:

Константа	Значение	Описание
vbNormal	0	Обычный
vbHidden	2	Скрытый
vbSystem	4	Системный
vbVolume	8	Метка тома; если она указана, все остальные атрибуты игнорируются
vbDirectory	16	Каталог или папка

Данные константы определяются в языке Visual Basic для приложений. Это означает, что их имена можно использовать в любом месте кода вместо фактических значений.

Функция Dir поддерживает использование подстановочных знаков для нескольких символов (*) и одиночного символа (?) с целью указания нескольких файлов.

При первом вызове функции Dir необходимо указать аргумент *путь*, в противном случае возникает ошибка. Если указаны атрибуты файла, наличие аргумента *путь* является обязательным.

Функция Dir возвращает первое имя файла, которое удовлетворяет аргументу *путь*. Для получения остальных файлов, имена которых удовлетворяют указанному пути, следует повторно вызвать функцию Dir без аргументов. Если файлов, имена которых удовлетворяют указанному пути, не осталось, возвращается пустая строка (“ ”). При следующем после возврата пустой строки вызове функции необходимо указать аргумент *путь*; в противном случае возникает ошибка. Изменить значение аргумента *путь* можно

в любой момент, не дожидаясь, пока закончатся файлы, имена которых удовлетворяют текущему пути. Рекурсивный вызов функции Dir запрещен. Вызов функции Dir с атрибутом vbDirectory не приводит к последовательному возврату подкаталогов.

DFirst, DLast. Используют для возвращения значений из случайно выбранных записей определенного поля в таблице или запросе.

Функции DFirst и DLast применяются в макросах, модулях, выражениях для запросов и вычисляемых элементах управления формы или отчета.

Синтаксисы:
DFirst(выражение, набор[, условие])
DLast(выражение, набор[, условие])

Функции DFirst и DLast используют аргументы, приведенные в табл. П2.21.

Таблица П2.21

Аргументы функций DFirst, DLast

Аргумент	Описание
выражение	Выражение, определяющее поле, в котором производится поиск первого или последнего значения. Данный аргумент может задаваться строковым выражением, определяющим поле в таблице или запросе, или представлять собой выражение, задающее выполнение вычислений над данными, содержащимися в поле. Допускается использование в аргументе *выражение* имени поля в таблице или элемента управления в форме, константы, а также встроенной или определяемой пользователем функции. Не допускается использование в аргументе *выражение* других статистических функций по подмножеству или статистических функций SQL
набор	Строковое выражение, определяющее набор записей, образующих подмножество
условие	Необязательное строковое выражение, ограничивающее диапазон данных, для которых определяется значение первого или последнего поля. Например, аргумент *условие* часто является эквивалентом предложения WHERE инструкции SQL, но без ключевого слова WHERE. Если аргумент *условие* опущен, DFirst и DLast выполняют действия над полем, заданным в аргументе *выражение*, для всего набора записей. Любое поле, указанное в аргументе *условие*, должно принадлежать подмножеству, заданному аргументом *набор*; в противном случае функции Dfirst и DLast возвращают значение Null

DLookup. Возвращает значение конкретного поля в указанном наборе (подмножестве) записей.

Функцию DLookup используют в программах на Visual Basic, макросах, выражениях для запросов, а также для определения вычисляемого элемента управления в форме или отчете.

Допускается использование функции DLookup для вывода значения поля в случае, когда это поле не принадлежит к базовому источнику записей формы или отчета. Предположим, например, что базовой таблицей формы является таблица ЗАКАЗАНО. В форме выводятся поля *КодЗаказа*, *КодТовара*, *Цена*, *Количество* и *Скидка*. Однако поле *Марка* находится в таблице ТОВАРЫ. В этом случае функция DLookup позволяет создать в этой форме вычисляемое поле, в котором будет выводиться значение поля *Марка*.

Синтаксис:
DLookup(выражение, набор[, условие])

Функция DLookup использует аргументы, приведенные в табл. П2.22.

Таблица П2.22

Аргументы функции DLookup

Аргумент	Описание
выражение	Выражение, определяющее нужное поле. Данный аргумент может задаваться строковым выражением, определяющим поле в таблице или запросе, или представлять собой выражение, задающее выполнение вычислений над данными, содержащимися в поле. Допускается использование в аргументе *выражение* имени поля в таблице или элемента управления в форме, константы, а также встроенной или определяемой пользователем функции. Не допускается использование в аргументе *выражение* других статистических функций по подмножеству или статистических функций SQL
набор	Строковое выражение, определяющее набор записей, образующих подмножество. Может представлять собой имя таблицы или запроса
условие	Необязательное строковое выражение, ограничивающее диапазон данных, в которых производится поиск значений. Например, аргумент *условие* часто является эквивалентом предложения WHERE инструкции SQL, но без ключевого слова WHERE. Если аргумент *условие* опущен, DLookup выполняет действия над полем, заданным в аргументе *выражение*, для всего набора записей. Любое поле, указанное в аргументе *условие*, должно принадлежать подмножеству, заданному аргументом *набор*; в противном случае функция DLookup возвращает значение Null

DLookup возвращает значение поля или выражения, определенного в аргументе *выражение*. Выбираемые в таблице или запросе значения принадлежат подмножеству записей, определяемых аргументом *набор* и удовлетворяющих условиям отбора, задаваемым в аргументе *условие*.

Если ни одна из записей набора не удовлетворяет аргументу *условие* или набор не содержит записей, функция DLookup возвращает значение Null.

Если указанным условиям удовлетворяет несколько полей, DLookup возвращает значение первого найденного поля. Рекомендуется указывать условия, обеспечивающие уникальность значения, возвращаемого функцией DLookup. Одним из способов обеспечения уникальности возвращаемых значений является указание условий для ключевого поля, например для поля *КодСотрудника* :

```
Dim varX As Variant
varX = DLookup("[Фамилия]", "Сотрудники", "[КодСотрудника] = 1")
```

DMax, DMin. Возвращают минимальное и максимальное значения поля в указанном наборе (подмножестве) записей.

Функции DMin и DMax используют в макросах или программах на Visual Basic, выражениях для запросов, а также для определения вычисляемого элемента управления.

Например, с помощью функций DMin и DMax определяют вычисляемые поля в отчете, в которые выводятся минимальная и максимальная суммы заказов конкретного клиента. Функцию DMin можно включить в запросе в выражение, с помощью которого будут отбираться заказы со скидкой, превышающей минимально допустимую.

Синтаксисы:
DMin(выражение, набор[, условие])
DMax(выражение, набор[, условие])

Функции DMin и DMax используют аргументы, приведенные в табл. П2.23.

Таблица П2.23

Аргументы функций DMin, DMax

Аргумент	Описание
выражение	Выражение, определяющее нужное поле. Данный аргумент может задаваться строковым выражением, определяющим поле в таблице или запросе, или представлять собой выражение, задающее выполнение вычислений над данными,

Окончание табл. П2.23

Аргумент	Описание
выражение	содержащимися в поле. Допускается использование в аргументе *выражение* имени поля в таблице или элемента управления в форме, константы, а также встроенной или определяемой пользователем функции. Не допускается использование в аргументе *выражение* других статистических функций по подмножеству или статистических функций SQL
набор	Строковое выражение, определяющее набор записей, образующих подмножество. Может представлять собой имя таблицы или запроса
условие	Необязательное строковое выражение, ограничивающее диапазон данных, для которых определяется минимальное или максимальное значение поля. Например, аргумент *условие* часто является эквивалентом предложения WHERE инструкции SQL, но без ключевого слова WHERE. Если аргумент *условие* опущен, DMin и DMax выполняют действия над полем, заданным в аргументе *выражение*, для всего набора записей. Любое поле, указанное в аргументе *условие*, должно принадлежать подмножеству, заданному аргументом *набор*; в противном случае функции DMin и DMax возвращают значение Null

Минимальное и максимальное значения поля отбираются в соответствии с условиями, указанными в аргументе *условие*. Если выражение определяет числовые данные, DMin и DMax возвращают числовые значения. Если выражение определяет строковые значения, то возвращается строка, являющаяся первой или последней в алфавитном порядке.

Пустые (Null) значения поля, определяемого аргументом *выражение*, игнорируются. Однако, если ни одна из записей набора не удовлетворяет аргументу *условие* или аргумент *набор* не содержит записей, функции DMin и DMax возвращают значение Null.

При любом использовании функций DMin или DMax в макросе, функции или модуле, выражении для запросов или вычисляемом элементе управления необходимо обеспечить правильное составление аргумента *условие*.

Функции DMin и DMax могут быть включены в строку *Условие отбора* бланка запроса, выражение для вычисляемого поля, а также в строку *Обновление запроса на обновление*.

В выражениях для вычисляемых полей в итоговых запросах используют как функции DMin и DMax, так и функции Min и Max. При использовании функций DMin и DMax значения находятся до группировки данных. При использовании функций Min и Max сначала выполняется группировка данных, а потом определяются соответственно минимальное и максимальное значения.

Функции DMin и DMax используют в вычисляемом элементе управления, если необходимо указать условия отбора, ограничивающие диапазон данных. Например, чтобы определить максимальную стоимость доставки заказа в Крым, следует ввести в ячейку свойства *Данные* (ControlSource) поля формы следующее выражение:

```
=DMax("[СтоимостьДоставки]", "Заказы", "[ОбластьПолучателя] = 'Крым'")
```

Если требуется просто найти минимальное или максимальное значение поля для всех записей в наборе, определяемом аргументом *набор*, следует использовать соответственно функции Min или Max.

Допускается использование функций DMin и DMax в модуле, макросе или вычисляемом элементе управления в форме, когда поле, для которого проводится поиск первого или последнего значения, не принадлежит к базовому источнику записей формы.

DoEvents (VBA). Передает управление операционной системе для обработки других событий.

Синтаксис:
DoEvents( )

Функция DoEvents возвращает значение типа Integer, представляющее собой число открытых форм в независимо установленных версиях языка Visual Basic. Во всех других приложениях функция DoEvents возвращает нуль.

Если часть программы занимает основное процессорное время, следует периодически использовать функцию DoEvents для отказа от управления в пользу операционной системы, чтобы такие события, как ввод данных с клавиатуры и нажатия кнопок мыши, обрабатывались без существенной задержки.

DstDev, DstDevP. Возвращают значение среднеквадратичного отклонения для выборки или ограниченной выборки значений, содержащихся в указанном наборе записей (подмножестве).

Функции DStDev и DStDevP используют в макросах, программах на Visual Basic, выражениях для запросов, а также для определения вычисляемого элемента управления в форме или отчете.

DStDevP возвращает смещенное значение среднеквадратичного отклонения, а DStDev — несмещенное значение (ограниченную выборку).

Например, функцию DStDev можно использовать в модуле для расчета среднеквадратичного отклонения студенческих экзаменационных оценок.

Синтаксисы:
DStDev(выражение, набор[, условие])
DStDevP(выражение, набор[, условие])

Функции DStDev и DStDevP используют аргументы, приведенные в табл. П2.24.

Таблица П2.24

Аргументы функций DStDev, DStDevP

Аргумент	Описание
выражение	Выражение, определяющее нужное поле. Данный аргумент может задаваться строковым выражением, определяющим поле в таблице или запросе, или представлять собой выражение, задающее выполнение вычислений над данными, содержащимися в поле. Допускается использование в аргументе *выражение* имени поля в таблице или элемента управления в форме, константы, а также встроенной или определяемой пользователем функции. Не допускается использование в аргументе *выражение* других статистических функций по подмножеству или статистических функций SQL
набор	Строковое выражение, определяющее набор записей, образующих подмножество. Может представлять собой имя таблицы или запроса
условие	Необязательное строковое выражение, ограничивающее диапазон данных, для которых определяется среднеквадратичное отклонение. Например, аргумент *условие* часто является эквивалентом предложения WHERE инструкции SQL, но без ключевого слова WHERE. Если аргумент *условие* опущен, DStDev и DStDevP выполняют действия над полем, заданным в аргументе *выражение*, для всего набора записей. Любое поле, указанное в аргументе *условие*, должно принадлежать подмножеству, заданному аргументом *набор*; в противном случае функции DStDev и DStDevP возвращают значение Null

Если аргумент *набор* определяет меньше двух записей или если меньше двух записей удовлетворяют аргументу *условие*, DStDev и DStDevP возвращают пустое (Null) значение, показывающее, что расчет среднеквадратичного отклонения невозможен.

При любом использовании функций DStDev или DStDevP в макросе, функции, модуле, выражении для запроса или вычисляемом элементе управления необходимо обеспечить правильное составление аргумента *условие*.

Функции DStDev и DStDevP могут быть включены в строку *Условие отбора* бланка запроса на выборку. Например, они позволяют создать запрос по таблицам ЗАКАЗЫ и ТОВАРЫ, в котором будет выводиться список всех товаров, для которых стоимость доставки превышает сумму средней стоимости и среднеквадратичного отклонения стоимости доставки. В этом случае в ячейку строки условий для поля *СтоимостьДоставки* следует ввести следующее выражение:

```
>(DStDev("[СтоимостьДоставки]", "Заказы") + DAvg("[Сто-
имостьДоставки]", "Заказы"))
```

Функции DStDev и DStDevP могут быть также включены в выражение для вычисляемого поля и строку *Обновление запроса на обновление*.

В выражениях для вычисляемых полей в итоговых запросах используют как функции DStDev и DStDevP, так и функции StDev и StDevP. При использовании функций DStDev и DStDevP значения находятся до группировки данных. При использовании функций StDev и StDevP сначала выполняется группировка данных, а потом определяются значения среднеквадратичного отклонения.

Функции DStDev и DStDevP используют в вычисляемом элементе управления, если необходимо указать условия отбора, ограничивающие диапазон данных. Например, для вывода среднеквадратичного отклонения стоимости доставки заказов в Крым следует ввести в ячейку свойства *Данные* (ControlSource) поля формы следующее выражение:

```
=DStDev("[СтоимостьДоставки]", "Заказы", "[Область-
Получателя] = 'Крым'")
```

DSum. Возвращает сумму набора значений в указанном наборе (подмножестве) записей.

Функцию DSum используют в макросах, программах на Visual Basic, выражениях для запросов, а также для определения вычисляемого элемента управления.

Например, функция DSum позволяет подсчитать в запросе общую сумму продаж, совершенных конкретным сотрудником за указанный период времени, или создать вычисляемое поле, в котором выводится сумма с накоплением для продаж конкретного товара.

Синтаксис:
DSum(выражение, набор[, условие])

Функция DSum использует аргументы, приведенные в табл. П2.25.

Таблица П2.25

Аргументы функции DSum

Аргумент	Описание
выражение	Выражение, определяющее нужное поле. Данный аргумент может задаваться строковым выражением, определяющим поле в таблице или запросе, или представлять собой выражение, задающее выполнение вычислений над данными, содержащимися в поле. Допускается использование в аргументе *выражение* имени поля в таблице или элемента управления в форме, константы, а также встроенной или определяемой пользователем функции. Не допускается использование в аргументе *выражение* других статистических функций по подмножеству или статистических функций SQL
набор	Строковое выражение, определяющее набор записей, образующих подмножество. Может представлять собой имя таблицы или запроса
условие	Необязательное строковое выражение, ограничивающее диапазон данных, для которых производится суммирование значений. Например, аргумент *условие* часто является эквивалентом предложения WHERE инструкции SQL, но без ключевого слова WHERE. Если аргумент *условие* опущен, DSum выполняет расчеты над полем, заданным в аргументе *выражение*, для всего набора записей. Любое поле, указанное в аргументе *условие*, должно принадлежать подмножеству, заданному аргументом *набор*; в противном случае функция DSum возвращает значение Null

DVar, DVarP. Возвращают значение дисперсии для выборки или ограниченной выборки значений, содержащихся в указанном наборе записей (подмножестве).

Функции DVar и DVarP используют в макросах, программах на Visual Basic, выражениях для запросов, а также для определения вычисляемого элемента управления в форме или отчете.

DVarP возвращает значение смещенной дисперсии, а DVar — несмещенной дисперсии.

Например, функцию DVar можно использовать в модуле для расчета разброса студенческих экзаменационных оценок.

Синтаксисы:
DVar(выражение, набор[, условие])
DVarP(выражение, набор[, условие])

Функции DVar и DVarP используют аргументы, приведенные в табл. П2.26.

Таблица П2.26

Аргументы функций DVar и DVarP

Аргумент	Описание
выражение	Выражение, определяющее нужное поле. Данный аргумент может задаваться строковым выражением, определяющим поле в таблице или запросе, или представлять собой выражение, задающее выполнение вычислений над данными, содержащимися в поле. Допускается использование в аргументе *выражение* имени поля в таблице или элемента управления в форме, константы, а акже встроенной или определяемой пользователем функции.Не допускается использование в аргументе *выражение* других статистических функций по подмножеству или статистических функций SQL
набор	Строковое выражение, определяющее набор записей, образующих подмножество. Может представлять собой имя таблицы или запроса
условие	Необязательное строковое выражение, ограничивающее диапазон данных, для которых определяется дисперсия. Например, аргумент *условие* часто является эквивалентом предложения WHERE инструкции SQL, но без ключевого слова WHERE. Если аргумент *условие* опущен, DVar и DVarP выполняют действия над полем, заданным в аргументе *выражение*, для всего набора записей. Любое поле, указанное в аргументе *условие*, должно принадлежать подмножеству, заданному аргументом *набор*; в противном случае функции DVar и DVarP возвращают значение Null

Environ (VBA). Возвращает значение типа String, содержащее переменную среды операционной системы.

Синтаксис:
Environ({envstring | number})

Синтаксис функции Environ содержит аргументы, представленные в табл. П.2.27.

Таблица П2.27

Аргументы функции Environ

Аргумент	Описание
envstring	Необязательный. Представляет собой строковое выражение, содержащее имя переменной среды
number	Необязательный. Представляет собой числовое выражение, соответствующее номеру нужной переменной среды в таблице переменных среды. Может задаваться любым числовым выражением и округляется до ближайшего целого числа перед выполнением операции

Если переменная с указанным именем не найдена в таблице переменных среды, возвращается пустая строка (“ ”). В противном случае функция Environ возвращает текст, связанный с указанной переменной, т.е. весь текст, расположенный после знака равенства в строке таблицы переменных среды, соответствующей этой переменной.

Если указан аргумент number, возвращается строка, занимающая указанную позицию в таблице переменных среды, т.е. в этом случае возвращается весь текст, в том числе envstring. Если указанная позиция пуста, функция Environ возвращает пустую строку.

EOF (VBA). Возвращает значение типа Integer, содержащее логическое значение True, при достижении конца файла.

Синтаксис:
EOF(номерФайла)

Обязательный аргумент *номерФайла* является выражением типа Integer, содержащим любой допустимый номер файла.

С помощью функции EOF можно избежать ошибок, возникающих при попытках чтения или записи после достижения конца файла.

Функция EOF возвращает значение False до достижения конца файла. При использовании с файлами, открытыми в режиме Random или Binary, функция EOF возвращает значение True, если последней выполненной инструкции Get не удалось считать целую запись; в противном случае возвращается значение False.

Попытка чтения файлов, открытых для доступа в режиме Binary, с помощью функции Input до возвращения функцией EOF значения True приводит к ошибке. При чтении двоичных файлов с помощью функции Input следует вместо функции EOF использовать функции LOF и Loc или с функцией EOF использовать инструкцию Get.

Error (VBA). Возвращает сообщение об ошибке, соответствующее определенному ее коду.

Синтаксис:
Error[(кодОшибки)]

Необязательный аргумент *кодОшибки* может представлять собой любое допустимое значение кода ошибки. Если в аргументе *кодОшибки* задан допустимый, но не определенный код ошибки, метод Error возвращает строку *Ошибка*, *определяемая приложением или объектом*. Недопустимое значение аргумента *кодОшибки* приводит к возникновению ошибки. Если аргумент *кодОшибки* опущен, возвращается сообщение, соответствующее последней ошибке выполнения. Если ошибка выполнения не возникала или аргумент *кодОшибки* имеет значение 0, функция Error возвращает пустую строку (" ").

Для определения последней ошибки выполнения следует проверить значения свойств объекта Err. Значение, возвращаемое функцией Error, соответствует значению свойства Description объекта Err.

Eval. Позволяет найти выражение, сводящееся к строковому или числовому значению.

Строка, которая собирается в результате определенных действий, обрабатывается функцией Eval как реальное выражение.

Функция Eval сначала находит значение строкового выражения, а потом возвращает соответствующее значение. Например, Eval("1 + 1") возвращает 2.

Синтаксис:
Eval(строковоеВыражение)

Аргумент *строковоеВыражение* представляет собой выражение, задаваемое строкой. Например, аргумент *строковоеВыражение* может задавать функцию, возвращающую строку или число, или являться ссылкой на элемент управления в форме. Необходимо, чтобы значением аргумента *строковоеВыражение* были строка или число; однако значением этого аргумента не может быть объект Microsoft Access.

При передаче имени функции в качестве аргумента *строковоеВыражение* в функцию Eval необходимо заключить это имя в кавычки. Например:

```
1. Debug.Print Eval("СписокИмен()")
   'СписокИмен — функция, определенная пользователем
2. Debug.Print Eval("StrComp(""Joe"",""joe"", 1)")
3. Debug.Print Eval("Date()")
```

Функцию Eval используют в вычисляемых элементах управления в формах и отчетах, а также в макросах и модулях. Функция Eval возвращает значение типа Variant, принадлежащее либо к строковому, либо к числовому типу.

Аргумент *строковоеВыражение* должен представлять собой выражение, сохраненное в виде строки. Попытка передать в функцию Eval обычное строковое значение, не представляющее собой выражение, например, Eval("Smith"), приведет к ошибке.

Функция Eval позволяет определить значение, сохраняемое в свойстве Value элемента управления. Приведем пример, где полная ссылка на элемент управления передается в функцию Eval, которая возвращает значение, сохраняемое в свойстве Value этого элемента управления:

```
Dim strCtl As String
Set ctl = Forms!Сотрудники!Фамилия
strCtl = "Forms!Сотрудники!Фамилия"
MsgBox ("Текущее значение поля" '& strCtl &'" : "&
Eval(strCtl))
```

Exp (VBA). Возвращает значение типа Double, содержащее результат возведения числа **e** (основание натуральных логарифмов) в указанную степень.

Синтаксис:
Exp(число)

Обязательный аргумент *число* представляет собой значение типа Double или любое допустимое числовое выражение.

Если значение аргумента *число* превышает 709,782712893, возникает ошибка. Константа **e** приблизительно равняется 2,718282.

FileAttr (VBA). Возвращает значение типа Long, представляющее собой режим файла для файлов, открытых с помощью инструкции Open.

Синтаксис:
FileAttr(filenumber, returntype)

Синтаксис функции FileAttr содержит аргументы, приведенные в табл. П.2.28.

Таблица П2.28

Аргументы функции FileAttr

Аргумент	Описание
filenumber	Обязательный. Имеет значение типа Integer. Представляет собой любой допустимый номер файла

Аргумент	Описание
returntype	Обязательный. Имеет значение типа Integer. Представляет собой число, указывающее характер возвращаемых данных. Значение 1 задает возвращение значения, указывающего режим файла. В 16-разрядных системах значение 2 задает возвращение дескриптора файла в операционной системе. В 32-разрядных системах значение 2 данного аргумента не поддерживается и приводит к ошибке

Между режимами файла и значениями, возвращаемыми в случае, если аргумент returntype имеет значение 1, существует следующее соответствие:

Режим	Значение
Input	1
Output	2
Random	4
Append	8
Binary	32

FileDateTime (VBA). Возвращает значение типа Variant (Date), содержащее дату и время создания или последнего изменения файла.

Синтаксис:
FileDateTime(путь)

Обязательный аргумент *путь* является строковым выражением, указывающим имя файла. Аргумент *путь* может содержать имя каталога или папки и диска.

FileLen (VBA). Возвращает значение типа Long, содержащее размер файла в байтах.

Синтаксис:
FileLen(путь)

Обязательный аргумент *путь* является строковым выражением, определяющим файл. Аргумент *путь* может содержать имя каталога или папки и диска.

Если в момент вызова функции FileLen указанный файл открыт, возвращается размер этого файла до его открытия.

Для определения размера открытого файла следует использовать функцию LOF.

First (DAO), Last (DAO). Возвращают значение поля из первой или последней записи результирующего набора запроса.

Синтаксисы:
First(выражение)
Last(выражение)

Аргумент *выражение* является строковым выражением, которое определяет поле, содержащее данные для подсчета, или выражение, выполняющее вычисления с данными из этого поля. Операнды аргумента *выражение* могут включать в себя имя поля таблицы, константу или функцию. (Функция может быть внутренней или определяться пользователем, но не может быть ни одной из других статистических функций SQL.)

Функции First и Last могут рассматриваться как аналоги методов MoveFirst и MoveLast объекта доступа к данным (DAO) Recordset. Эти функции возвращают значение указанного поля, находящееся соответственно в первой или последней записях результирующего набора запроса.

Поскольку записи обычно возвращаются без какого-либо специального порядка (кроме случаев, когда запрос содержит предложение ORDER BY), значит, эти функции возвращают случайные записи.

Fix (VBA), Int (VBA). Возвращают значение типа, совпадающего с типом аргумента, которое содержит целую часть числа.

Синтаксисы:
Int(число)
Fix(число)

Обязательный аргумент *число* представляет собой значение типа Double или любое допустимое числовое выражение. Если аргумент *число* имеет значение Null, возвращается Null.

Обе функции Int и Fix отбрасывают дробную часть числа и возвращают целое значение.

Различие между функциями Int и Fix состоит в том, что для отрицательного значения аргумента *число* Int возвращает ближайшее отрицательное целое число, меньшее либо равное указанному, а Fix — ближайшее отрицательное целое число, большее либо равное указанному. Например, функция Int преобразует –8.4 в –9, а функция Fix преобразует –8,4 в –8.

Выражение Fix(число) эквивалентно следующему выражению:

```
Sgn(число) * Int(Abs(число))
```

Format (VBA). Возвращает значение типа Variant (String), содержащее выражение, отформатированное согласно инструкциям, заданным в описании формата.

Синтаксис:
Format(выражение[, формат[, первыйДеньНедели[, перваяНеделяГода]]])

Синтаксис функции Format содержит аргументы, приведенные в табл. П2.29.

Таблица П2.29

Аргументы функции Format

Аргумент	Описание
выражение	Обязательный. Представляет собой любое допустимое выражение
формат	Необязательный. Представляет собой любое допустимое именованное или определяемое пользователем выражение формата
первыйДеньНедели	Необязательный. Представляет собой константу, определяющую первый день недели
перваяНеделяГода	Необязательный. Представляет собой константу, определяющую первую неделю года

FreeFile (VBA). Возвращает значение типа Integer, представляющее собой следующий номер файла, доступный для использования с инструкцией Open.

Синтаксис:
FreeFile[(диапазонНомеров)]

Необязательный аргумент *диапазонНомеров* является выражением типа Variant, указывающим диапазон, из которого возвращается следующий свободный номер файла. Значение 0 (используется по умолчанию) задает возвращение номера файла из диапазона от 1 до 255, а значение 1 — из диапазона от 256 до 511.

Функцию FreeFile используют для возвращения незанятого номера файла.

FV (VBA). Возвращает значение типа Double, указывающее будущее значение суммы регулярных платежей при заданной учетной ставке.

Синтаксис:
FV(rate, nper, pmt[, pv[, type]])

Синтаксис функции FV содержит аргументы, приведенные в табл. П2.30.

Таблица П2.30

Аргументы функции FV

Аргумент	Описание
rate	Обязательный. Представляет собой выражение типа Double, указывающее учетную ставку за период
nper	Обязательный. Представляет собой выражение типа Integer, указывающее полное число периодов (выплат) за рассматриваемый срок
pmt	Обязательный. Представляет собой выражение типа Double, указывающее размер выплат за период. Каждый платеж включает в себя как собственно возвращаемую сумму, так и проценты, которые не изменяются от одного платежа к другому
pv	Необязательный. Представляет собой выражение типа Variant, указывающее сумму на текущий момент. Например, при займе на покупку автомобиля сумма займа является текущим значением при расчете будущих платежей. Если этот аргумент опущен, подразумевается значение 0
type	Необязательный. Представляет собой выражение типа Variant, указывающее режим выплат. Значение 0 означает, что платежи вносятся в конце периода, а значение 1, — что платежи вносятся в начале периода. Если этот аргумент опущен, подразумевается значение 0

GetAllSettings (VBA). Возвращает список записей и их значений (созданных с помощью функции SaveSetting) из раздела, соответствующего приложению, в реестре Windows.

Синтаксис:
GetAllSettings(appname, section)

Синтаксис функции GetAllSettings содержит аргументы, приведенные в табл. П2.31.

Таблица П2.31

Аргументы функций GetAllSettings

Аргумент	Описание
appname	Обязательный. Представляет собой строковое выражение, содержащее имя приложения или проекта, для которого определяются записи

Аргумент	Описание
section	Обязательный. Представляет собой строковое выражение, содержащее имя раздела, для которого определяются записи. Функция GetAllSettings возвращает значение типа Variant

GetAttr (VBA). Возвращает значение типа Integer, содержащее атрибуты файла, каталога или папки.

Синтаксис:
GetAttr(путь)

Обязательный аргумент *путь* является строковым выражением, указывающим имя файла. Аргумент *путь* может содержать имя каталога или папки и диска.

Значение, возвращаемое функцией GetAttr, является суммой следующих значений:

Константа	Значение	Описание
vbNormal	0	Обычный
vbReadOnly	1	Только чтение
vbHidden	2	Скрытый
vbSystem	4	Системный
vbDirectory	16	Каталог или папка
vbArchive	32	Файл был изменен после последнего резервирования

Данные константы определяются в языке Visual Basic для приложений. Это означает, что их имена можно использовать в любом месте кода вместо фактических значений.

GetObject (VBA). Возвращает ссылку на объект ActiveX, сохраненный в файле.

Синтаксис:
GetObject([pathname] [, class])

Синтаксис функции GetObject содержит аргументы, приведенные в табл. П2.32.

Таблица П2.32

Аргументы функции GetObject

Аргумент	Описание
pathname	Необязательный; имеет значение типа Variant (String). Представляет собой полный путь и имя файла, содержащего объект, который следует загрузить. Если данный аргумент опущен, значит, должен быть указан аргумент class
class	Необязательный; имеет значение типа Variant (String). Это строка, представляющая собой класс объекта

Аргумент class использует следующий синтаксис:

```
имяПриложения.типОбъекта
```

который содержит элементы, приведенные в табл. П2.33.

Таблица П2.33

Элементы аргумента class

Элемент	Описание
ИмяПриложения	Обязательный; имеет значение типа Variant (String). Представляет собой имя приложения, являющегося источником объекта
типОбъекта	Обязательный; имеет значение типа Variant (String). Представляет собой тип или класс создаваемого объекта

Функцию GetObject используют для доступа к объекту ActiveX, сохраненному в файле, и присвоения объекта объектной переменной. Присвоить объект, возвращаемый функцией GetObject, объектной переменной позволяет инструкция Set. Например:

```
Dim CADObject As Object
Set CADObject = GetObject("C:\CAD\SCHEMA.CAD")
```

Эти инструкции запускают приложение, указанное с помощью аргумента pathname, и активизируют объект, находящийся в указанном файле.

GetSetting (VBA). Возвращает значение записи из раздела, соответствующего приложению в реестре Windows.

Синтаксис:
GetSetting(appname, section, key[, default])

Синтаксис функции GetSetting содержит аргументы, приведенные в табл. П2.34.

Таблица П2.34

Аргументы функции GetSettings

Аргумент	Описание
appname	Обязательный. Представляет собой строковое выражение, содержащее имя приложения или проекта, для которого определяются записи
section	Обязательный. Представляет собой строковое выражение, содержащее имя раздела, в котором находится запись
key	Обязательный. Представляет собой строковое выражение, содержащее имя возвращаемой записи
default	Необязательный. Представляет собой выражение, содержащее значение, возвращаемое в том случае, если значение записи не задано. Если данный аргумент не определен, то подразумевается пустая строка (“”)

Hex (VBA). Возвращает значение типа String, задающее шестнадцатеричное представление указанного числа.

Синтаксис:
Hex(число)

Обязательный аргумент *число* представляет собой любое допустимое числовое выражение или строковое выражение.

Если аргумент *число* не является целым числом, перед преобразованием он округляется до ближайшего целого числа:

Аргумент *число*	Возвращаемое значение
Null	Пустое значение
Empty	Нуль (0)
Любое другое число	До восьми шестнадцатеричных символов

Для явного представления шестнадцатеричного числа без вызова функции следует поставить перед ним символы &H. Например, &H10 это десятичное число 16 в шестнадцатеричном представлении.

Hour (VBA). Возвращает значение типа Variant (Integer), содержащее целое число (от 0 до 23), которое представляет собой часы в значении времени.

Синтаксис:
Hour(время)

Обязательный аргумент *время* может быть любым значением типа Variant, числовым выражением, строковым выражением или любым их сочетанием, которое представляет собой значение времени. Если *время* содержит значение Null, возвращается значение Null.

HyperlinkPart. Возвращает сведения о данных, сохраняемых с типом данных гиперссылки.

Поле гиперссылки функции HyperlinkPart может содержать до трех частей в следующем формате:

```
отображаемыйТекст#адрес#допАдрес
```

Синтаксис:
объект.HyperlinkPart(гиперссылка As Variant[, часть As Integer])

Функция HyperlinkPart содержит аргументы, приведенные в табл. П2.35.

Таблица П2.35

Аргументы функции HyperlinkPart

Аргумент	Описание
объект	Необязательный. Представляет собой объект Application
гиперссылка	Значение типа Variant, представляющее собой данные, сохраняемые в поле гиперссылки
часть	Необязательный. Значением данного аргумента является встроенная константа, представляющая собой сведения о данных, возвращаемые функцией HyperlinkPart

Возможные значения констант:

Константа	Значение	Описание
acDisplayedValue	0	Значение используется по умолчанию
acDisplayText	1	Часть поля гиперссылки *отображаемыйТекст*
acAddress	2	Часть поля гиперссылки *адрес*
acSubAddress	3	Часть поля гиперссылки *допАдрес*

Функция HyperlinkPart используется для возвращения одного из трех значений из поля гиперссылки или отображаемого значения. Аргумент *часть* определяет, какое именно значение будет воз-

вращено. Этот аргумент является необязательным. Если он не используется, то возвращается значение, которое Microsoft Access отображает для данной гиперссылки (т.е. значение, соответствующее значению аргумента *часть* по умолчанию, представляемому константой acDisplayedValue). Возвращаемые значения представляют собой соответственно одну из трех частей поля гиперссылки (*отображаемыйТекст*, *адрес* или *допАдрес*) или отображаемое значение, которое в Microsoft Access выводится для гиперссылки.

Если функция HyperlinkPart используется в запросе, то аргумент *часть* является обязательным. При этом использование приведенных констант невозможно, и вместо них задаются фактические значения.

Если возвращается значение из части поля гиперссылки *отображаемыйТекст*, то отображаемое Microsoft Access значение совпадает с содержимым этой части. Если часть поля гиперссылки *отображаемыйТекст* не содержит значения, то отображаемое значение совпадает с содержимым первой заполненной части поля гиперссылки, т.е. части *адрес* или *допАдрес*.

В табл. П2.36 приведены значения, возвращаемые функцией HyperlinkPart для данных, сохраняемых в поле гиперссылки.

Таблица П2.36

Значения, возвращаемые функцией HyperlinkPart

Данные в поле гиперссылки	Возвращаемые значения
#http://www.microsoft.com#	acDisplayedValue: http://www.microsoft.com acDisplayText: acAddress: http://www.microsoft.com acSubAddress:
Microsoft#http://www.microsoft.com#	acDisplayedValue: Microsoft acDisplayText: Microsoft acAddress: http://www.microsoft.com acSubAddress:
Клиенты##Form Клиенты	acDisplayedValue:Клиенты acDisplayText: Клиенты acAddress: acSubAddress: Form Клиенты
##Form лиенты	acDisplayedValue: Form Клиенты acDisplayText: acAddress: acSubAddress: Form Клиенты

Если пользователь включает часть *адрес* в поле гиперссылки с помощью диалогового окна *Вставить гиперссылку* (которое открывается при выборе команды *Гиперссылка* в меню *Вставка*) или

непосредственно вводит адрес в поле, то Microsoft Access добавляет два символа ##, разделяющие части гиперссылки.

Для добавления или изменения в поле гиперссылки части *отображаемыйТекст* следует выделить гиперссылку в таблице с помощью правой кнопки мыши, а затем выбрать в контекстном меню команду *Гиперссылка* и ввести текст в поле *Отображать текст*.

Если данные вводятся в поле гиперссылки непосредственно или добавляются с помощью методов объектов доступа к данным (DAO), то обязательно должны быть вставлены два символа ##, разделяющие части гиперссылки.

IIf (VBA). Возвращает одно из двух значений в зависимости от истинности указанного выражения.

Синтаксис:
IIf(expr, truepart, falsepart)

Синтаксис функции IIf содержит аргументы, приведенные в табл. П2.37.

Таблица П2.37

Аргументы функции IIf

Аргумент	Описание
expr	Обязательный. Представляет собой проверяемое выражение
truepart	Обязательный. Представляет собой значение или выражение, возвращаемое, если expr имеет значение True
falsepart	Обязательный. Представляет собой значение или выражение, возвращаемое, если expr имеет значение False

Функция IIf вычисляет оба выражения (truepart и falsepart) несмотря на то, что возвращается только одно из них. В некоторых случаях это приводит к нежелательным побочным эффектам. Например, если при вычислении выражения falsepart имеет место деление на нуль, то возникает ошибка, даже если значение expr имеет значение True.

Input (VBA). Возвращает значение типа String, содержащее символы из файла, открытого в режиме Input или Binary.

Синтаксис:
Input(число, [#]номерФайла)

Синтаксис функции Input содержит аргументы, приведенные в табл. П2.38.

Таблица П2.38

Аргументы функции Input

Аргумент	Описание
число	Обязательный. Представляет собой любое допустимое числовое выражение, задающее число возвращаемых символов
номерФайла	Обязательный. Представляет собой любой допустимый номер файла

Данные, считываемые с помощью функции Input, обычно записываются в файл с помощью инструкции Print # или Put. Эта функция применима только к файлам, открытым в режиме Input или Binary.

В отличие от инструкции Input # функция Input возвращает все прочитанные символы, в том числе запятые, символы возврата каретки, символы перевода строки, кавычки и начальные пробелы.

Попытка чтения файлов, открытых для доступа в режиме Binary, с помощью функции Input до возвращения функцией EOF значения True приводит к ошибке. При чтении двоичных файлов с помощью функции Input следует вместо функции EOF использовать функции LOF и Loc или же с функцией EOF использовать инструкцию Get.

InputBox (VBA). Выводит на экран диалоговое окно, содержащее сообщение и поле ввода, устанавливает режим ожидания ввода текста пользователем или нажатия кнопки, а затем возвращает значение типа String, содержащее текст, введенный в поле.

Синтаксис:

InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])

Синтаксис функции InputBox содержит аргументы, приведенные в табл. П2.39.

Таблица П2.39

Аргументы функции InputBox

Аргумент	Описание
prompt	Обязательный. Представляет собой строковое выражение, отображаемое как сообщение в диалоговом окне. Максимальная длина строки prompt составляет приблизительно 1024 символа и зависит от ширины используемых символов. Строковое значение prompt может содержать несколь-

Окончание табл. П2.39

Аргумент	Описание
prompt	ко физических строк. Для разделения строк допускается использование символа возврата каретки (Chr(13)), символа перевода строки (Chr(10)) или комбинации этих символов (Chr(13) & Chr(10))
title	Необязательный. Представляет собой строковое выражение, отображаемое в строке заголовка диалогового окна. Если этот аргумент опущен, в строку заголовка помещается имя приложения
default	Необязательный. Представляет собой строковое выражение, отображаемое в поле ввода как используемое по умолчанию, если пользователь не введет другую строку. Если этот аргумент опущен, поле ввода изображается пустым
xpos	Необязательный. Представляет собой числовое выражение, задающее расстояние по горизонтали между левой границей диалогового окна и левым краем экрана (в твипах). Если этот аргумент опущен, диалоговое окно выравнивается по центру экрана по горизонтали
ypos	Необязательный. Представляет собой числовое выражение, задающее расстояние по вертикали между верхней границей диалогового окна и верхним краем экрана (в твипах). Если этот аргумент опущен, диалоговое окно занимает по вертикали примерно одну треть высоты экрана
helpfile	Необязательный. Представляет собой строковое выражение, определяющее имя файла справки, содержащего справочные сведения о данном диалоговом окне. Если этот аргумент указан, необходимо указать также аргумент context
context	Необязательный. Представляет собой числовое выражение, определяющее номер соответствующего раздела справочной системы. Если этот аргумент указан, необходимо указать также аргумент helpfile

InStr (VBA). Возвращает значение типа Variant (Long), указывающее позицию первого вхождения одной строки внутрь другой строки.

Синтаксис:
InStr([start,]string1, string2[, compare])

Синтаксис функции InStr содержит аргументы, приведенные в табл. П2.40.

Таблица П2.40

Аргументы функции InStr

Аргумент	Описание
start	Необязательный. Представляет собой числовое выражение, задающее позицию, с которой начинается каждый поиск. Если этот аргумент опущен, поиск начинается с первого символа строки. Если start имеет значение Null, возникает ошибка. Указывать аргумент start надо обязательно, если указан аргумент compare
string1	Обязательный. Представляет собой строковое выражение, в котором выполняется поиск
string2	Обязательный. Представляет собой искомое строковое выражение
compare	Необязательный. Указывает способ сравнения строк. Данный аргумент может быть опущен или иметь значение 0, 1 или 2. Для выполнения двоичного сравнения следует указать 0 (это значение используется по умолчанию), для выполнения посимвольного сравнения без учета регистра — 1. Только в Microsoft Access допускается использование значения 2 для выполнения сравнения строк на основании сведений, содержащихся в базе данных. Если данный аргумент имеет значение Null, возникает ошибка; если он опущен, способ сравнения строк определяется значением параметра инструкции Option Compare

Возвращаемые значения:

Условие	Значение функции InStr
string1 является пустой строкой	0
string1 имеет значение Null	Пустое значение
string2 является пустой строкой	Start
string2 имеет значение Null	Пустое значение
string2 не найдена	0
string2 найдена в string1	Позиция обнаруженной подстроки
start > string2	0

IsArray (VBA). Возвращает значение типа Boolean, показывающее, является ли переменная массивом.

Синтаксис:
IsArray(имяПеременной)

Обязательный аргумент *имяПеременной* является идентификатором переменной.

Функция IsArray возвращает значение True, если переменная содержит массив; в противном случае возвращается значение False. Функцию IsArray используют для проверки значений типа Variant, содержащих массивы.

IsDate (VBA). Возвращает значение типа Boolean, показывающее, может ли значение выражения быть преобразовано в значение даты.

Синтаксис:
IsDate(выражение)

Обязательный аргумент *выражение* представляет собой значение типа Variant, содержащее выражение даты или строковое выражение, распознаваемое как значение даты или времени.

IsEmpty (VBA). Возвращает значение типа Boolean, показывающее, была ли инициализирована переменная.

Синтаксис:
IsEmpty(выражение)

Обязательный аргумент *выражение* представляет собой значение типа Variant, содержащее числовое или строковое выражение. Однако, поскольку функция IsEmpty предназначена для проверки того, была ли инициализирована конкретная переменная, в аргументе *выражение* обычно указывают имя переменной.

IsError (VBA). Возвращает значение типа Boolean, показывающее, является ли аргумент *выражение* значением ошибки.

Синтаксис:
IsError(выражение)

Обязательный аргумент *выражение* должен принадлежать к подтипу VarType vbError типа Variant.

IsMissing (VBA). Возвращает значение типа Boolean, показывающее, был ли передан в процедуру необязательный аргумент.

Синтаксис:
IsMissing(имяАргумента)

Обязательный аргумент *имяАргумента* содержит имя необязательного аргумента процедуры.

IsNull (VBA). Возвращает значение типа Boolean, показывающее, является ли результатом выражения пустое значение (Null).

Синтаксис:
IsNull(выражение)

Обязательный аргумент *выражение* представляет собой значения типа Variant, содержащее числовое или строковое выражение.

Функция IsNull возвращает значение True, если выражение имеет значение Null; в противном случае IsNull возвращает значение False. Если выражение содержит несколько переменных, то значение Null любой из этих переменных приводит к значению True, возвращаемому для всего выражения.

Значение Null указывает, что переменная типа Variant не содержит допустимых данных. Не следует путать значение Null со значением Empty, указывающим, что переменная не была инициализирована. Это значение также не эквивалентно пустой строке (“ ”), которую иногда называют строкой нулевой длины.

IsNumeric (VBA). Возвращает значение типа Boolean, показывающее, имеет ли выражение числовое значение.

Синтаксис:
IsNumeric(выражение)

Обязательный аргумент *выражение* представляет собой значение типа Variant, содержащее числовое или строковое выражение.

IsObject (VBA). Возвращает значение типа Boolean, показывающее, является ли идентификатор объектной переменной.

Синтаксис:
IsObject(идентификатор)

Обязательный аргумент *идентификатор* представляет собой имя переменной.

Функция IsObject используется только для проверки принадлежности выражения типа Variant к подтипу VarType vbObjec, что возможно только в случае, если выражение типа Variant задает (или задавало) ссылку на объект, или если оно имеет значение Nothing.

Функция IsObject возвращает значение True, если идентификатор определяет переменную, описанную с типом Object или любым допустимым типом класса, а также если идентификатор представляет собой подтип VarType vbObject типа Variant либо определяемый пользователем объект. В противном случае возвраща-

ется значение False. IsObject возвращает значение True даже для переменной со значением Nothing.

Для проверки допустимой ссылки на объект следует использовать перехват ошибок. Проверить, имеет ли ссылка на объект значение Nothing, позволяет функция IsNothing.

LBound (VBA). Возвращает значение типа Long, содержащее минимальный доступный индекс указанной размерности массива.

Синтаксис:
LBound(имяМассива[, размерность])

Синтаксис функции LBound содержит элементы, приведенные в табл. П2.41.

Таблица П2.41

Элементы функции LBound

Элемент	Описание
имяМассива	Обязательный. Представляет собой имя переменной массива, удовлетворяющее стандартным правилам именования переменных
размерность	Необязательный; значения типа Variant (Long). Представляет собой целое число, указывающее размерность, нижнюю границу которой возвращает функция

LCase (VBA). Возвращает значение типа String, представляющее собой строку, преобразованную к нижнему регистру.

Синтаксис:
LCase(строка)

Обязательный аргумент *строка* представляет собой любое допустимое строковое выражение. Если аргумент *строка* имеет значение Null, возвращается значение Null.

Left (VBA). Возвращает значение типа Variant (String), содержащее указанное число первых символов строки.

Синтаксис:
Left(string, length)

Синтаксис функции Left содержит аргументы, приведенные в табл. П2.42.

Таблица П2.42

Аргументы функции Left

Аргумент	Описание
string	Обязательный. Представляет собой строковое выражение, из которого извлекаются символы. Если данный аргумент имеет значение Null, возвращается значение Null
length	Обязательный; значение типа Variant (Long). Представляет собой числовое выражение, указывающее число возвращаемых символов.Если данный аргумент равен 0, возвращается пустая строка (“”). Если же его значение больше либо равняется числу символов в строке string, возвращается вся строка

Len (VBA). Возвращает значение типа Long, содержащее число символов в строке или число байт, необходимое для размещения переменной.

Синтаксис:
Len(строка | имяПеременной)

Синтаксис функции Len содержит аргументы, приведенные в табл. П2.43.

Таблица П2.43

Аргументы функции Len

Аргумент	Описание
строка	Любое допустимое строковое выражение. Если данный аргумент имеет значение Null, возвращается значение Null
имяПеременной	Любое допустимое имя переменной. Если данный аргумент имеет значение Null, возвращается значение Null. Если же он имеет значение типа Variant, функция Len обрабатывает его так же, как и значение типа String, и всегда возвращает число содержащихся в нем символов

LoadPicture. Загружает рисунок в элемент ActiveX.

Синтаксис:
LoadPicture(имяФайла)

Функция LoadPicture использует аргументы, приведенные в табл. П2.44.

Таблица П2.44

Аргументы функции LoadPicture

Аргумент	Описание
имяФайла	Строковое выражение, определяющее имя загружаемого файла. Допустимы точечные рисунки (bitmap) (.bmp), значки (.ico), файлы в формате .rle или метафайлы (.wmf)

Для динамической загрузки рисунка в элемент ActiveX следует присвоить значение, возвращаемое функцией LoadPicture, свойству *Рисунок* (Picture) этого элемента. Приведем пример загрузки точечного рисунка в специальный элемент управления с именем *ЭлементOLE* в форме ЗАКАЗЫ:

```
Set Forms!Заказы!ЭлементOLE.Picture = LoadPicture
("Stars.bmp")
```

Loc (VBA). Возвращает значение типа Long, определяющее текущее положение указателя чтения/записи внутри открытого файла.

Синтаксис:
Loc(номерФайла)

Обязательный аргумент *номерФайла* типа Integer представляет собой любой допустимый номер файла.

Значения, возвращаемые для каждого режима файла, следующие:

Режим	Возвращаемое значение
Random	Номер последней записи, считанной или записанной в этот файл
Sequential	Номер текущего байта, деленный на 128. (Следует отметить, что значение, возвращаемое функцией Loc для файлов в режиме последовательного доступа, не является ни нужным, ни необходимым.)
Binary	Номер последнего считанного или записанного байта

LOF (VBA). Возвращает значение типа Long, представляющее собой размер файла (байт), открытого с помощью инструкции Open.

Синтаксис:
LOF(номерФайла)

Обязательный аргумент *номерФайла* является выражением типа Integer, содержащим допустимый номер файла.

Для определения размера неоткрытого файла следует использовать функцию FileLen.

Log (VBA). Возвращает значение типа Double, содержащее натуральный логарифм числа.

Синтаксис:
Log(число)

Обязательный аргумент *число* представляет собой значение типа Double или любое допустимое положительное числовое выражение.

Натуральным называют логарифм с основанием **e**. Константа **e** приблизительно равняется 2,718282.

Для вычисления логарифма числа x с основанием n следует разделить натуральный логарифм числа x на натуральный логарифм числа n:

```
Logn(x) = Log(x) / Log(n)
```

Приведем пример специальной процедуры Function, предназначенной для вычисления логарифмов с основанием 10:

```
Static Function Log10(X)
   Log10 = Log(X) / Log(10#)
End Function
```

LTrim (VBA), RTrim (VBA), Trim (VBA). Возвращают значение типа Variant (String), содержащее копию строки, из которой удалены пробелы, находившиеся в начале строки (LTrim), в конце строки (RTrim) или в начале и конце строки (Trim).

Синтаксисы:
LTrim(строка)
RTrim(строка)
Trim(строка)

Обязательный аргумент *строка* представляет собой любое допустимое строковое выражение. Если аргумент *строка* имеет значение Null, возвращается значение Null.

Min (DAO), Max (DAO). Возвращают минимальное и максимальное значения из набора значений, содержащихся в указанном поле запроса.

Синтаксисы:
Min(выражение)
Max(выражение)

Аргумент *выражение* является строковым выражением, которое определяет поле, содержащее вычисляемые данные, или вы-

ражение, выполняющее вычисления с данными из этого поля. Операнды аргумента *выражение* могут включать в себя имя поля таблицы, константу или функцию. (Функция может быть внутренней или определяться пользователем, но не может быть ни одной из других статистических функций SQL.)

Функции Min и Max используются для определения наименьшего и наибольшего значений из поля на основе выборки или группировки. Например, можно применить эти функции для возврата наименьшей и наибольшей стоимости доставки. Если не указан способ группировки, используется вся таблица.

Функции Min и Max используются в выражении запроса и в свойстве SQL объекта QueryDef или при создании объекта Recordset на основе запроса SQL.

Mid (VBA). Возвращает значение типа Variant (String), содержащее указанное число символов строки.

Синтаксис:
Mid(string, start[, length])

Синтаксис функции Mid содержит аргументы, приведенные в табл. П2.45.

Таблица П2.45

Аргументы функции Mid

Аргумент	Описание
string	Обязательный. Представляет собой строковое выражение, из которого извлекаются символы. Если данный аргумент имеет значение Null, возвращается значение Null
start	Обязательный; значение типа Long. Представляет собой позицию символа в строке string, с которого начинается нужная подстрока. Если данный аргумент больше числа символов в строке string, функция Mid возвращает пустую строку (“ ”)
length	Необязательный; значение типа Variant (Long). Представляет собой число возвращаемых символов. Если этот аргумент опущен или превышает число символов, расположенных справа от позиции start, возвращаются все символы от позиции start до конца строки

Для определения числа символов в аргументе string следует использовать функцию Len.

Эквивалентной функцией обработки строк по байтам является функция MidB. В этом случае аргументы указывают число байт (а не символов).

Minute (VBA). Возвращает значение типа Variant (Integer), содержащее целое число (от 0 до 59), которое представляет собой минуты в значении времени.

Синтаксис:
Minute(время)

Обязательный аргумент *время* может быть любым значением типа Variant, числовым выражением, строковым выражением или любым их сочетанием, представляющим собой значение времени. Если аргумент *время* содержит значение Null, возвращается значение Null.

Month (VBA). Возвращает значение типа Variant (Integer), содержащее целое число (от 0 до 12), которое представляет собой месяц в значении даты.

Синтаксис:
Month(дата)

Обязательный аргумент *дата* может быть любым значением типа Variant, числовым выражением, строковым выражением или любым их сочетанием, представляющим собой дату. Если аргумент *дата* содержит значение Null, возвращается значение Null.

MsgBox (VBA). Выводит на экран диалоговое окно, содержащее сообщение, устанавливает режим ожидания нажатия кнопки пользователем, а затем возвращает значение типа Integer, указывающее, какая кнопка была нажата.

Синтаксис:
MsgBox(prompt[, buttons] [, title] [, helpfile, context])

Синтаксис функции MsgBox содержит аргументы, приведенные в табл. П2.46.

Таблица П2.46

Аргументы функции MsgBox

Аргумент	Описание
prompt	Обязательный. Представляет собой строковое выражение, отображаемое как сообщение в диалоговом окне. Максимальная длина строки prompt составляет приблизительно 1024 символов и зависит от ширины используемых символов. Строковое значение prompt может содержать несколько физических строк. Для разделения строк допускается использование символа возврата каретки (Chr(13)), символа перевода строки (Chr(10)) или комбинации этих символов (Chr(13) & Chr(10))

Окончание табл. П2.46

Аргумент	Описание
buttons	Необязательный. Представляет собой числовое выражение, определяющее сумму значений, которые указывают число и тип отображаемых кнопок, тип используемого значка, основную кнопку и модальность окна сообщения. Значение по умолчанию данного аргумента равняется 0
title	Необязательный. Представляет собой строковое выражение, отображаемое в строке заголовка диалогового окна. Если данный аргумент опущен, в строку заголовка помещается имя приложения
helpfile	Необязательный. Представляет собой строковое выражение, определяющее имя файла справки, содержащего справочные сведения о данном диалоговом окне. Если данный аргумент указан, необходимо также указать аргумент context
context	Необязательный. Представляет собой числовое выражение, определяющее номер соответствующего раздела справочной системы. Если данный аргумент указан, необходимо также указать аргумент helpfile

В табл. П2.47 перечислены допустимые константы аргумента buttons.

Таблица П2.47

Константы аргумента buttons

Константа	Значение	Описание
vbOKOnly	0	Отображается только кнопка [OK]
vbOKCancel	1	Отображаются кнопки [OK] и [Отмена] (Cancel)
vbAbortRetryIgnore	2	Отображаются кнопки [Прервать] (Abort), [Повторить] (Retry) и [Пропустить] (Ignore)
vbYesNoCancel	3	Отображаются кнопки [Да] (Yes), [Нет] (No) и [Отмена] (Cancel)
vbYesNo	4	Отображаются кнопки [Да] (Yes) и [Нет] (No)
vbRetryCancel	5	Отображаются кнопки [Повторить] (Retry) и [Отмена] (Cancel)
vbCritical	16	Используется значок [Критическое сообщение]

Окончание табл. П2.47

Константа	Значение	Описание
vbQuestion	32	Используется значок [Предупреждающий запрос]
vbExclamation	48	Используется значок [Предупреждение]
vbInformation	64	Используется значок [Информационное сообщение]
vbDefaultButton1	0	Основной является первая кнопка
vbDefaultButton2	256	Основной является вторая кнопка
vbDefaultButton3	512	Основной является третья кнопка
vbDefaultButton4	768	Основной является четвертая кнопка
vbApplicationModal	0	Соответствует модальному окну на уровне приложения: чтобы продолжить работу с текущим приложением, необходимо ответить на данное сообщение
vbSystemModal	4096	Соответствует модальному окну на уровне системы: все приложения будут недоступны до тех пор, пока пользователь не ответит на данное сообщение

Первая группа значений (0...5) указывает число и тип кнопок, отображаемых в диалоговом окне, вторая (16; 32; 48; 64) — задает тип используемого значка, третья (0; 256; 512) — определяет кнопку, которая является основной, а четвертая (0; 4096) — определяет модальность окна сообщения.

При определении значения аргумента buttons суммируется не более одного значения из каждой группы.

Константы, приведенные в табл. П2.47, определены в языке Visual Basic для приложений. Использование имен этих констант вместо их значений допускается в любом месте программы.

Возвращаемые значения:

Константа	Значение	Нажатая кнопка
vbOK	1	[OK]
vbCancel	2	[Отмена] (Cancel)
vbAbort	3	[Прервать] (Abort)
vbRetry	4	[Повторить] (Retry)
vbIgnore	5	[Пропустить] (Ignore)
vbYes	6	[Да] (Yes)
vbNo	7	[Нет] (No)

Если указаны аргументы helpfile и context, пользователь имеет возможность нажатием клавиши [F1] вызвать контекстную справку. Некоторые главные приложения (например, Microsoft Excel) также автоматически добавляют в диалоговое окно кнопку [Справка].

Если диалоговое окно содержит кнопку [Отмена] (Cancel), нажатие клавиши [ESC] эквивалентно нажатию этой кнопки. Если диалоговое окно содержит кнопку [Справка] (Help), значит, существует связанный с ним раздел справочной системы. Однако никакое значение не возвращается до тех пор, пока не будет нажата какая-либо другая кнопка.

Функцию MsgBox с двумя или большим числом аргументов можно использовать только в выражении. При этом наличие запятых, соответствующих отсутствующим аргументам, является обязательным.

Now (VBA). Возвращает значение типа Variant (Date), содержащее текущую дату и время по системному календарю и часам компьютера.

Синтаксис:
Now

Nz. Возвращает нуль, пустую строку (“ ”) или другое указанное значение, если переменная типа Variant имеет значение Null.

Например, функцию Nz используют для преобразования значений Null в другое значение при работе с выражениями, не допускающими пустых значений.

Синтаксис:
Nz(variant[, представление])

Функция Nz использует аргументы, приведенные в табл. П2.48.

Таблица П2.48

Аргументы функции Nz

Аргумент	Описание
variant	Переменная с типом данных Variant
представление	Необязательный (если не используется в запросе). Имеет значение типа Variant, которое возвращается, если аргумент variant имеет значение Null. Данный аргумент позволяет возвращать значение, отличное от нуля или пустой строки. Если функция Nz используется в выражении запроса без аргумента *представление*, то результатом будет пустая строка в полях, содержащих значения null

Если аргумент variant имеет значение Null, функция Nz возвращает нуль или пустую строку в зависимости от контекста, требующего числовое или строковое значение.

Если аргумент variant имеет значение, отличное от значения Null, то функция Nz возвращает значение аргумента variant.

Функцию Nz используют при работе с выражениями, в которых могут оказаться пустые значения. Для того чтобы такое выражение возвращало непустое значение при любых значениях входящих в него компонентов, следует с помощью функции Nz определить замену пустых значений на нули, пустые строки или любое специальное значение, представляющее собой пустые значения.

Например, выражение 2 + varX возвращает значение Null, если переменная varX типа Variant имеет значение Null. Однако выражение 2 + Nz(varX) в этом случае возвращает значение 2.

Функцию Nz часто используют как альтернативу функции IIf. Для примера приведем конструкцию, в которой для получения нужных результатов требуются две инструкции с функцией IIf (первое выражение, содержащее функцию IIf, используется для проверки на пустые значения и преобразования пустых значений в нулевые):

```
varTemp = IIf(IsNull(доставка), 0, доставка)
varResult = IIf(varTemp > 50, "Больше", "Меньше")
```

Приведем также конструкцию функции Nz, позволяющую выполнить те же действия в одной программной строке:

```
varResult = IIf(Nz(доставка) > 50, "Больше", "Меньше")
```

Если указать необязательный аргумент *представление*, его значение будет возвращаться вместо пустого значения аргумента variant. Использование этого необязательного аргумента позволяет исключить одно выражение, содержащее функцию IIf.

Для примера приведем выражение, использующее функцию IIf для возвращения строки, если переменная *доставка* имеет значение Null:

```
varResult = IIf(IsNull(доставка), "Бесплатно", доставка)
```

В следующем приведенном примере необязательный аргумент функции Nz указывает строку, которая возвращается, если переменная *доставка* имеет значение Null:

```
varResult = Nz(доставка, "Бесплатно")
```

Oct (VBA). Возвращает значение типа Variant (String), содержащее восьмеричное представление указанного числа.

Синтаксис:
Oct(число)

Обязательный аргумент *число* представляет собой любое допустимое числовое или строковое выражение.

Если аргумент *число* не является целым числом, он округляется перед преобразованием до ближайшего целого числа:

Аргумент *число*	Возвращаемое значение
Null	Null
Empty	Нуль (0)
Любое другое число	До 11 восьмеричных символов

Для явного представления восьмеричного числа без вызова функции следует поставить перед ним символы &O. Например, &O10 — это десятичное число 8 в восьмеричном представлении.

Partition (VBA). Возвращает значение типа Variant (String), указывающее положение числа в вычисляемом наборе диапазонов.

Синтаксис:
Partition(number, start, stop, interval)

Синтаксис функции Partition содержит аргументы, приведенные в табл. П2.49.

Таблица П2.49

Аргументы функции Partition

Аргумент	Описание
number	Обязательный. Представляет собой целое число, для которого проверяется положение относительно набора диапазонов
start	Обязательный. Представляет собой целое число, задающее начало набора диапазонов. Это число должно быть неотрицательным
stop	Обязательный. Представляет собой целое число, задающее конец набора диапазонов, и которое должно быть больше, чем значение start
interval	Обязательный. Представляет собой целое число, задающее размер каждого диапазона значений в наборе диапазонов значений от start до stop, и которое не может быть меньше 1

Функция Partition определяет диапазон, который содержит указанное значение number, и возвращает значение типа Variant (String), описывающее этот диапазон. Обычно функция Partition используется в запросах. Например, можно создать запрос на выборку, показывающий распределение стоимости заказов по диапазонам (например, от 1 до 1000 тыс. р.; от 1001 до 2000 тыс. р. и т. п.).

QBColor (VBA). Возвращает значение типа Long, представляющее собой код цвета в модели RGB, соответствующий указанному номеру цвета.

Синтаксис:
QBColor(цвет)

Обязательный аргумент *цвет* задается целым числом в диапазоне от 0 до 15.

Допустимые значения аргумента *цвет* следующие:

Номер	Цвет	Номер	Цвет
0	Черный	8	Светло-серый
1	Темно-синий	9	Синий
2	Темно-зеленый	10	Зеленый
3	Бирюзовый	11	Голубой
4	Малиновый	12	Красный
5	Сиреневый	13	Розовый
6	Оливковый	14	Желтый
7	Темно-серый	15	Белый

Аргумент *цвет* описывает номера цветов, используемые другими версиями Basic (например, Microsoft Visual Basic для MS-DOS и Basic Compiler). Начиная с наименее значимого байта, возвращаемое значение содержит величины красного, зеленого и синего компонентов, описывающих соответствующий цвет в системе RGB, используемой Visual Basic для приложений.

RGB (VBA). Возвращает значение типа Long, т. е. целое число, представляющее собой цвет в модели RGB.

Синтаксис:
RGB(red, green, blue)

Синтаксис функции RGB содержит аргументы, приведенные в табл. П2.50.

Таблица П2.50

Аргументы функции RGB

Аргумент	Описание
Red	Обязательный; значение типа Variant (Integer). Это число в интервале от 0 до 255, представляющее собой красный компонент цвета
Green	Обязательный; значение типа Variant (Integer). Это число в интервале от 0 до 255, представляющее собой зеленый компонент цвета
Blue	Обязательный; значение типа Variant (Integer). Это число в интервале от 0 до 255, представляющее собой синий компонент цвета

Для описания цветов, являющихся в приложениях аргументами методов и значениями свойств, принимающих спецификацию цвета, подразумевается, что цвет задается с помощью значения в модели RGB. Значения аргументов функции RGB указывают относительную интенсивность красного, зеленого и синего компонентов, образующих отображаемый цвет.

Значение любого аргумента функции RGB, превышающее 255, считается равным 255.

Приведем некоторые стандартные цвета и соответствующие им комбинации компонентов RGB:

Цвет	Красный	Зеленый	Синий
Черный	0	0	0
Синий	0	0	255
Зеленый	0	255	0
Голубой	0	255	255
Красный	255	0	0
Розовый	255	0	255
Желтый	255	255	0
Белый	255	255	255

Right (VBA). Возвращает значение типа Variant (String), содержащее указанное число последних символов строки.

Синтаксис:
Right(string, length)

Синтаксис функции Right содержит аргументы, приведенные в табл. П2.51.

Таблица П2.51

Аргументы функции Right

Аргумент	Описание
String	Обязательный. Представляет собой строковое выражение, из которого извлекаются символы. Если данный аргумент имеет значение Null, возвращается значение Null
Length	Обязательный; значение типа Variant (Long). Представляет собой числовое выражение, указывающее число возвращаемых символов. Если данный аргумент равен 0, возвращается пустая строка (“ ”). Если же он превышает число символов в строке string, возвращается вся строка

Для определения числа символов в строке string следует использовать функцию Len.

Эквивалентной функцией обработки строк по байтам является функция RightB. В этом случае аргумент length указывает число байтов (а не символов), которые следует вернуть.

Rnd (VBA). Возвращает значение типа Single, содержащее случайное число.

Синтаксис:
Rnd[(число)]

Необязательный аргумент *число* представляет собой значение типа Single или любое допустимое числовое выражение.

Возвращаемые значения:

Значение аргумента *число*	Rnd возвращает
Меньше нуля	Каждый раз одно и то же число, используя при этом аргумент *число* в качестве опорного числа
Больше нуля	Следующее случайное число в последовательности
Равняется нулю	Случайное число, возвращенное при предыдущем вызове этой функции
Не указано	Следующее случайное число в последовательности

Функция Rnd возвращает значение, меньшее единицы и большее или равное нулю.

Аргумент *число* определяет способ генерации случайного числа функцией Rnd.

При использовании одинаковых опорных чисел получаются одинаковые последовательности случайных чисел, поскольку при генерации каждого следующего члена последовательности используется предыдущий член.

Перед вызовом функции Rnd используется инструкция Randomize без аргумента для инициализации генератора случайных чисел значением, возвращаемым системным таймером.

Приведем формулу, предназначенную для получения случайных целых чисел в заданном диапазоне:

```
Int((верхняяГраница – нижняяГраница+ 1) * Rnd + нижняяГраница)
```

Здесь *верхняяГраница* — максимальное число в диапазоне, а *нижняяГраница* — минимальное.

Для повторения последовательности случайных чисел следует вызвать функцию Rnd с отрицательным аргументом сразу после использования инструкции Randomize с числовым аргументом. Повторное использование инструкции Randomize с тем же числовым аргументом не приводит к повторению предыдущей последовательности случайных чисел.

Second (VBA). Возвращает значение типа Variant (Integer), содержащее целое число (от 0 до 59), которое представляет собой секунды в значении времени.

Синтаксис:
Second(время)

Обязательный аргумент *время* может быть любым значением типа Variant, числовым выражением, строковым выражением или любым их сочетанием, которое представляет собой значение времени. Если аргумент *время* содержит значение Null, возвращается значение Null.

Seek (VBA). Возвращает значение типа Long, определяющее текущее положение указателя чтения (записи) внутри файла, открытого с помощью инструкции Open.

Синтаксис:
Seek(номерФайла)

Обязательный аргумент *номерФайла* является выражением типа Integer, содержащим допустимый номер файла.

Функция Seek возвращает значение в интервале от 1 до 2 147 483 647 (т.е. $2^{31}-1$).

В режиме Random возвращаемым значением является номер записи, которая будет считана или записана следующей, а в режимах Binary, Output, Append, Input — номер байта, с которого начинается выполнение следующей операции ввода/вывода (первому байту файла соответствует номер 1, второму — 2 и т.п.).

Sgn (VBA). Возвращает значение типа Variant (Integer), соответствующее знаку указанного числа.

Синтаксис:
Sgn(число)

Обязательный аргумент *число* может представлять собой любое допустимое числовое выражение.

Если значение аргумента *число* больше нуля, функция Sgn возвращает 1, если его значение равно нулю, возвращается 0, а если значение меньше нуля, возвращается –1.

Знак аргумента *число* определяет значение, возвращаемое функцией Sgn.

Shell (VBA). Запускает исполняемую программу и при успешном ее запуске программы возвращает значение типа Variant (Double), представляющее собой идентификатор программы (в противном случае возвращается нуль).

Синтаксис:
Shell(pathname[,windowstyle])

Синтаксис функции Shell содержит аргументы, приведенные в табл. П2.52.

Таблица П2.52

Аргументы функции Shell

Аргумент	Описание
pathname	Обязательный. Имеет значение типа Variant (String), представляющее собой имя выполняемой программы и любые требуемые аргументы или ключи командной строки; допускает включение каталога или папки и диска
windowstyle	Необязательный. Имеет значение типа Variant (Integer), соответствующее типу окна, в котором выполняется программа. Если данный аргумент опущен, программа запускается в свернутом окне и получает фокус

Аргумент windowstyle может определяться строковой константой или иметь числовое значение:

Константа	Значение	Описание
vbHide	0	Окно скрыто, и фокус передается скрытому окну
vbNormalFocus	1	Окно имеет фокус и восстанавливает свои стандартные размер и положение
vbMinimizedFocus	2	Окно отображается в виде значка с фокусом
vbMaximizedFocus	3	Окно развертывается на полный экран с фокусом
vbNormalNoFocus	4	Восстанавливаются предыдущие размер и положение окна. Активным остается текущее окно
vbMinimizedNoFocus	6	Окно отображается в виде значка. Активным остается текущее окно

При успешном запуске указанного файла функцией Shell она возвращает идентификатор (ID) запущенной программы. Идентификатор задачи ID является уникальным номером, указывающим на выполняемую программу. Если функция Shell не может запустить указанную программу, возникает ошибка. При использовании функции MacID с функцией Shell в Microsoft Windows возникает ошибка.

Функция Shell запускает другие программы в асинхронном режиме. Это означает, что для продолжения выполнения инструкций, следующих за Shell, не требуется завершения программы, запущенной с ее помощью.

Приведем пример использования функции Shell для запуска приложения, указанного пользователем:

```
'В Microsoft Windows:
'1 в качестве второго аргумента открывает приложение
'в окне обычного размера и передает ему фокус
Dim RetVal
RetVal = Shell("C:\WINDOWS\CALC.EXE", 1)
'Запускает калькулятор.
```

Sin (VBA). Возвращает значение типа Double, содержащее синус угла.

Синтаксис:
Sin(число)

Обязательный аргумент *число* представляет собой значение типа Double или любое допустимое числовое выражение, задающее угол в радианах.

Функция Sin определяет отношение длины двух сторон прямоугольного треугольника (противолежащего катета и гипотенузы) по указанному углу (в радианах).

Значение, возвращаемое данной функцией, лежит в диапазоне от –1 до 1.

Для преобразования градусов в радианы следует умножить градусы на π/180. Для преобразования радиан в градусы следует умножить радианы на 180/π.

Space (VBA). Возвращает значение типа Variant (String), содержащее указанное число пробелов.

Синтаксис:
Space(число)

Обязательный аргумент *число* указывает нужное число пробелов в строке.

Функцию Space удобно использовать для форматирования и очистки данных в строках фиксированной длины.

Spc (VBA). Используется вместе с инструкцией Print # или методом Print для установки позиции вывода.

Синтаксис:
Spc(n)

Обязательный аргумент n задает число пробелов, которые следует вставить перед выводом на экран или печать следующего выражения в списке.

Если n меньше, чем ширина строки вывода, следующий символ печатается сразу после указанного числа пробелов. Если n больше, чем ширина строки вывода, следующая позиция печати вычисляется по формуле

```
текущаяПозицияВывода + (n Mod ширина)
```

Например, если текущая позиция печати равняется 24, а ширина строки вывода равняется 80, то выражение Spc(90) установит следующую позицию печати равной 34 (текущая позиция печати + остаток от деления 90 на 80). Если разность между текущей позицией печати и шириной строки вывода меньше, чем n (или n Mod ширина), функция Spc задает переход в начало следующей строки и вставку пробелов:

```
n  -  (ширина - текущаяПозицияВывода)
```

При использовании метода Print и пропорционального шрифта поле печати разбивается на позиции фиксированной ширины, которая равняется средней ширине всех символов текущего размера в используемом шрифте. Такую же ширину имеют пробелы, задаваемые функцией Spc. Следует отметить, что не существует зависимости между числом напечатанных символов и числом позиций фиксированной ширины, занимаемых этими символами. Например, прописная буква "W" шире одной позиции, а строчная буква "I" уже.

Sqr (VBA). Возвращает значение типа Double, содержащее квадратный корень указанного числа.

Синтаксис:
Sqr(число)

Обязательный аргумент *число* представляет собой значение типа Double или любое допустимое неотрицательное числовое выражение.

StDev (DAO), StDevP (DAO). Возвращают соответственно смещенное и несмещенное значения среднеквадратичных отклонений, вычисляемых по набору значений, содержащихся в указанном поле запроса.

Синтаксисы:
StDev(выражение)
StDevP(выражение)

Аргумент *выражение* является строковым выражением, которое определяет поле, содержащее вычисляемые числовые данные, или выполняет вычисления с данными из этого поля. Операнды аргумента *выражение* могут включать в себя имя поля таблицы, константу или функцию. (Функция может быть внутренней или определяться пользователем, но не может быть ни одной из других статистических функций SQL.)

Функции StDevP и StDev вычисляют соответственно смещенное и несмещенное среднеквадратичные отклонения.

Если базовый запрос содержит меньше двух записей (или не содержит записей для функции StDevP), эти функции возвращают значение Null (что означает невозможность вычисления среднеквадратичного отклонения).

Функции StDev и StDevP используются в выражении запроса, а также в свойстве SQL объекта QueryDef или при создании объекта Recordset на основе запроса SQL.

Str (VBA). Возвращает значение типа Variant (String), являющееся строковым представлением числа.

Синтаксис:
Str(число)

Обязательный аргумент *число* имеет тип Long и может задаваться любым допустимым числовым выражением.

При преобразовании числа в строку в начале строки обязательно резервируется позиция для знака числа. Если аргумент *число* является положительным, возвращенная строка будет содержать пробел на месте знака.

Чтобы представить числовое значение как дату, время, денежное значение или в специальном формате, следует использовать функцию Format, которая в отличие от функции Str не резервирует позицию для знака положительного числа.

В качестве допустимого десятичного разделителя функция Str воспринимает только точку (.). При наличии другого десятичного разделителя (например, запятой) для преобразования чисел в строки следует использовать функцию CStr.

StrComp (VBA). Возвращает значение типа Variant (Integer), представляющее собой результат сравнения строк.

Синтаксис:
StrComp(string1, string2[, compare])

Синтаксис функции StrComp содержит аргументы, приведенные в табл. П2.53.

Таблица П2.53

Аргументы функции StdComp

Аргумент	Описание
string1	Обязательный. Представляет собой любое допустимое строковое выражение
string2	Обязательный. Представляет собой любое допустимое строковое выражение
compare	Необязательный. Указывает способ сравнения строк. Данный аргумент может быть опущен или иметь значение 0; 1; 2. Чтобы выполнить двоичное сравнение, следует указать значение 0 (используется по умолчанию). Чтобы выполнить посимвольное сравнение без учета регистра, следует указать 1. Только в Microsoft Access допускается использование значения 2 для выполнения сравнения на основании сведений, содержащихся в базе данных. Если аргумент compare имеет значение Null, возникает ошибка. Если же он опущен, способ сравнения строк определяется значением параметра инструкции Option Compare

Возвращаемые значения:

Условие	Возвращаемое значение
string1 меньше, чем string2	–1
string1 равняется string2	0
string1 больше, чем string2	1
string1 или string2 имеет значение Null	Null

StrConv (VBA). Возвращает значение типа Variant (String), содержащее преобразованную строку.

Синтаксис:
StrConv(string, conversion)

Синтаксис функции StrConv содержит аргументы, приведенные в табл. П2.54.

Таблица П2.54

Аргументы функции StdConv

Аргумент	Описание
string	Обязательный. Представляет собой строковое выражение, которое следует преобразовать
conversion	Обязательный; значение типа Integer. Представляет собой сумму значений, указывающих тип преобразования, которое следует выполнить

Аргумент conversion может определятся строковой константой или иметь числовое выражение:

Константа	Значение	Описание
vbUpperCase	1	Преобразование строки к верхнему регистру
vbLowerCase	2	Преобразование строки к нижнему регистру
vbProperCase	3	Преобразование первой буквы каждого слова в строке в прописную
vbWide*	4*	Преобразование однобайтовых символов в двухбайтовые
vbNarrow*	8*	Преобразование двухбайтовых символов в однобайтовые
vbKatakana**	16**	Преобразование символов хираганы в символы катаканы
vbHiragana**	32**	Преобразование символов катаканы в символы хираганы

Константа	Значение	Описание
vbUnicode	64	Преобразование строки в Unicode с помощью используемой по умолчанию системной кодовой страницы
vbFromUnicode	128	Преобразование строки из Unicode с помощью используемой по умолчанию системной кодовой страницы

* Применимо к дальневосточным национальным настройкам.
** Применимо только к японскому языку.

Указанные константы определяются в языке Visual Basic для приложений. Это означает, что их имена можно использовать в любом месте кода вместо фактических значений.

Большинство из констант могут быть объединены (например, vbUpperCase + vbWide). Исключением являются взаимно противоречащие константы (например, vbUnicode + vbFromUnicode). Использование констант vbWide, vbNarrow, vbKatakana и vbHiragana в несовместимой национальной настройке приводит к ошибкам.

Допустимые символы, служащие разделителями слов: Null — Chr$(0), горизонтальная табуляция — Chr$(9), перевод строки — Chr$(10), вертикальная табуляция — Chr$(11), конец страницы — (Chr$(12)), возврат каретки — Chr$(13), пробел (однобайтовая система) — Chr$(32).

Код пробела в двухбайтовых системах (DBCS) зависит от текущей страны.

String (VBA). Возвращает значение типа Variant (String), содержащее повторяющуюся строку указанной длины.

Синтаксис:
String(number, character)

Синтаксис функции String содержит аргументы, приведенные в табл. П2.55.

Таблица П2.55

Аргументы функции String

Аргумент	Описание
number	Обязательный; значение типа Long. Представляет собой длину возвращаемой строки. Если данный аргумент имеет значение Null, возвращается значение Null
character	Обязательный; значение типа Variant. Представляет собой код символа или строковое выражение, первый символ которого используется при создании возвращаемой строки. Если данный аргумент имеет значение Null, возвращается значение Null

Значения аргумента character, превышающие 255, преобразуются функцией String в допустимые коды символов по следующей формуле:

```
character Mod 256
```

Sum (DAO). Возвращает сумму набора значений, содержащихся в заданном поле запроса.

Синтаксис:
Sum(выражение)

Аргумент *выражение* является строковым выражением, которое определяет поле, содержащее добавляемые числовые данные, или выполняет вычисления с данными из этого поля. Операнды аргумента *выражение* могут включать в себя имя поля таблицы, константу или функцию. (Функция может быть внутренней или определяться пользователем, но не может быть ни одной из других статистических функций SQL.)

Функция Sum выполняет суммирование значений в поле. Например, функцию Sum можно использовать для определения полной стоимости доставки.

Функция Sum пропускает записи с полями, содержащими значения Null.

Приведем пример вычисления суммы произведений полей *Цена* и *Количество*:

```
SELECT
Sum(Цена * Количество)
AS [Общий доход] FROM Заказано
```

Функция Sum используется в выражении запроса, а также в свойстве SQL объекта QueryDef или при создании объекта Recordset на основе запроса SQL.

Switch (VBA). Вычисляет список выражений и возвращает значение типа Variant или выражение, соответствующее первому выражению в списке, которое имеет значение True.

Синтаксис:
Switch(выражение1, значение1[, выражение2, значение2 ... [, выражение n, значение n]])

Синтаксис функции Switch содержит аргументы, приведенные в табл. П2.56.

Таблица П2.56

Аргументы функции Switch

Аргумент	Описание
выражение	Обязательный. Представляет собой выражение типа Variant, подлежащее вычислению
значение	Обязательный. Представляет собой числовое значение или выражение, возвращаемое, если соответствующий аргумент *выражение* имеет значение True

Список аргументов функции Switch содержит пары выражений и значений. Выражения вычисляются в том порядке, в котором они включены в список (слева направо). Возвращается значение, соответствующее первому истинному выражению в списке. Если разбиение выражений и значений на пары выполнено неправильно, возникает ошибка выполнения. Например, если аргумент *выражение1* имеет значение True, функция Switch возвращает значение1. Если же он имеет значение False, а выражение2 имеет значение True, функция Switch возвращает значение2 и т.д.

Функция Switch возвращает значение Null в следующих случаях:

- ни одно из выражений не имеет значение True.
- первому выражению, имеющему значение True, соответствует значение Null.

Функция Switch вычисляет все выражения несмотря на то, что возвращается только одно из соответствующих им значений. В некоторых случаях это приводит к нежелательным побочным эффектам. Например, если при вычислении одного из выражений имеет место деление на нуль, возникает ошибка.

SysCmd. Используется для выполнения следующих действий:

- вывод в строке состояния индикатора выполнения или указанного текста;
- возвращение сведений о Microsoft Access и файлах приложения;
- возвращение сведений о состоянии объекта базы данных, показывающих, открыт ли данный объект, является ли он новым, был ли он изменен, но не сохранен.

Например, в специальной программе мастера функция SysCmd позволяет вывести индикатор выполнения, демонстрирующий успешное выполнение программы при создании мастером формы.

Синтаксис:

ReturnValue = SysCmd(действие[, текст][, значение])

ObjectState = SysCmd(действие[, типОбъекта][, имяОбъекта])

Функция SysCmd использует аргументы, приведенные в табл. П2.57.

Таблица П2.57

Аргументы функции SysCmd

Аргумент	Описание
действие	Одна из встроенных констант, определяющих тип выполняемого действия (см. табл. П2.58)
текст	Строковое выражение, определяющее текст, выводящийся в строке состояния с выравниванием по левому краю. Данный аргумент применяется, когда в аргументе *действие* указаны константы acSysCmdInitMeter, acSysCmdUpdateMeter или acSysCmdSetStatus. При других значениях аргумента *действие* этот аргумент не применяется
значение	Числовое выражение, определяющее состояние индикатора выполнения. Данный аргумент применяется, когда в аргументе *действие* указана константа acSysCmdInitMeter. При других значениях аргумента *действие* этот аргумент не применяется
типОбъекта	Одна из следующих встроенных констант: acTable; acQuery; acForm; acReport; acMacro; acModule. Данный аргумент применяется, когда в аргументе *действие* указана константа acSysCmdGetObjectState. При других значениях аргумента *действие* этот аргумент не применяется
имяОбъекта	Строковое выражение, являющееся допустимым именем объекта базы данных для типа, указанного в аргументе *типОбъекта*. Данный аргумент применяется, когда в аргументе *действие* указана константа acSysCmdGetObjectState. При других значениях аргумента *действие* этот аргумент не применяется

Набор констант для индикатора выполнения приведен в табл. П2.58. При успешном выполнении указанных действий функция SysCmd возвращает значение Null. В противном случае Microsoft Access генерирует ошибку выполнения.

Таблица П2.58

Набор констант для индикатора выполнения

Константа	Действие
acSysCmdInitMeter	Инициализирует индикатор выполнения. При использовании данной константы пользователь должен задать значения аргументов *текст* и *значение*

Окончание табл. П2.58

Константа	Действие
acSysCmdUpdateMeter	Обновляет индикатор выполнения с помощью указанного значения. При использовании данной константы необходимо задать значение аргументов *текст* и *значение*
acSysCmdRemoveMeter	Удаляет индикатор выполнения
acSysCmdSetStatus	Выводит текст в строке состояния
acSysCmdClearStatus	Сбрасывает текст в строке состояния
acSysCmdRuntime	Возвращает значение True (–1), если запущена версия Microsoft Access, предназначенная только для выполнения
acSysCmdAccessVer	Возвращает номер версии Microsoft Access
acSysCmdIniFile	Возвращает имя файла .ini, используемого Microsoft Access
acSysCmdAccessDir	Возвращает имя каталога, в котором хранится файл Msaccess.exe
acSysCmdProfile	Возвращает значение параметра /profile, указанного при запуске Microsoft Access с командной строки
acSysCmdGetWorkgroupFile	Возвращает путь к файлу рабочей группы (System.mdw)
acSysCmdGetObjectState	Возвращает сведения о состоянии объекта базы данных. При использовании данной константы необходимо задать значение аргументов *типОбъекта* и *имяОбъекта*

Для изменения текста, выводящегося в строке состояния, следует вызвать функцию SysCmd с константой acSysCmdSetStatus в аргументе *действие* и новым текстом сообщения в аргументе *текст*. Например, при выполнении операции сортировки удобно вывести в строке состояния сообщение *Сортировка...*, а после завершения сортировки удалить это сообщение. Максимальная длина строки в аргументе *текст* составляет примерно 80 символов. Поскольку текст в строке состояния выводится с помощью пропорционального шрифта, реальная длина сообщения определяется конкретным содержимым строки *текст*.

Увеличение ширины текста сообщения приводит к уменьшению ширины индикатора выполнения. Если сообщение, размер которого превышает ширину строки состояния, задается при константе acSysCmdInitMeter в аргументе *действие*, то функция SysCmd

игнорирует сообщение и не выводит никакого текста в строке состояния. Если же такой текст задается при константе acSysCmdSetStatus в аргументе *действие*, то функция SysCmd выводит сообщение, но обрезает текст по размеру строки состояния.

Не допускается указание в аргументе *текст* пустой строки (" "). Чтобы удалить существующее сообщение, выведенное в строке состояния, следует задать в аргументе *текст* единственный пробел. Удаление текста из строки состояния иллюстрируют следующие инструкции:

```
varReturn = SysCmd(acSysCmdInitMeter, "", 100)
varReturn = SysCmd(acSysCmdSetStatus, "")
```

Если в строке состояния выведен индикатор выполнения, то указание текста сообщения путем вызова функции SysCmd с константой acSysCmdSetStatus в аргументе *действие* приводит к автоматическому удалению индикатора.

Вызов функции SysCmd с другими константами действия позволяет получить системную информацию о Microsoft Access (в том числе номер исполняемой версии, является ли данная версия только исполняемой, адрес каталога исполняемого файла и имя файла инициализации Microsoft Access).

Как общие, так и специализированные настройки Microsoft Access сохраняются в реестре Windows, поэтому при разработке приложения использование файла .ini является излишним. Константа acSysCmdIniFile определят совместимость с предыдущими версиями Microsoft Access.

Чтобы получить сведения о состоянии конкретного объекта базы данных, следует вызвать функцию SysCmd с константой acSysCmdGetObjectState в аргументе *действие* и заданными аргументами *типОбъекта* и *имяОбъекта*. Различают четыре состояния объекта: неоткрытый, или несуществующий; открытый; новый; измененный, но не сохраненный.

Например, при разработке программы мастера, вставляющего новое поле в таблицу, требуется определить, была ли структура таблицы изменена, но еще не сохранена, чтобы сохранить ее перед новым изменением. Для этого следует проверить значение, возвращаемое функцией SysCmd.

Функция SysCmd с константой acSysCmdGetObjectState в аргументе *действие* может возвращать любую комбинацию следующих констант:

Константа	Состояние объекта базы данных
acObjStateOpen	Открыт
acObjStateNew	Новый
acObjStateDirty	Изменен, но не сохранен

Если объект, указанный в аргументе *имяОбъекта* не открыт или не существует, функция SysCmd возвращает нуль.

Tab (VBA). Используется вместе с инструкцией Print # или методом Print для указания позиции вывода.

Синтаксис:
Tab[(n)]

Необязательный аргумент n задает номер столбца, к которому следует перейти перед выводом на экран или печать следующего выражения из списка. Если данный аргумент опущен, Tab устанавливает курсор в начало следующей зоны печати. Это позволяет использовать функцию Tab в качестве разделителя списка вместо запятой, если в текущей национальной настройке запятая используется в качестве десятичного разделителя.

Если текущая позиция печати на текущей строке больше аргумента n, функция Tab вызывает переход к n-му столбцу на следующей строке вывода. Если n меньше 1, Tab переходит к столбцу 1. Если n больше, чем ширина строки вывода, следующая позиция печати вычисляется по формуле

```
n Mod ширина
```

Например, если ширина строки равняется 80, то выражение Tab(90) установит следующую позицию печати равной 10 (остаток от деления 90 на 80). Если n меньше, чем ее текущая позиция печати, то печать начинается в вычисленной позиции в следующей строке. Если вычисленная позиция печати больше, чем ее текущая позиция, то печать начинается в вычисленной позиции в той же строке.

Крайней левой позицией печати строки вывода всегда является 1. При печати в файл с помощью инструкции Print # крайней правой позицией ее является текущая ширина результирующего файла, которая устанавливается с помощью инструкции Width #.

При использовании метода Print и функции Tab поле печати разбивается на позиции фиксированной ширины, которая равняется средней ширине всех символов текущего размера в используемом шрифте. Следует отметить, что не существует зависимости между числом напечатанных символов и числом позиций фиксированной ширины, занимаемых этими символами. Например, прописная буква “W” шире одной позиции, а строчная буква “l” уже.

Tan (VBA). Возвращает значение типа Double, содержащее тангенс угла.

Синтаксис:
Tan(число)

Обязательный аргумент *число* представляет собой значение типа Double или любое допустимое числовое выражение, задающее угол в радианах.

Функция Tan определяет отношение двух сторон прямоугольного треугольника (противолежащего и прилегающего катетов) по указанному углу (в радианах).

Для преобразования градусов в радианы следует умножить градусы на $\pi/180$. Для преобразования радиан в градусы следует умножить радианы на $180/\pi$.

Time (VBA). Возвращает значение типа Variant (Date), содержащее текущее время по системным часам компьютера.

Синтаксис:
Time

Для установки системного времени используется инструкция Time.

Timer (VBA). Возвращает значение типа Single, представляющее собой число секунд, прошедших после полуночи.

Синтаксис:
Timer

TimeSerial (VBA). Возвращает значение типа Variant (Date), содержащее время, соответствующее указанным часу, минуте и секунде.

Синтаксис:
TimeSerial(hour, minute, second)

Синтаксис функции TimeSerial содержит аргументы, приведенные в табл. П2.59.

Таблица П2.59

Аргументы функции TimeSerial

Аргумент	Описание
hour	Обязательный; значение типа Variant (Integer). Представляет собой число от 0 (12:00 AM) до 23 (11:00 PM) или числовое выражение

Окончание табл. П2.59

Аргумент	Описание
minute	Обязательный; значение типа Variant (Integer). Представляет собой любое числовое выражение
second	Обязательный; значение типа Variant (Integer). Представляет собой любое числовое выражение

Значение каждого аргумента функции TimeSerial должно лежать в соответствующем диапазоне: 0...23 для часов и 0...59 для минут и секунд. Кроме того, можно использовать числовые выражения для описания времени на определенное число часов, минут и секунд более позднего или раннего, чем указанное. Приведем пример, в котором аргументами функции TimeSerial являются не абсолютные значения, а числовые выражения:

```
TimeSerial(12 – 6, -15, 0)
```

Следовательно, возвращается время на 6 ч и 15 мин более раннее, чем полдень, т.е. 5:45:00.

Если значение какого-либо аргумента превышает максимально допустимое для него, то соответствующим образом увеличивается более старший компонент времени. Например, 75 мин означают 1 ч и 15 мин. Однако если значение любого аргумента лежит вне диапазона 32 768...32 767 или сочетание всех трех аргументов описывает дату, лежащую вне допустимого диапазона дат, возникает ошибка.

TimeValue (VBA). Возвращает значение типа Variant (Date), содержащее время.

Синтаксис:
TimeValue(время)

Обязательный аргумент *время* обычно задается строковым выражением, представляющим собой время от 0:00:00 (12:00:00 AM) до 23:59:59 (11:59:59 PM). Кроме того, в качестве аргумента *время* можно использовать любое выражение, представляющее собой время в этом диапазоне. Если аргумент *время* содержит значение Null, возвращается значение Null.

При вводе времени можно использовать как 12-часовой, так и 24-часовой формат. Например, и “2:24PM”, и “14:24” являются допустимыми значениями аргумента *время*.

Если аргумент *время* содержит сведения о дате, эти сведения не возвращаются функцией TimeValue. Однако если аргумент *время* содержит недопустимые сведения о дате, возникает ошибка.

TypeName (VBA). Возвращает значение типа String.

Синтаксис:
TypeName(имяПеременной)

Обязательный аргумент *имяПеременной* представляет собой выражение типа Variant, определяющее любую переменную, за исключением переменной с определяемым пользователем типом.

Строка, возвращаемая функцией TypeName, может быть любой из представленных в табл. П2.60.

Таблица П2.60

Строка, возвращаемая функцией TypeName

Возвращаемая строка	Значение переменной
ТипОбъекта	Объект указанного типа
Byte	Байтовое
Integer	Целое
Long	Длинное целое
Single	С плавающей точкой обычной точности
Double	С плавающей точкой двойной точности
Currency	Денежное
Decimal	Типа Decimal
Date	Дата
String	Строковое
Boolean	Логическое
Error	Значение ошибки
Empty	Не инициализированное
Null	Пустое
Object	Объект
Unknown	Объект неизвестного типа
Nothing	Объектная переменная, не содержащая ссылки на объект

Если аргумент *имяПеременной* представляет собой массив, то возвращается одна из перечисленных в табл. П2.60 строк (или Variant) с добавлением пустых скобок.

Например, если аргумент *имяПеременной* представляет собой массив целых значений, то TypeName возвращает строку "Integer()".

UBound (VBA). Возвращает значение типа Long, содержащее максимальный доступный индекс указанной размерности массива.

Синтаксис:
UBound(имяМассива[, размерность])

Синтаксис функции UBound содержит аргументы, приведенные в табл. П2.61.

Таблица П2.61

Аргументы функции UBound

Аргумент	Описание
имяМассива	Обязательный. Представляет собой имя переменной массива, удовлетворяющее стандартным правилам именования переменных
размерность	Необязательный; значение типа Variant (Long). Представляет собой целое число, указывающее размерность, верхнюю границу которой возвращает функция. Для первой размерности используется значение 1, для второй — 2 и т. д. Если данный аргумент опущен, подразумевается значение 1

Функция UBound вместе с функцией LBound используется для определения размеров массива. Функция LBound применяется для выяснения нижнего предела размерности массива.

UCase (VBA). Возвращает значение типа Variant (String), содержащее строку, преобразованную к верхнему регистру.

Синтаксис:
UCase(строка)

Обязательный аргумент *строка* представляет собой любое допустимое строковое выражение. Если аргумент *строка* имеет значение Null, возвращается значение Null.

К верхнему регистру преобразуются только строчные буквы; прописные буквы и прочие символы остаются неизменными.

Val (VBA). Возвращает числа, содержащиеся в строке, как числовое значение соответствующего типа.

Синтаксис:
Val(строка)

Обязательный аргумент *строка* является любым допустимым строковым выражением.

Функция Val прекращает чтение строки на первом символе, который она не может распознать в качестве части числа. Символы, которые часто рассматриваются в качестве частей числовых значений типа знака доллара и запятых, не распознаются. Однако эта функция распознает префиксы оснований &O (для восьмеричных) и &H (для шестнадцатеричных значений). Пробелы, символы табуляции и символы перевода строк удаляются из значения аргумента.

Пример возвращения функцией Val числового значения 1615198:

```
Val("1615 198-я ул. Н.Й.")
```

Пример возвращения функцией Val десятичного значения –1 для шестнадцатеричного значения:

```
Val("&HFFFF")
```

Функция Val распознает в качестве разделителя целой и дробной частей значения только точку (.). При использовании других разделителей целой и дробной частей значения (например, в национальных версиях приложений) следует применять для преобразования строки в число функцию CDbl.

Var (DAO), VarP (DAO). Возвращают соответственно значения смещенной и несмещенной дисперсий, вычисляемых по набору значений, содержащихся в указанном поле запроса.

Синтаксисы:
Var(выражение)
VarP(выражение)

Аргумент *выражение* является строковым выражением, которое определяет поле, содержащее вычисляемые числовые данные, или выполняет вычисления с данными из этого поля. Операнды аргумента *выражение* могут включать в себя имя поля таблицы, константу или функцию. (Функция может быть внутренней или определяться пользователем, но не может быть ни одной из других статистических функций SQL.)

Функции VarP и Var вычисляют соответственно значения смещенной и несмещенной дисперсий.

Если базовый запрос содержит меньше двух записей, функции Var и VarP возвращают значение Null (что означает невозможность вычисления дисперсии).

Функции Var и VarP используются в выражении запроса или в инструкции SQL.

VarType (VBA). Возвращает значение типа Integer, указывающее подтип переменной.

Синтаксис:
VarType(имяПеременной)

Обязательный аргумент *имяПеременной* представляет собой выражение типа Variant, содержащее имя любой переменной (за исключением переменной с определяемым пользователем типом).

Возвращаемые значения:

Константа	Значение	Описание
vbEmpty	0	Empty (не инициализирована)
vbNull	1	Null (не содержит данных)
vbInteger	2	Целое
vbLong	3	Длинное целое
vbSingle	4	С плавающей точкой обычной точности
vbDouble	5	С плавающей точкой двойной точности
vbCurrency	6	Денежное
vbDate	7	Дата
vbString	8	Строковое
vbObject	9	Объект
vbError	10	Ошибка
vbBoolean	11	Логическое
vbVariant	12	Тип Variant (используется только для массивов типа Variant)
vbDataObject	13	Объект доступа к данным
vbDecimal	14	Значение типа Decimal
vbByte	17	Байтовое
vbArray	8192	Массив

Данные константы определены в языке Visual Basic для приложений. Использование имен этих констант вместо их значений допускается в любом месте программы.

Функция VarType никогда не возвращает значение vbArray само по себе. Эта константа всегда складывается с каким-либо другим значением, указывающим тип массива. Константа vbVariant возвращается только в сочетании с константой vbArray, чтобы показать, что в аргументе функции VarType указан массив типа Variant. Например, значение, возвращаемое массивом с целочисленны-

ми элементами, имеет вид vbInteger + vbArray, т.е. 8194. Если объект имеет свойство, используемое по умолчанию, то функция VarType(объект) возвращает тип этого свойства.

Weekday (VBA). Возвращает значение типа Variant (Integer), содержащее целое число, представляющее собой день недели.

Синтаксис:
Weekday(date, [firstdayofweek])

Синтаксис функции Weekday содержит аргументы, приведенные в табл. П2.62.

Таблица П2.62

Аргументы функции Weekday

Аргумент	Описание
date	Обязательный; значение типа Variant. Это числовое выражение, строковое выражение или любое их сочетание, представляющее собой дату. Если данный аргумент имеет значение Null, возвращается значение Null
firstdayofweek	Необязательный. Представляет собой константу, указывающую первый день недели. Если данный аргумент опущен, подразумевается константа vbSunday

Допустимые значения аргумента firstdayofweek:

Константа	Значение	Описание
vbUseSystem	0	Используется значение NLS API
vbSunday	1	Воскресенье (по умолчанию)
vbMonday	2	Понедельник
vbTuesday	3	Вторник
vbWednesday	4	Среда
vbThursday	5	Четверг
vbFriday	6	Пятница
vbSaturday	7	Суббота

Значения, возвращаемые функцией Weekday:

Константа	Значение	Описание
vbSunday	1	Воскресенье
vbMonday	2	Понедельник

Константа	Значение	Описание
vbTuesday	3	Вторник
vbWednesday	4	Среда
vbThursday	5	Четверг
vbFriday	6	Пятница
vbSaturday	7	Суббота

Year (VBA). Возвращает значение типа Variant (Integer), содержащее целое число, представляющее собой год.

Синтаксис:
Year(дата)

Обязательный аргумент *дата* может быть любым значением типа Variant, числовым выражением, строковым выражением или любым их сочетанием, представляющим собой дату. Если аргумент *дата* содержит значение Null, возвращается значение Null.

Комплекс лабораторных работ

ЛАБОРАТОРНАЯ РАБОТА № 1

Создание однотабличной базы данных

Цель работы

Приобретение специальных навыков работы в СУБД Access по созданию однотабличной базы данных

Общие сведения

СУБД Access позволяет хранить большие массивы данных в определенном формате и обрабатывать их, представляя в удобном для пользователей виде.

Access содержит набор инструментов для управления базами данных, включающий в себя конструкторы таблиц, форм, запросов и отчетов. Кроме того, Access можно рассматривать и как среду для разработки приложений. Используя макросы для автоматизации задач, можно создавать такие же мощные, ориентированные на пользователя приложения, как и приложения, разработанные с помощью «полноценных» языков программирования, дополнять их кнопками, меню и диалоговыми окнами.

Мастер (Wizard) — специальная программа для решения какой-то задачи или создания объекта определенного типа. Эта программа помогает пользователю за несколько минут выполнить рутинную работу, на которую иначе требуется несколько часов. Программа-мастер задает вопросы о содержании, стиле и формате объекта, а затем создает этот объект без какого-либо вмешательства со стороны пользователя. В Access имеется около сотни мастеров, предназначенных для проектирования баз данных, приложений, таблиц, форм, отчетов, графиков, почтовых наклеек, элементов управления и свойств.

Все составляющие базы данных (таблицы, отчеты, запросы, формы и объекты) в Access хранятся в едином дисковом файле. Основным структурным компонентом базы данных является таблица. В таблицах хранятся и все вводимые пользователем данные. Внешне каждая таблица Access 97 похожа на обычную таблицу и состоит из столбцов, называемых полями, и строк, называемых записями. Каждая запись таблицы содержит всю необходимую информацию об отдельном элементе базы данных. Например, за-

пись о преподавателе может содержать фамилию, имя, отчество, дату рождения, должность и т. п.

При разработке структуры таблицы прежде всего определяют названия полей, из которых она должна состоять, типы полей и их размеры. Каждому полю таблицы присваивается уникальное имя, которое не может содержать более 64 символов, при этом по имени должна узнаваться функция поля. Далее решают, данные какого типа будут содержаться в каждом поле. В Access можно выбирать любые из основных типов данных, которые соответственно присваиваются каждому полю. Значение типа поля может быть задано только в режиме конструктора. В табл. П3.1 представлены типы данных Access и дано их описание.

В Access существует четыре способа создания пустой таблицы.

1. Использование мастера баз данных для создания БД, содержащей все требуемые отчеты, таблицы и формы, за одну операцию (мастер баз данных создает новую БД, т. е. его нельзя использовать для добавления новых таблиц, форм, отчетов в уже существующую базу данных).

2. Использование мастера таблиц, позволяющего выбрать поля для данной таблицы из множества определенных ранее таблиц, таких как деловые контакты, список личного имущества или рецепты.

3. Ввод данных непосредственно в пустую таблицу в режиме таблицы. При сохранении новой таблицы в Access данные анализируются, и каждому полю присваиваются необходимый тип данных и формат.

4. Определение всех параметров макета таблицы в режиме конструктора.

Ввод данных в ячейки таблицы производится обычным образом. Однако для некоторых типов данных (числового, денежного, дата/время, логического) Access автоматически проверяет правильность их ввода. Например, если ввести букву в ячейку с числовым типом, то Access выдаст сообщение о неправильно введенном значении и не позволит перейти к другой ячейке, пока не будут введены правильные данные.

Для всех типов полей (кроме *Счетчик* и *Поле объекта OLE*) можно самостоятельно задавать ограничения для вводимых данных. Для этого в режиме конструктора надо выбрать вкладку *Общие*, перевести курсор в поле с именем *Условия на значение* и ввести ограничение на данные. Ограничение можно вводить и на текстовое поле. Обычно в этом случае задаются слова, которые могут присутствовать в данном поле. Вводить ограничения можно не только вручную, но и с помощью построителя выражений.

Можно использовать еще один удобный инструмент при вводе данных — параметр *Значение по умолчанию* (который находится также на вкладке *Общие*). Таким образом можно задать данные,

Таблица П3.1

Типы данных Access

Тип данных	Описание
Текстовый (значение по умолчанию)	Текст или числа, не требующие проведения расчетов, например номера телефонов (до 255 знаков)
Числовой	Числовые данные различных форматов, используемые для проведения расчетов
Дата/время	Предназначен для хранения информации о дате и времени с 100 по 9999 год
Денежный	Денежные значения и числовые данные, используемые в математических расчетах, проводящихся с точностью до 15 знаков в целой и до 4 знаков в дробной частях
Поле MEMO	Предназначен для хранения комментариев (до 65 535 символов)
Счетчик	Специальное числовое поле, в котором Access автоматически присваивает уникальный порядковый номер каждой записи. Значения полей этого типа обновлять нельзя
Логический	Принимает только одно из двух возможных значений (True/False, *Да/Нет*)
Поле объекта OLE	Объект (например, электронная таблица Microsoft Excel, документ Microsoft Word, рисунок, звукозапись или другие данные в двоичном формате), связанный или внедренный в таблицу Access
Гиперссылка	Строка, состоящая из букв и цифр и представляющая собой адрес гиперссылки, который может состоять максимум из трех частей: текста, выводимого в поле или элемент управления, и пути к файлу (в формате пути UNC) или странице (адрес URL). Чтобы вставить адрес гиперссылки в поле или элемент управления, следует выполнить команду *Вставка*, *Гиперссылка*
Мастер подстановок	Создает поле, в котором предлагается выбор значений из списка или из поля со списком, содержащего набор постоянных значений или значений из другой таблицы. В действительности это не тип поля, а способ его хранения

которые Access будет вводить по умолчанию при заполнении таблицы. Это удобно использовать, когда большинство значений данного поля одинаковы. Например, должность большинства препо-

давателей — доцент. Если ввести это слово в строку параметра *Значение по умолчанию*, то все ячейки поля *Должность* примут значение *Доцент*, и надо будет лишь изменить значения ячеек для преподавателей с другими должностями. Данные можно вводить и копированием их из одной ячейки в другую стандартными средствами Windows.

Для каждого типа данных (кроме *Счетчик*) предусмотрено пустое (нулевое) значение, при этом различают пустые (Null) значения и пустые строки. Если пустое поле имеет пустое (Null) значение, то это означает, что данные для него существуют, но пока неизвестны. Если же введена пустая строка (два знака прямых кавычек (“ ”), это означает, что данные не существуют вовсе. Access позволяет обрабатывать такие пустые значения.

Для удобства просмотра можно сортировать записи в таблице в определенной последовательности, например в таблице ПРЕПОДАВАТЕЛИ записи можно отсортировать в порядке убывания стажа преподавателей. Кнопки сортировки на панели инструментов (или команды меню *Записи*, *Сортировка*, *Сортировка по возрастанию*, *Сортировка по убыванию*) позволяют сортировать столбцы по возрастанию или убыванию. Прежде чем щелкнуть мышью по кнопке сортировки, следует выбрать поля, используемые для сортировки. Современные СУБД (такие, как Access) никогда не сортируют таблицы физически, как это делалось раньше. Средства сортировки данных (а также фильтрации, поиска и замены) реализованы в них как автоматически создаваемые запросы. Записи таблицы всегда располагаются в файле базы данных в том порядке, в котором они были добавлены в таблицу.

Отбор данных может производиться с помощью фильтра, т.е. набора условий, применяемых для отбора подмножества записей. В Access существуют фильтры четырех типов: фильтр по выделенному фрагменту, обычный фильтр, расширенный фильтр и фильтр по вводу.

Фильтрование данных в Access производится с помощью двух кнопок: [Фильтр по выделенному] и [Изменить фильтр] (команды меню *Записи*, *Фильтр*, *Изменить фильтр*). После нажатия второй кнопки от таблицы остается одна запись. При этом каждое поле становится полем со списком (когда в нем находится курсор), в котором можно выбрать все значения для данного поля. После щелчка мышью по кнопке [Применить фильтр] выбираются записи, соответствующие измененному фильтру. Еще более сложные условия фильтрации можно задать командами меню *Записи*, *Фильтр*, *Расширенный фильтр*.

Задание на лабораторную работу

1. Создать новую базу данных.
2. Создать таблицу базы данных.

3. Определить поля создаваемой таблицы в соответствии с табл. П3.2.

4. Сохранить созданную таблицу.

5. Заполнить созданную таблицу данными в соответствии с табл. П3.3.

6. Просмотреть, как будет выглядеть созданная таблица на листе бумаги.

7. С помощью мастера форм создать форму СОСТАВ ПРЕПОДАВАТЕЛЕЙ (в один столбец).

8. Просмотреть, как будет выглядеть форма на листе бумаги.

Таблица П3.2

Таблица данных ПРЕПОДАВАТЕЛИ

Имя поля	Тип данных	Размер, формат, свойства поля
Код преподавателя	Счетчик	
Фамилия	Текстовый	15
Имя	Текстовый	15
Отчество	Текстовый	15
Дата рождения	Дата/время	Краткий
Должность	Текстовый	9
Дисциплина	Текстовый	11
Телефон	Текстовый	9
Зарплата	Денежный	(Число десятичных знаков — 0)

Таблица П3.3

Информация для заполнения базы данных

Код преподавателя	Фамилия	Имя	Отчество	Дата рождения	Должность	Дисциплина	Телефон	Зарплата, р.
1	Ивушкин	Владимир	Семенович	04.12.49	Доцент	Информатика	123-45-67	1900
2	Морозов	Петр	Павлович	24.11.66	Профессор	Экономика	231-44-42	2200
3	Громов	Евгений	Иванович	21.11.67	Доцент	Математика	234-56-78	1700
4	Андреева	Светлана	Ивановна	10.01.71	Ассистент	Математика	345-67-89	1300
5	Пак	Ольга	Федоровна	01.06.55	Доцент	Экономика	456-78-90	1650
6	Бухтеева	Елена	Петровна	12.03.55	Доцент	Информатика	567-89-01	1600
7	Морозов	Алексей	Николаевич	22.12.58	Доцент	Физика	456-22-11	1800

Технология выполнения задания

1. Для создания новой базы данных следует:

• загрузить Access и в появившемся окне выбрать пункт *Новая база данных*;

• в окне *Файл новой базы данных* задать имя базы (пункт *Имя файла*) и выбрать папку (пункт *Папка*), где эта база данных будет находиться. По умолчанию Access предлагает имя базы db1, а тип файла — Базы данных Access. Задать имя *Преподаватели*, а тип файла оставить прежним (другие типы файлов используются в специальных случаях);

• щелкнуть мышью по кнопке [Создать].

2. Для создания таблицы базы данных необходимо:

• в окне базы данных выбрать вкладку *Таблицы*, а затем щелкнуть мышью по кнопке [Создать];

• в окне *Новая таблица* выбрать пункт *Конструктор* и щелкнуть мышью по кнопке [OK].

В результате проделанных операций откроется окно таблицы в режиме конструктора, в котором следует определить поля таблицы.

3. Для определения полей создаваемой таблицы следует:

• ввести в строку столбца *Имя поля* имя первого поля *Код преподавателя*;

• в строке столбца *Тип данных* щелкнуть мышью по кнопке списка и выбрать тип данных *Счетчик*. Поля вкладки *Общие* оставить такими, как предлагает Access.

Заполнение строки столбца *Описание* необязательно и обычно используется для внесения дополнительных сведений о поле.

4. Для сохранения таблицы необходимо:

• выбрать пункт меню *Файл*, *Сохранить*;

• в диалоговом окне *Сохранение* ввести имя таблицы ПРЕПОДАВАТЕЛИ;

• щелкнуть мышью по кнопке [OK].

5. Ввести данные в созданную таблицу в соответствии с табл. П3.3.

6. Для просмотра созданной таблицы следует:

• щелкнуть по кнопке  или выполнить команду *Файл*, *Предварительный просмотр*;

• закрыть окно просмотра.

7. Для создания формы СОСТАВ ПРЕПОДАВАТЕЛЕЙ необходимо:

• открыть вкладку *Формы* в окне базы данных;

• щелкнуть по кнопке [Создать];

• в появившемся окне выбрать (подвести курсор мыши и щелкнуть левой ее кнопкой) пункт *Мастер форм* (или *Создание формы с помощью мастера*);

• щелкнуть по значку списка *Таблицы и запросы* в левой части окна и выбрать в списке таблицу ПРЕПОДАВАТЕЛИ;

• щелкнуть мышью по кнопке [OK];

• в появившемся окне выбрать поля, которые будут присутствовать в форме. Так как в данном примере присутствовать будут все поля, щелкнуть мышью по кнопке  ;

• щелкнуть мышью по кнопке [Далее];

• так как в появившемся окне уже выбран вид *Форма в один столбец*, щелкнуть мышью снова по кнопке [Далее];

• в появившемся окне выбрать стиль оформления. После выбора стиля щелкнуть мышью по кнопке [Далее];

• в появившемся окне задать имя формы, набрав на клавиатуре параметр СОСТАВ ПРЕПОДАВАТЕЛЕЙ. Остальные параметры в окне оставить без изменений;

• щелкнуть мышью по кнопке [Готово].

8. Для просмотра созданной формы следует:

• щелкнуть мышью по кнопке  или выполнить команду *Файл, Предварительный просмотр*;

• закрыть окно просмотра.

Контрольные вопросы

1. Дать краткую характеристику СУБД Access.
2. Что такое реляционная СУБД?
3. Перечислить (кратко) сервисные возможности Access.
4. Перечислить типы данных, допустимых для использования в Access.
5. Что представляют собой и как осуществляются сортировка и фильтрация данных?
6. Кратко описать технологию создания БД.
7. Какими способами осуществляется заполнение БД?
8. Описать технологию ввода и просмотра данных посредством формы.

ЛАБОРАТОРНАЯ РАБОТА № 2

Формирование запросов и отчетов для однотабличной базы данных

Цель работы

Приобретение специальных навыков работы в СУБД Access по формированию запросов и отчетов для однотабличной базы данных.

Общие сведения

Формирование запросов на выборку. Запросы являются мощным средством обработки данных, хранимых в таблицах Access 97.

С помощью запросов можно просматривать, анализировать и изменять данные из нескольких таблиц. Они также используются в качестве источника данных для форм и отчетов. Запросы позволяют вычислять итоговые значения и выводить их в компактном формате, подобном формату электронной таблицы, а также выполнять вычисления над группами записей.

Запросы можно создавать самостоятельно и с помощью мастеров. Мастера запросов автоматически выполняют основные действия в зависимости от ответов пользователя на поставленные вопросы. Самостоятельно разработать запросы можно в режиме конструктора.

В Access (Access 97) можно создавать следующие типы запросов:

- запрос на выборку;
- запрос с параметрами;
- перекрестный запрос;
- запрос на изменение (удаление, обновление и добавление записей, создание таблицы);
- запросы SQL (запросы на объединение, запросы к серверу, управляющие и подчиненные запросы).

Запрос на выборку используется наиболее часто. При его выполнении данные, удовлетворяющие условиям отбора, выбираются из одной или нескольких таблиц и выводятся в определенном порядке.

Можно также использовать запрос на выборку, чтобы сгруппировать записи для вычисления сумм, средних значений, пересчета и других действий. Например, используя запрос на выборку, можно получить данные о среднем стаже доцентов и профессоров (на основе таблицы ПРЕПОДАВАТЕЛИ).

Запрос с параметрами — это запрос, при выполнении которого в соответствующем диалоговом окне пользователю выдается приглашение ввести данные, на основе которых он будет выполняться. Например, часто требуются данные о том, какие дисциплины ведут преподаватели. В этом случае вместо создания отдельных запросов по каждому преподавателю можно создать один запрос с параметрами, в котором в качестве параметра будет использоваться фамилия преподавателя. При каждом вызове этого запроса предлагается ввести фамилию преподавателя, после чего на экран выводятся все поля, указанные в запросе, например фамилия, имя, отчество преподавателя и читаемая им дисциплина.

Для создания нового запроса надо в окне базы данных выбрать вкладку *Запросы* и щелкнуть мышью по кнопке [Создать]. В открывшемся окне *Новый запрос* следует выбрать один из пяти пунктов: *Конструктор*, *Простой запрос*, *Перекрестный запрос*, *Повторяющиеся записи*, *Записи без подчиненных*.

Конструктор позволяет самостоятельно создать любой тип запроса, но этот режим рекомендуется пользователям, уже имеющим некоторый опыт создания запросов. Простой запрос позволяет создать с помощью мастера запрос на выборку из определенных полей таблиц или других запросов.

При выполнении запроса на выборку Access извлекает записи из таблиц и формирует результирующий набор данных, который выглядит, как таблица, не являясь ею. Результирующий набор данных — это *динамический* (или *виртуальный*) набор записей, не хранящийся в базе данных, т.е. после закрытия запроса результирующий набор данных этого запроса прекращает свое существование.

При сохранении запроса остается только его структура — перечень таблиц, список полей, порядок сортировки, ограничения на записи, тип запроса и т.д. Запрос при сохранении в базе данных по сравнению с результирующим набором данных имеет ряд преимуществ:

- на физическом носителе информации (обычно это жесткий диск) для него требуется меньше места;
- в нем могут использоваться обновленные версии любых записей, измененных со времени последнего запуска.

При каждом выполнении запрос обращается к базовым таблицам и снова создает результирующий набор данных. Поскольку сам по себе результирующий набор данных не сохраняется, запрос автоматически отображает любые изменения, которые произошли в базовых таблицах с момента его последнего запуска (даже в реальном времени в многопользовательской среде).

Для сохранения запроса следует произвести следующие действия. Выполнить команду *Файл*, *Сохранить* или щелкнуть мышью по кнопке [Сохранить] на панели инструментов. Если этот запрос сохраняется впервые, необходимо ввести его новое имя в диалоговом окне *Сохранение*.

Формирование отчетов. Отчет — это гибкое и эффективное средство для организации просмотра и распечатки итоговой информации. В отчете можно получить результаты сложных расчетов, статистических сравнений, а также поместить в него рисунки и диаграммы.

Пользователь имеет возможность разработать отчет самостоятельно или создать его с помощью мастера. Мастер по разработке отчетов быстро выполняет всю рутинную работу. После вызова мастера выводятся диалоговые окна с приглашением ввести необходимые данные, и отчет создается на основании ответов пользователя. Мастер необходим даже для опытных пользователей, так как позволяет быстро разработать макет, служащий основой создаваемого отчета. После этого можно, переключившись в режим конструктора, внести изменения в стандартный макет.

При работе с мастером в зависимости от того, какой отчет необходимо создать (т.е. каковы ответы на вопросы мастера), Access предлагает различные варианты макетов. Например, при создании простого отчета без группировки данных предлагается три варианта макета: в столбец, табличный и выровненный. При этом в небольшом окне представляется вид этих макетов. Если задаются уровни группировки данных (т.е. признаки, по которым их надо сгруппировать, например по должности), предлагается шесть видов макетов.

Основное различие между отчетами и формами заключается в их назначении. Если формы предназначены преимущественно для ввода данных, то отчеты — для их просмотра (на экране либо на бумаге). В формах используются вычисляемые поля (обычно с помощью вычислений на основе полей в текущей записи). В отчетах вычисляемые поля (итоги) формируются на основе общей группы записей, страницы записей или всех записей отчета. Все, что можно сделать с формой (за исключением ввода данных), можно сделать и с отчетом. Действительно, форму можно сохранить в виде отчета, а затем изменить элементы ее управления в окне конструктора отчета.

Для создания отчета надо открыть вкладку *Отчеты* и щелкнуть мышью по кнопке [Создать], после чего откроется окно *Новый отчет*, в котором приведены шесть пунктов меню, т.е. шесть способов создания отчета: *Конструктор*, *Мастер отчетов*, *Автоотчет в столбец*, *Автоотчет ленточный*, *Мастер диаграмм* и *Почтовые наклейки*.

Конструктор позволяет самостоятельно создать отчет, однако это непросто даже для опытного пользователя.

Мастер отчетов автоматически создает отчет на основе выбранных полей таблиц (запросов) и макетов отчетов. Этот способ создания отчетов является наиболее удобным как для начинающих, так и для опытных пользователей.

Автоотчет в столбец и *Автоотчет ленточный* — простейшие способы создания отчетов, т.е. здесь достаточно указать только имя таблицы (запроса), на основе которой будет создаваться отчет, все остальное сделает *Мастер отчетов*.

Мастер диаграмм поможет создать отчет в виде диаграммы.

Почтовые наклейки — способ создания отчета, отформатированного для печати почтовых наклеек.

Задание на лабораторную работу

1. На основе таблицы ПРЕПОДАВАТЕЛИ создать простой запрос на выборку, в котором должны отображаться фамилии, имена, отчества преподавателей и их должности.
2. Сохранить запрос.

3. На основе таблицы ПРЕПОДАВАТЕЛИ создать отчет с группированием данных по должностям.

Технология выполнения задания

1. Для создания простого запроса следует:
   - в окне базы данных открыть вкладку *Запросы*;
   - в открывшемся окне щелкнуть мышью по кнопке [Создать];
   - из появившихся пунктов окна *Новый запрос* выбрать *Простой запрос* и щелкнуть мышью по кнопке [OK];
   - в появившемся окне в строке *Таблицы и запросы* выбрать таблицу ПРЕПОДАВАТЕЛИ (если других таблиц или запросов не было создано, она будет одна в открывшемся списке);
   - в окне *Доступные поля* перевести выделение на параметр *Фамилия*;
   - щелкнуть мышью по кнопке  , в результате чего слово *Фамилия* перейдет в окно *Выбранные поля*;
   - аналогично в окно *Выбранные поля* перевести поля *Имя*, *Отчество*, *Должность*;
   - щелкнуть мышью по кнопке [Далее];
   - в строке параметра *Задайте имя запроса* ввести новое имя *Должности преподавателей*;
   - щелкнуть мышью по кнопке [Готово], после чего на экране появится таблица с результатами запроса.
2. Для сохранения запроса необходимо:
   - щелкнуть мышью по кнопке  или выполнить команду *Файл*, *Сохранить*;
   - закрыть окно запроса.
3. Для создания отчета следует:
   - открыть вкладку *Отчеты* и щелкнуть мышью по кнопке [Создать];
   - в открывшемся окне выбрать пункт *Мастер отчетов*;
   - щелкнуть мышью по значку раскрывающегося списка в нижней части окна;
   - выбрать из появившегося списка таблицу ПРЕПОДАВАТЕЛИ;
   - щелкнуть мышью по кнопке [OK]. В появившемся окне выбрать поля, которые будут присутствовать в форме. Так как в данном примере присутствовать будут все поля из таблицы, надо щелкнуть мышью по кнопке  ;
   - щелкнуть мышью по кнопке [Далее];
   - в появившемся окне приведен перечень полей. Перевести выделение на поле *Должность*;

• щелкнуть мышью по кнопке  , задав таким образом группировку данных по должности;

• щелкнуть мышью по кнопке [Далее];

• так как параметры в появившемся окне оставляем без изменений, надо снова щелкнуть мышью по кнопке [Далее];

• в появившемся окне *Вид макета для отчета*, а также в следующем окне *Требуемый стиль* выбрать стиль оформления отчета;

• щелкнуть мышью по кнопке [Далее];

• в появившемся окне ввести название отчета ПРЕПОДАВАТЕЛИ;

• щелкнуть мышью по кнопке [Готово], после чего на экране появится сформированный отчет;

• просмотреть, а затем закрыть отчет.

Контрольные вопросы

1. Что такое запросы? Какими возможностями они обладают?
2. Перечислить и охарактеризовать основные типы запросов, используемых в СУБД Access.
3. Что такое отчеты? Какими возможностями они обладают?
4. Кратко описать технологию создания запросов на выборку.
5. Описать технологию создания отчетов с группировкой данных (на примере создания отчета по таблице ПРЕПОДАВАТЕЛИ с группировкой данных по должностям).

ЛАБОРАТОРНАЯ РАБОТА № 3

Разработка информационно-логической модели реляционной базы данных

Цель работы

Приобретение специальных навыков работы в СУБД Access по разработке информационно-логической модели и созданию структуры реляционной базы данных.

Общие сведения

Слово «реляционная» происходит от английского relation — отношение. Отношение — математическое понятие, но в терминологии моделей данных отношения удобно изображать в виде таблиц, строки которых соответствуют кортежам отношения, а столбцы — атрибутам. Ключом называют любую функцию от атрибутов кортежа, которая может быть использована для его иденти-

фикации. Такая функция может быть значением одного из атрибутов (простой ключ) или задаваться алгебраическим выражением, включающим в себя значения нескольких атрибутов (составной ключ). Несмотря на то что данные в строках каждого из столбцов составного ключа могут и повторяться, комбинация данных каждой строки этих столбцов является уникальной. Например, в таблице СТУДЕНТЫ есть столбцы *Фамилия* и *Год рождения*. В каждом из столбцов есть некоторые повторяющиеся данные, т. е. одинаковые фамилии и одинаковые года рождения. Но если студенты, имеющие одинаковые фамилии, имеют разные года рождения, то эти столбцы можно использовать в качестве составного ключа.

В Access можно выделить три типа ключевых полей: простой ключ, составной ключ и внешний ключ.

Одно из важнейших достоинств реляционных баз данных состоит в том, что можно хранить логически сгруппированные данные в разных таблицах и задавать связи между ними, объединяя эти таблицы в единую базу.

Для задания связи таблицы должны иметь поля с одинаковыми именами или хотя бы с одинаковыми форматами данных. Связь между таблицами устанавливает отношения между совпадающими значениями в этих полях. Такая организация позволяет уменьшить избыточность хранимых данных, упрощает их ввод и организацию запросов и отчетов. Поясним это на следующем примере.

В Access можно задать три типа связей между таблицами: *один ко многим*, *многие ко многим*, *один к одному*.

Наиболее часто используемый тип связи между таблицами один ко многим. При такой связи каждой записи в таблице А может соответствовать несколько записей в таблице *B* (поля с этими записями называют внешними ключами), а запись в таблице *B* не может иметь более одной соответствующей ей записи в таблице *A*.

При связи типа многие ко многим одной записи в таблице *A* может соответствовать несколько записей в таблице *B*, а одной записи в таблице *B* — несколько записей в таблице *A*. Такая схема реализуется только с помощью третьей (связующей) таблицы, ключ которой состоит по крайней мере из двух полей, одно из которых является общим с таблицей *A*, а другое — общим с таблицей *B*.

При связи типа один к одному запись в таблице *A* может иметь не более одной связанной записи в таблице *B*, и наоборот. Этот тип связи используется не очень часто, поскольку такие данные можно поместить в одну таблицу. Связь типа один к одному применяют для разделения очень широких таблиц, отделения части таблицы в целях ее защиты, а также для сохранения сведений, относящихся к подмножеству записей в главной таблице.

Тип создаваемой связи зависит от полей, для которых определяется эта связь:

- связь типа один ко многим задается в том случае, когда только одно из полей является ключевым или имеет уникальный индекс, т.е. значения в этом поле не повторяются;
- связь типа один к одному задается в том случае, когда оба связываемых поля являются ключевыми или имеют уникальные индексы;
- связь типа многие ко многим фактически представляет собой две связи типа один ко многим через третью таблицу, ключ которой состоит по крайней мере из двух полей, общих для двух других таблиц.

Целостность данных определяет систему правил, используемых в СУБД Access для поддержания связей между записями в связанных таблицах (таблицах, объединенных с помощью связи), а также обеспечивает защиту от случайного удаления или изменения связанных данных. Контролировать целостность данных можно при выполнении следующих условий:

- связанное поле (поле, посредством которого осуществляется связь) одной таблицы является ключевым полем или имеет уникальный индекс;
- связанные поля имеют один тип данных. (Здесь существует исключение: поле счетчика может быть связано с числовым полем, если оно имеет тип *Длинное целое*.);
- обе таблицы принадлежат одной базе данных Access.

Для установки целостности данных база данных, в которой находятся таблицы, должна быть открыта. Для связанных таблиц из баз данных других форматов установить целостность данных невозможно.

Задание на лабораторную работу

1. Создать базу данных Деканат в соответствии с логической схемой, приведенной на рис. П3.1.
2. Создать структуру таблицы СТУДЕНТЫ.
3. Создать структуру таблицы ДИСЦИПЛИНЫ.
4. Изменить структуру таблицы ПРЕПОДАВАТЕЛИ.
5. Создать структуру таблицы ОЦЕНКИ.
6. Разработать схему данных, т.е. задать связи между созданными таблицами.
7. Создать форму *Студенты*.
8. Заполнить данными таблицу СТУДЕНТЫ, используя форму *Студенты*.
9. Создать форму *Дисциплины*.
10. Заполнить данными таблицу ДИСЦИПЛИНЫ используя форму *Дисциплины*.

Рис. П3.1. Логическая схема базы данных:
1 : M — связь типа один ко многим

11. Создать форму *Оценки*.

12. Заполнить данными таблицу ОЦЕНКИ, используя форму *Оценки*.

Контрольные вопросы

1. Что такое реляционная БД?
2. Пояснить, чему соответствуют в таблице строки, столбцы?
3. Что такое ключ? Какими бывают ключи?
4. Охарактеризовать существующие типы связей между таблицами.
5. Пояснить, что такое целостность данных.
6. Пояснить построение информационно-логической модели БД на примере БД Деканат.

ЛАБОРАТОРНАЯ РАБОТА № 4

Использование языка VBA при работе с основными объектами базы данных

Цель работы

Ознакомление с возможностями языка Visual Basic for Applications (VBA) при создании и работе с базами данных, а также с основными объектами базы данных — таблицами, полями, формами.

VBA является полнофункциональным объектно-ориентированным языком программирования, позволяющим создавать приложения пользователя в среде СУБД Access. VBA — это общее средство программирования для всего семейства Microsoft Office, включая Word, Excel, Access, Outlook, Project.

VBA целесообразно использовать для создания нестандартных процедур обработки событий и функций, выполняющих сложные вычисления, которые невозможно записать в виде выражений.

Основой программ на VBA являются процедуры, состоящие из инструкций, выполняющих необходимые операции и вычисления. Процедуры хранятся в модулях, из которых они запрашиваются на выполнение. Собственно модуль не исполняется, а служит для объединения процедур по их функциональному назначению или привязке к форме или отчету.

Различают процедуры-подпрограммы и процедуры-функции.

Процедуры-подпрограммы (Sub) не возвращают значения вызывающей процедуре.

Процедура-подпрограмма может выполнять любые действия, например корректировать данные базы, выполнять вычисления в полях, открывать формы, печатать отчеты.

Формат процедуры-подпрограммы имеет следующий вид:

```
[Private!Public] [Static] Sub <Имя процедуры>[([список аргументов>)]
    [<описание переменных>]
    [<инструкции>]
    [Exit Sub]
    [<инструкции>]
End Sub
```

Здесь Sub, End — отмечают соответственно начало и конец тела процедуры; Public — указывает, что процедура Sub является общей, т.е. доступной для всех других процедур во всех модулях; Private — указывает, что процедура Sub доступна только для процедур того модуля, в котором она описана; Static — указывает, что значения локальных переменных процедуры сохраняются между вызовами этой процедуры; Exit Sub — приводит к немедленному завершению процедуры Sub.

Процедура-функция (Function) возвращает значение, которое присваивается ее имени внутри процедуры.

Формат процедуры-функции имеет следующий вид:

```
[Private!Public] [Static] Function <Имя процедуры>[(<список аргументов>)]
```

```
    [<описание переменных>]
    [<инструкции>]
    [<имя процедуры>=<выражение>]
    [Exit Function]
    [<инструкции>]
    [<имя процедуры>=<выражение>]
End Function
```

В теле процедуры-функции (в отличие от процедуры-подпрограммы) присутствует инструкция присваивания имени процедуры значения, вычисляемого выражением. Эта инструкция позволяет возвратить значение из процедуры-функции в место ее вызова.

При описании переменных обычно используется инструкция присвоения Dim, которая присваивает выражение переменной или константе. Инструкции присвоения всегда включают в себя знак равенства (=). Для присвоения значения переменной, описанной как объект, применяется инструкция Set.

Задание на лабораторную работу

1. Написать процедуру VBA, создающую новую базу данных.
2. Написать процедуру, создающую в текущей базе данных таблицу СТУДЕНТЫ с полями *Номер студента*, *ФИО*, *Предмет1*, *Предмет2*, *Предмет3*, *Предмет4*, *Средний балл*.
3. Внести в созданную таблицу пять записей во все поля (кроме поля *Средний балл*).
4. Создать процедуру, подсчитывающую средний балл всех студентов и заносящую рассчитанные значения в поле *Средний балл*.
5. Создать форму, отображающую данные таблицы СТУДЕНТЫ и содержащую кнопку, запускающую процедуру расчета среднего балла.

Технология выполнения задания

1. Открыть новую базу данных и вкладку *Модули*, написать процедуру, создающую новую базу данных в соответствий с приведенным примером:

```
'Создание новой базы данных
Sub CreateDatabaseX()
    'Описание переменных
     Dim myWs As Workspace
     Dim myDb As Database
    'Определяем стандартный объект Workspace (рабо-
    'чее пространство)
     Set myWs = DBEngine.Workspaces(0)
      'Создаем новую базу данных
```

```
        'с указанным используемым порядком символов
        'dbLangGeneral
        Set myDb = myWs.CreateDatabase("C:\NewDB.mdb",
        dbLangGeneral)
        myDb.Close
    End Sub
```

В результате выполнения процедуры на диске С должна появиться новая база данных с названием NewDB.mdb.

2. Написать процедуру, создающую в текущей базе данных таблицу СТУДЕНТЫ с полями *Номер студента*, *ФИО*, *Предмет1*, *Предмет2*, *Предмет3*, *Предмет4*, *Средний балл* в соответствии с приведенным примером:

```
    'Создание новой таблицы СТУДЕНТЫ в текущей базе данных
    Sub CreateTableDefX()
    'Определяем переменные
    Dim myDb As Database
    Dim myTab As TableDef
    Dim myF As Field

    Set myDb = CurrentDb()

    'Создаем новый объект TableDef – таблицу СТУДЕНТЫ
    Set myTab = myDb.CreateTableDef("Студенты")

    'Создаем новый объект Field – текстовое поле Номер
    'студента и добавляем его к семейству полей объекта
    'таблицы СТУДЕНТЫ
    Set myF = myTab.CreateField("Номер студента",
dbInteger)
    myTab.Fields.Append myF
    'Создаем новый объект Field – текстовое поле ФИО
    'и добавляем его к семейству полей объекта таблицы
    'СТУДЕНТЫ
    Set myF = myTab.CreateField(ФИО, dbText)
    myTab.Fields.Append myF
    'Создаем новый объект Field – поле Предмет1 и
    'добавляем его к семейству полей объекта таблицы
    'СТУДЕНТЫ
    Set myF = myTab.CreateField("Предмет1", dbInteger)
    myTab.Fields.Append myF

    'Аналогично поступаем с другими полями таблицы
    Set myF = myTab.CreateField("Предмет2", dbInteger)
    myTab.Fields.Append myF

    Set myF = myTab.CreateField("Предмет3", dbInteger)
    myTab.Fields.Append myF
```

```
  Set myF = myTab.CreateField("Предмет4", dbInteger)
  myTab.Fields.Append myF

  Set myF = myTab.CreateField("Средний балл", dbDouble)
  myTab.Fields.Append myF

  'Добавляем объект таблицу СТУДЕНТЫ к семейству таб-
'лиц базы данных
  myDb.TableDefs.Append myTab

  End Sub
```

3. Открыть созданную таблицу и внести пять записей во все поля (кроме поля *Средний балл*).

4. Создать процедуру, подсчитывающую средний балл всех студентов и заносящую рассчитанные значения в поле *Средний балл* в соответствии с приведенным примером:

```
Private Sub SB()
Dim myDb As Database        'объектная переменная типа
                            'базы данных
Dim myRec As Recordset      'объектная переменная типа
                            'набора записей
Dim sb As Double            'переменная для вычисления
                            'среднего балла
Dim i As Integer            'переменная цикла
Dim max As Integer          'переменная для хранения чис-
                            'ла записей в таблице

Set myDb = CurrentDb()      'Работаем с текущей базой
                            'данных
'Открываем набор записей таблицы СТУДЕНТЫ и присваива-
'ем ссылку на него объектной переменной myRec
Set myRec = myDb.OpenRecordset("Студенты")
i = 0
myRec.MoveLast              'Идем к последней записи таб-
                            'лицы
max = myRec.RecordCount     'При этом RecordCount содер-
                            'жит число записей в
                            'таблице, которое нужно нам
                            'для вычисления
                            'среднего бала в каждой
                            'строке таблицы
myRec.MoveFirst             'Переходим к первой записи
                            'таблицы и вычисляем средний
                            'балл
Do While i < max
```

```
sb = (myRec!Предмет1 + myRec!Предмет2 + myRec!Предмет3 +
+ myRec!Предмет4)/4
  myRec.Edit                    'Заносим значение среднего
                                'балла в одноименное поле
  myRec![Средний балл] = sb
  myRec.Update                  'Для внесения данных в поля
                                'таблицы обязательно исполь-
                                'зуются команды Edit и
                                'Update.
  myRec.MoveNext                'Переходим к следующей запи-
                                'си таблицы и повторяем все
  i = i + 1                     'пока не достигнем послед-
                                'ней записи.
Loop

  'Закрываем набор записей.
  myRec.Close
End Sub
```

5. Создать форму, отображающую данные таблицы СТУДЕНТЫ и содержащую кнопку, запускающую процедуру расчета среднего балла.

Контрольные вопросы

1. Дать определение макроса. Какими возможностями обладают макросы?
2. Дать определение модуля. Какими возможностями обладают модули?
3. Описать технологию создания процедур на VBA.
4. Описать технологию создания баз данных на VBA.
5. Описать технологию создания таблиц на VBA.
6. Описать технологию создания процедуры обработки событий на VBA.

ЛАБОРАТОРНАЯ РАБОТА № 5

Использование языка VBA для фильтрации данных в базе

Цель работы

Ознакомление с возможностями фильтрации данных в базе с помощью языка Visual Basic for Applications (VBA).

Задание на лабораторную работу

1. Создать таблицы базы данных и установить связи между ними.
2. Создать простой запрос к таблицам базы данных.
3. Создать формы, образующие сложную форму с подчиненной.

4. Разместить в форме элемент управления *Набор вкладок* и написать процедуру обработки события, осуществляющую отбор (фильтрацию) данных в форме в соответствии с выбором вкладки.

Технология выполнения задания

1. Открыть новую базу данных (можно использовать уже имеющуюся), для чего выполнить следующее:

• создать таблицу КАТЕГОРИЯ ИСТОЧНИКА с полями *Код категории* (ключевое поле) и *Наименование категории* (книги, журналы, статьи, справочники и т.д.);

Создать таблицу КАТАЛОГ ИСТОЧНИКОВ с полями *Код источника* (ключевое поле), *Код категории*, *Автор*, *Наименование*, *Год издания*, *Число страниц* и т.д.;

• установить связь между таблицами типа один ко многим от таблицы КАТЕГОРИЯ ИСТОЧНИКА к таблице КАТАЛОГ ИСТОЧНИКОВ;

• занести в таблицы непротиворечивые данные.

2. Создать простой запрос к таблицам КАТАЛОГ ИСТОЧНИКОВ и КАТЕГОРИЯ ИСТОЧНИКА, отбирающий все поля и все записи.

3. Создать форму на основе запроса, созданного в п. 2.

4. Создать пустую форму и разместить в ней подчиненную форму, созданную в п. 3. В этой же форме разместить элемент управления *Набор вкладок* с наименованием *Категория* и вкладками, имя каждой из которых соответствует используемым категориям (книги, журналы и т.д.). Нажатие на соответствующую вкладку набора должно вызывать отбор (фильтрацию) данных в подчиненной форме.

5. В окне свойств элемента управления *Набор вкладок* в графе *Изменение* выбрать опцию *Процедура обработки события*, которая будет реагировать на выбор соответствующей вкладки и выводить в подчиненной форме только записи соответствующей категории.

В теле процедуры должен содержаться набор записей следующего типа:

```
If Me![Категория]=1 Then
   Me![Каталог источников].Form.RecordSource =
   "SELECT * FROM [Каталог источников Запрос] WHERE
   [Каталог источников Запрос].[Наименование кате-
   гории] Like 'Книги';"
End If
    If Me![Категория]=2 Then
    . . . . . .
End If
. . . . . .
```

Здесь выражение Me![Категория]=1 указывает, что если в текущей форме в элементе управления *Набор вкладок* с именем *Категория* выбрана вкладка с индексом 1 (в данном случае вкладка *Книги* имеет индекс 1, вкладка *Журналы* — 2 и т.д.), то источником строк (RecordSource) для подчиненной формы с именем *Каталог источников* является запрос с именем *Каталог источников Запрос* с условием отбора записей по наименованию категории, соответствующему категории *Книги*.

6. Проверить правильность работы формы с отбором записей по категории.

Контрольные вопросы

1. Описать технологию создания процедуры обработки событий на VBA.
2. Написать процедуру создания формы на VBA.
3. Написать процедуру создания отчета на VBA.
4. Описать основные свойства форм, доступные при программировании на VBA.

СПИСОК ЛИТЕРАТУРЫ

1. *Бекаревич Ю. Б., Пушкина Н. В.* Microsoft Access 2000. — СПб.: БХВ-Санкт-Петербург, 1999.

2. *Васильев А., Андреев А.* VBA в Office 2000: учебный курс. — СПб.: Питер, 2001.

3. *Вербовецкий А. А.* Основы проектирования баз данных. — М.: Радио и связь, 2000.

4. *Дейт К. Дж.* Введение в системы баз данных. — М.: Вильямс, 1999.

5. Информатика: Учебник / Под ред. Н. В. Макаровой. — М.: Финансы и статистика, 1998.

ОГЛАВЛЕНИЕ