MICROSOFT ACCESS

ОГЛАВЛЕНИЕ

1.1. Архитектура MICROSOFT ACCESS

Access представляет собой простую, но достаточно мощную настольную реляционную СУБД и преимущественно предназначен для создания некоммерческих приложений или приложений средней мощности, не связанных с интенсивной обработкой данных. Access входит в состав MS Office и достаточно тесно интегрирован с остальными ее компонентами.

Среди достоинств Access можно выделить также следующие:

1. Простота освоения. Возможность использования непрофессионалом;
2. Визуальное программирование. Простую БД можно создать только с помощью мыши, даже не прибегая к программированию;
3. Наличие мастеров, которые помогают решать такие сложные задачи, как анализ данных в таблицах на избыточность, создание связанных форм, отчетов и т.д.;
4. Возможность динамической проверки результатов без всякой компиляции. Конструируя таблицу, можно легко переходить в режим таблицы и при обнаружении недостатков возвращаться назад для корректировки. При этом модернизацию можно провести немедленно или отложить на длительный срок;
5. Тесная интеграция с остальными компонентами MS Office.

Так как Access является интерпретатором (отчасти в силу того, что он основан на языке Visual Basic), появляется необходимость в совмещении в одном приложении функций разработки приложения и собственно функций готового приложения. Для доступа к объектам базы в режиме разработки служит специальное окно БД,

содержащее список всех объектов БД. С каждым объектом БД можно работать в двух режимах. Первый режим можно условно назвать режимом выполнения или рабочего состояния. Второй режим – это режим конструктора, в котором производится создание и модификация объекта. Наличие окна БД определяет открытую БД. Закрытие окна приводит к закрытию БД. После полной разработки БД появление данного окна можно отключить. Исключив также все команды разработчика из меню и панелей инструментов, можно создать распространяемую версию БД. Такая база данных Access будет функционировать при условии наличия на компьютере СУБД Access. При необходимости можно воспользоваться пакетом Access Developer Toolkit для создания инсталляционной версии БД, где СУБД Access будет представлена несколькими файлами DLL. Для работы с распространяемой версией БД пользователям предоставляется интерфейс, обычно содержащий главную переключательную (по числу решаемых задач) и несколько обычных форм.

БД, написанная на Access, состоит из объектов. Microsoft Access называет объектами все, что может иметь некоторое имя. В базе данных Access основными объектами являются таблицы, запросы, формы, отчеты, макросы и модули. В новых версиях Access имеются также страницы доступа к данным, позволяющие создавать Web интерфейс БД, и проекты, служащие для создания клиентских приложений для работы с MS SQL Server. Ниже приведены основные характеристики основных объектов базы данных.

Таблица

Объект, который определяется и используется для хранения и визуализации данных.

Каждая таблица обычно содержит информацию об объекте определенного типа. Таблицы связаны между собой, повторяя реальные связи, существующие между объектами. Дополнительно к таблице можно определить несколько индексов для ускорения доступа к данным. В Access объект таблица используется не только для хранения данных, но и для их визуализации, поскольку таблица сама по себе является прекрасным способом отображения реляционных данных.

Запрос

Объект, который позволяет пользователю получить нужные данные из одной или нескольких таблиц. С помощью режима таблицы имеется возможность просмотреть все данные. Но что делать, если записей в таблице слишком много или требуемые данные находятся в нескольких таблицах. Тогда для отбора необходимых данных и применяются запросы. Они позволяют выбрать только нужные данные, соответствующие определенному критерию. Для создания запроса предоставляется два вида языков: QBE (Query By Example - запрос по образцу) и SQL. Можно создавать запросы на выбор, обновление, удаление или на добавление данных. С помощью запросов существует возможность создавать новые таблицы, используя данные одной или нескольких существующих таблиц и т.д. В Access не делается различия между запросами и таблицами. Соответственно запрос можно основывать и на других запросах, что очень удобно.

Форма

Объект, предназначенный для ввода и отображения данных на экране или управления работой приложения. Таблицы – полезное средство для просмотра и изменения данных, но работать с ними бывает не всегда удобно, а представление данных в них не достаточно наглядно. Для редактирования данных и вывода их на экран в удобном виде предназначены формы. Форма представляет собой бланк, подлежащий заполнению, или маску, накладываемую на набор данных. Формы можно использовать для того, чтобы реализовать требования пользователя к представлению данных из запросов или таблиц. С помощью формы можно красочно оформить данные, представить их в цвете, можно добавить такие элементы, как поля со списком, комбинированные списки, флажки, кнопки и многое другое. Можно также добавить рисунки и диаграммы и производить вычисления над данными таблиц и запросов.

Формы также предназначены для автоматизации приложения, так как являются средством организации интерфейса в Access. С помощью формы вы можете в ответ на некоторые события запустить макрос или процедуру VBA. С помощью макросов или процедур VBA можно связать несколько форм или отчетов между собой. Например, находясь в одной форме можно открыть другую форму и связать выводимые в них данные таким образом, чтобы при переходе от записи к записи в этой форме в другой форме выводились соответствующие данные.

Отчет

Объект, предназначенный для создания печатного документа. В Access допускается экспорт отчета в документ другого приложения, например в Word или Excel. Мы уже можем распечатать информацию в виде таблицы или формы, но создать полноценный выходной документ с использованием всех средств форматирования текста и дополнительной обработки данных (например, вычисления промежуточных и окончательных итогов) можно только с помощью отчетов. В таблице можно изменять размер шрифта и ширину столбца. В формах есть возможность производить разнообразное форматирование текста, но они плохо приспособлены для проведения сложных вычислений, группировки данных, расчета промежуточных и общих итогов, и на экране показывают чаще всего только одну запись. Отчеты же совмещают достоинства таблицы и формы для печатного предоставления данных. Прежде чем выводить отчет на печать, его можно предварительно просмотреть на экране.

Макрос

Объект, представляющий собой структурированное описание одного или нескольких действий, которые должен выполнить Access в ответ на определенное событие.

Событие – это распознаваемое изменение состояния любого объекта MS Access или операционной системы. Может представлять любое действие, выполняемое вами или компьютером. Например, перемещение мыши, ввод символа, нажатие на клавишу клавиатуры или мыши, потеря или получение фокуса некоторого элемента окна и т.д. Такие действия, как открытие окна, состоят из нескольких последовательно происходящих событий. Управление работы приложения в Access достигается за счет обработки событий, которые возникают при работе. Такое управление работы приложения очень удобно, т.к. появляется возможность оторваться от предопределенной линии поведения приложения и управлять им динамически.

Модуль

Объект, содержащий программы на Visual Basic for Applications (VBA), сохраненные под общим именем. Хотя в Access можно без труда создать приложение, состоящее только из таблиц, форм, отчетов и макросов, все же возникает потребность выполнения таких действий, которые нельзя запрограммировать с помощью макросов. Необходимость в модулях появляется также при необходимости обработки ошибок и оформления в виде некоторой процедуры или функции одинаковых вычислений или действий. Модули могут быть независимыми объектами, содержащими функции, которые можно вызывать из любого места приложения, но они могут быть непосредственно «привязаны» к отдельным формам или отчетам для реакции на те или иные происходящие в них события.

Концептуальные взаимосвязи объектов Access показаны на рисунке Рис. 1. В таблицах хранятся данные, которые можно просматривать и редактировать напрямую или с помощью запросов. Используя формы, можно выводить данные на экран или изменять их. Необходимо отметить, что формы и отчеты могут использовать данные непосредственно из таблиц или через запросы. Для выполнения нужных вычислений и преобразования данных запросы могут использовать встроенные функции Access, или функции, написанные в VBA.

События, связанные с формами или отчетами, могут запускать макросы или процедуры VBA. Из макросов и модулей можно существует возможность изменять внешний вид объектов и ход выполнения

приложения; открывать, фильтровать и изменять данные в формах и отчетах; выполнять запросы, создавать новые таблицы. В VBA можно создать, модифицировать и удалить любой объект Access, извлекать и обрабатывать данные из других БД и т.д.

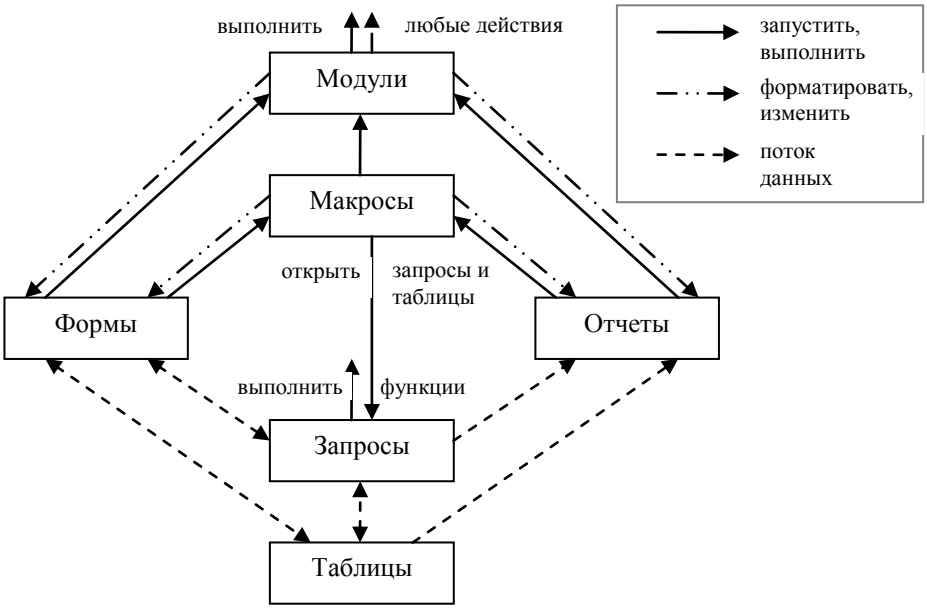


Рис. 1. Архитектура Microsoft Access

1.2. Разработка таблиц

В Access разработка таблиц производится с помощью конструктора (смотри Рис. 2), а также с помощью Мастера или путем прямого ввода данных.

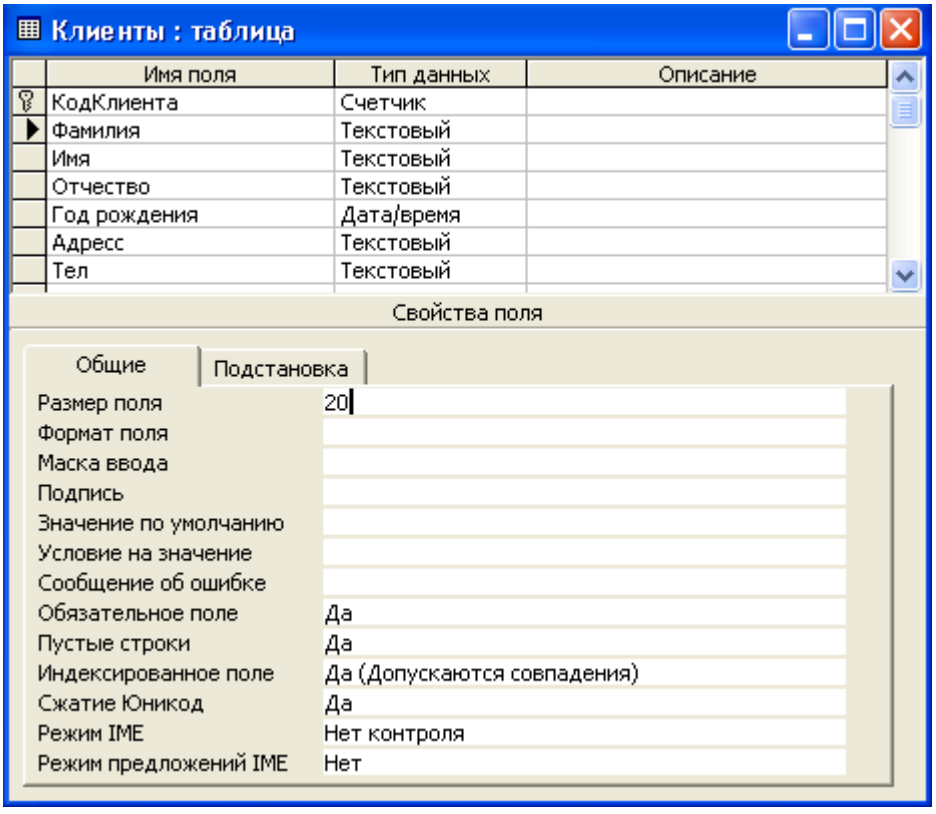


Рис. 2. Окно конструктора таблиц

Для создания таблицы необходимо определить имена полей, их типы данных и свойства. При желании можно воспользоваться готовыми структурами таблиц из встроенных примеров или воспользоваться построителем  При этом появляется окно, позволяющее выбирать готовые описания полей. Имена полей должны быть уникальными в пределах таблицы, иметь не более 64 символов (включая пробелы), не должны начинаться с пробела или управляющего символа и не содержать точки, восклицательного знака, апострофа и квадратных скобок.

Access поддерживает 9 типов данных:

1	Текстовый	Text	Текстовые значения до 255 символов (255 байт)
2	Поле Мемо	Memo	Текст или битовый массив до 64 кбайт (65535 символов)
3	Числовой	Number	Целые и вещественные числа (1, 2, 4, 8 или 12 байт)
4	Дата/время	Date/Time	Значения даты/время (8 байт)
5	Денежный	Currency	Денежные значения и числовые данные (от одного до четырех знаков в дробной части), используемые в математических расчетах, проводящихся с точностью до 15 знаков в целой и до 4 знаков в дробной части (8 байт)
6	Счетчик	AutoNumber	Уникальные последовательно возрастающие или случайные числа, автоматически вводящиеся при добавлении каждой новой записи в таблицу. Таблица должна иметь не более одного поля Счетчик. (4 байта)
7	Логический	Yes/No	Логические значения (1 бит)
8	Поле объекта OLE	OLE Object	Объекты OLE. До 1 Гбайт (ограничивается объемом диска)
9	Гиперссылка	Hyperlink	Текст или комбинация текста и чисел, хранимые как текст и используемые в качестве адреса гиперссылки. Адрес гиперссылки может состоять максимум из трех частей: текст – текст, выводимый в поле или в элементе управления; адрес – путь к файлу (в формате UNC или URL) и дополнительный адрес – расположение внутри файла или страницы. Длина каждой из трех частей гиперссылки не более 2048 знаков.

Тип *Мемо* используется для хранения текстовых массивов данных, превышающих 255 символов. Тип *Дата/Время* применяется для хранения и выполнения вычислений календарных дат или значений времени. Храниться в виде вещественного числа, целая часть которого определяет дату, дробная – время. Тип *Счетчик* является разновидностью числового типа (длинное целое) и создан для автоматической генерации значений первичного ключа (таблица не может иметь более одного поля типа *Счетчик*). *Денежный* тип данных удобен в денежных расчетах, так как предотвращает округления в процессе вычислений. Логический тип данных используется для хранения значений *Истина* и *Ложь* и особенно полезен для отметки выполнения некоторых действий или наличия чего-либо. Тип *Поле объекта OLE* позволяет хранить картинки, диаграммы, звуковые фрагменты и прочие объекты, созданные в приложениях, поддерживающих интерфейс OLE. Например, Access позволяет хранить и редактировать документы Microsoft Word, электронные таблицы Microsoft Excel, картинки Microsoft PowerPoint и Paint, звуковые файлы WAV и т.д. Данные типа *Мемо* и *Поле объекта OLE* хранятся в отдельной присоединенной таблице, разгружая основную таблицу от хранения больших объемов данных и таким образом ускоряя операции загрузки, поиска, сортировки и фильтрации. И, наконец, в поле типа можно выбрать значение *Мастер подстановок*, позволяющий определить подстановку значений поля из фиксированного списка значений или из другой таблицы.

Нужно быть особо внимательным при задании типа поля, поскольку он определяет диапазон принимаемых значений и размер хранимых данных. Например, задав номер телефона числовым, мы рискуем потерять первые числа номера, если они равны 0 (а эта ситуация часто встречается в международных кодах и

номерах первой необходимости). При выборе числового типа следует дополнительно обдумать вопрос о значении свойства *Размер поля*, поскольку ваш выбор будет определять точность значений данных и объем памяти, необходимый для их хранения.

1.2.1. Свойства полей

После задания типа для каждого из полей необходимо определить их свойства. Задание отдельных свойств наряду с типом поля определяет целостность хранящихся данных. Детальная проработка типов и свойств полей фактически определяет саму БД, поскольку БД в Access становиться практически полнофункциональной сразу после создания таблиц. Список свойств зависит от выбранного типа данных. Ниже перечислены основные свойства полей таблицы:

Размер поля (Field Size)

Это свойство задает максимальный размер данных, для хранения которых предназначено данное поле. Поле с текстовым типом данных может иметь размер от 1 до 255 (символов); по умолчанию устанавливается 50 символов. Для числового типа данных размер поля может быть следующим:

Байт	Byte	Целые числа от 0 до 255. Занимает при хранении 1 байт.
Целое	Integer	Целые числа от -32768 до +32767. Занимает 2 байта.
Длинное целое	Long Integer	Целые числа от -2147483648 до +2147483647. Занимает 4 байта.
Одинарное с плавающей точкой	Single	Числа с точностью до 7 знаков. От -3,402823E38 до –1,401298E–45 для отрицательных значений, и от 1,401298E–45 до 3,402823E38 для положительных. Занимает 4 байта.
Двойное с плавающей точкой	Double	Числа с точностью до 15 знаков. Числа от -1,79769313486231E308 до -4,94065645841247E–324 для отрицательных значений и от 4,94065645841247E–324 до 1,79769313486231E308 для положительных. Занимает 8 байт.
Действительное	Real	Числа с точностью до 28 знаков. Числа от –10^28–1 до 10^28–1. Занимает 12 байт.

Формат поля (Format)

С помощью этого свойства можно задать формат представления данных при выводе на экран или печать. Свойство *Формат поля* определяет только способ отображения данных и не влияет на способ их хранения. Для типов данных *Числовой, Денежный, Счетчик* существует стандартный набор форматов поля. Некоторые форматы приведены ниже:

Основной	General	(Значение по умолчанию.) Числа отображаются так, как они были введены.
Денежный	Currency	Символы валют и два знака после десятичного разделителя.
Фиксированный	Fixed	По крайней мере один знак до и два знака после десятичного разделителя. Число десятичных знаков определяется в одноименном свойстве.
С разделителями разрядов	Standard	Используются общие настройки для отрицательных значений, символов десятичного разделителя и десятичных разрядов.
Процентный	Percent	Значение умножается на 100 и к нему добавляется знак процента %.
Экспоненциальный	Exponential	Числа выводятся в экспоненциальной нотации.

Для типа *Дата/Время* также существует набор предопределенных форматов. К ним относятся *Полный, Длинный, Средний и Короткий* форматы даты и *Длинный, Средний и Короткий* форматы времени. Эти форматы определены в региональных установках панели управления каждого компьютера. Соответственно их можно изменить на другие по собственному желанию. Специальные форматы даты и времени создаются с помощью следующих символов:

Символ	Описание
: (двоеточие)	Разделитель компонентов времени. Знак разделителя задается в диалоговом окне *Язык и стандарты* (панель управления Microsoft Windows).
/	Разделитель компонентов даты. Знак разделителя задается в диалоговом окне *Язык и стандарты* (панель управления Microsoft Windows).
c	Задает встроенный «Полный формат даты».
d	Номер дня месяца, состоящий из одной или двух цифр (1-31).
dd	Номер дня месяца, состоящий из двух цифр (01-31).
ddd	Сокращенное название дня недели (Пн-Вс).
dddd	Полное название дня недели (понедельник-воскресенье).
ddddd	Задает встроенный «Краткий формат даты».
dddddd	Задает встроенный «Длинный формат даты».
w	Номер дня недели (1-7).
ww	Номер недели в году (1-53).
m	Номер месяца, состоящий из одной или двух цифр (1-12).
mm	Номер месяца, состоящий из двух цифр (01-12).
mmm	Первые три буквы названия месяца (янв-дек).
mmmm	Полное название месяца (Январь-Декабрь).
q	Номер квартала в году (1-4).
y	Номер дня в году (1-366).
yy	Последние две цифры номера года (01-99).
yyyy	Полный номер года (0100-9999).
h	Число часов, состоящее из одной или двух цифр (0-23).
hh	Число часов, состоящее из двух цифр (00-23).
n	Число минут, состоящее из одной или двух цифр (0-59).
nn	Число минут, состоящее из двух цифр (00-59).
s	Число секунд, состоящее из одной или двух цифр (0-59).

Для логического типа данных используется следующий набор форматов:

Да/Нет	Yes/No
Истина/*Ложь*	True/False
Вкл/Выкл	On/Off

По умолчанию Access выводит числа в стандартном формате, а денежные значения – в денежном. Можно также создать пользовательский формат, который будет использовать Access, (в зависимости от того, будет ли число положительным, отрицательным, равным нулю или равным *Null*), определив до четырех соответствующих спецификаций формата, разделенных точкой с запятой. Рассмотрим некоторые полезные спецификации формата:

Символ	Описание
0	используется, чтобы показать, что в этой позиции будет находиться цифра. Если в этой позиции в числе нет цифры, то Access выводит 0.
#	используется, чтобы показать, что в этой позиции будет находиться цифра. Если в этой позиции в числе нет цифры, то Access выводит пробел.
- + $ ( ) пробел	эти символы можно использовать в любом месте строки спецификации.
“текст”	выводит текст, заключенный в кавычки.
\	используется для вывода любого символа, следующего за \ (то же, что и заключение в одиночные кавычки).
!	выравнивание по левому краю. По умолчанию числовые данные выравниваются по правому краю, текстовые – по левому.
*	используется, чтобы в качестве заполнителя неиспользованных позиций употреблялся следующий за * символ.
%	используется для умножения числа на 100 и дополнения справа символом процента.
E- или e	выводит числа в экспоненциальном формате.

E+ или e+	дополнительно всегда показывает знак чисел: + для положительных и – для отрицательных.
[Цвет]	отображает отформатированные данные заданным цветом, название которого указано в скобках. Допустимые имена цветов: Черный (Black), Синий (Blue), Зеленый (Green), Бирюзовый (Cyan), Красный (Red), Лиловый (Magenta), Желтый (Yellow), Белый (White).

Например, если вы хотите вывести число с двумя десятичными знаками после десятичной точки, запятой в качестве разделителя тысяч, если число положительное, и заключенным в круглые скобки и красным цветом символов, если число отрицательное, “Ноль”, если равно 0, и “Не введено” для пустого значения *Null*:

#, ##0.00; (#, ##0.00) [Красный]; “Ноль”; “Не введено”

Для текстовых данных также можно задать пользовательский формат, который может состоять из одной, двух либо трех частей. Если включена вторая спецификация, то Access будет использовать ее для вывода значения *Null*; если включена третья часть, то Access использует вторую для вывода пустой строки, а третью для *Null*. Для текстовых значений определены следующие спецификации формата:

Символ	**Описание**
@	используется для вывода любого имеющегося в этой позиции символа или пробела.
&	используется для вывода любого имеющегося в этой позиции символа. Если в этой позиции нет символа, то ничего не выводит.
<, >	преобразование к нижнему или верхнему регистру.
пробел, “текст”,\, !, *, [цвет]	аналогично приведенным для числовых значений.

Например, формат вида

>; “Безразлично”; “Не определено”

позволяет вывести в верхнем регистре значение поля и «озвучить» визуально не определяемую разницу между пустой строкой и пустым значением, а

** @@@@@@@@

записывает сумму чека прописью и заполняет свободное пространство звездочками, например: “Ten Dollars and*********50 Cents”.

Формат вывода логических данных также состоит из трех частей, хотя первая из них не используется. Т.е. первая часть спецификации формата оставляется пустой, далее следует формат для значения *Истина*, далее для значения *Ложь*. Например:

;“Счет выписан” [Красный]; “Счет не выписан” [Синий].

Число десятичных знаков (Decimal Places)

Для числового и денежного типов данных можно задать число знаков, выводимых после десятичной точки. По умолчанию устанавливается значение *Авто,* при котором для *Денежного, Фиксированного, С разделителями тысяч* и *Процентного* форматов поля выводятся два десятичных знака после запятой, а для *Стандартного* формата число выводимых знаков определяется точностью числовых значений. Можно задать фиксированное число десятичных знаков от 0 до 15. Игнорирование задания формата для вещественных чисел приводит к серьезным ошибкам. Так как числа выравниваются по правому краю, то при их отображении в окошке ввода/вывода небольшого размера может поместиться не все число, а только его дробная часть. Значимая целая часть может быть вообще проигнорирована.

Маска ввода (Input Mask)

Для текстового, числового и денежного типов данных, а также для типа *Дата/Время* можно задать маску ввода, которая будет контролировать ввод данных с клавиатуры с помощью некоторого шаблона. Маску можно

использовать для выполнения таких простых действий, как преобразование всех введенных символов к верхнему регистру или более сложных, например, добавление скобок и разделителей к номеру телефона. Можно также включать постоянные строки символов, которые затем будут отображаться на экране и по вашему желанию включаться в таблицу наравне с вводимыми. Например, можно установить, чтобы Access выводил разделители _/_/_ для поля типа *Дата*, или задать маску ввода паспорта в виде Серия __ № _____.

Маска ввода состоит из трех частей, разделенных символом точкой с запятой. Первая часть содержит собственно маску ввода, состоящую из символов, указанных в таблице, необязательная вторая часть указывает, требуется ли сохранять постоянные символы маски в этом поле. Если постоянные символы маски должны быть включены, то 0, если надо сохранять только введенные пользователем символы, то 1. Необязательный третий компонент задает символ-указатель. По умолчанию используется ‘_’. Чтобы отобразить пустую строку, можно ввести пробел, заключенный в кавычки (" "). Маска ввода задается с помощью следующих символов:

Символ	**Описание**
0	Цифра (0-9, обязательный знак; знаки (+) и (-) не разрешены).
9	Цифра или пробел (необязательный знак; знаки (+) и (-) не разрешены).
#	Цифра или пробел (необязательный знак; незаполненные позиции выводятся как пробелы в режиме редактирования, но удаляются при сохранении данных; знаки (+) и (-) не разрешены).
L	Буква (от А до Я, обязательный знак).
?	Буква (от А до Я, необязательный знак).
A	Буква или цифра (обязательный знак).
a	Буква или цифра (необязательный знак).
&	Любой знак или пробел (обязательный знак).
C	Любой знак или пробел (необязательный знак).
. , : ; - /	Десятичный разделитель, разделители групп разрядов, времени или даты. (Используемые знаки разделителей определяются настройками, выбранными на панели управления Microsoft Windows в диалоговом окне Язык и стандарты.)
<	Преобразует все знаки к нижнему регистру.
>	Преобразует все знаки к верхнему регистру.
!	Указывает заполнение маски ввода справа налево, а не слева направо. Заполнение маски знаками всегда происходит слева направо. Восклицательный знак в маске ввода можно помещать в любую позицию.
\	Указывает, что следующий знак будет отображаться как текстовая константа (например, \A отображается как «A»).

Например, маска ввода номера телефона – “8- (”####”) “##0\-####,

маска ввода паспорта “Серия ”LL” №0000000.

При вводе данных в поле, для которого определена маска ввода, всегда используется режим замены. При удалении знака путем нажатия клавиши BACKSPACE знак заменяется на пробел. При копировании или перемещении содержимого поля, для которого определена маска ввода, в буфер обмена текстовые константы маски копируются вне зависимости от режима их сохранения. Маска ввода используется только при вводе знаков с клавиатуры и игнорируется при всех остальных операциях (при импорте данных, при выполнении запроса на изменение, а также при вводе знаков с помощью инструкции VBA).

Маска ввода определяет также и формат вывода, но если определена маска ввода и одновременно задан формат поля, то при выводе данных приоритет имеет формат поля. Это означает, что при форматировании данных сохраненная маска ввода игнорируется. Данные в базовой таблице при этом не изменяются, поскольку формат поля определяет только режим отображения данных.

Подпись поля (Caption)

Можно определить более описательное имя поля, которое Access будет выводить в элементах управления форм и в заголовках отчетов. К примеру, если вы определили имя поля без пробелов, то можно использовать свойство *Подпись поля* для того, чтобы задать имя, которое включает в себя пробелы.

Значение по умолчанию (Default Value)

Можно определить значение поля по умолчанию для всех типов полей, кроме *Счетчика, Мемо* и *Объекта OLE*. Например, для числовых полей, участвующих в некоторых вычислениях, желательно установить значение поля по умолчанию в 0 или в 1, в зависимости от того, в какой операции будет участвовать это поле, в сложении или в умножении. В противном случае, если это поле будет пустым, то и весь результат операции будет равен пустому значению.

Условие на значение (Validation Rule)

Можно задать выражение, которое при вводе или редактировании значения этого поля всегда должно быть истинным (при этом название поля опускается). Например, <100 означает, что значение поля должно быть меньше 100. Условие задается логическим выражением, которое в общем случае состоит из операторов сравнения и значений, используемых для сравнения. Его элементами могут быть операции сравнения <, <=, >, >=, =, <>, логические операторы NOT, AND, OR, скобки и функции IN, BETWEEN, LIKE. Строки заключаются в кавычки, а константы типа *Дата/Время* – в символы #.

Функция IN проверяет на равенство любому значению из списка: IN (“Минск”, “Москва”, “Киев”);

Функция BETWEEN задает диапазон значений. Границы диапазона разделяются оператором And: BETWEEN 50 And 100.

Функция LIKE проверяет на соответствие заданному шаблону символов. В качестве символов шаблона используются:

* – любое число произвольных символов;

? – один произвольный символ;

– одна произвольная цифра;

[] – диапазон допустимых символов. К примеру, [А - Я], [3 - 9]. Если же эти символы необходимо исключить, то перед ними ставится !: [!А - Я]. Например, LIKE “######”проверяет почтовый индекс, а LIKE “*@*” – наличие символа @ в электронном адресе.

Можно также задать условия на значения между двумя полями одной и той же таблицы, которые Access будет использовать для проверки каждой записываемой строки. Это условие проверяется только при сохранении записи, так как в таком условии одно поле сравнивается с другим, а пока вы не начнете сохранять запись, Access может не знать, окончательно вы ввели данные или потом захотите их изменить. Следовательно, это условие нельзя записать в свойстве *Условие на значение* отдельного поля. Для задания условия такого вида необходимо выбрать команду *Свойства таблицы* меню *Вид*. Имена полей заключаются в квадратные скобки [ ], а в строке *Сообщение об ошибке* можно задать свое предупреждающее сообщение. Например, имеет смысл проверить, не превышает ли дата увольнения даты приема на работу [Дата увольнения] > [Дата приема].

Сообщение по ошибке (Validation Text)

Можно ввести текст, который Microsoft Access будет выводить на экран, когда вводимое значение не удовлетворяет условию на значение.

Обязательное поле (Required)

Если необходимо, чтобы данное поле всегда содержало некоторое значение (т.е. было отлично от *Null*), то значение этого свойства должно быть *Да*.

Пустые строки (Allow Zero Length)

Для *Текстовых* и *Мемо* полей можно разрешить ввод пустых строк. Реляционные базы данных обеспечивают возможность хранения в полях таблиц специального значения *Null*, называемого пустым значением, для обозначения некоторого неизвестного значения. Пустое значение имеет особые свойства. Пустое значение не может быть равно никакому другому значению, в том числе другому пустому значению. Это означает, что невозможно объединить две таблицы по значению *Null*. Условие «A = B», где A или B содержит значение *Null*, имеет значение *Ложь*. Пустые значения также не учитываются в групповых функциях, таких как Sum (Сумма) или Avg (Среднее значение). Можно лишь проверить наличие значения *Null*, сравнив значения поля со специальной константой NULL или используя встроенную функцию IsNulI. С другой стороны, можно установить значение *Текстового* или *Мемо* поля равным пустой строке (""), и это будет означать, что значение поля известно, но поле пустое. При сравнении пустые строки считаются равными друг другу, следовательно, можно соединять таблицы по пустым строкам и даже подсчитывать их количество. Для того чтобы разрешить пользователю ввод пустых строк в *Текстовые* и *Мемо* поля, следует установить свойства *Пустые строки* в значение *Да*. Если этого не сделать, Access преобразует введенные пользователем пустые строки, а также строки, состоящие только из пробелов, в строки, содержащие значение *Null*.

Почему же так важно различать строки со значением *Null* и пустые строки? Приведем пример: предположим, у вас имеется база данных, в которой хранятся результаты опросов клиентов о предпочитаемых ими характеристиках автомобилей. Если бы в анкете отсутствовал ответ на вопрос о предпочитаемом цвете автомобиля, это соответствовало бы хранению в этом поле значения *Null*. При этом не имеет смысла сравнивать «неизвестное» значение с другими и использовать эту строку при вычислениях итогового и среднего значений. С другой стороны, некоторые клиенты могли бы ответить на вопрос о предпочитаемом цвете: «Безразлично». В этом случае вам известен «неопределенный» ответ на ваш вопрос, и вы ставите ему в соответствие пустую строку. Вы можете подсчитать все «неопределенные» ответы и включить их в расчеты итогового и среднего значений.

Пустое значение выбирается также и в том случае, если вы не знаете, есть у клиента факс или нет, в то время как пустая строка может означать, что определенно нет.

Индексированное поле (Indexed)

Для полей с типом данных *Текстовый, Числовой, Денежный, Дата/Время* и *Счетчик* можно задать построение индекса с целью ускорения доступа к данным, поиска и сортировки. При создании индекса можно также включить проверку на уникальность значений индексируемого поля. С одной стороны, индексы ускоряют поиск и доступ к данным, с другой – замедляют все операции обновления данных, поскольку при этом производится обновление всех индексов, определенных для таблицы. Поэтому индексы рекомендуется создавать лишь для тех полей, по которым наиболее часто выполняется поиск либо сортировка. Поля *Мемо*, *Гиперссылки* и *Объект OLE* не допускают индексирования.

Создать составной индекс (по нескольким полям) можно с помощью специального окна *Индексы*, доступного в режиме конструирования таблицы (смотри Рис. 3). Достаточно ввести имя индекса (*ПолноеИмя* в примере) и в колонке *Имя поля* определить поля, входящие в состав индекса, не повторяя имя самого индекса.

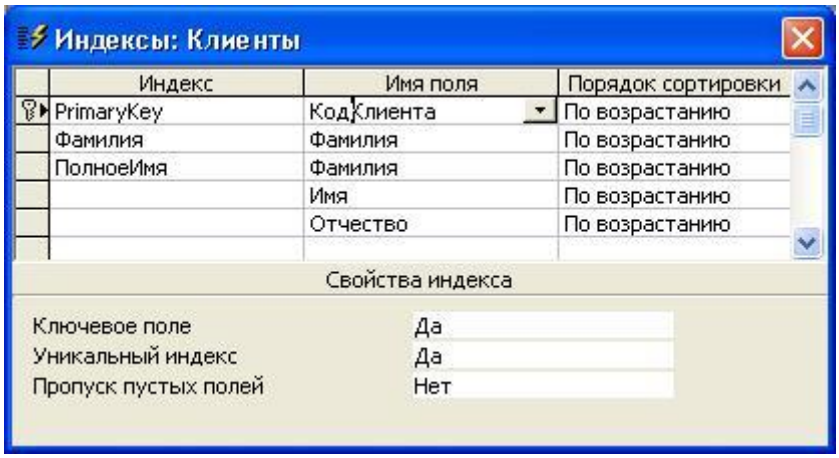


Рис. 3. Окно определения индексов

Сжатие Юникод (Unicode Compression)

В Microsoft Access 2000 и более поздних версиях для хранения информации полей типа *Текстовый*, *Поле Мемо* или *Гиперссылка* используется кодировка Юникод. В Юникод каждый знак представляется 16 битами, поэтому для хранения данных в полях типа *Текстовый*, Поле *Мемо* и *Гиперссылка* требуется больше места, чем в Microsoft Access 97 или более ранних версиях, в которых каждый знак представляется одним байтом. Когда свойство Сжатие Юникод поля имеет значение *Да*, все знаки, первый байт которых равен 0, будут сжиматься при сохранении и восстанавливаться при выборке. Так как первым байтом латинских букв является 0, то кодировка Юникод при включенном сжатии не накладывает дополнительных требований к объему дискового пространства, необходимого для хранения в полях данных, состоящих только из латинских букв.

Задание свойств *Размер поля*, *Маска ввода*, *Условие на значение*, *Значение по умолчанию, Обязательное* и *Индексированное поле* являются очень важными, поскольку определяют целостность данных. Свойства *Размер поля*, *Маска ввода*, *Условие на значение* и *Значение по умолчанию* задают целостность значений; свойство *Обязательное поле* – целостность записи, поскольку гарантирует отсутствие полностью пустых записей; определение индекса с контролем уникальности значений задает целостность ключей, поскольку гарантирует отсутствие повторяющихся значений потенциальных ключей.

1.2.2. Задание ключевых полей и создание связей между таблицами

Для Access обязательным является определение ключевого поля для таблицы. Для его определения достаточно выделить поле и выбрать команду *Ключевое поле* меню *Правка*. Если требуется определить составной ключ, но необходимо выделить требуемые поля при нажатой клавише Ctrl, а затем выбрать команду *Ключевое поле*. При определении ключевого поля автоматически создается уникальный индекс, определяющий физический порядок записей в таблице. Этот индекс является первичным индексом для таблицы и имеет зарезервированное имя *Primary Key*. Для составных ключей существенным может оказаться порядок образующих ключ полей, так как упорядочение записей будет проводиться вначале по первому полю, затем по второму и т.д. Для внешних полей при создании связи также происходит автоматическое создание индекса (в данном случае вторичного).

После определения нескольких таблиц необходимо задать связи между ними. Для этого требуется закрыть все участвующие в связях таблицы, открыть окно *Схема данных* (*Relationships*), добавить в окно необходимые таблицы и задать связь посредством переноса (с помощью мыши) поля из одной таблицы в соответствующее ему поле в другой таблице. Поля таблиц, участвующих в связи, должны иметь одинаковый тип и размер поля. Тип связи определяется автоматически. После создания связи она будет отображается линией между таблицами. Для удаления достаточно ее выделить и нажать клавишу Del.

Для определения ссылочной целостности данных следует установить опцию *Обеспечение целостности данных* (Enforce Referential Integrity). Дополнительно можно установить каскадное удаление и обновление записей. Каскадное обновление позволяет вам при замене значения ключа со стороны 1 поменять его во всех связанных с ним таблицах со стороны ∞. Каскадное удаление позволяет удалить все записи со стороны ∞ при удалении записи со стороны 1.

1.2.3. Работа с таблицей в режиме заполнения

После определения таблиц и установления связей между ними можно приступить к вводу записей в таблицы. При редактировании доступными являются поля только текущей записи, поэтому необходимо всегда помнить о том, что вначале надо перейти к желаемой записи (найти ее), а только потом редактировать. Если необходимо обновить сразу группу записей, то для этого следует воспользоваться запросами действий. В левой части таблицы имеется селекторный столбец, содержащий индикаторы состояния записи. Треугольник означает, что запись является текущей. При редактировании записи треугольник сменяется на карандаш. В распределенных базах данных Access блокирует редактируемую запись до ее сохранения. В остальных БД эта строка отображается в виде перечеркнутого круга. И, наконец, звездочка означает пустую строку в конце таблицы, которую можно использовать для создания новой записи. Эта запись присутствует всегда, если в данной таблице разрешено добавление записей.

В процессе добавления новых записей курсор автоматически перемещается в следующую строку. Введенная запись сохраняется автоматически, как только ее покидает курсор, т.е. для сохранения записи достаточно просто перейти на соседнюю. Это позволяет избежать промежуточного сохранения таблицы и при любом сбое будет потеряна только редактируемая запись. Сохранить принудительно запись можно, нажав Shift+Enter в любом месте записи либо просто Enter в конце записи. Отменить сделанные изменения или вставку новой записи можно нажатием клавиши Esc. Переход по полям от первого к последнему в записи и далее в начало следующей записи производится с помощью клавиши Tab.

Для облегчения ввода новых записей Access предоставляет несколько «горячих» комбинаций клавишей:

Ctrl + ;	вводит текущую дату
Ctrl + :	вводит текущее время
Ctrl + Alt + Space	вводит значение поля, установленного по умолчанию
Ctrl + ’	вводит значение того же поля из предыдущей записи
Ctrl + Enter	вставляет символ “возврат каретки” в поле *Мемо* либо в текстовое поле
Ctrl + +	добавляет новую запись
Ctrl + -	удаляет текущую запись.

Если в таблицу необходимо встроить объект из другого приложения, поддерживающего OLE интерфейс, следует обратиться к команде *Объект* меню *Вставка*. Эта команда будет доступна в том случае, если курсор будет находиться в этом поле и полю будет присвоен тип *Поле объекта OLE*. При этом имеется возможность, выбрав объект, запустить связанную с ним родительскую программу и провести изменения в объекте. Помимо встраивания объекта имеется также возможность установить просто связь таблицы с некоторым объектом, отметив в соответствующем диалоговым окне флажок *Связь*. Кроме внедрения и установления связей с объектом можно вставить в таблицу простую копию объекта, никак не связанную с оригиналом. Для этого необходимо воспользоваться командой *Специальная вставка*.

Старшие версии Access имеют расширенные средства копирования и импорта данных из сторонних источников информации: БД других форматов, текст, электронные таблицы, HTML и XML документы и т.д. Кроме импорта Access позволяет также присоединять таблицы из других БД, находящихся даже на удаленных

компьютерах. Оба приложения могут работать с такой таблицей одновременно. Для Access присоединение таблиц – это единственный способ создания распределенной БД, работающей в режиме «файл-сервер» с реализацией полноценного многопользовательского доступа к одним и тем же данным. При присоединении таблицы ее структура и данные не копируются, в Access хранится лишь информация о связи. С присоединенной таблицей можно работать точно так же, как и со своей таблицей. Ограничением является запрет на изменение ее структуры, хотя можно менять название таблицы (задать псевдоним присоединенной таблицы) и ряд свойств полей, определяющих способ ввода и отображения данных в Access.

Для поиска информации в таблице можно воспользоваться командой *Поиск*, а также применить сортировку и фильтрацию. Команда *Поиск* позволяет искать как в определенном поле, так и по всем полям сразу. Сортировку в Access можно производить по всем полям (кроме *Объект OLE*) независимо от того, был ли определен заранее соответствующий индекс. Фильтр является логическим выражением. Вывод тех записей таблицы, которые не удовлетворяют этому условию, подавляется, хотя доступ к ним через запросы, макросы или программы сохраняется. Синтаксис записи условия фильтра подобен синтаксису определения условия на значение, кроме необходимости явного определения имен полей, по которым производится фильтрация. Объединяя такие условия с помощью логических операций, можно создавать сложные фильтры. Фактически, фильтр – это условие отбора в запросах SQL, т.е. та его часть, которая записывается после оператора WHERE. Фильтры в Access бывают четырех типов:

1. Фильтр по выделенному (Filter by Selection). Для его применения достаточно выделить содержимое некоторого поля, или даже его часть, и вызвать соответствующую команду. Производит фильтрацию только по одному полю, сравнивая значения данного поля всех записей с выделенной. Допустимо также исключить выделенное (в старших версиях Access);
2. Фильтр для (Filter for). Применение его подобно предыдущему, только отсутствует необходимость вначале искать и выделять требуемое значение фильтрации. Достаточно просто ввести условие и применить команду (доступно через локальное меню). Также производит фильтрацию только по одному полю;
3. Фильтр по форме (Filter by Form). Позволяет визуально создавать сложные фильтры, состоящие из нескольких условий, объединенных логическими операциями AND или OR. Производит фильтрацию по нескольким полям одновременно;
4. Расширенный фильтр (Advanced Filter). Позволяет визуально создавать фильтры произвольной сложности. Для его создания применяется язык QBE, который будет рассмотрен в следующем параграфе.

Особенностью работы с базами данных является то, что реального удаления записей после выполнения операции удаления не происходит. Запись лишь помечается на удаление. Access дополнительно сразу не удаляет и свои объекты, поэтому размер БД только увеличивается. Хотя БД специально созданы для безопасного хранения информации, данные могут повреждаться вследствие аварийного закрытия приложений и ряда других причин. Для сжатия и восстановления БД любая СУБД и Access в том числе предоставляет набор средств. В новых версиях Access сжатие и восстановление объединены в одну операцию *Сжатие и Восстановление БД*. Периодическое выполнение данной операции очень полезно, поскольку проверяется и восстанавливается вся структура БД.

1.3. Запросы

Запросы в Access можно создавать как с помощью визуальных средств – QBE (Query By Example – запрос по образцу) и Мастера, так и с помощью SQL (Structured Query Language – структурированный язык запросов). В этом параграфе мы более подробно рассмотрим QBE (режим конструктора). Окно QBE состоит из панели задания таблиц и панели спецификации запроса (смотри Рис. 4). Перед созданием запроса в QBE появляется окно выбора имеющихся таблиц и запросов. Для создания запроса необходимо выбрать по крайней мере одну таблицу (следует выбирать только те таблицы, из которых необходимо извлекать данные, и те, которые требуются для установления связей). Если между таблицами были определены связи в окне *Схема данных*, то они автоматически будут установлены и в запросе. В противном случае их можно установить аналогичным способом с помощью перетягивания требуемого поля из одной таблицы в другую.

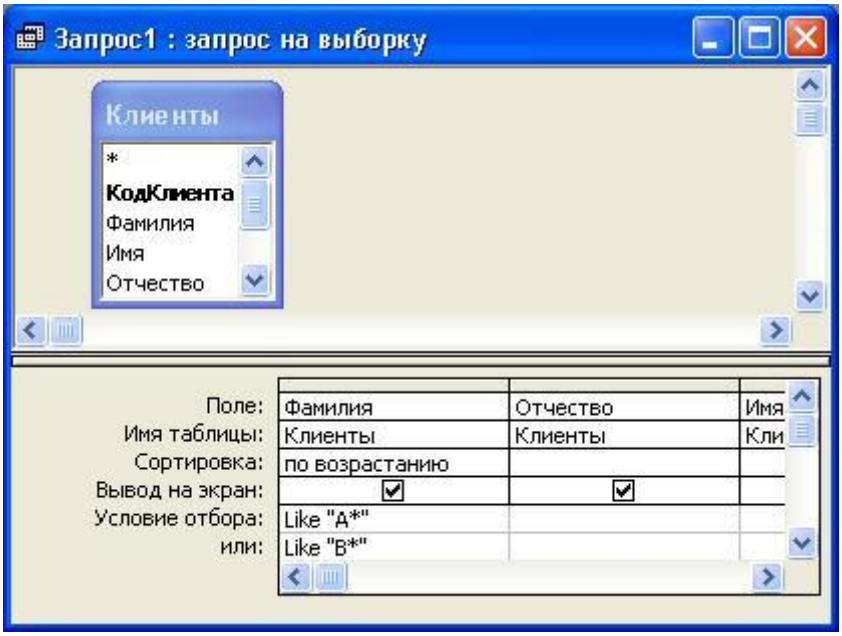


Рис. 4. Окно QBE

В верхней строке спецификации запроса отображаются выбранные поля. Для выбора поля (эквивалентно добавлению названия поля в список полей во фразе SELECT в SQL) достаточно перетянуть его с помощью мыши из таблицы в требуемый столбец спецификации, представляющий описание одного поля запроса, или выбрать его из выпадающего списка. Для полей можно дополнительно установить порядок сортировки и вывод на экран. Перетянув выбранные поля в область спецификации запроса, мы таким образом определим запрос. Если требуются все поля некоторой таблицы, то можно перетянуть первую строку этой таблицы с изображением *, означающим “Все поля”. Остается только сохранить запрос и запустить его на выполнение либо с помощью кнопки , либо просто открыв в режиме таблица. Поля в таблице, являющейся результатом выполнения запроса, отображаются в том порядке, в котором они указаны в спецификации запроса.

С помощью такого запроса мы выберем указанные поля для всех записей. Чтобы ограничить число записей, можно воспользоваться условием отбора. Правила записи условий отбора (и список доступных функций) практически идентичны правилам задания условий на значения в таблице. Условие отбора можно задать для каждого поля, входящего в запрос. При этом все условия отбора объединяются по логической операции И. Для реализации условия отбора по ИЛИ служат нижние строки спецификации запроса, как показано на Рис. 4. Если поле включается в запрос только для того, чтобы указать условие отбора, то вывод его на экран можно отключить, сняв пометку с флажка в строке *Вывод на экран*.

В поле спецификации запроса можно определять также вычисляемые поля. Правила записи вычисляемых полей совпадают с определенными в SQL. Отличается лишь задание альтернативного имени. Альтернативное имя задается перед вычисляемым полем и отделяется двоеточием, например: ПолноеИмя: [Фамилия] &" "& [Имя] &" "& [Отчество]. Это поле будет отображаться наряду с другими. Для облегчения ввода длинных выражений можно увеличить область ввода нажатием Shift + F2. Для проведения внешнего объединения в режиме конструктора необходимо вызвать редактор параметров объединения (с помощью двойного нажатия клавиши мыши на линии связи между таблицами) и выбрать необходимый тип объединения. На связи появится стрелка, направленная к таблице со стороны ∞ (для левого объединения).

При открытии запроса в режиме конструктора в меню появляется дополнительная группа команд меню Запрос, позволяющая создавать различные типы запросов. Можно визуально создавать все запросы действий, перекрестный запрос и некоторые виды запросов SQL. Дополнительно из этого меню можно открыть окно задания типов параметров (аналог инструкции SQL PARAMETERS). Методика визуального создания запросов действия и перекрестного запроса в QBE следующая. Вначале необходимо создать запрос выборки, отбирающий необходимые данные (полезно проверить его на выполнение и убедится, что вы выбираете действительно нужные записи), затем с помощью соответствующей команды меню перевести его в нужный вид. Поле спецификации запросов будет дополнено соответствующей строкой (*Обновление*, *Удаление* и *Перекрестная таблица*). Остается задать необходимые значения и запустить запрос на выполнение.

Запрос с группировкой можно создать по тому же сценарию. Вначале создается запрос выборки требуемых данных, затем вызывается команда *Групповые операции* (доступна из локального меню поля спецификации запроса и общего меню). Эта команда дополнит поле спецификации строкой *Групповые операции*. В этой строке автоматически устанавливается функция *Группировка* (GroupBy). Кроме функции группировки в этой строке для полей запроса можно задать вычисление групповых функций или задать условие отбора. При задании группировки необходимо помнить о том, что в запросе с группировкой могут участвовать только те поля, которые принимают единственное значение для группы, т.е. сами поля, для которых определена функция *Группировка*, выражения с ними или некоторые групповые функции над другими полями, возвращающими единственное значение для группы.

Для полей запроса можно поменять некоторые свойства, определенные для таблицы. Дополнительно в свойствах самого запроса можно задать выбор только уникальных значений (предикат DISTINCT в SQL), уникальных строк (предикат DISTINCTROW в SQL), установить блокировку записей, фильтр, тип набора записей (определяет возможность редактирования запроса) и многое другое. Внешне таблица, являющаяся результатом выполнения запроса, ничем не отличается от других таблиц БД. С ней можно работать точно так же, как и с обычной таблицей, т.е. просматривать данные, производить поиск, сортировку, фильтрацию, и даже изменять данные. Изменения будут немедленно отражены и в соответствующей таблице. Разумеется, не все запросы являются редактируемыми. Можно редактировать только однотабличные и некоторые двухтабличные без предикатов, группировки и внешнего объединения. Узнать, является ли запрос редактируемым, можно по наличию последней пустой строки, обозначенной с помощью * в селекторном столбце.

Запрос в Access можно сохранить в виде фильтра, и в этом случае его действие не будет отличаться от расширенного фильтра для таблицы. Хотя запросы и можно сохранять в виде фильтров, между ними имеются существенные различия:

1. Фильтры не позволяют объединять данные из нескольких таблиц, т.е. фильтр определяется только для отдельной таблицы;
2. Фильтры подавляют вывод всей строки, а не отдельных полей;

3. После применения фильтра возможность доступа к не вошедшим в выходной набор записям сохраняется, а после применения запроса – нет.

1.4. Формы

Формы являются средством ввода-вывода данных и организации интерфейса в MS Access. С помощью форм можно:

1. выводить и редактировать данные;
2. вводить данные;
3. управлять ходом выполнения приложения;
4. выводить сообщения;
5. печатать информацию.

В форме обычно выводятся поля одной записи некоторой таблицы или запроса. В этом случае она просто представляет удобный бланк для заполнения таблицы либо форматированного вывода данных, в которой кроме полей ввода/вывода можно разместить названия полей, подсказки, переключатели, флажки, списки, кнопки и другие элементы управления. Но форма может и не быть основанной на определенной таблице либо запросе (называемом базовым набором данных). Тогда она служит для вывода некоторой разнородной информации либо для управления ходом выполнения приложения. Например, заставка приложения тоже является формой.

Создавать формы в Access можно с помощью автоформ, мастера и с помощью конструктора. Наиболее удобно и быстро создавать формы с помощью автоформ. Для этого только необходимо указать базовую таблицу или запрос и вид формы. Автоформы позволяют создавать достаточно сложные формы на основе анализа структуры базовой таблицы или запроса. Но основным средством создания форм является конструктор.

В режиме конструктора (смотри Рис. 5) форма имеет область заголовка, область данных и область примечаний. Размеры этих областей можно легко менять.

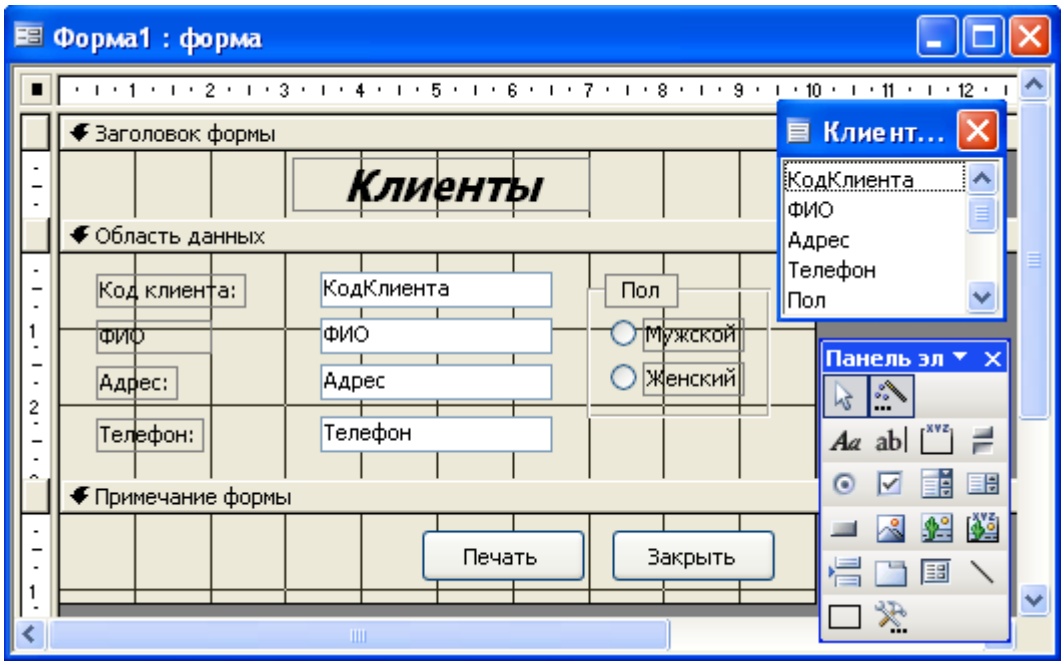


Рис. 5. Форма в режиме конструктора

Обычно информация, выводимая из базовой таблицы либо запроса, размещается в области данных (информация, представленная в этой области, меняется от записи к записи). Информацию, которая не меняется от записи к записи, лучше всего разместить в области заголовка либо примечания. В этих областях можно выводить

реквизиты фирмы, итоговые значения, а также те элементы, которые относятся ко всем записям базовой таблицы, к примеру, кнопки и другие элементы управления ходом выполнения приложения. Например, в область заголовка можно поместить список выбора товара, а в области данных – поля, описывающие товар. Тогда после выбора товара из списка в области данных появится его описание.

Access использует те элементы управления, которые Windows предоставляет для своих приложений. Условно все элементы управления можно разбить на три группы: присоединенные, свободные и вычисляемые. Присоединенные элементы управления связаны с полями базового набора данных и, соответственно, отображают и позволяют редактировать данные, содержащиеся в связанных с ними полях таблицы. Свободные элементы служат для разнообразия интерфейса и ввода/вывода некоторой не содержащейся в базовом наборе информации. Вычисляемые элементы позволяют вычислять свои значения на основе значений других элементов управления либо полей базового набора. При открытии формы в режиме конструктора (смотри Рис. 5) на экран выводится *Панель элементов*, на которой размещены все доступные элементы управления. Рассмотрим их подробнее:

Надпись  применяется для размещения в форме статического текста: названия полей, поясняющих надписей и т.п. Большинство других элементов управления помещается на форму вместе с присоединенной к ним надписью.

Поле  применяется для ввода/вывода и редактирования данных. Позволяет реализовать прямой доступ к данным после привязки к соответствующему полю таблицы с помощью свойства *Данные*. Вместе с надписью является основным элементом управления.

Выключатель  , переключатель  и флажок  применяются для ввода/вывода и редактирования логических данных. Будучи объединенными в группы, позволяют реализовать наглядный выбор одного варианта из нескольких возможных.

Группа  применяется для предоставления пользователю выбора из небольшого набора вариантов. Выбор вариантов осуществляется с помощью встроенных в группу выключателей, переключателей либо флажков. При этом привязку к некоторому полю имеет только группа, а встроенные элементы управления не привязываются. Значением группы является некоторое уникальное значение (обычно это порядковый номер выбранного элемента управления, встроенного в группу).

Список  применяется для отображения набора данных в виде списка или предоставления пользователю выбора из сравнительно большого набора вариантов. Набор данных может быть представлен фиксированным списком значений, таблицей либо запросом. Для выбора определенного значения достаточно просто выделить его в списке. Список может состоять из одного или нескольких столбцов, но в привязанное поле таблицы передается значение только одного столбца. Сделав размер присоединенного столбца нулевым, можно организовать вывод одних данных и подстановку других. Например, вывод названия товара и подстановку его кода.

Поле со списком  применяется для предоставления пользователю выбора значения из раскрывающегося списка и ввода/вывода данных. Является комбинацией элементов управления *Поле* и *Список*. Обычно элемент управления *Поле со списком* применяется для отображения внешних ключей таблицы, т.е. тех полей, которые содержат значения, соответствующие первичным ключам базовой таблицы. Тогда аналогично элементу управления С*писок* можно организовать вывод значащих полей базовой таблицы, а подставлять в связанное поле соответствующее значение первичного ключа.

Кнопка  применяется для выполнения некоторых действий типа закрытия формы. Привязав к событию *Нажатие кнопки* макрос или процедуру VBA, можно выполнить ряд действий, определенных в макросе (процедуре).

Рисунок  , Свободная рамка объекта  и Присоединенная рамка объекта 

применяются для отображения графических объектов и других объектов, поддерживающих интерфейс OLE. Отличие состоит в том, что *Рисунок* предназначен для встраивания статических изображений, *Свободная рамка объекта* – для отображения или встраивания не связанного с таблицей объекта OLE, а *Присоединенная рамка объекта* – для отображения содержимого поля типа *Поле объекта OLE*.

Вкладка  применяется для отображения нескольких вкладок в форме. С помощью набора вкладок можно представить большую совокупность данных компактным образом, разместив на каждой вкладке ряд элементов управления.

Разрыв страницы  применяется для создания форм, имеющих несколько страниц. Часть формы после элемента управления *Разрыв страницы* будет отображаться на следующей странице. Для переключения между страницами необходимо написать группу макросов или процедур VBA и активировать их, например, с помощью элемента управления *Кнопка*.

Подчиненная форма  применяется для внедрения в форму другой формы. Если основная форма основана на главной таблице, а подчиненная – на таблице, связанной с главной отношением ∞ – 1, то отображение данных в основной и подчиненной форме можно легко синхронизировать.

Линия  и Прямоугольник  применяются для оформления внешнего вида форм.

Кроме вышеприведенных элементов управления можно также воспользоваться большой совокупностью установленных на компьютер ActiveX объектов, таких как календарь, стандартный диалог открытия и сохранения документов и т.д. Список доступных ActiveX объектов открывается по кнопке  , размещенной на *Панели элементов*.

Создание формы в режиме конструктора заключается в размещении требуемых элементов управления на форме и задания их свойств. Список полей базового набора, появляющийся при отображении формы в режиме конструктора, облегчает создание связанных элементов управления. Например, для создания поля с подписью, связанного с некоторым полем таблицы, достаточно перетянуть его в один из разделов формы.

Формы могут быть следующих видов: простая, составная, ленточная, многостраничная и диалоговая. Простой называется форма, которая основана на данных одной таблицы или запроса. Если число полей в базовой таблице столь велико, что не помещается на весь экран, то можно создать многостраничную форму (многостраничную форму можно создать как с помощью элемента управления *Разрыв страницы*, так и с помощью элемента управления *Вкладка*). Если же, наоборот, число полей невелико, то можно создать ленточную форму, где записи выводятся так же, как и в таблице, но для каждой записи отводится больше места для удобного представления полей. Любую форму, основанную на некотором наборе записей, можно также просматривать и в виде таблицы. Диалоговая форма обычно применяется для создания диалоговых окон, а не для вывода и редактирования данных, т.е. для задания некоторых параметров, действий и т.д. В то же время любую форму можно сделать модальной (монопольной). Тогда доступ к другим окнам будет блокирован до закрытия данной формы.

Для одновременного отображения и редактирования записей из двух таблиц, связанных отношением 1 - ∞, наиболее удобной является составная форма (master-detail form). Тогда записи из таблицы со стороны 1 будут отображаться в основной форме, а из таблицы со стороны ∞, имеющие совпадающие значения в связанных полях, – в подчиненной. Подчиненная форма располагается внутри основной (является элементом управления *Подчиненная форма* для основной формы) и обычно представлена в виде таблицы или ленточной формы. Создать такую форму можно даже автоматически с помощью автоформ на основе запроса, объединяющего данные из главной и подчиненной таблиц. Пример составной формы представлен на Рис. 6. Такая форма позволяет просматривать и обновлять информацию о клиенте и обо всех сделанных им заказах. В составной форме значения внешнего ключа подчиненной таблицы (в нашем случае *Код клиента* таблицы *Заказы*) назначаются

автоматически при добавлении записей. Естественно, вначале производится ввод нового клиента, а затем его заказов.

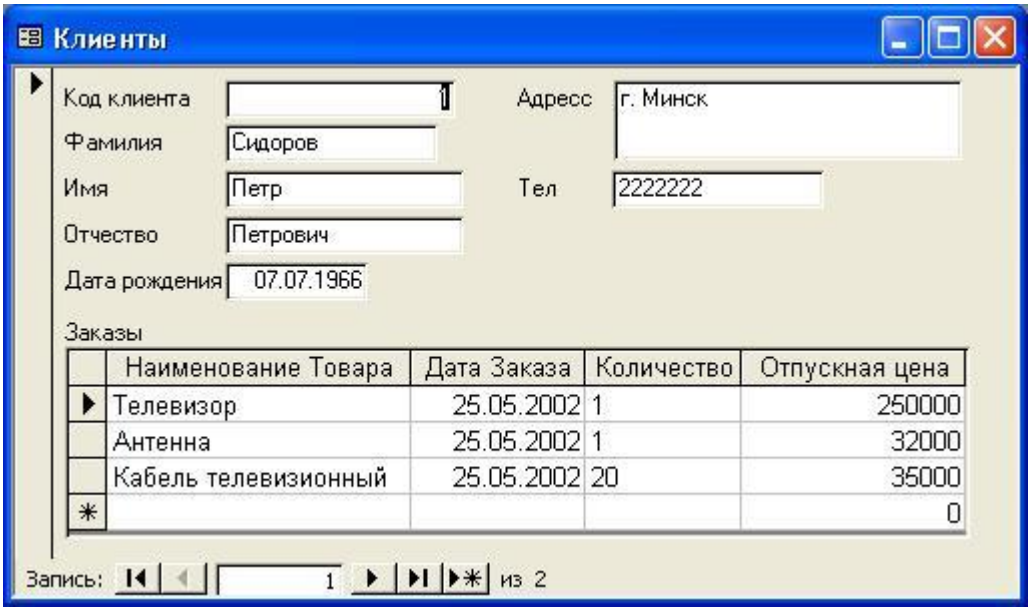


Рис. 6. Составная форма

Для просмотра данных в таблице со стороны ∞ вместо подчиненной формы можно также использовать список. С другой стороны, если есть желание в форме, основанной на подчиненной таблице, вместо значений внешних полей (зачастую неинформативных табельных номеров) видеть соответствующие им значения некоторых других полей базовой таблицы, то наилучшим средством является применение комбинированных списков. В этом случае комбинированный список будет отображать заданные поля соответствующей записи со стороны 1, а подставлять вместо них значение первичного ключа. Например, поле *Наименование товара* в форме, представленной на Рис. 6, является комбинированным списком, подставляющим названия товаров из поля *Товар* таблицы *Товары*. В противном случае в этом поле отображались бы табельные номера товаров.

Вычисления в формах можно производить не только на основе значений других элементов управления или полей базового набора, но и на основе данных, хранящихся в других таблицах. Можно также рассчитывать итоговые значения по всему множеству данных из базового набора или других запросов и таблиц, либо по его подмножеству, определяемому логическим выражением. Для выполнения вычислений необходимо поставить знак равенства в свойстве *Данные* и ввести выражение, например =[Цена]*[Количество]. Операндами выражения могут служить названия полей базового набора (все поля базового набора, а не только те, которые включены в форму) или названия элементов управления, т.е. предыдущий пример может быть записан и следующим образом = Поле_Цена*Поле_Количество. Итоговые вычисления производятся с помощью агрегатных функций, имеющих тот же смысл и написание, как в запросах. Аргументами их могут быть только имена полей. Например, = Sum([Цена]*[Количество]) или = Count([Код заказа]). Размещать такие поля следует в области примечаний формы. Тогда множеством данных будут все записи, отображаемые в форме. Если поместить расчет итогов в область примечаний подчиненной формы, то можно подсчитать общую сумму и количество товаров, заказанных каждым клиентом.

Итоговые функции по подмножеству позволяют производить вычисления или просто выводить в форме данные, не принадлежащие базовому набору. Аргументами этих функций являются название поля, по которому производится вычисление, название таблицы или запроса и условие отбора записей, имеющее тот же синтаксис, что и условие отбора в запросах. Функция DLookup используется обычно для вывода значения поля, не принадлежащего базовому набору. Например, в форме *Товары* можно вывести сведения о поставщике,

хранящиеся в связанной отношением ∞ – 1 таблице *Поставщики*: =DLookUp("[Должность]";"[Поставщики]";"[КодПоставщика]=" & [КодПоставщика]). Здесь условие отбора "[КодПоставщика]=" & [КодПоставщика] реализовано с помощью конкатенации условия [КодПоставщика]= (где *КодПоставщика* – это поле таблицы *Поставщики*) и значения, возвращаемого одноименным элементом управления формы *Товары*. Функции DCount, DSum позволяют организовать вычисления общей суммы и количества товаров, купленных клиентом: =DCount("[Код заказа]";"[Заказы]";"[Код клиента]=" & [КодКлиента]) и =DSum("[Цена]*[Количество]";"[Заказы]";"[Код клиента]=" & [КодКлиента]). С помощью функции DMax можно организовать счетчик записей: =DMax("[Код заказа]";"[Заказы]") + 1.

Форма, каждый раздел формы и все элементы управления имеют свой список свойств. С помощью этого списка можно управлять выводом данных на экран, форматом их представления, обработкой событий и многим другим. Список свойств для каждого типа объектов различен. Некоторые из них могут иметь до 80 различных свойств (смотри Рис. 7).

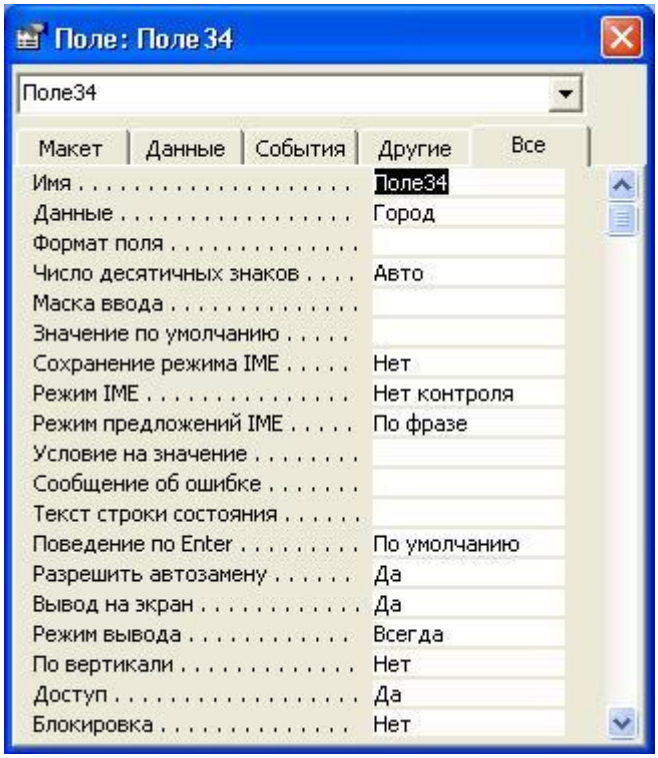


Рис. 7. Свойства элемента управления “Поле”

Для удобства просмотра свойства разделены на группы: свойства данных, свойства макета, свойства событий и другие (не относящиеся к вышеперечисленным группам). Научившись работать с ними, можно создавать очень сложные формы. Отметим, что все эти свойства доступны для макросов и программ, написанных на Visual Basic. Таким образом, меняя значения соответствующих свойств элементов управления (и свойств самих форм), можно управлять формами и представлением в них данных. Технология программирования в Access предполагает овладение техникой обработки событий и манипулирования свойствами объектов.

В списке свойств наиболее значимыми являются имя элемента и источник данных. По имени элемента вы можете ссылаться на него даже из другой формы (при условии, что форма будет открыта), а источник данных предопределяет то, что же появится в этом элементе (если, конечно, он предназначен для отображения данных). В качестве источника данных можно задать поле базовой таблицы, инструкцию SQL, которая найдет необходимые данные даже в некоторой другой базе данных, либо задать вычисляемое выражение (предполагается обязательное наличие знака равенства в начале выражения, например, =Цена*Количество). Для элемента управления *Поле со списком* и *Список* дополнительно нужно определить источник строк для списка и его тип. Источником строк

может быть список фиксированных значений, таблица или запрос. В последнем случае источником строк обычно является инструкция SQL. Если источник строк имеет несколько столбцов, то еще необходимо указать связанный столбец, т.е. тот столбец, откуда будут браться значения для присоединенного поля.

Для элементов управления *Поле*, *Поле со списком* и *Список* имеются три свойства, определяющие способ вывода данных в форме. К ним относятся: *Формат поля*, *Число десятичных знаков* и *Маска ввода*. Access копирует эти свойства из полей базовой таблицы, но при желании их можно переустановить.

Для самой формы имеется ряд свойств, предназначенных для управления работой с данными. Выделим наиболее важные из них:

1. Источник данных (RecordSource) – определяет базовую таблицу/запрос для формы;
2. Фильтр (Filter) - позволяет задать фильтр для базовой таблицы/запроса формы;
3. Порядок сортировки (OrderBy) – указывает сортировку базовой таблицы/запроса;
4. Разрешить изменение (AllowEdits) – определяет, можно ли изменять сохраненные записи через форму
5. Разрешить удаление (AllowDeletions) – определяет, можно ли удалять записи через форму;
6. Разрешить добавление (AllowAdditions) определяет, можно ли добавлять записи через форму;
7. Ввод данных (DataEntry) определяет для формы режим только для ввода данных. Это свойство не определяет возможность добавления записи; оно только устанавливает, будут ли выводиться существующие записи;
8. Тип набора записей (RecordsetType) определяет тип набора записей. Можно выбрать следующие значения:
   - Динамический набор – (значение по умолчанию) допускается редактирование элементов управления в единственной таблице или в нескольких таблицах со связями типа «один-к-одному». Не допускается изменение данных для элементов управления, присоединенных к полям таблиц со связями типа «один-ко-многим» со стороны «один», если не разрешено каскадное изменение между таблицами,
   - Динамический набор (Несогл.) – допускается редактирование всех таблиц и элементов управления, присоединенных к их полям,
   - Статический набор – не допускаются изменения данных в таблицах и в присоединенных к их полям элементах управления;
9. Блокировка записей (RecordLocks) определяет способы блокировки записей и их реализацию при попытке двух пользователей одновременно изменить одну и ту же запись. Можно выбрать следующие значения:
   - Отсутствует – (Значение по умолчанию.) Применяется в формах, которые могут изменяться одновременно несколькими пользователями. Такую блокировку называют также «нежесткой». Если два пользователя пытаются сохранить изменения одной и той же записи, то Microsoft Access выведет соответствующее сообщение пользователю, который предпринял такую попытку вторым. Ему предоставляются следующие возможности: отменить внесенные изменения, скопировать запись в буфер обмена или переписать запись, измененную другим пользователем. Данное значение обычно применяется для форм, которые доступны только для чтения, или для форм в базах данных, рассчитанных на одного пользователя. Этот режим используется также в сетевых базах данных, чтобы разрешить нескольким пользователям одновременно вносить изменения в одну и ту же запись.

– Всех записей – блокируются все записи в базовой таблице или запросе при открытии формы. Другим пользователям разрешается просматривать записи, но они не могут изменять, добавлять или удалять любые записи до закрытия формы.
– Изменяемой записи – страница записей блокируется, как только любой из пользователей начинает вносить изменения в любое поле одной из этих записей, и остается заблокированной до тех пор, пока пользователь не перейдет к другой записи. Одна запись может изменяться одновременно только одним пользователем. Такую блокировку иногда называют «жесткой».

1.5. Отчеты

Отчеты являются наилучшим средством для предоставления данных в виде печатного документа и сочетают в себе возможности форматированного вывода данных с группированным представлением большого количества записей из нескольких таблиц. Так как число записей и число полей может быть велико, то отчеты могут выводится на нескольких страницах. Кроме вывода самих данных отчеты позволяют производить разнообразные вычисления и расчет итоговых значений. Отчет является автоматически генерируемым законченным документом и его нельзя редактировать. Единственно, что возможно, так это просмотреть его в режиме предварительного просмотра перед печатью либо внедрить в состав другого документа, например, в документ MS Word или MS Excel.

Отчеты и формы имеют много общего. Однако, в отличие от форм, отчеты не предназначены для ввода и правки данных в таблицах и в них нет смысла размещать такие элементы управления, как кнопки, поля со списком, группы и т.п. Отчеты обладают следующими возможностями:

- имеют широкие возможности для форматирования текста;
- имеют широкие возможности для группировки, а также для вычисления промежуточных и общих итогов. В отчете каждая группа и итоги по ней представлены отдельно, и можно определить до 10 уровней группировки;
- можно производить сложные вычисления не только внутри некоторой группы, но и по нескольким группам одновременно;
- позволяют легко создать такие документы, как счета, почтовые наклейки и т.д.

Создавать отчеты в Access можно с помощью автоотчетов, мастера и с помощью конструктора. Наиболее удобно создавать формы с помощью автоотчетов. Для этого необходимо только указать базовую таблицу или запрос и вид отчета. Мастер позволяет создавать более сложные отчеты с группировкой и расчетом итогов. Но основным средством разработки отчетов является конструктор.

Отчет, как и форма, имеет область заголовка, примечания и область данных. Дополнительно к ним (как для любого печатного документа) можно определить верхний и нижний колонтитулы. Заголовок и примечание отчета, верхний и нижний колонтитулы не являются обязательными и их можно добавить или удалить по желанию с помощью соответствующей команды меню *Вид*. Но, в отличие от форм, число областей не является фиксированным и зависит от данных, размещаемых в отчете. Некоторые области могут повторяться многократно, отражая соответствующую информацию из базовой таблицы (запроса), а некоторые будут представлены в единственном числе. К примеру, область данных повторяется для каждой записи базового набора, а заголовок отчета всего один раз на первой странице. Рассмотрим подробнее, из каких областей состоит отчет, и какие данные можно в них размещать:

1. Область заголовка и область примечания отчета. Эти области располагаются соответственно один раз в начале и конце отчета. Обычно в область заголовка отчета включают название отчета, дату его

создания и логотип предприятия. В области примечания помещают расчет итоговых результатов по данным всего отчета и некоторую служебную информацию. Если отчет многостраничный, то эта информация будет размещена на последней странице.

2. Верхний и нижний колонтитулы. Эти области предназначены для вывода информации, повторяющейся для каждой страницы. Например, там можно поместить номер страницы, шапку для вывода некоторой табличной информации и т.п.
3. Область данных. В области данных размещается вывод основных данных. Вертикальные размеры этой области должны быть минимальными (фактически по размеру элементов управления, отвечающих за вывод данных), поскольку область данных повторяется для каждой записи базового набора. В свою очередь область данных также может иметь сложный вид, т.к. обычно в отчете выводится группированная информация. Тогда для каждой группы можно дополнительно задать свои заголовок и примечание.
4. Область заголовка и область примечания группы. В этих областях помещается та информация, которая является общей для группы. Формирование данных областей производится одновременно с заданием группировки. Например, здесь можно выводить заголовок группы (поле, по которому производится группировка и выбирается обычно в качестве заголовка группы). В области примечания обычно размещается расчет итогов (так называемых промежуточных итогов) по каждой группе.

Так как отчеты содержат большое количество записей, то для удобства просмотра они должны быть определенным образом сгруппированы и упорядочены. Если отчет основан на запросе, то он игнорирует все условия сортировки и группировки, заданные для него. Группировка и сортировка данных задается прямо в отчете с помощью окна *Сортировка и группировка* (смотри Рис. 8).

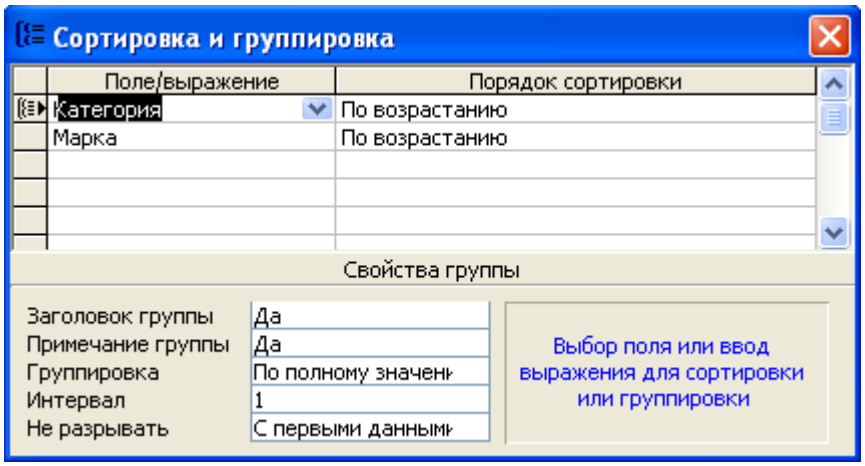


Рис. 8. Окно *Сортировка и группировка*

Вызвать появление этого окна можно с помощью кнопки  или соответствующей команды меню. В данном окне можно определить до 10 полей или выражений для группировки данных (т.е. группировка может осуществляться по значению выражения, например, = [Фамилия] & " " & [Имя] & " " & [Отчество]). Первое поле или выражение определяет основную группу, последующие – подгруппы внутри группы предыдущего уровня. В этом же окне можно включить либо выключить вывод областей заголовка и примечания для каждой из групп. Дополнительно можно установить еще три свойства: *Группировка*, *Интервал* и *Не разрывать*. Свойство *Не разрывать* устанавливается в *Полная группа*, если есть желание, чтобы записи, относящиеся к одной группе, размещались на одной странице. Свойство *Группировка* и *Интервал* связаны между собой и позволяют выделять данные в новую группу не по полному изменению значения поля, участвовавшего в группировке, а по смене, скажем, его части, или когда значения поля попадают в некоторый диапазон. Вид и задаваемый размер диапазона

зависят от типа поля. Для текстовых значений можно указать, чтобы Access начинал новую группу при изменении первого или нескольких начальных символов значения поля, а для числовых или типа *Дата/время* задать интервал значений. Для задания количества первых символов или интервала значений используется свойство *Интервал*. Т.е. для численных данных свойство *Группировка* устанавливается в *Интервал*, а в свойстве *Интервал* задается интервал значений. Например, 10. Тогда будут формироваться следующие группы: от -20 до –11, от -10 до –1, от 0 до 9, от 10 до 20 и т.д. Для поля типа *Дата/время* свойство *Группировка* может принимать следующие значения: *По годам*, *По кварталам*, *По месяцам*, *По неделям*, *По часам*, и *По минуте*. Тогда значение свойства *Интервал* будет определять количество соответственно лет, кварталов, месяцев и т.д. в каждой группе. Создание отчетов в режиме конструктора почти ничем не отличается от создания форм. Требуется только установить сортировку и группировку, включить области заголовков и примечаний и разместить требуемые поля (или другие элементы управления) в соответствующих областях отчета. При этом необходимо помнить, что имеет смысл использовать только те элементы управления, которые можно вывести на печать, и что вывод данных будет определяться областью размещения соответствующих элементов управления. Например, одно и тоже вычисляемое поле может выполнять совершенно разные действия. Поле со свойством *Данные* =Count([Код заказа]) будет подсчитывать количество заказов для каждого клиента, если оно помещено в область заголовка или примечаний для группы *Клиенты*, и общее количество заказов, если будет помещено в области заголовка/примечаний всего отчета.

При создании отчета важно не только определить данные, которые должны быть представлены в отчете, но и нужным образом оформить отчет как документ: правильно разбить его на страницы, пронумеровать страницы, выделить наиболее важные данные. Если не все нужные данные помещаются на одной странице, Access позволяет вставить принудительный разрыв страницы. Текущая дата вводится с помощью функций Now() или Date(). Номер страницы представляется с помощью системных переменных Page и Pages. Например: "Страница" & [Page] & "из" & [Pages]. В результате получите: Страница 5 из 25. Вставить вывод даты и номера страницы можно также с помощью команд меню.

Для вычисления промежуточных (по группе) итогов необходимо разместить вычисляемое несвязанное поле в области примечаний группы. Например, Sum([Количество])*[Цена]) в качестве значения “данные” для вычисляемого поля. То же поле, но размещенное в области примечаний отчета, уже будет вычислять общий итог по всему отчету. Для вычисляемых полей в отчетах появляется новое интересное свойство – *Сумма с накоплением*. Если поле размещено в области данных и это свойство установлено в *Для группы*, то Access будет суммировать данные в каждой записи сбрасывать итоговое значение в 0 после начала каждой группы. Если поле размещено в области примечаний группы, Access будет накапливать итоговые суммы для всех групп этого уровня до тех пор, пока не встретится уровень группировки более высокого уровня. Таким образом, можно подсчитать количество купленных товаров по категориям и т.д. Если же свойство *Сумма с накоплением* установлено в *Для всего*, то Access будет суммировать данные по всем записям до конца отчета. Примером использования свойства *Сумма с накоплением* может служить нумерация строк в отчете. Если разместить несвязанное поле в области данных отчета и задать свойство *Данные* =1, а свойство *Сумма с накоплением* в *Для группы*, то можно пронумеровать к примеру заказы в пределах группы. Если установить свойство *Сумма с накоплением* в *Для всего*, то можно установить сквозную нумерацию.

Иногда требуется, чтобы определенные поля выводились в отчете в зависимости от их значения или от значения других элементов в отчете. Например, разумно не выводить поле Скидка, если она равна 0. К тому же желательно, чтобы остальные поля при этом перемещались вверх, чтобы в отчете не оставалось пустого места. В Access существует функция IIF(усл; выр 1; выр 2), возвращающая в зависимости от выполнения условия значения либо *выр 1*, либо *выр 2*. Для того чтобы поле “исчезало”, когда его значение окажется пустой строкой,

необходимо установить свойство *Сжатие* в *Да*. Свойство *Не выводить повторы* (HideDuplicates) дополнительно позволяет не выводить элемент управления отчета, если значение этого элемента управления совпадает с его значением в предыдущей записи.

Как и формы, отчеты могут быть также составными, т.е. иметь подчиненные отчеты. Использование подчиненных отчетов в ряде случаев позволяет просто упростить вывод связанных записей и задание группировки в отчете. Но в ряде случаев без подчиненных отчетов не обойтись. Например, требуется уровень вложенности более двух, представление итогов в виде отдельной таблицы наряду с выводом всех данных или главный отчет может быть простым контейнером для размещения нескольких подчиненных отчетов из несвязанных источников данных.

В качестве примера, рассмотрим создание отчета, предоставляющего информацию по клиентам и сделанным ими заказам. Дополнительно потребуем от отчета вычисления промежуточных и общих итогов. Для этого в окне С*ортировка и группировка* создадим одну группу по ключевому полю таблицы *Клиенты* и включим отображение заголовка и примечания для группы. Далее разместим элементы управления *Надпись* и *Поле* как показано на Рис. 9 и определим для них соответствующие данные.

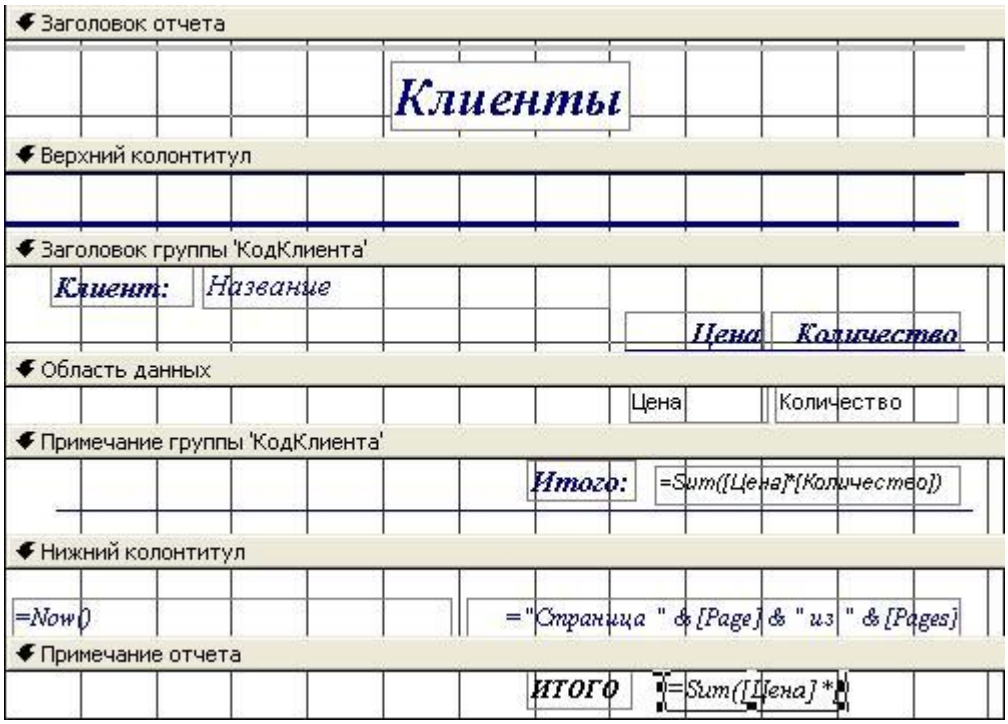
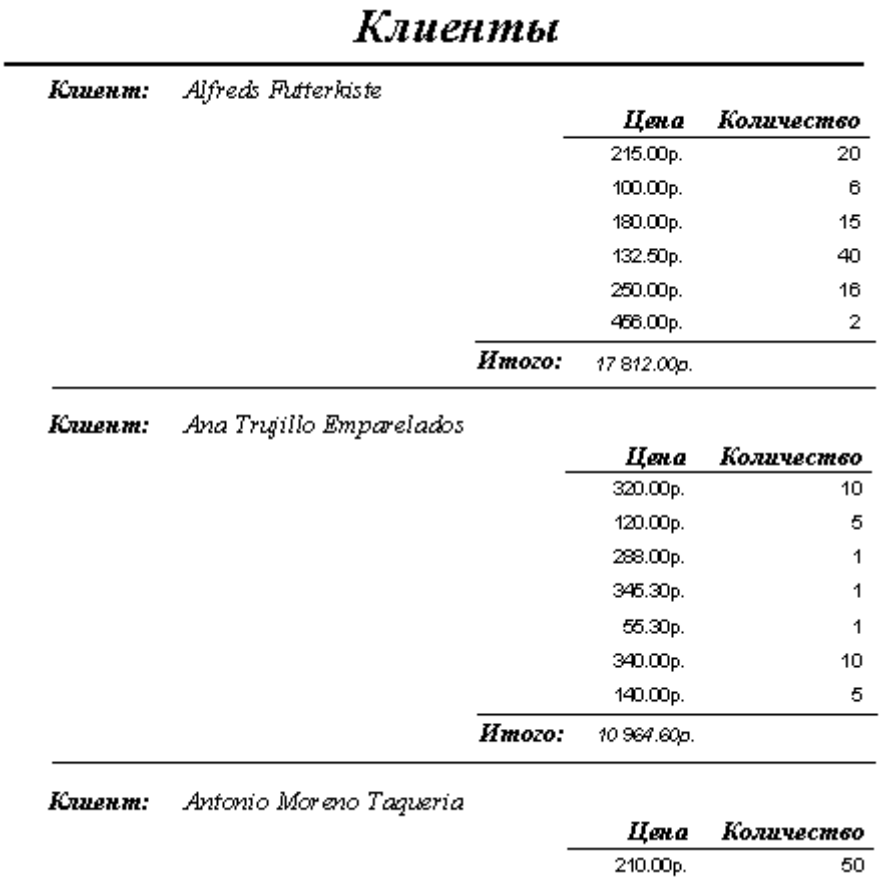


Рис. 9. Отчет в режиме конструктора

Тот же отчет в режиме предварительного просмотра будет выглядеть следующим образом (смотри Рис. 10)



Клиенты

Клиент: Alfreds Futterkiste

Цена	Количество
215.00р.	20
100.00р.	6
180.00р.	15
132.50р.	40
250.00р.	16
456.00р.	2
Итого: 17 812.00р.	

Клиент: Ana Trujillo Emparelados

Цена	Количество
320.00р.	10
120.00р.	5
288.00р.	1
345.30р.	1
55.30р.	1
340.00р.	10
140.00р.	5
Итого: 10 964.60р.	

Клиент: Antonio Moreno Taqueria

Цена	Количество
210.00р.	50

Рис. 10. Отчет в режиме предварительного просмотра

1.6. Автоматизация приложения

При создании распространяемой версии необходимо исключить доступ пользователей к объектам БД в режиме разработки. Если этого не сделать, то БД может быть легко повреждена при неквалифицированных действиях пользователей или даже случайно. Для этого в первую очередь необходимо отключить появление окна БД, содержащего список всех объектов БД и предоставляющего доступ к ним в режиме разработки. Работа с базой данных должна быть организована через интерфейс пользователя. Создание удобного и интуитивно понятного интерфейса, предоставляющего доступ ко всем объектам БД и автоматизирующего выполнение основных задач, является неотъемлемой частью разработки любой БД, даже если вы не планировали ее распространять. Интерфейс пользователя в Access реализуется с помощью форм, панелей инструментов и меню. Поскольку Access является СУБД и предоставляет полный набор операций, доступных через меню и встроенные панели инструментов, то создание пользовательского меню и панелей инструментов не является первоочередной задачей. Гораздо более важно научиться создавать удобные в работе и богатые функциональными возможностями формы. Если разрабатываемая БД небольшая и фактически предназначена для выполнения одной или небольшого количества взаимосвязанных задач, то достаточно создать одну главную форму, автоматически вызываемую при открытии БД, и ряд вспомогательных форм, открываемых обычно с помощью командных кнопок, размещенных на главной форме. Закрытие этой формы можно связать с закрытием всего приложения. Если же разрабатываемая БД большая и предназначена для выполнения ряда задач, то не обойтись без создания главной переключательной формы, которая автоматически вызывалась бы при открытии БД и оставалась открытой все время работы приложения. Создать такую форму можно самостоятельно в режиме конструктора или с помощью мастера создания переключательных форм, а задать ее автоматическое появление на экран наряду

с заданием названия приложения, значка приложения и т.п. можно с помощью диалогового окна *Параметры запуска*, вызываемого по одноименной команде меню.

Автоматизация приложения в Access достигается за счет обработки событий, происходящих в формах и отчетах с помощью макросов и модулей. Макросы и программы, написанные на языке Visual Basic for Applications и входящие в состав модулей, позволяют выполнить ряд запрограммированных действий и изменять состояние любого объекта БД. Из макросов и модулей можно ссылаться на любой объект, читать и менять его свойства.

1.6.1. Ссылки на объекты

При связывании, проведении вычислений и автоматизации форм и отчетов часто приходится ссылаться на саму форму/отчет или на какой-нибудь элемент управления в ней, чтобы считать или установить его свойства или значение. Доступ к объектам БД в Access можно получить, вначале указав имя коллекции, а затем через восклицательный знак имя объекта в коллекции. Помимо операции “!” можно указать в скобках индекс или имя объекта. Индекс является порядковым номером открытой формы и начинается с нуля. Коллекции форм и отчетов являются глобальными и к объекту такой коллекции можно обращаться напрямую. Например: Reports![Годовой отчет], Forms!Заказы (иначе Forms(0) или Forms("Заказы")).

Для указания ссылки на свойство объекта используется операция “.”, за которой следует имя свойства

Forms![Catalog Item].Visible

Получив ссылку на объект или свойство, можно считать ее значение или, напротив, присвоить ей некоторое допустимое значение. Например:

frm = Forms![Catalog Item],

Forms![Catalog Item].Visible = False

По аналогии с коллекциями для основных типов объектов сами формы и отчеты становятся коллекциями имеющихся в них элементов управления. Синтаксис ссылки на конкретный элемент управления и на его свойства аналогичен вышеприведенному.

Forms![Заказы]![код_заказа] и Forms![Заказы]![код_заказа].Visible

Несколько сложнее дело обстоит со ссылками на подчиненные формы или отчеты. Когда форма внедряется внутрь другой формы, то она будет являться элементом управления *Подчиненная форма* для основной формы (т.е. ссылка к самой подчиненной форме или к ее свойствам ничем не будет отличаться от ссылки, скажем, на элемент *Поле*).

Forms![Заказы]![Подчиненная Заказы].Visible

Одним из свойств элемента управления *Подчиненная форма* является свойство *Форма* (Form), которое позволяет ссылаться в свою очередь на подчиненную форму и на элементы управления в ней

Forms![Заказы]![Подчиненная Заказы].Form![Код_заказа]. Для ссылки на свойство элемента управления внутри подчиненной формы остается добавить еще точку и имя свойства. При работе с макросами и модулями необходимо иметь в виду, что доступ возможен только к элементам управления открытых форм или отчетов. Если же есть желание иметь доступ к элементам управления, но не видеть форму открытой, то можно просто установить свойство (Visible) этой формы в False, т.е. скрыть ее.

Объекты в Access могут иметь свойство, используемое по умолчанию. Это свойство применяется в том случае, когда имя свойства в ссылке явно не указано. Например, у элемента управления *Поле* по умолчанию используется свойство *Значение* (Value), поэтому ссылка Forms!Товары!Цена позволяет получить доступ к значению, отображенному в текстовом поле Цена. Следовательно, можно просто присвоить некоторое значение

элементу управления *Поле*, а вместе с ним тому полю базового набора, с которым этот элемент управления связан. Например, Forms!Товары!Цена = 100 или наоборот, price = Forms!Товары!Цена.

1.6.2. События Access

Событие – это распознаваемое изменение состояния любого объекта MS Access или операционной системы. Может представлять любое действие, инициируемое пользователем или самой системой. Такие действия, как открытие окна, состоят из нескольких последовательно происходящих событий. Поскольку автоматизация приложения происходит за счет обработки событий, то конечный результат будет зависеть не только от умения правильно обрабатывать отдельные события, но и от знания последовательности их выполнения. Большое количество действий, например, вставка либо удаление записи, имеют по два события для их обработки. Первое происходит непосредственно перед выполнением действия, второе – сразу после него. События, происходящие перед выполнением некоторого действия, можно отменять. Отмена события фактически отменяет выполнение самого действия. Определить, является ли событие отменяемым, можно по наличию в списке его аргументов параметра Cancel. Тогда для отмены события достаточно будет установить его в True. Здесь мы рассмотрим основные события (весь список можно найти в справочной системе), условно разделив их на следующие группы:

- *События данных*. Происходят при изменении данных в элементе управления или в форме и при переходе от записи к записи:
  - *Текущая запись*. Происходит при обращении к источнику данных формы/отчета. Возникает как при открытии формы, так и при переходе от записи к записи.
  - *Удаление.* Происходит непосредственно перед реальным удалением записи.
  - *До подтверждения Del*. Используется для отмены появления окна запроса на подтверждение удаления и самого удаления.
  - *После подтверждения Del*. Используется для проверки статуса удаления записи.
  - *До вставки*. Происходит сразу перед вставкой записи (ввод первого символа в любое поле). Может использоваться для проверки, разрешена ли вставка записи.
  - *После вставки*. Происходит сразу после вставки записи.
  - *До обновления*. Возникает при обновлении данных как в самой записи, так и в элементе управления. Используется обычно для проверки условий целостности.
  - *После обновления*. Возникает сразу после обновления данных в записи или элементе управления. При обработке события можно восстановить старые данные, сохраняемые в свойстве Old Value элемента управления.
  - *Изменение*. Событие редактора. Возникает при вводе/удалении каждого символа.
  - *Внесены изменения*. Позволяет проверить, были ли изменения в записи.
  - *Отсутствие в списке*. Возникает в поле со списком при вводе значения, отсутствующего в списке.
- *События окна*. Происходят при открытии и закрытии форм и отчетов и при изменении их размеров:
  - *Открытие*. Происходит непосредственно перед отображением первой записи.
  - *Закрытие*. Происходит непосредственно перед тем, когда форма будет удалена из экрана.
  - *Загрузка*. Происходит после события *Открытие* при загрузке данных в форму.
  - *Выгрузка*. Происходит перед событием *Закрытие* при выгрузке данных.
  - *Изменение размера*. Происходит при изменении размеров формы/отчета, а также при их открытии.

- *События фокуса*. Происходят при получении и потери фокуса и при активации/деактивации элемента управления или самой формы:
  - *Вход*.
  - *Выход*.
  - *Получение фокуса*. Происходит, когда форма или элемент управления получает фокус, т.е. становится подсвеченной либо выделяется другим цветом.
  - *Потеря фокуса*. Происходит, когда форма или элемент управления теряет фокус.
  - *Включение*. Возникает, когда форма/отчет становится активной. Применяется для связывания появления панели инструментов или т.п. с появлением формы.
  - *Отключение*. Возникает, когда активной становится другая форма/отчет.
- *События клавиатуры*. Происходят при нажатии на клавиши клавиатуры:
  - *Клавиша вниз*. Происходит при нажатии клавиши. Имеет два параметра KeyKode и Shift. По значениям этих параметров можно определить, что было нажато.
  - *Клавиша вверх*. Происходит при отпускании клавиши. Имеет те же параметры.
  - *Нажатие клавиши*. Генерируется непрерывно, когда клавиша нажимается и остается нажатой.
- *События мыши*. Происходят при нажатии клавиш мыши или при ее перемещении:
  - *Нажатие кнопки*. Происходит при нажатии левой клавиши мыши. Это событие вызывается также автоматически при нажатии клавиши Enter для имеющей фокус кнопки или выборе элемента из списка.
  - *Двойное нажатие кнопки*. Происходит после двойного нажатия клавиши мыши.
  - *Перемещение указателя*. Генерируется непрерывно, пока происходит перемещение мыши. Система определяет границы объектов и генерирует это событие поочередно для всех объектов, в которые попадает указатель мыши. Имеет параметры Button, Shift, X и Y. Параметры Button и Shift служат для определения состояния нажатых кнопок мыши и клавиш Alt, Shift, Ctrl.
  - *Колесико мыши, Кнопка вниз* и *Кнопка вверх*. Дополнительные события мыши.

Если форма имеет хотя бы один доступный элемент управления, то при ее открытии возникает следующая последовательность событий: *Открытие* → *Загрузка* → *Изменение размера* → *Включение* → *Текущая запись* → *Вход* (э)→ *Получение фокуса* (э). Буква э в скобках означает, что событие относится к элементу управления, а не к форме. В следующих примерах буква *ф* в скобках будем обозначать событие, происходящее для самой формы, если это и так не ясно из контекста. Соответственно, при закрытии формы: *Выход* (э)→ *Потеря фокуса* (э) → *Выгрузка* → *Отключение* → *Закрытие*. При переключении между двумя открытыми формами цепочка событий следующая: *Потеря фокуса* (э1) → *Отключение* (ф1) → *Включение* (ф2)→ *Вход* (э2)→ *Получение фокуса* (э2), а при переключении между двумя элементами управления одной формы: *Выход* (э1) → *Потеря фокуса* (э1) → *Вход* (э2)→ *Получение фокуса* (э2).

При изменении данных в элементе управления при каждом нажатии клавиши клавиатуры возникает следующая цепочка событий: *Клавиша вниз* → *Нажатие клавиши* → *Внесены изменения* → *Изменение* → *Клавиша вверх.* При последующем переходе к другому элементу управления дополнительно происходят: *До обновления* → *После обновления* → *Выход* → *Потеря фокуса*.

При добавлении записи в форму (производится ввод первого символа в поле пустой записи) цепочка событий следующая: *Текущая запись* → *Вход* (э)→ *Получение фокуса* (э) → *До вставки*→ *Изменение* (э) → *До обновления* (ф) → *После обновления* (ф) → *После вставки*. При удалении: *Удаление* → *До подтверждения Del* →

После подтверждения Del → *Текущая запись* → *Вход* (э)→ *Получение фокуса* (э). Последние три события происходят вследствие того, что при удалении фокус переходит на следующую запись.

1.6.3. Макросы

Макрос представляет собой набор из одной или более макрокоманд, которые выполняются либо последовательно, либо в порядке, заданном определенными условиями. Каждая макрокоманда имеет собственное имя и один или несколько аргументов. Использование макросов оправдано тем, что для их создания нет необходимости в серьезном изучении языка программирования. Использование макрокоманд интуитивно понятно, поскольку название макрокоманды фактически совпадает с выполняемым ей действием. Например, *ЗадатьЗначение*, *ОткрытьФорму*. Но программирование с помощью макросов имеет ряд серьезных ограничений. Набор макросов ограничен, макросы не могут возвращать значения, с их помощью невозможна обработка ошибок и т.д.

С помощью макросов можно выполнять следующие действия:

1. Открывать и закрывать любые объекты вашей БД в любом режиме. Для форм дополнительно можно ограничивать число выводимых записей по некоторому условию. Для отчетов можно запускать их на печать и экспортировать в формате TXT, RTF, XLS и HTML;
2. Выполнять запросы и задавать их параметры с помощью элементов управления любой открытой формы;
3. Запускать другие макросы и процедуры VBA, прерывать выполнение макросов и отменять событие, вызвавшее макрос, и даже выходить из приложения;
4. Устанавливать значения любых свойств любого элемента управления формы или отчета;
5. Осуществлять поиск записей в базовой таблице или запросе форм и отчетов и переходить к любой из них;
6. Переопределять либо создавать системное меню и выполнять любую команду любого меню Access;
7. Перемещать, изменять размеры, сворачивать, скрывать и т.д. формы и отчеты внутри рабочего окна Access. Можно также передавать фокус любой форме и любому элементу управления в ней;
8. Выводить на экран информационные сообщения либо выдавать звуковые сигналы;
9. Переименовывать, удалять, импортировать/экспортировать объекты и внешние данные;
10. Запускать другие приложения, осуществлять обмен информацией с помощью механизма DDE или буфера обмена. Можно даже передать последовательность нажатий клавиш в открытое приложение.

Макросы создаются с помощью конструктора макросов (смотри Рис. 11). Макросы можно также объединить в группы для уменьшения их общего числа и лучшей ориентации среди них. Для создания группы макросов необходимо вывести столбец *Имя макроса* в окне конструктора макросов с помощью соответствующей команды меню. Чтобы сослаться на макрос, входящий в группу, требуется указать имя группы макросов и через точку имя макроса. Можно задать условие на выполнение определенных макрокоманд. Для этого необходимо добавить столбец *Условие* в окно конструктора макросов с помощью соответствующей команды меню. Создание условий ничем не отличается от задания условий на значения для форм и таблиц.

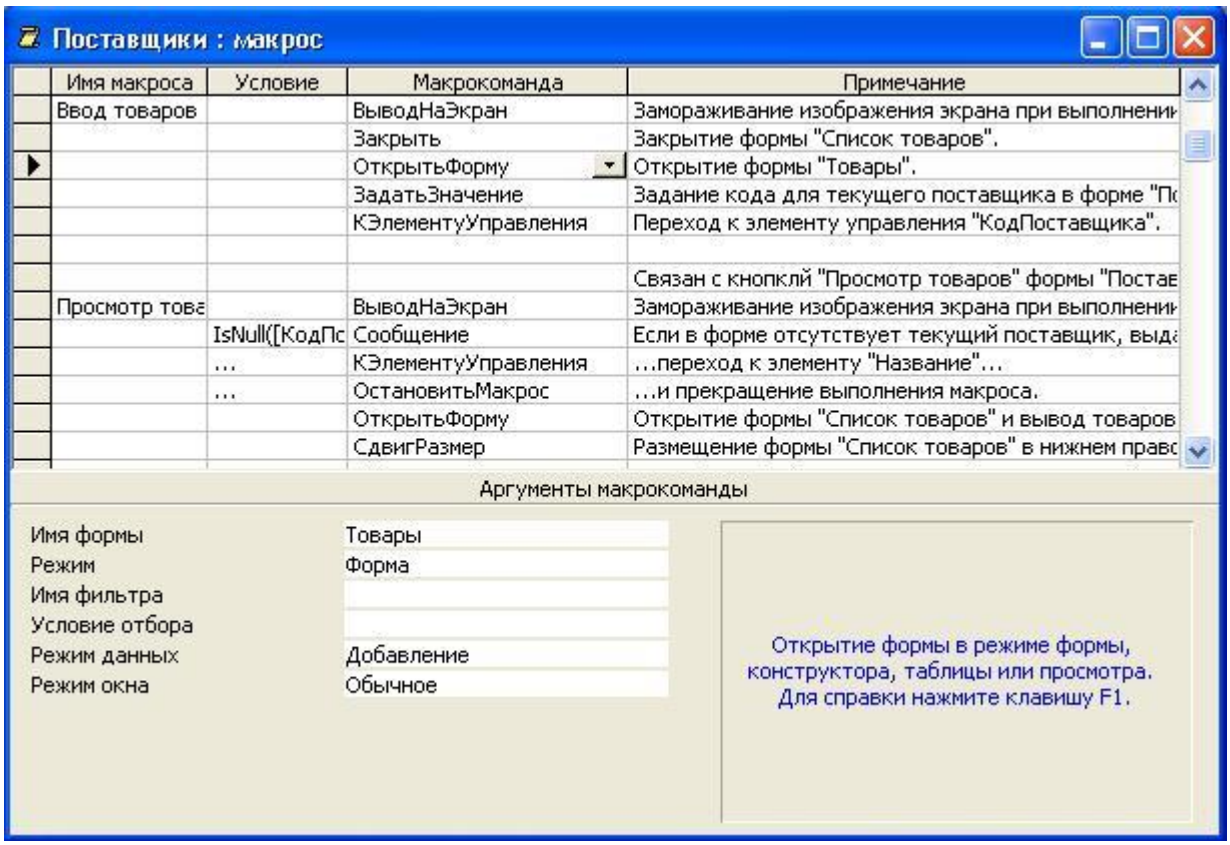


Рис. 11. Окно конструктора макросов

Если условие действует на группу макрокоманд, то чтобы не повторять одно и то же условие для нескольких последовательно идущих макрокоманд, можно вместо условия поставить многоточие “…”. Если условие принимает значение *Истина*, то Access запускает макрокоманду в этой строке и все макрокоманды в расположенных ниже строках, которые имеют многоточие вместо условия. Затем продолжается выполнение дополнительных макрокоманд внутри данного макроса, пока не встретится макрокоманда с другим условием, другой макрос или конец группы макросов. Если условие принимает значение *Ложь*, то Access игнорирует макрокоманду данной строки и всех следующих за ней макрокоманд с многоточием и переходит к строке, содержащей новое условие или не содержащей условий вообще. Для макросов невозможно задание альтернативного ветвления, поэтому для создания макроса с альтернативным ветвлением необходимо вначале определить группу макрокоманд с одним условием, а затем сразу же другую группу с альтернативным условием.

Макросы можно запустить на выполнение из другого макроса или процедуры VBA, но чаще всего макросы привязываются к определенным событиям в формах и отчетах и служат для их обработки. Для связи макроса с определенным событием необходимо задать имя макроса в строке обработки этого события в бланке свойств того объекта, события которого хотите обработать. Почти все макрокоманды имеют аргументы, т.е. требуют задания дополнительных данных, определяющих способ их выполнения. Например, с событием *Нажатие кнопки* элемента управления *Кнопка* можно связать макрос, содержащий макрокоманду *ОткрытьФорму*. Тогда после нажатия кнопки произойдет открытие формы, определенной в аргументе макрокоманды.

Существует зарезервированное имя макроса AutoExec для автоматического выполнения при открытии базы данных. Т.е. если присвоить это имя какому-либо макросу, то он начнет выполняться сразу же после запуска БД. Если же необходимо отключить его выполнение, то при открытии БД надо удерживать нажатой клавишу Shift. При одновременном задании параметров запуска вначале происходит обработка действий, определенных в окне *Параметры Запуска*, а затем макроса AutoExec.

Почти все макросы можно продублировать с помощью функций VBA. Поэтому, начиная с Access 97, фирма Microsoft рекомендует использовать программирование в VBA, так как это ускоряет работу с БД и предоставляет дополнительные возможности и преимущества.

1.6.4. Модули

Модули являются объектом базы данных, объединяющим одну или несколько процедур либо функций, написанных на языке Visual Basic for Applications (VBA). Модули следует использовать вместо макросов в следующих случаях:

- Необходима обработка ошибок;
- Необходимо возвращать и передавать параметры;
- Необходимо создание общих функций;
- Необходимо во время работы (динамически) создавать новые объекты БД;
- Необходим вызов функций Windows или обмен данными через OLE;
- Необходима функциональность, недостижимая с помощью макросов;
- Необходимо производить действия более чем с одной БД одновременно.

Входящие в модуль процедуры/функции объединены общей областью декларации. В ней устанавливаются определения и правила, являющиеся общепринятыми для модуля, а также переменные, общие для всех процедур/функций модуля. Процедуры и функции могут иметь аргументы. Отличием функции от процедуры является то, что первая может в точку вызова вернуть некоторое значение, а вторая – нет. Функции и процедуры, записанные в модулях, доступных через закладку *Модули* окна базы данных (стандартные или общедоступные модули), являются доступными для всех объектов БД. Их можно использовать в следующих местах:

- во всех процедурах и других функциях вашей БД;
- в выражениях вычисляемых полей в формах, объектах и запросах;
- в выражениях, определяющих условия выполнения макрокоманд в макросах.

Кроме общедоступных модулей в каждой форме и отчете имеются свои локальные (иначе привязанные) модули. Программы локальных модулей загружаются динамически вместе с объектом, их содержащим. Процедуры в них являются локальными по отношению к данному объекту и используются преимущественно для обработки событий. Имя таких процедур состоит из имени объекта и имени события и не подлежит изменению. Если требуется обработка некоторого события, то проще всего создать процедуру его обработки из окна свойств того объекта, события которого необходимо обработать. Для этого надо нажать кнопку с многоточием в строке обрабатываемого события и выбрать построение программ. После этого автоматически будет создано тело процедуры, связанной с определенным событием.

При наборе команд в окне модуля Access проверяет правильность написания инструкций VBA. Перед первым использованием процедуры Access автоматически ее компилирует и сохраняет в псевдоисполняемом коде. Если некоторая форма содержит очень много кода внутри своего модуля, то рекомендуется большую часть его выделить в виде функций и поместить в отдельный стандартный модуль. Тогда загрузка самой базы будет происходить медленнее, но форма будет открываться значительно быстрее.

1.6.5. Visual Basic for Applications

Visual Basic for Applications (VBA) является интерпретируемым языком, т.е. его инструкции интерпретируются каждый раз при выполнении программы. Этот язык является общим для всех приложений MS

Office; на нем можно писать с равным успехом программы как для Access, так и для Word либо Excel. Вначале давайте кратко рассмотрим синтаксис языка VBA.

Объявление переменных

Переменные в Access можно объявлять явно и неявно. Имена переменных должны начинаться с буквы и не могут содержать пробелов или других знаков пунктуации, кроме знака подчеркивания. Также нельзя использовать зарезервированные слова. Число символов в имени не должно превышать 225. При неявном задании переменной ей автоматически присваивается тип Variant. Это универсальный тип данных. Переменная такого типа может принимать любые значения (этот тип по умолчанию имеют все данные, непосредственно полученные из таблиц и запросов) и дополнительно пустое значение *Null*. Проверить, содержит ли переменная какое-либо значение, можно с помощью функции *IsNull*().

Переменные определенных типов можно задавать либо с помощью суффиксов (определенного символа, записываемого сразу после имени переменной), либо с помощью явного объявления типа. Имеются следующие типы данных:

Byte	Currency @
Integer %	String $
Long &	Date
Single !	Boolean
Double #	Variant

Для явного описания переменной используется оператор Dim.

Dim <имя переменной> As <тип переменной>.

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

Строки могут быть постоянной и переменной длины. Оператор Dim <имя переменной> As String объявляет строку переменной длины, а оператор Dim <имя переменной> As String*<количество символов> - фиксированной длины. Например:

Dim str1 As String Dim str2 As String*20.

Можно ввести требование, чтобы все переменные были описаны явно. Для этого в разделе описаний этого модуля необходимо задать декларацию Option Explicit. Тогда Access будет автоматически обнаруживать ошибки в написании имен переменных и контролировать совпадение типов переменной и назначаемых ей значений.

Переменные имеют определенную область видимости. Существует 4 уровня видимости:

1. Локальный уровень или уровень процедуры. Переменные существуют только внутри процедуры, где они описаны и используются;
2. Уровень формы (отчета). Переменные описываются в разделе описаний модуля формы (отчета) и являются доступными только для процедур этой формы (отчета) в то время, когда форма (отчет) открыта;
3. Уровень модуля. Переменные описываются в разделе описаний стандартного модуля и доступны для всех процедур модуля при открытой БД;
4. Глобальный уровень. Переменная записывается в разделе описаний модуля при помощи инструкций Public или Global.

Если пытаться использовать неявно заданную переменную в то время, когда она недоступна, то Access взамен ее создаст новую переменную с тем же именем, что может привести к трудно обнаруживаемым логическим ошибкам. Это может служить дополнительным поводом для явного задания всех переменных.

Все переменные имеют определенное время жизни. Для локальных переменных время жизни определяется временем использования кода. Каждый раз при новом вызове процедуры переменной приписывается либо *Null*, либо 0, либо пустая строка в зависимости от типа. Для создания переменных со временем жизни, равным времени жизни приложения, используется инструкция Static. Тем не менее, статическая переменная не может быть использована в других процедурах. Изменяется лишь время ее жизни, а не область видимости. 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. Все переменные в определенной процедуре будут статическими, если перед самой процедурой поставить инструкцию Static.

С помощью ключевого слова Dim также можно определять многомерные массивы. Размерность и число элементов массива определяется внутри скобок после имени переменной. Например:

Dim myArray(20) As String*10

Dim myArray(5, 5) As Double или Dim myArray(3, 3, 3) As Double

Нумерация элементов массива начинается с 0, поэтому для первого примера будет зарезервирована память для массива, состоящего из 21 элемента, для второго – 6 на 6. При объявлении массива можно определять не только верхнюю границу массива, но и нижнюю. Например, Dim my Array(1 то 20) As String.

В VBA допускается использование динамических массивов. Для этого при описании переменной число элементов массива опускается, а затем применяется инструкция ReDim для объявления реального размера массива. При этом все элементы массива обнуляются. Для сохранения значений элементов массива применяется инструкция Preserve.

Dim dynArray( ) As Double

ReDim Preserve dynArray (5, 2, 4)

Во время выполнения программы нижнюю и верхнюю границы массива можно определить с помощью функций LBound() и UBound() соответственно. Удалить массив из памяти можно с помощью оператора Erase.

Константы в VBA описываются с помощью ключевого слова Const. Вместе с объявлением переменной необходимо также произвести ее инициализацию.

Const pi As Double = 3.1415

Кроме переменных вышеприведенных типов в VBA можно задавать объектные переменные, применяющиеся для хранения ссылок на объекты Access. Для каждой коллекции основных объектов в Access имеется соответствующий ей объектный тип, а также общий тип Object, принимающих ссылки на любые объекты. Например:

Dim myObject As Object – переменная любого объектного типа,

Dim myControl As Control – переменная типа элемента управления,

Dim myForm As Form – переменная типа формы.

Присваивать конкретные значения объектным переменным можно с помощью инструкции Set.

Set myForm = Forms![Моя Форма].

Пользователь может также создавать свой тип данных на основе существующих. Пользовательский тип данных вводится между ключевыми словами Type…End Type.

```
Type tPhone
    number As String*15
    type As String*10
End Type
```

Обращение к полю пользовательского типа производится через операцию “.”.

```
Dim tel As tPhone
tel.number = 3334455
tel.type = "Мобильный".
```

Заполнить объектные переменные, содержащие много свойств, а также переменные пользовательского типа данных можно с помощью инструкции With … End With.

```
Dim myForm As Form
Set myForm = Forms![Имя формы]
With myForm
   .Top=1000
   .Left=1000
   .Width=5000
   . Height=4000
End With.
```

Создание комментариев

Любой текст, следующий за символом ‘, воспринимается как комментарий.

Процедуры и функции

Основными компонентами программ на VBA являются процедуры и функции. Они представляют собой фрагменты программного кода, заключенные между операторами Sub и End Sub или между Function и End Function. В общем случае процедуры и функции записываются следующим образом:

```
Sub <имя процедуры< [(<аргументы>)]
   <операторы>
End Sub
Function <имя функции> [(<аргументы>)] As <тип возвращаемого значения>
    <операторы>
   <имя функции> = <возвращаемое значение>
End Function
```

Список аргументов разделяется запятыми. Функция отличается от процедуры тем, что ее имя выступает также в качестве переменной и используется для возвращения значения в точку вызова функции. Для вызова процедуры из другой процедуры или функции используется инструкция Call. Вначале идет имя процедуры, а затем в скобках список фактических значений ее аргументов. Процедуры можно также вызывать просто по их имени без использования инструкции Call. В этом случае список аргументов не заключается в скобки. Функции вызываются так же, как и процедуры, но гораздо чаще они вызываются по их имени с заключенным в скобки списком фактических значений аргументов в правой части оператора присваивания. Например:

```
Call mySub("Ландера", 4, i+1)
mySub "Ландера", 4, i+1
total_price = myFunc([Цена]*[Количество], [Доставка])
```

Допускается два различных способа передачи переменных процедуре или функции: по ссылке или по значению. По умолчанию переменные передаются по ссылке (объявление ByRef). При изменении значения переменной внутри процедуры это изменение остается и при выходе из этой процедуры. Переменные можно передавать и по значению. Тогда изменение значения такой переменной не будет воздействовать на значение переменной, переданной в процедуру или функцию. Для этого используют объявление ByVal. В общем виде

объявление аргументов производится следующим образом ByVal/ByRef <имя аргумента> [( )] [As <tun>]. Например:

```
Sub mySyb(srteet As String, ByVal building As Integer, ByRef apt As Byte)
   Forms![Клиенты]![Адресс] = srteet & " " & building & ", " & apt
End Sub
Function myFunc(full_price As Currency, shipment As Single) As Currency
  myFunc = full_price * 1.2 + shipment
End Function
```

Для принудительного выхода из процедуры и функции используются соответственно Exit Sub и Exit Function.

Управление выполнением программы

Управление выполнением программы достигается за счет оператора безусловного перехода, операторов ветвления (условных операторов) и циклов. Условные операторы выполняют группу команд в зависимости от условия. К ним относится оператор If и Select Case. В простейшем виде оператор If можно записать в одну строку.

```
If <условие> Then <оператор>
```

Более сложный вариант использования оператора If приведен ниже:

```
If <условие> Then
   <операторы>
Else
    <операторы>
End If
```

Для задания выбора действий на основе проверки целой группы условий используется расширенный вариант записи оператора If.

```
If <условие> Then
   <операторы>
Else If <условие 2> Then
    <операторы>
Else
    <операторы>
End If
```

Если выбор действий зависит от различных значений одного и того же выражения, то вместо вложенного оператора If предпочтительнее использовать оператор Select Case … End Select. Этот оператор определяет, является ли выражение истинным, а также оценивает, заключены ли в определенных пределах значения этого выражения.

```
Select Case <имя переменной>
   Case <выражение 1> [ , <выражение 2>, …]
```

(Операторы, выполняемые, если значения переменной удовлетворяют или выражение 1, или выражение 2, или …)

```
   [Case <выражение 3> To <выражение 4>]
```

(Операторы, выполняемые, если значение переменной находится в диапазоне, определяемом выражениями 3 и 4)

```
   [Case Is <выражение отношения>]
```

(Операторы, выполняемые, если значение переменной удовлетворяет выражению отношения.)

[Case Else <операторы, выполняемые, если не было выполнено ни одно из вышеперечисленных условий>]

End Select

Любой строке в программе на VBA можно присвоить метку Метка:. Перейти на эту строку программы можно с помощью оператора безусловного перехода GoTo <метка>. Однако операторы GoTo нарушают стиль структурного программирования, и они применяются только для обработки ошибок в приложении.

Операторы цикла в VBA различаются на две основные группы: циклы с перечислением (For … Next) и циклы с условием (Do … Loop).

Оператор For…Next повторяет тело цикла заданное число раз.

For <счетчик> = <начальное значение> To <конечное значение> [Step <приращение>]

<операторы>

Next <счетчик>

Для выхода из цикла используется оператор Exit For.

Операторы циклов Do While … Loop, While … Wend являются синонимами. While … Wend оставлен для совместимости со старыми версиями. Эти операторы повторяют тело цикла, пока условие принимает значение True.

Do While <условие [=True]>

<операторы>

Loop

Оператор Do Until … Loop выполняет тело цикла до тех пор, пока не будет выполнено условие

Do Until <условие [< > True]>

<операторы>

Loop

Чтобы обеспечить выполнение операторов тела цикла по крайней мере один раз, можно использовать следующие варианты этих циклов.

Do

<операторы>

Loop While <условие [=True]>

или

Do

<операторы>

Loop Until <условие [=False]>

Для выхода из этих циклов используется оператор Exit Do.

Для построения итератора по всем элементам массива или коллекции используется инструкция For Each

For Each <элемент> In <коллекция>

<операторы>

Next <элемент>

где <элемент> – это переменная, используемая для ссылки на элементы семейства объектов. Следующий пример заполнит выпадающий список элемента управления ПолеСоСписком1 названиями всех элементов управления, расположенных в форме

Dim ctrl As Control

For Each ctrl In Form

ПолеСоСписком1.AddItem ctrl.Name

```
Next ctrl
```

1.6.6. Обработка ошибок на этапе выполнения

Как тщательно не проверялся бы код, на этапе его выполнения неизбежно возникают ошибки. Попытка деления на ноль – это типичный пример ошибки времени выполнения. В Access обработка ошибок выполняется с помощью инструкции On Error. Имеется три вида инструкций On Error:

1. On Error GoTo Метка – осуществляет переход на строку с меткой (Метка) при возникновении произвольной ошибки на этапе выполнения. Обычная часть кода, обрабатывающая ошибки, помещается в корпус процедуры. После обработки ошибки можно либо вызвать повторение той части кода, где произошла ошибка, либо проигнорировать ошибку и продолжить выполнение последующих инструкций. Для возвращения на строку с ошибкой используется ключевое слово Resume.
2. On Error Resume Next – игнорирует ошибку и продолжает выполнение последующих инструкций.
3. On Error GoTo 0 отключает обработку ошибок.

После обработки первой ошибки инструкция On Error GoTo Метка продолжает выполняться для всех последующих ошибок, пока не закончится выполнение данной процедуры либо обработка ошибок не будет явно отключена инструкцией On Error GoTo 0. Если нет обработки какой-то ошибки или обработка выключена, то при возникновении необработанной ошибки приложение сразу же выдаст сообщение об ошибке и прекратит работу.

Если процедура обработки некоторого события создается с помощью мастера, то он автоматически дополняет процедуру кодом обработки ошибок.

```
Private Sub MyID_DblClick(Cancel As Integer)
On Error GoTo Err_MyID_DblClick
    Текст процедуры
Exit_MyID_DblClick:
Exit Sub
Err_MyID_DblClick:
MsgBox Err.Description
Resume Exit_MyID_DblClick
End Sub
```

Объект Err содержит информацию об ошибках выполнения и обычно используется вместе со структурой Select Case, чтобы определить, какое действие предпринять в зависимости от кода ошибки.

```
Select Case Err
  Case 58 To 76
     Call FileError- процедура обработки ошибок работы с файлами
  Case 281 To 297
     Call DDEError  - процедура для обработки ошибок  DDE
  Case 340 To 344
     Call ArrayError - процедура ошибок массивов
End Select
Err=0  - отключение обработки необработанных ошибок.
```

1.6.7. Работа с объектами и коллекциями

В VBA можно работать с объектами и коллекциями Access, библиотекой доступа к данным DAO (Data Access Objects), библиотекой прямого доступа к данным баз данных ODBC (ODBC Direct), библиотекой доступа к данным ADO (ActiveX Data Objects) и другими библиотеками. В Access, DAO и ADO все объекты расположены внутри коллекций и доступны в VBA. Каждый объект имеет свойства и методы. Все библиотеки организованы в виде иерархии объектов. Объекты имеют коллекции (семейства) подчиненных объектов и т.д. В Access имеется 8 базовых объектов; их иерархия представлена на Рис. 12.

Объекты MS Access:

1. Application – активное приложение;
2. Control – элемент управления;
3. DoCmd – объект вызова макрокоманд в VBA коде;
4. Form – открытая форма;
5. Module – объект, ссылающийся на стандартные модули;
6. Reference – объект, содержащий ссылки на объекты;
7. Report – открытый отчет;
8. Screen – ссылка на экран;

Названия семейств формируются путем возведения в множественное число названия соответствующего объекта. В свою очередь большинство объектов имеют присоединенные коллекции свойств (Properties), а формы и отчеты – коллекции разделов и т.д. Так как все объекты в Access хранятся внутри иерархически связанных коллекций, то доступ к объекту на нижней ступени иерархии можно получить, указав все имена коллекций, разделенных точкой, начиная от корневого объекта. Например, Application.Forms("Заказы").Controls(0).Properties(0). Большинство коллекций, к примеру, коллекции форм и отчетов, являются глобальными. Тогда к объекту этой коллекции можно обращаться напрямую: Forms("Заказы") или Forms!Заказы.

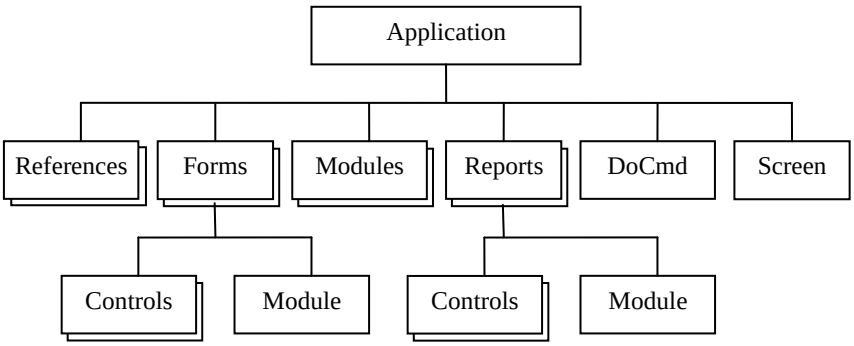


Рис. 12. Иерархия объектов Access

Поскольку библиотека DAO всегда поставляется с Access, рассмотрим ее структуру и основные методы более подробно (смотри Рис. 13).

Объекты библиотеки DAO:

Database – открытая база данных;

DBEngine – ссылка на Microsoft Jet (ядро БД);

Error – объект ошибок;

Field – поле в таблицах, запросах, динамических наборах и т.д.;

Index – индекс;

Parameter – параметр запроса;

QueryDef – сохранённый запрос;

Recordset – динамический набор данных;

Relation – связь между таблицами;

TableDef – сохраненная таблица;

Workspace – активная сессия.

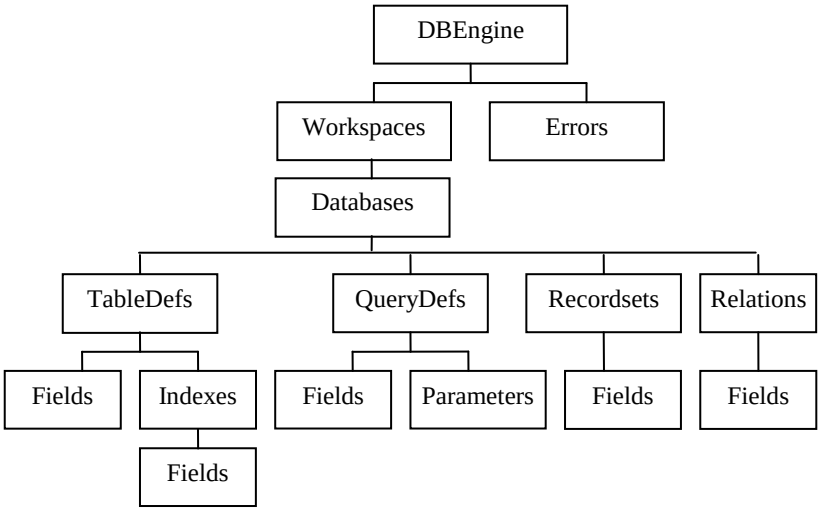


Рис. 13. Иерархия объектов DAO

В программах на VBA имеется набор свойств объектов, которые возвращают ссылки на подчиненные объекты:

Me – ссылка на активную форму или отчет (доступна в присоединенном модуле);

ActiveControl – ссылка на активный элемент управления;

ActiveForm – ссылка на активную форму (доступна в объекте Screen);

ActiveReport – ссылка на активный отчет (доступна в объекте Screen);

Application – ссылка на открытое приложение;

Parent – ссылка на родительский объект, т.е. на коллекцию;

DBEngine – возвращает ссылку на Application.

Работа с записями и полями

Применить фильтрацию, изменить значение какого-либо поля в базовом наборе данных либо даже сменить базовый набор можно непосредственно с помощью свойств самой формы и элементов управления. Например:

```
Me![Полная цена] = Me![Цена] * Me![Количество]
Me.Filter = "[Адрес] Like '*" & Me![ПолеУсловияПоиска] & "*'"
Me.FilterOn = True
```

Если есть желание читать и менять данные в некотором поле базового набора, но не видеть его на экране, то достаточно поместить элемент управления *Поле* на форму и задать свойство Visible в False. Но прямой доступ к базовому набору в Access невозможен. Можно лишь создать динамическую копию базового набора и синхронизировать все производимые действия с ней с базовым набором самой формы. Для прямого доступа к записям и полям используется объект Recordset. Имеются четыре типа Recordset объектов – *table, dynaset, snapshot* и *forwarrd-only*:

1. *Table*: Может быть создан только на основе существующей или присоединенной таблицы. Предоставляет доступ ко всем методам и свойствам таблицы, а также к индексам, что дает намного более быстрый метод поиска (метод Seek);
2. *Dynaset* - может быть создан на основе таблицы или запроса. Позволяет обновлять данные в многотабличных запросах и в запросах к внешним БД. Обновление *Dynaset* объекта приводит к автоматическому обновлению всех участвующих в нем таблиц;
3. *Snapshot* - создает статическую копию и существует только в то время, когда он создан. Последующие изменения таблиц на него не воздействуют;
4. *Forward-only* создает статическую копию с просмотром только в прямом порядке.

Для создания объекта типа Recordset используется метод OpenRecordset:

Set variable = database.OpenRecodset (source [type, options, lockedits]), или

Set variable = object.OpenRecodset ([type, options, lockedits]),

где database – это переменная типа Database; object – переменная типа TableDef или QueryDef; source – ссылка на объект типа TableDef или QueryDef; type – тип динамического набора (может принимать следующие значения: dbOpenTable, dbOpenDynaset, dbOpenSnspshot dbOpenForwardOnly); options может принимать следующие значения: dbAppendOnly, dbReadOnly, dbForwardOnly, …; lockedits – аргумент, определяющий разрешение конфликтов в многопользовательских БД (может принимать следующие значения: DbReadOnly, dbPessimistic, dbOptimistic). Например:

```
Dim db As Database
Dim rst As Recordset
Set db = CurrentDb()
Set rst = db.OpenRecordset("Клиенты", dbOpenDynaset)
```

Открыть Recordset можно и основываясь на переменной типа формы (допустим только для форм, основанных на таблице или запросе) с помощью метода RecordsetClone. Метод RecordsetClone создает динамический набор, основываясь на свойстве *Источник данных* для формы

```
Dim rstOrders As Recordset
Set rstOrders = Forms![Заказы].RecordsetClone
```

или просто Me.RecordsetClone

Recordset можно создать, также основываясь на строке SQL

```
Set rst = db.OpenRecordset("SELECT * FROM Товары WHERE Цена > 1000", dbOpenDynaset, dbReadOnly)
```

После завершения работы с динамическим набором его необходимо закрыть. Существуют два общих способа закрытия объектов в VBA. Первый заключается в вызове метода Close, второй – в присвоении соответствующей объектной переменной значения Nothing. Например, rst.Close или Set rst = Nothing.

Recordset имеет текущее приложение – current position. Для синхронизации текущего положения динамического набора с текущей записью формы можно использовать свойство Bookmark (необходимо всегда помнить, что при обращении к динамическому набору данные берутся именно из текущей записи). Для перемещения по динамическому набору имеется ряд методов: MoveFirst, MoseLast, MoveNext, MovePrevions, Move[n]. Например:

```
Dim rst As Recordset
  Set rst = Me.RecordsetClone
  rst.MoveNext
  Me.Bookmark = rst.Bookmark
```

Для определения начала и конца набора можно использовать свойства BOF (before of file – начало файла) и EOF (end of file – конец файла). Если в наборе нет записей, то BOF и EOF равны True. Если в наборе есть записи, то при открытии курсор обычно устанавливается на первой записи и BOF и EOF = False. Если курсор находится перед первой записью, то BOF = True, и если курсор находится после последней записи, то EOF = True.

Число записей в динамическим наборе можно получить с помощью свойства RecordCount. Это свойство возвращает реальное число записей только для динамического набора, основанного на таблице, т.е. имеющего тип dbOpenTable. После открытия динамических наборов любых других типов число записей в нем будет неизвестно до тех пор, пока курсор не будет перемещен на последнюю запись. Для непосредственной проверки на наличие в наборе записей лучше проверить свойство EOF. Если EOF будет равно True, то RecordCount также будет равен 0.

```
Public Function RecCount() As Long
Dim rstCount As Recordset
Dim dbs As Database
Set dbs = CurrentDB()
Set rstCont = dbs.OpenRecordset("Заказы")
If rstCount.Eof Then
   RecCount = 0
Else
   rstCount.MoveLast
   RecCount = rstCount.RecordCount
End If
rstCount.Close
Set dbs = Nothing
End Function
```

Поиск определенной записи

Для наборов, основанных на таблице, можно использовать метод Seek, который ищет запись, основываясь на сравнении данных некоторого индекса с заданными значениями (самый быстрый метод). Для примера запишем программу, которая будет проверять дублирование значений ключевого поля таблицы *Клиенты* перед сохранением записи. Для проверки результатов поиска в объекте Recordset имеется свойство NoMatch. Если NoMatch равно False, то запись с заданными условиями поиска отсутствует в наборе.

```
Dim rst As Recordset
Set rst = CurrentDb.OpenRecordset("Клиенты", dbOpenTable)
rst.Index = "PrimaryKey"
rst.Seek "=", Me![КодКлиента]
If Not rst.NoMatch Then
   MsgBox  "Клинт с таким табельным номером уже существует в базе"
   DoCmd.GoToControl "КодКлиента"
End If
rst.Close
```

Метод Seek всегда начинает поиск с начала набора, поэтому поиск по одному и тому же условию будет всегда приводить к одной и той же записи. Для поиска по динамическим наборам остальных типов можно применять также методы FindFirst, FindLast, FindNext и FindPrevions. Например, подсчитаем количество заказов определенного клиента:

```
intCount = 0
rstOrders.FindFirst "КодКлиента=" & Me![КодКлиента]
Do While rstOrders.NoMatch
   rstOrders.FindNext "КодКлиента=" & Me![КодКлиента]
   intCount = intCount + 1
Loop
```

Для поиска записей можно также применять сортировку и фильтрацию. Для сортировки записей лучше всего открыть новый динамический набор, основанный на запросе SQL с оператором ORDER BY. Для фильтрации можно задать свойство Filter объекта Recordset и затем обновить набор с помощью метода OpenRecordset. Ниже приведен текст программы, осуществляющей фильтрацию произвольного набора:

```
Function FilterField(rstTemp As Recordset, strField As String, strFilter As String) As Recordset
   rstTemp.Filter = strField & " = '" & strFilter & "'"
   Set FilterField = rstTemp.OpenRecordset
End Function
```

Обновление динамических наборов

Обновление данных возможно только при работе с динамическими наборами типов dbOpenTable и dbOpenDynaset. Для редактирования некоторой записи необходимо вначале установить курсор на ней, перевести динамический набор в режим редактирования (метод Edit), а затем сохранить изменения с помощью метода Update.

```
rst.Edit
rst![Товар] = "Pentium 100"
rst![Цена] = 100
rst.Update
```

Метод Cancel отменяет внесенные изменения. Для удаления текущей записи существует метод Delete. Этот метод удаляет запись без выдачи каких-либо предупреждений и не меняет положения курсора. Для перехода на следующую запись необходимо вызвать метод MoveNext. Для добавления новой записи служит метод AddNew. После заполнения новой записи значениями ее необходимо сохранить с помощью метода Update, потому как если после добавления новой записи перейти к другой записи без сохранения или просто закрыть динамический набор, то добавляемая запись будет потеряна. Следующий пример изменяет значения полей *Цена* и *Количество* таблицы *Заказы*, отфильтрованной для определенного клиента.

```
Dim rst As Recordset
Dim dbs As Database
Set dbs = CurrentDb()
Set rst = dbs.OpenRecordset("Заказы", dbOpenDynaset)
rst.Filter = "[КодКлиента]=" & Me![Код клиента]
Set rst = rst.OpenRecordset
If Not rst.EOF Then
  With rst
    Do Until .EOF
      .Edit
      ![Цена] = rst![Цена] * 1.2
      ![Количество] = ![Количество] * 2
      .Update
```

```
      .MoveNext
    Loop
    .Close
  End With
End If
dbs.Close
```

Работа с полями

Получить доступ к значениям и свойствам полей некоторой таблицы можно, открыв соответствующий динамический набор. Коллекция полей Fields является коллекцией по умолчанию для объекта типа Recordset. Тогда получить доступ к некоторому полю можно по его имени или по индексу в коллекции, например: rst.Fields(Field1”), rst.Fields(4) или rst![Field1]. После получения ссылки на объект типа Field можно читать и изменять его свойства, например:

```
Dim fld As Field
fld.DefaultValue = 1
fld.ValidationRule = "BETWEEN 1 AND 1000"
fld.Value = 100
```