Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра информационных технологий автоматизированных систем

О. В. Герман, Т. В. Тиханович

БАЗЫ И БАНКИ ДАННЫХ

Лабораторный практикум для студентов специальности
«Автоматизированные системы обработки информации»
дневной и дистанционной форм обучения

Минск БГУИР 2009

УДК 004.65(075.8)
ББК 32.973.26 – 018.2я73
Г38

Р е ц е н з е н т:

доцент кафедры ИСИТ БГТУ, канд. физ.-мат. наук
Н. И. Гурин

Герман, О. В.

Г38 Базы и банки данных: лаб. практикум для студ. спец. «Автоматизированные системы обработки информации» днев. и дистанц. форм обуч. / О. В. Герман, Т. В. Тиханович. – Минск : БГУИР, 2009.– 68 с.: ил.

ISBN 978-985-488-410-3

Практикум содержит краткие теоретические сведения и практические задания для проведения четырех четырехчасовых лабораторных работ по курсу «Базы и банки данных». Материал охватывает основные разделы теоретического курса дисциплины и позволяет практически освоить важнейшие аспекты программирования баз данных в средах Visual FoxPro 9.0, MS SQL Server 2005 (2000), VBA. Достаточно полно представлены сведения по языку SQL (T-SQL).

Пособие предназначено для студентов, магистрантов, аспирантов и всех тех, чья деятельность связана с изучением и реализацией современных баз данных.

УДК 004.65(075.8)
ББК 32.973.26 – 018.2я73

ISBN 978-985-488-410-3

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА №1

ЯЗЫК SQL

1.1 ЦЕЛЬ РАБОТЫ

1) изучение основных понятий современных баз данных;
2) изучение основ языка SQL (Structured Query Language).

1.2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.2.1 Основные понятия теории баз данных

1.2.1.1 Таблицы и атрибуты

Современную реляционную базу данных (БД) можно рассматривать как совокупность взаимосвязанных таблиц. Для первоначального знакомства введем в рассмотрение простую таблицу 1.1, содержащую информацию о складе.

Таблица 1.1 – Склад

Код	Название	Цена	Количество
11	Бумага	5000	500
13	Клей	1000	20
21	Тушь	3000	100
34	Краски	10000	40
37	Перья	5000	500
43	Бумага	8000	100

Такая таблица на языке БД представляет собой **сущность** или **отношение**, **записи** таблицы – **экземпляры сущности**, а поля таблицы – **атрибуты**. Значения полей, например, «бумага», «клей», «тушь» и т.д. представляют собой значения атрибута. Совокупность всех таких значений определяет **домен** значений атрибута (иначе – **тип данных** атрибута). В последующем заголовок таблицы будем также называть **схемой отношения**.

Реляционная БД может содержать не одну, а много таблиц. Заголовки всех таблиц образуют **схему БД**.

Важно отметить, что некоторые атрибуты или подмножества атрибутов являются уникальными. Например, в таблице 1.1 уникален Код товара. Следовательно, если нужно найти целиком запись, соответствующую, например, коду 37, то значение 37 можно использовать как ключ.

Ключевым атрибутом является такой атрибут, который уникальным образом определяет запись таблицы. Однако таблица может не содержать одиночных ключевых атрибутов вовсе и только определенные сочетания атрибутов могут уникально идентифицировать запись в таблице. Так, в таблице 1.1 атри-

бут **Название** не является ключевым, но в сочетании с атрибутом **Цена** образует ключ.

Создавая свою БД, пользователь самостоятельно создает и ключевые атрибуты (ключи), причем из всех ключей он выделяет **первичный** (или главный) ключ. Ключ, не являющийся первичным, называется вторичным. Вторичных ключей может быть много, первичный ключ всегда единствен. Первичный ключ должен иметь уникальные значения.

Итак, ключи позволяют рассматривать функциональные зависимости на множестве атрибутов таблицы БД.

1.2.1.2 Отношения между атрибутами

Говорим, что атрибут **B функционально зависит** от атрибута **A**, если атрибут **A** уникально определяет атрибут **B**. Наличие функциональных зависимостей в таблице позволяет реализовать процедуру поиска записи по ключу. Функциональные зависимости могут быть разных типов: однозначные, многозначные, частичные, полные.

Зависимость атрибута **B** от ключа **Z** называется **полной**, если **B** не зависит функционально ни от какого подмножества **Z**, кроме него самого.

Зависимость атрибута **B** от ключа **Z** называется **частичной**, если **B** зависит от части ключа **Z**. Так, в **таблице 1.1** атрибут **Код** полностью зависит от ключа **Название-Цена**, но атрибут **Количество** зависит частично от ключа **Название-Цена**.

Если **Z** – ключевой атрибут, то таблицу можно проиндексировать по этому ключу. **Проиндексировать таблицу** – значит создать для нее индексный файл, который позволяет быстро выполнить поиск нужной информации по значению ключа.

1.2.1.3 Отношения между таблицами

Введем в рассмотрение таблицу 1.2.

Таблица 1.2 – Производители

Код	Название	Фирма	Город
11	Бумага	Папир	Борисов
13	Клей	Ориенталь	Минск
21	Тушь	20 Век	Минск
34	Краски	Колер	Витебск
37	Перья	Зотов и Ко	Смоленск
43	Бумага	Фантом	Орехово

Нетрудно видеть, что и таблица 1.1, и таблица 1.2 имеют общие колонки (колонки могут быть общими, даже если их названия в разных таблицах не совпадают). Наличие общих колонок позволяет установить связи между таблицами по этим колонкам. Такая связь может иметь тип 1:1, 1:M, M:1 и M:M. Наличие установленных связей между таблицами позволяет говорить об ограничениях

целостности, т. е. о таком отношении между таблицами, при котором изменение каких-либо атрибутов в одной таблице приводит к адекватному их изменению в другой таблице. Для того чтобы разобраться в этом подробнее, соединим наши таблицы по полю **Код** и будем считать таблицу 1.1 главной (родительской), а таблицу 1.2 – дочерней (рисунок 1.1).

Поле **Код** таблицы 1.1 является **внешним ключом** для таблицы 1.2, поскольку позволяет отыскивать записи во второй таблице. Поле Код в таблице 1.1 является **первичным ключом**. Когда для первичного ключа родительской таблицы имеется внешний ключ дочерней таблицы, то такая ситуация называется **ссылочной целостностью**. При наличии ссылочной целостности каждому значению внешнего ключа должен соответствовать первичный ключ родительской таблицы. Ограничение ссылочной целостности означает, что при попытке, например, ввести новое значение в поле внешнего ключа дочерней вершины выполняется проверка наличия такого же значения в поле первичного ключа родительской таблицы. Если значение ключа отсутствует в родительской таблице, то операция завершается неудачей.

Код	Название	Цена	Количество
11	бумага	5000	500
13	клей	1000	20
21	тушь	3000	100
34	краски	10000	40
37	перья	5000	500
43	бумага	8000	100

1:1

Код	Название	Фирма	Город
11	бумага	Папир	Борисов
13	клей	Ориенталь	Минск
21	тушь	20 Век	Минск
34	краски	Колер	Витебск
37	перья	Зотов и Ко	Смоленск
43	бумага	Фантом	Орехово

Рисунок 1.1 – Связь между таблицами по атрибуту Код

Важной операцией для связанных таблиц является их соединение по общему атрибуту. Как правило, при соединении двух таблиц в результирующую таблицу попадают какие-то столбцы из первой таблицы и какие-то из второй. Например, результатом соединения двух рассматриваемых таблиц по полю **Код**, в котором из первой таблицы выбираем атрибуты **Название** и **Цена**, а из второй – **Фирма**, будет таблица 1.3.

Видим, что записи в таблице 1.3 получаются из записей таблицы 1.1 и таблицы 1.2, которым соответствуют одинаковые значения поля **Код**.

Таблица 1.3 – Результат соединения таблиц по полю Код

Название	Цена	Фирма
Бумага	5000	Папир
Клей	1000	Ориенталь
Тушь	3000	20 Век
Краски	10000	Колер
Перья	5000	Зотов и Ко
Бумага	8000	Фантом

1.2.1.4 Операции над таблицами

Наряду с приведенной выше операцией соединения таблиц рассмотрим другие основные операции над таблицами:

- создание таблицы;
- удаление таблицы/ удаление записи;
- изменение таблицы;
- выбор записей из таблицы;
- расщепление таблицы.

Расщепление таблицы. Эта операция используется в первую очередь на этапе проектирования БД и связана с нормализацией БД. На вопросах нормализации останавливаемся коротко ниже.

Операция расщепления может приводить к аномалии обратного соединения. Рассмотрим следующий пример (таблица 1.4).

Таблица 1.4 – Пример таблицы до расщепления

Код	Товар	Фирма	Цена
11	Бумага	Ориенталь	5000
13	Бумага	Папир	7000
14	Краска	Норд	10000
27	Кисточки	Фокус	4000

Расщепим эту таблицу по полю **Товар**, (таблицы 1.5, 1.6).

Таблица 1.5 – Товар

Код	Товар
11	Бумага
13	Бумага
14	Краска
27	Кисточки

Таблица 1.6 – Фирма

Товар	Фирма	Цена
Бумага	Ориенталь	5000
Бумага	Папир	7000
Краска	Норд	10000
Кисточки	Фокус	4000

Видим, что операция расщепления сводится к выборке указанных столбцов.

1.2.1.5 Хранимые процедуры

Хранимые процедуры – это программы, которые сохраняются в БД наряду с собственно данными. Необходимость в хранимых процедурах связана с часто используемыми действиями, например, формированием резервной копии таблицы, собиранием статистической информации или выполнением табличных расчетов. Вызвать хранимую процедуру можно с любого удаленного сетевого компьютера, используя для этого требуемый синтаксис.

1.3 ЯЗЫК SQL

1.3.1 Создание базы данных, таблиц, курсоров и представлений

Язык **SQL** (Structured Query Language) является стандартным языком для работы с широким кругом БД. Этот язык все же несколько отличается при переходе от одной системы к другой, например, от MS ACCESS к SQL Server. В частности, язык SQL системы SQL Server содержит некоторое расширение, которое известно как T(ransact)-SQL.

Для создания БД используем следующий вариант SQL-запроса:

CREATE DATABASE mydb ON (FILENAME=’c:\msdev\db.mdf’, SIZE=2, NAME=’log1’)

Здесь FILENAME – полное имя файла с БД на диске;

NAME – логическое имя БД, которое по умолчанию совпадает с именем файла;

SIZE – исходный размер файла (указывается в мегабайтах).

Из других параметров, которые могут указываться в команде, отметим FILEGROWTH – задает размер приращения БД в процентах (%) или мегабайтах, MAXSIZE – определяет максимальный размер БД.

Заметим, что SQL не критичен к регистру букв.

Теперь приведем примеры команд для создания таблиц в существующей базе данных.

CREATE TABLE Склад (Код int NOT NULL CONSTRAINT fircst PRIMARY KEY,
Название varchar(40),
Цена int,
Количество int)

Данная команда создает таблицу 1.1. Имя создаваемой таблицы **Склад**. В команде указываются имена столбцов: **Код**, **Название**, **Цена**, **Количество**. Для поля **Код** указывается тип int, для поля **Название** – тип varchar, для поля

Цена и **Количество** – int. Поскольку мы хотим определить в качестве первичного ключа поле **Код**, то это выполняется с помощью ограничения

CONSTRAINT fircst PRIMARY KEY

Указывается, что ограничение имеет имя **fircst** и устанавливает данное поле в качестве первичного ключа (PRIMARY KEY). Объявить поле первичным ключом можно было и проще без использования ограничения:

CREATE TABLE Склад (Код int NOT NULL PRIMARY KEY,
Название varchar(40),
Цена int,
Количество int)

Преимущество использования ограничения состоит в том, что ограничение можно удалить, тем самым поле **Код** перестанет являться первичным ключом. Удаление нашего ограничения можно выполнить так:

DROP CONSTRAINT fircst

Типами данных в SQL являются следующие:

- int – целое число (32 разряда)
- smallint – целое число (16 разрядов)
- tinyint – байт (8 разрядов)
- float – вещественное число с плавающей точкой
- numeric(p,s) – число с фиксированной точкой с количеством цифр p и дробной частью из s цифр
- text – большой текст (длиной до 2 147 483 647 символов)
- money – денежный тип
- datetime – дата/время
- binary(n) – двоичный вектор длиной n
- varbinary(n) – двоичный вектор переменной длины с начальной длиной n
- image – изображение
- nvarchar – строка в кодировке Unicode (2 байта на символ) переменной длины
- nchar – строка в кодировке Unicode (2 байта на символ) фиксированной длины

Спецификатор NOT NULL не разрешает при вводе записи оставлять данное поле без значения.

В ограничениях можно указывать условия на значения данных с помощью спецификатора CHECK:

CREATE TABLE Склад (Код int NOT NULL PRIMARY KEY,
Название varchar(40),
Цена int ,
Количество int,
CONSTRAINT price CHECK (Цена > 0 AND Цена <1000000))

Проверяемое условие вставлено в ограничение с именем price. Оно состоит из двух простых условий:

Цена > 0;

Цена <1 000 000.

Некоторые столбцы можно создавать как вычисляемые выражения (для MS SQL Server). Например, создадим столбец НАЛОГ как выражение ЦЕНА*КОЛИЧЕСТВО*0.13:

CREATE TABLE Склад (Код int NOT NULL PRIMARY KEY,
Название varchar(40),
Цена int NOT NULL,
Количество int NOT NULL,
НАЛОГ numeric(10,3) AS ЦЕНА *Количество*0.13,
CONSTRAINT price CHECK (Цена > 0 AND Цена <1000000))

Вычисляемый столбец определяется как выражение, помещаемое вслед за ключевым словом AS.

Рассмотрим создание триггеров.

В системе FoxPro создание триггера выполняется таким образом (пример):

CREATE TRIGGER ON Склад FOR INSERT AS Price<100

Этот триггер не именован и задает условие проверки на вставляемую запись в виде Price < 100. В качестве условия можно записывать более сложные функции. В MS SQL Server синтаксис команды создания триггера более широкий:

CREATE TRIGGER mytrig ON Склад FOR INSERT
AS
IF Price<0
print “Нельзя вводить отрицательную цену”
Return

В этом примере создается триггер с именем mytrig, который реагирует на ввод новой записи в случае Price < 0 выдачей сообщения. Вместо ключевых слов For Insert можно указывать For Update или For Delete. Вариант For Update соответствует обновлению текущей записи, а For Delete – удалению текущей записи. Для возврата значения из триггера следует использовать команды print или RaiseError:

CREATE TRIGGER mytrig ON Склад FOR Update
AS
IF Price<0
RaiseError(‘Нельзя вводить отрицательную цену’,1,10)
Return

Теперь обратимся к курсорам. **Курсоры** – это временные таблицы с записями в оперативной памяти. Достаточно часто курсоры получают вследствие выборки данных по команде выборки. В связи с тем, что команда выборки под-

робно еще не рассматривалась, придется использовать ее с минимальными объяснениями. Рассмотрим вариант использования курсоров в языке Foxpro:

```
CLOSE ALL
Open DataBase mydb
CREATE CURSOR mycur (name c(4), group n)
b1 = SQLConnect (“Connect1”)
if b1>0
   messageBox(“OK”)
   s=”Select * from Db1!stud”
   =SQLSetProp(b1,’asynchronous’,.F.)
i= SQLPrepare(b1,s,”mycur”)
y=SQLEXEC(b1,s,”mycur”)
  thisform.text1.value=mycur.name;
  SQLDisconnect(b1)
   else
      messageBox(“No connection exists”)
   endif
CLOSE ALL
```

Здесь объявлен курсор с именем **mycur**, состоящий из двух полей – **name** и **group**. Указывается в команде создания курсора, что поле **name** имеет символьный тип c(40) с максимальной длиной 40 символов, а поле **group** – числовой тип n. Поскольку курсор заполняется записями из некоторой таблицы, то названия полей, указанных при создании курсора, должны содержаться среди полей этой таблицы.

В данном примере курсор заполняется записями из сетевой базы данных с именем Db1. Для этого нужно предварительно установить соединение с этой базой данных. В языке Visual Foxpro для этих целей применяют некоторый диалект языка SQL, который называется языком сквозных SQL-запросов. Попытка установить соединение выполняется посредством команды

```
b1 = SQLConnect (“Connect1”)
```

Эта команда заносит в переменную b1 номер логического канала, если соединение выполнено успешно, или «-1», если соединение установить не удалось. Константа «Connect1» задает имя соединения, которое должно быть определено предварительно средствами системы Visual FoxPro. Если соединение успешно установлено (b1>0), то выдаем сообщение

```
messageBox(“OK”)
```

и формируем строку команды для выборки записей из таблицы **stud** базы данных Db1:

```
s=”Select * from Db1!stud”
```

В языке FoxPro имя базы данных отделяется от имени таблицы знаком восклицания. В то же время имя поля таблицы отделяется от имени таблицы точкой: mycur.name.

Команда «=SQLSetProp(b1,’asynchronous’,.F.)» задает свойство с именем ASYNCHRONOUS режима чтения данных из сетевой таблицы, равным FALSE (пишется .F.). Это означает, что основная программа будет дожидаться получения записей из сетевой базы данных. Команда «i=SQLPrepare(b1,s,”mycur”)» передает скомпилированную команду на выборку записей для ускорения последующего ее выполнения, хотя фактически для выборки записей служит следующая команда:

y=SQLEXEC(b1,s,”mycur”)

Нетрудно понять, что текст команды на выборку помещается в переменной s.

Наконец отметим, что команда «thisform.text1.value=mycur.name»; отображает содержимое первой выбранной записи (а именно, значение поля name) в элементе с именем text1 в форме приложения. Разрыв соединения реализует команда «SQLDisconnect(b1)».

Представление (View) – это программно создаваемая и визуально отображаемая таблица, заполняемая данными из других таблиц, временно существующая в оперативной памяти. Для создания представления используют запрос языка SQL, реализация которого обеспечивает выборку данных для представления. Для ссылки на представление используется ключевое слово VIEW. На представлении можно выполнять операции поиска, замены и удаления записей. На представлении можно также выполнять SQL-запросы. Изменение данных в представлении автоматически ведет к изменению данных в тех таблицах, из которых построено представление. Представление может быть получено как результат выборки данных через соединение с удаленными компьютерами. Курсоры и представления во многом, но не во всём, дублируют друг друга. Важное различие между ними заключается в том, что

- изменение данных в курсоре необязательно приводит к изменению данных в исходной таблице (определяется типом курсора)
- представление – это визуально отображаемая таблица; курсор – нет.

Рассмотрим создание представлений в Visual FoxPro .

CREATE SQL VIEW students;
AS SELECT * From Stud

Здесь создается представление с именем **students**. Представление заполняется данными из таблицы **Stud** текущей открытой базы данных. В FoxPro для продолжения текста команды с новой строки используется символ точки с запятой (;).

Можно создать удаленное представление, которое будет заполняться данными, полученными из удаленного сетевого компьютера. Как и в случае с курсором, необходимо в этом случае использовать именованное соединение.

Пример:

CREATE SQL VIEW students;
CONNECTION Connect1;
AS SELECT * FROM Stud

В отличие от предыдущего примера указано удаленное соединение с именем Connect1, которое должно быть предварительно построено средствами Visual Foxpro.

1.3.2 Добавление и удаление записей

Удаление целиком всей таблицы выполняется командой

DROP TABLE Склад

В команде указывается имя удаляемой таблицы (в данном случае – Склад).

Можно удалить все записи из таблицы Склад, но оставив саму таблицу:

DELETE FROM Склад

Удаление записей выполняется, как правило, по условию, которому должны отвечать удаляемые записи. Такие условия вводятся с помощью ключевого слова **WHERE.**

Пример:

DELETE FROM Склад **WHERE** Цена>5000

Удаляются все записи, где цена больше 5000. Можно указывать сложные условия, связываемые операциями **AND** (И), **OR**(ИЛИ) или **NOT** (Не).

Пример:

DELETE FROM Склад **WHERE** Цена>5000 **AND Not**(Название=”бумага”)

Удаляются все записи, где одновременно **Цена**>5000 и **Название** не равно «бумага».

Большие возможности предоставляют функции SQL. Среди них выделим в первую очередь групповые функции: SUM, MAX, MIN, AVG, COUNT. Функция SUM (AVG) подсчитывает сумму (среднее) значений указанного поля (выражения). Функция MAX (MIN) вычисляется максимальное (минимальное) значение данного поля (выражения). Функция COUNT подсчитывает число записей с заданным свойством поля (выражения). Предварительно заметим, что групповые функции нельзя записывать в секции WHERE SQL-запроса. В следующем примере удаляются все записи, в которых поле Цена меньше значения максимальной цены.

DELETE FROM Склад **WHERE** Цена < (**Select max**(Цена) **FROM** Склад)

Команда Select, указанная в этом запросе, осуществляет выборку максимальной цены из таблицы Склад. Главная часть запроса представлена секцией

DELETE FROM Склад **WHERE** Цена <

и требует удалить из таблицы Склад все записи, в которых значение поля Цена меньше (<) этого максимального значения, определенного командой Select.

В следующем примере удаляются все записи, в которых поле Название содержит неопределенные значения (NULL):

Пример:

DELETE FROM Склад **WHERE** isNULL(Цена)

В этом примере используется функция isNULL, которая проверяет, содержит ли поле значение NULL, и в случае, если ответ положительный, возвращает истинное значение.

Добавление новых записей в таблицу выполняет команда **INSERT**.

Пример:

INSERT into Склад values(55,”обои”,12000,200)

Эта команда вставляет в таблицу с именем Склад новую запись со следующими значениями полей:

Код=55, Название=”обои”, Цена=12000, Количество=200.

В следующем примере явно указываются поля таблицы, в которые будут помещены значения для вставляемой записи. Остальные поля останутся без значений (будут содержать Null)

INSERT into Склад(Код, Название) values(55,”обои”)

1.3.3 Обновление записей

Обновление (**UPDATE**) записей представляет изменение одного или нескольких полей в выбранной записи или записях. При этом выбор записей выполняется с помощью секции WHERE. Рассмотрим синтаксис команды обновления на примере.

UPDATE Склад **Set** Цена=Цена*1.2

Данная команда обновляет всю таблицу, устанавливая новое значение поля Цена равным предыдущему значению этого поля, умноженному на коэффициент 1.2.

UPDATE Склад **Set** Цена=Цена*1.2 **WHERE** Количество< 1000

Здесь обновление коснется только тех записей, в которых значение поля Количество меньше 1000.

1.3.4 Поиск и выборка записей

Ключевым словом команды является слово **SELECT**. В секции **WHERE** указываются условия, которым должны удовлетворять отбираемые записи. В секции **From** указывается имя одной или нескольких таблиц, из которых производится выборка записей.

Пример:

SELECT Название, Цена **From** Склад

Данная команда выполняет выборку полей Название и Цена из всех записей таблицы Склад.

Если нужно выбрать вообще все поля из записей, то следует указать символ звездочки: «*»

SELECT * **From** Склад

В следующем примере производится выборка по условию:

SELECT Название, Цена **From** Склад **WHERE** Цена>2000

Выдаются значения полей Название и Цена тех записей, где значение поля Цена>2000.

Рассмотрим следующий пример:

SELECT Название, Цена*Количество*0.13 **As** Налог **From** Склад

В данном примере выбирается значение поля Название и параллельно с ним вычисляется выражение «Цена*Количество*0.13», которое будет выведено в колонке с именем **Налог**. Для указания имени колонки, под которым будет выведено выражение, используется ключевое слово **AS**.

Таким образом, можно изменить заголовки полей таблицы, выводимой по команде SELECT:

SELECT Название **AS** ТОВАР, Количество **AS** ВСЕГО **FROM** Склад

Допустим, что величина налога зависит от выручки за реализацию того или иного товара и вычисляется по формуле

ЕСЛИ Цена*Количество <50000 ТО Налог=Цена*Количество*0.13
ЕСЛИ Цена*Количество >= 50000 ТО Налог=Цена*Количество*0.20

Для представления в команде SELECT подобного сложного условия следует использовать функцию **IIF** языка SQL.

Пример:

IIF(Цена*Количество<50000, Цена*Количество*0.13, Цена*Количество*0.2)

Первым аргументом функции IIF является вычисляемое выражение «Цена*Количество<50000». Первый аргумент представляет собой логическое условие. Если условие истинно, то функция IIF возвращает в качестве ответа значе-

ние второго аргумента. В данном примере вторым аргументом является выражение «Цена*Количество*0.13».

Если условие, задаваемое первым аргументом, ложно, то функция IIF возвращает в качестве ответа значение третьего аргумента, т.е. «Цена*Количество*0.2».

В команде SELECT можно выполнять группирование записей по общим значением поля. Рассмотрим таблицу 1.7.

Таблица 1.7 – Пример таблицы Склад

Код	Название	Цена	Количество
11	Бумага	10000	500
13	Клей	1000	20
21	Клей	3000	100
34	Краски	10000	40
37	Краски	9000	500
43	Бумага	8000	100

Заметим, что в колонке **Название** встречаются по нескольку раз одинаковые слова. Это позволяет провести группирование записей по полю **Название**. Например, можно подсчитать среднее значение цены на бумагу, клей, краски или максимальное значение цены на эти товары, или, наконец, сумму количеств этих товаров на складе.

Пример:

SELECT Название, **avg**(Цена) **AS** Средняя_Цена **From** Склад
Group By Название

В следующем *примере* подсчитаем, сколько раз товары с одинаковым именем встретились в таблице:

SELECT Название, **count**(Название) **AS** Число_вхождений **From** Склад
Group By Название

Следующий *пример* позволяет найти минимальную стоимость клея:

SELECT min(Цена), Название
Where Название=”клей”
Group By Название

Заметим, что обратный порядок секций не допустим:

SELECT min(Цена), Название
Group By Название
Where Название=”клей”

Однако в SQL при группировании записей можно использовать секцию **Having**, которая играет роль, аналогичную секции **Where,** причем допустимо записать так:

SELECT min(Цена), Название

Group By Название
HAVING Название=”клей”

Между секцией **HAVING** и секцией **WHERE** имеется следующее принципиальное различие: секция Having выполняет группирование на уже сформированном наборе записей, т.е. записи сначала отбираются, а затем фильтруются по условию Having. Секция Where работает в процессе отбора записей.

Для представления результатов выборки в отсортированном виде используют ключевое слово ORDER BY. Например, записи в порядке алфавитного следования имен товаров:

SELECT * **From** Склад **Order By** Название

Если требуется порядок заменить на обратный, то пишем

SELECT * FROM Склад **Order By** Название **Decs**

Для вывода по SELECT постоянных значений (текстовых констант) можно использовать следующий *пример* для подражания:

SELECT Название, Цена, ”USA долл.” **From** Склад

Результат выборки представлен в таблице 1.8.

Таблица 1.8 – Результат выборки из таблицы Склад

Название	Цена	Expr1002
бумага	700	USA долл.
клей	2500	USA долл.
краски	1800	USA долл.
обои	1000	USA долл.
карандаши	7000	USA долл.
перья	1100	USA долл.
тетради	4555	USA долл.

Рассмотрим теперь выборку из нескольких таблиц. В качестве примера возьмем таблицу 1.1 и таблицу 1.2. Выборка из двух и более таблиц выполняется из результата их соединения. Соединение таблиц осуществляется по общему столбцу. Следовательно, сначала таблицы нужно соединить по общему столбцу, а затем указать условие отбора записей.

Пример:

SELECT Склад.Название, Склад.Цена, Производители.Фирма **From**
Склад **Inner Join** Производители **on**
Склад.Название=Производители.Название

В этом примере использовано внутреннее объединение таблиц в стандарте **ANSI**. Внутреннее объединение учитывает только те записи (в одной таблице), которым соответствуют записи из другой таблицы, не содержащие в общем столбце значений Null. При выборке из нескольких таблиц сначала перечисляются имена результирующих колонок, указываемые после точки, разделяющей

имя колонки от имени таблицы: Склад.Название, Склад.Цена, Производители.Фирма. Затем указывается, как строится результирующая таблица:

From Склад **Inner Join** Производители

Для упорядоченного вывода результатов выборки по команде SELECT следует использовать опцию ORDER BY.

Пример:

SELECT * from SCLAD ORDER BY НАЗВАНИЕ

Записи выводятся в алфавитном порядке по полю **Название**. Порядок, обратный алфавитному, задается так:

SELECT * from SCLAD ORDER BY НАЗВАНИЕ DESCENDING

Имеется возможность группировать записи при выборке. При этом можно выполнять групповые функции на каждой группе записей. К групповым функциям относятся такие функции, как SUM (суммирование), AVG (вычисление среднего), COUNT (подсчет числа записей), MIN/MAX (определение максимума/минимума) и др.

Пример для таблицы 1.1:

SELECT Название, avg(Цена) from Склад **GROUP BY** Название

Эта команда предписывает сгруппировать записи по полю **Название**. Это значит, что в каждую группу попадают записи с одинаковым значением поля Название. Для каждой группы вычисляется среднее значение цены в пределах группы.

SELECT Название, count(Название) as Всего From Склад Group By Название

1.3.5 Некоторые вложенные функции SQL

Рассмотрим некоторые важные функции T-SQL, достаточно часто используемые в запросах.

IF EXISTS …

Проверяет наличие хотя бы одной выбранной записи по запросу.

```
IF EXISTS (Select * From Склад Where Цена>8000)
   Select   ”Имеются товары с высокой ценой” From Склад
 Else
   Select “Нет дорогих товаров”
```

ABS(…) возвращает модуль величины, указываемой в скобках

CHAR(…) преобразует целое число в строку символов

GetDate() возвращает текущую системную дату

IsdATE(выражение) возвращает истину, если выражение соответствует корректной дате

isNULL(выражение, константа) заменяет значение NULL заданной константой

isNumeric(выражение) возвращает истину, если выражение является числом

Len(символьная строка) возвращает длину строки

Lower(строка) преобразует символы строки в нижний регистр

LTrim(строка) удаляет ведущие пробелы

STR(число) преобразует число в строку

SubString(строка, начало, длина) выделяет подстроку из строки, начиная с заданного символа начала и включая указанное число символов

Upper(выражение) преобразует символы выражения в верхний регистр

Year(дата) возвращает год из даты в виде целого числа из четырех цифр

Month(дата) возвращает месяц в виде целого числа

Можно выбрать указанное число записей из набора:

SELECT TOP 2 * **from** Склад

Эта команда выбирает только две первые записи из таблицы Склад. Можно произвести отбор первых 10% записей так:

SELECT TOP 10 PERCENT * **from** Склад

Для выборки неповторяющихся записей следует указать:

SELECT * **from** Склад **DISTINCT**
(dictinct – различные).

Для выборки строк по шаблону применяется функция LIKE. Пример

Select Название **From** Склад **Where** Название **Like** “б%”

Для проверки попадания значения в диапазон используется конструкция **Select** с **In**:

Select Название **from** Склад **Where** Цена **In** (**Select max**(Цена) **from** Склад)

Этот запрос выдает названия всех товаров, цена на которые попадает в запрос

Select max(Цена) **from** Склад

т. е. является максимальной ценой.
Другой способ проверки попадания в диапазон:

Select Название **from** Склад **Where** Цена **Between** 1000 **AND** 5000

Здесь использовано ключевое слово **Between** (между).

При проверке алгебраических соотношений между числовыми величинами используют следующие операции: «=» – равно, «!=» – не равно, «>» – больше, «!>» – не больше, «<» – меньше, «!<» – не меньше, «>=» –больше или равно, «<=» – меньше или равно, «<>» – не равно.

1.3.6 Изменение таблиц

Для изменения таблиц используется команда **ALTER**. Следующий *пример* показывает применение команды ALTER для добавления нового столбца в таблицу вместе с новым ограничением:

ALTER TABLE Склад **ADD**
Дата_поступления **DATE CONSTRAINT** ctr **CHECK**
(**Year(**Дата_поступления)=2006)

В этом примере в таблицу Склад добавляется новое поле Дата_поступления с типом DATE, а также ограничение с именем ctr, которое проверяет, что значение года в поле Дата_поступления равно 2006.

1.4 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

В качестве БД выберем СУБД Visual FoxPro 9.0. Запустим систему Visual FoxPro 9.0. Отобразится следующий экран (рисунок 1.2). Создаем новую базу данных (нажимаем кнопку New Project в левом нижнем окне). Вводим по приглашению системы собственное имя проекта (например proj1). Откроется окно менеджера проектов (рисунок 1.3).

В этом окне (см. рисунок 1.3) выберем закладку Data (щелкнем по значку «+» слева) и нажмём кнопку New. Зададим имя новой базы данных, например myDb.

Для работы с SQL потребуется командное окно FoxPro для ввода и исполнения команд SQL. В командном окне создайте две таблицы: Склад и Производители. *Например*, таблица Склад создается таким образом:

CREATE TABLE sclad(kod int PRIMARY KEY,title varchar(20), price int , amount int)

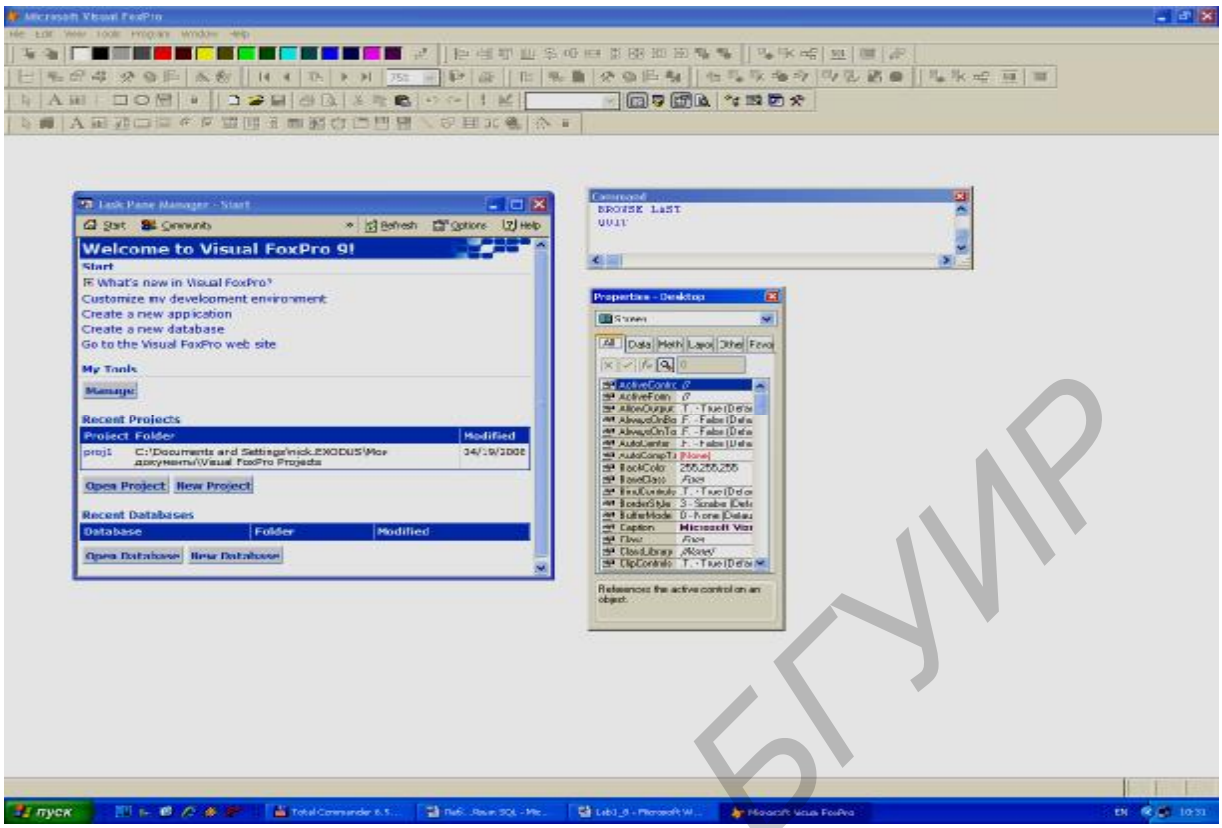


Рисунок 1.2 – Рабочее окно с Visual FoxPro 9.0

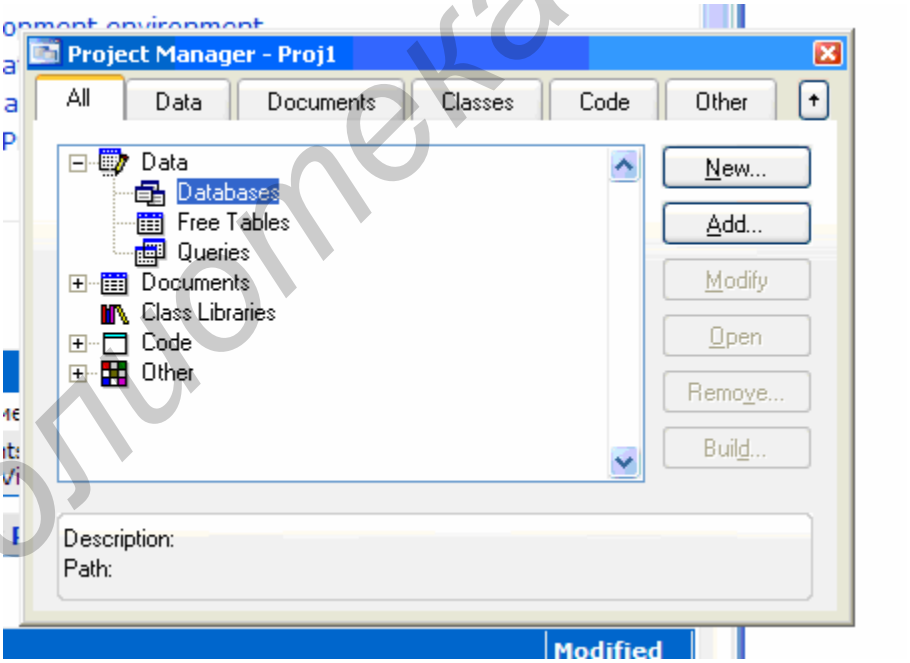


Рисунок 1.3 – Окно менеджера проектов

Чтобы открыть эту таблицу, используйте команду FoxPro:

USE sclad

Для просмотра таблицы введите команду

BROWSE

На данном этапе в таблице SCLAD нет записей. Добавьте в таблицу несколько записей. *Например*, это можно сделать следующей командой:

INSERT INTO Sclad values(1,«гвозди»,10000,10)

Результат добавления записи просмотрите командой Browse.
Попробуйте ввести еще раз команду

INSERT INTO Sclad values(1, «гвозди», 10000,10)

Система выдаст сообщение о неуникальности ключа!
Выполните удаление записей командой

DELETE FROM Sclad

Снова введите команду Browse. Записи останутся в таблице. FoxPro не удаляет записи сразу, а только помечает их на удаление. Физическое удаление помеченных записей производится командой

PACK

Снова вставьте несколько записей. Выполним изменение записи.

UPDATE sclad **SET** price=10,amount=1000 **WHERE** kod=1

Просмотрите результат этой команды с помощью Browse.

Конечно, работать с командным окном не удобно – нужно каждый раз вводить и выполнять всего лишь одну команду. Чтобы облегчить работу, воспользуемся **хранимыми процедурами**.

В мастере проектов на закладке Data выберите пункт Stored Procedures (рисунок 1.4) и нажмите кнопку New. Откроется окно редактора, куда необходимо поместить код наподобие следующего:

Procedure Insert1
  Insert INTO sclad values(3,”болты”,200,1000)
ENDPROC

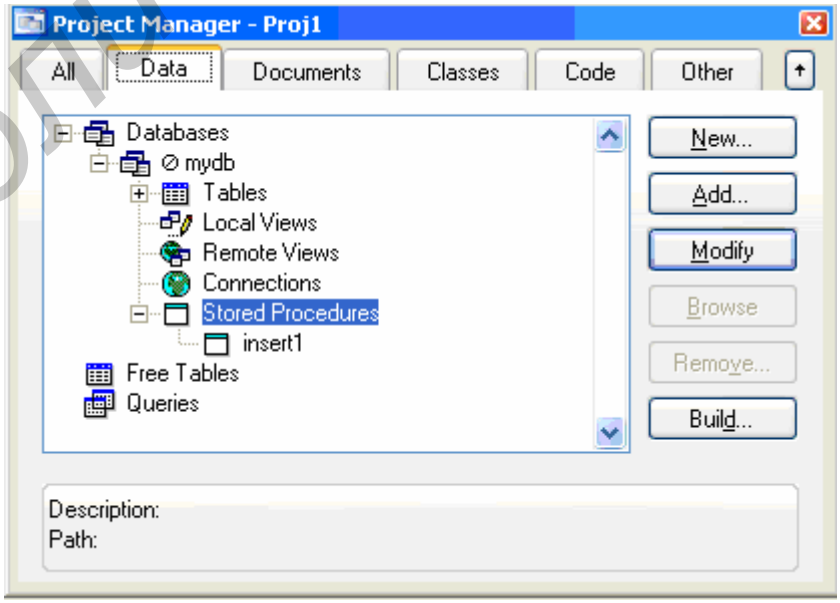


Рисунок 1.4 – Мастер проектов

Для сохранения этой процедуры нажмите комбинацию клавиш CTRL+W. Теперь можно выполнить то же из командного окна. Введите строку

DO insert1

Результат просмотрите командой Browse. Возможно еще больше облегчить себе задачу, если автоматизировать ввод кода, названия товара, цены и количества. Несмотря на то что это уводит в область программирования на FoxPro, результат должен быть получен. Напишем следующий код:

```
PROCEDURE insert1
 k=RECCOUNT()+1
 t=”?”
 p=0
 am=0
@ 0,0 CLEAR
 @ 1,1 CLEAR to 1,50
 @1,1 SAY ”kod=” GET k size 1,6
 READ
 @ 1,1 CLEAR to 1,50
 @1,1 SAY ”title=” GET t size 1,15
 READ
 @ 1,1 CLEAR to 1,50
 @1,1 SAY ”price=” GET p size 1,6
 READ
 @ 1,1 CLEAR to 1,50
 @1,1 SAY ”amount=” GET am size 1,6
 READ
 INSERT INTO sclad values(k,t,p,am)
 BROWSE
ENDPROC
```

Сначала необходимо объявить переменные: k – код записи, t – название товара, p – цена и am – количество. Обозначения переменных произвольны. **RECCOUNT()** возвращает число записей в таблице. Команда

@ 1,1 **CLEAR** to 1,50

очищает область экрана с первой строки первого столбца по первую строку пятидесятого столбца. Чтение данных выполняют команды

```
@1,1 SAY ”price=” GET p size 1,6
READ
```

Здесь выводится слово «price=» с первой строки в первом столбце в области, состоящей из одной строки и шести столбцов. Команда **READ** реагирует на нажатие ВВОДа для присваивания значений переменным.

1.5 ЗАДАНИЕ

Написать хранимые процедуры для следующих видов запросов:

1) три примера на изменение записи – изменить цену, название и количество товара. Название товара ввести с экрана;
2) вывести количество товара данного наименования;
3) вывести названия и цену товара, причем названия товара упорядочить по алфавиту;
4) вывести название товара и выручку как вычисляемое поле=цена*количество;
5) вывести название товара с минимальной ценой, цену также отобразить;
6) создать выборку из двух таблиц – товары и производители. Вывести название товара, цену, фирму и ее адрес;
7) вывести название фирмы, производящей самый дешевый товар;
8) привести два различных примера работы SELECT … group by с группированием;
9) вывести название фирмы, продающей товар по самой низкой цене. Название товара ввести с экрана;
10) продемонстрировать использование курсора;
11) продемонстрировать работу представления;
12) вывести таблицу по SELECT с вычисляемым полем НАЛОГ. Для вычисления налога использовать функцию IIF (налог считать так: если выручка от продажи товара больше 10 000, то налог равен 20% от выручки, иначе – 13% от выручки.

1.6 КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Что представляет собой реляционная база данных?
2 Что такое атрибут, домен?
3 Функциональная зависимость между атрибутами и её смысл?
4 Что такое полный ключ? Частичный ключ?
5 Объясните понятие курсора и представления.
6 Объясните основные команды SQL (по выбору преподавателя).

ЛАБОРАТОРНАЯ РАБОТА №2

ВИЗУАЛЬНЫЙ ИНТЕРФЕЙС Visual FoxPro

2.1 ЦЕЛЬ РАБОТЫ

1) изучение основных средств визуального интерфейса;
2) обработка событий.

2.2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.2.1 Общие сведения

Основная работа с базой данных выполняется через форму (набор форм). На форме размещают различные элементы интерфейса, такие, как кнопки, текстовые поля, списки, рисунки и пр.

Для открытия окна конструктора форм при создании новой формы можно воспользоваться одним из следующих способов:

1) выполнить команду **File | New**, выбрать опцию **Form** и нажать кнопку **New File**;
2) в командном окне набрать команду **CREATE FORM**.

Для отображения на экране панелей инструментов установите метки в соответствующих опциях меню **View** или установите флажки выбора панелей инструментов в окне диалога **Toolbars**.

Панель инструментов **Form Controls** используется для размещения объектов в форме. Краткое описание кнопок этой панели приведено в таблице 2.1.

Таблица 2.1 – Панель инструментов **Form Controls**

Наименование	Назначение
1	2
Select Objects	Указатель выделения
View Classes	Выбирает класс для создания объектов в форме
Label	Текстовый объект
Text Box	Поле ввода
Edit Box	Поле редактирования текста
Command Button	Кнопка
Option Group	Переключатель
Check Box	Флажок
Grid	Таблица
Combo Box	Поле ввода со списком
List Box	Список
Spinner	Поле ввода значения

Продолжение таблицы 2.1

1	2
Line	Линия
Shape	Контур
Container	Контейнер
Image	Рисунок
Command Group	Группа кнопок
Timer	Таймер
Page Frame	Страница
OLE Bound Control	Отображает содержимое OLE-объекта, хранящегося в поле типа General
OLE Container Control	Создает OLE-объект
Separator	Создает разделитель – объект, формирующий промежутки между элементами управления на панели инструментов
HyperLink	Гиперссылка
Builder Lock	Закрепляет выбор построителя
Button Lock	Закрепляет выбранную кнопку на панели инструментов

Панель инструментов **Form Designer** содержит кнопки вызова панелей инструментов **Form Controls**, **Color Palette**, **Layout**, а также выполняет некоторые дополнительные функции управления формой.

2.2.2 Создание окружения

Подключение источников данных к формам организовывается с помощью настройки среды окружения.

Любая форма имеет встроенный объект **Data Environment**, доступ к которому возможен из меню **View | Data Environment** или из контекстного меню **Data Environment** после щелчка правой кнопкой мыши на форме. В любом случае открывается диалоговое окно **Data Environment**, в которое можно добавить таблицу, представление или курсор.

2.2.3 Размещение объектов на форме

Для того чтобы разместить любой объект на форме, необходимо выполнить следующие действия:

1) выбрать объект на панели **Form Controls**;
2) установить указатель мыши на место предполагаемого расположения текстового объекта и, удерживая кнопку мыши в нажатом состоянии, переместить курсор по диагонали так, чтобы получилась рамка требуемого размера;
3) установить соответствующие свойства объекта;
4) определить соответствующие методы объекта.

2.2.4 Текстовые поля

Размещение текста в форме осуществляется с помощью инструмента **Label**. Под текстом понимается любая текстовая информация: заголовки, наименования полей и поясняющая информация.

Текст задается в свойстве **Caption**, тип шрифта, размер и цвет символов устанавливаются с помощью свойств **FontName**, **FontSize** и **ForeColor** соответственно.

2.2.5 Поля ввода и редактирования

2.2.5.1 Поля ввода

Поля ввода (объект **Text Box**) могут использоваться для отображения и ввода значений в поля таблицы. Чтобы связать его с полем таблицы, необходимо в свойстве **ControlSource** указать имя поля открытой таблицы.

Установка свойства **Alignment** позволяет задать способ выравнивания информации, отображаемой в поле.

Для задания стиля и цвета рамки поля используются свойства **BorderStyle** и **BorderColor**, а для определения цвета фона неактивного поля – свойство **DisabledBackColor**.

Свойства **FontName** и **FontSize** определяют тип и размер шрифта, а **ForeColor** – цвет информации в поле ввода.

Если информация в поле должна быть доступна только для чтения, необходимо установить значение свойства **ReadOnly** равным **True**.

Для определения значения поля по умолчанию используется свойство **Value**.

2.2.5.2 Поля редактирования

Поля редактирования **Edit Box** очень удобны для редактирования символьных полей большого размера и **Memo**-полей.

Поскольку поле редактирования предназначено для просмотра и редактирования полей большого размера, в правой части поля расположена полоса прокрутки, предназначенная для просмотра информации, не поместившейся в окне просмотра. Отличительной особенностью поля редактирования по сравнению с полем ввода является наличие у него свойства **ScrollBars**. Это свойство может принимать два значения: **None** – полоса прокрутки отсутствует; **Vertical** – поле имеет вертикальную полосу прокрутки.

2.2.6 Кнопки управления

Для создания кнопок управления используется инструмент **Command Button**. На кнопке можно расположить текст или графическое изображение.

При размещении текстовой информации корректируется свойство **Caption**, для размещения графического изображения используется свойство **Picture**.

После создания кнопки необходимо определить команды, которые будут выполняться при ее нажатии. Для этой цели служит метод **Click**, который автоматически вызывается при нажатии на кнопку.

Для определения текста необходимо в окне свойств объекта выбрать **Click Event**. На экране откроется окно процедур, куда помещаются команды, которые должны выполняться при нажатии на данную кнопку. Например, при нажатии на кнопку выхода из формы на экран будет выдаваться запрос о желании действительно выйти из формы (при утвердительном ответе форма будет закрыта):

```
* Запрашиваем и выходим, если ”Да”
IF MessegeBox(”Выйти из формы? ”, 4+32+256, ”Выход”) = 6
  _screen.ActiveForm.Release()
ELSE
  _screen.ActiveForm.Refresh()
ENDIF
```

2.2.7 Флажки и переключатели

Для индикации состояния, которое может иметь только одно из двух допустимых значений, используются объекты типа **CheckBox** (флажки). Рассмотрим создание флажка для редактирования поля типа **Logical** (это поле может принимать значения 0 или 1).

Свойство **Caption** определяет заголовок, отображаемый справа от флажка. В случае необходимости можно скорректировать свойства **ForeColor**, **BackColor**, **FontName**, определяющие цвет шрифта, фон и вид шрифта.

Для связывания флажка с полем таблицы используется свойство **ControlSource**. В нем задается в качестве значения имя **Поля** таблицы.

Объекты типа **Option Group** (переключатели) позволяют выбрать одно из нескольких значений поля или переменной. **Option Group** являются составными объектами, содержащими внутри себя элементы, наделённые собственными свойствами.

Рассмотрим создание переключателя для просмотра и редактирования поля **TypePay** (Вид оплаты) таблицы **Ordsalem** (Заказы и продажи), которое может принимать одно из значений: «Наличные», «Безналичные», «Бартер», «Кредитная карточка», «В рассрочку». Необходимо выполнить следующую последовательность действий:

1) разместить на форме инструмент **Option Group**;

2) в окне свойств скорректировать свойство **Button Count**, задав количество опций равным 5;

3) скорректировать свойство **ControlSource**, задав имя поля **TypePay**;

4) в раскрывающемся списке объектов в верхней части окна **Properties** выбрать первую опцию переключателя **Option1**, скорректировать свойства **Caption**, **ForeColor**, **BackColor**, **FontName**, определяющие заголовок, цвет шрифта, цвет фона и вид шрифта;

5) аналогично скорректировать свойства для остальных объектов.

Теперь при редактировании заказов и продаж в поле **TypePay** таблицы **Ordsalem** будет заноситься значение, которое устанавливается с помощью переключателя.

2.2.8 Списки

Список предназначен для отображения на экране элементов списка, которые могут быть определены с помощью следующих средств: массив, меню, список файлов, значения поля таблицы, структура таблиц и т.д.

Различают два вида списков: **List Box** и **Combo Box**. Оба списка имеют одинаковые свойства, но **Combo Box** в отличие от **List Box** не занимает много места в форме.

Тип источника данных определяется свойством **RowSourceType**, допустимые значения которого приведены в таблице 2.2.

Таблица 2.2 – Допустимые значения свойства **RowSourceType**

Значение	Источник данных
None	Элементы списка определяют программно, используя методы **Addltem** или **AddListItem**
Value	Список задается в виде строки, элементы в которой разделяются запятыми
Alias	В качестве источника данных используется таблица. Количество выводимых полей таблицы определяется значением свойства **ColumnCount**
SQL Statement	Список содержит данные, полученные в результате выполнения **SQL** оператора
Query	Список содержит данные, полученные в результате выполнения указанного запроса. Запрос задается именем файла с расширением .qpr
Array	Источником данных является заданный массив
Fields	Значения элементов списка определяются полями таблицы
Files	Перечень файлов текущего каталога. Можно задать в свойстве **RowSource** шаблон выбора файлов
Structure	Источник данных – структура таблицы
Popup	Список содержит пункты всплывающего меню

Списки используют свойства, размещенные в таблице 2.3.

Таблица 2.3 – Свойства и методы объектов типа **List Box**

Свойство	Назначение
ColumnCount	Число столбцов в списке
ColumnWidths	Ширина столбцов
ControlSource	Источник данных, с которым связан объект
FirstElement	Задает первый элемент массива, который будет отображаться в списке. Данное свойство доступно только в том случае, если свойство **RowSource** задает в качестве источника данных массив
IncrementalSearch	Определяет, позволяет ли объект последовательный поиск
MultiSelect	Определяет множественный выбор в списке
NumberOfElement	Определяет количество элементов массива, отображаемых в списке. Данное свойство доступно только в том случае, если свойство **RowSource** задает в качестве источника массив
RowSource	Указывает источник данных списка

Рассмотрим создание списка для редактирования поля **Country** (Страна) таблицы **Customer**, которое может принимать одно из значений, заданных в поле **CountryName** таблицы **Countries**.

Добавим в окружение формы таблицу **Countries** с наименованиями стран. Затем разместим на форме инструмент **Combo Box**. Чтобы связать созданное поле с полем таблицы **Customer**, свойство **ControlSource** устанавливаем равным **Customer.Country**. Далее корректируем свойство **RowSourceType**, устанавливая его равным **Fields**, поскольку список стран находится в таблице. И в заключение свойство **RowSource** устанавливаем равным **Countries.Country**.

2.2.9 Графические изображения и объекты типа General

2.2.9.1 Графические изображения

В формы можно вставлять различные графические изображения, позволяющие облегчать восприятие информации. Для этого используется инструмент **Image**.

Установив данный объект на форме, в свойстве **Picture** необходимо указать имя BMP-файла.

2.2.10 Запуск формы на выполнения

Создадим форму, показанную на рисунке 2.1. С этой целью воспользуемся мастером форм и разместим на форме три ярлыка (**Label**) три текстовых поля (**Textbox**) и две кнопки.

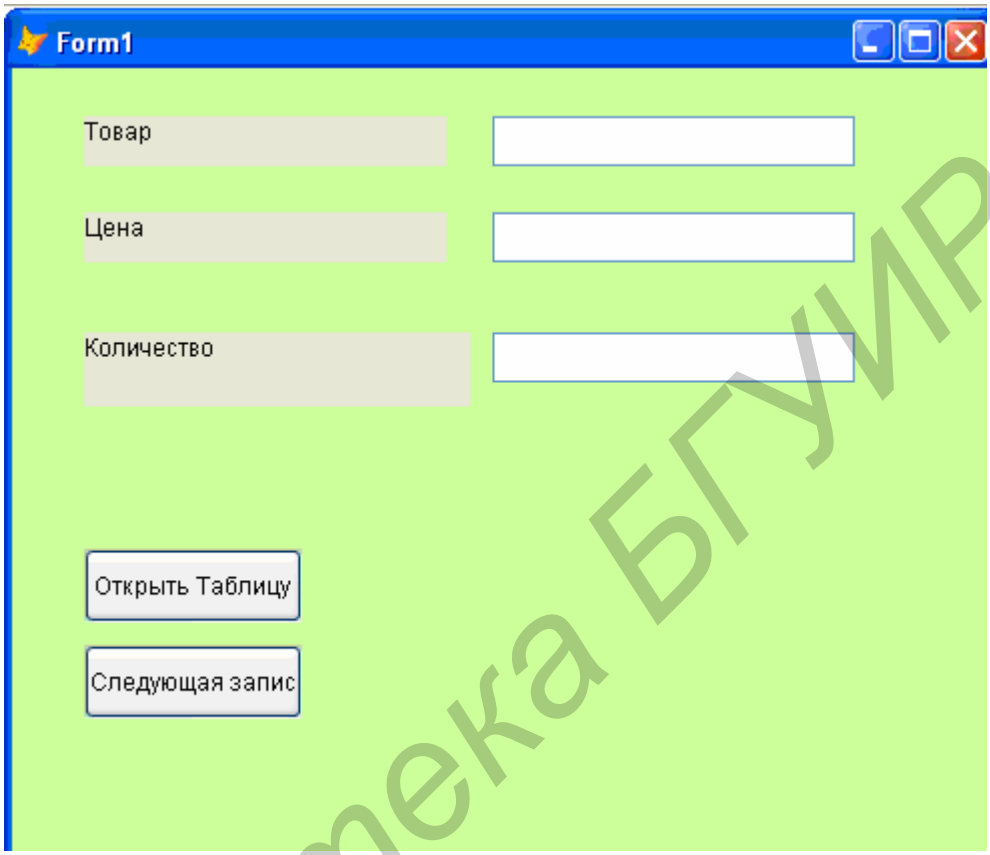


Рисунок 2.1 – Пример создаваемой формы

В окне свойств зададим названия ярлыков, а также установим цвет формы (светло-зеленый). Наша задача – запрограммировать кнопки. Однако перед тем, как выполнить эту задачу, сделаем нашу форму стартовым объектом проекта. Это значит, что при запуске проекта данная форма будет стартовать первой. Для этого в окне диспетчера проекта на вкладке **Documents** щелкните правой кнопкой мыши на имени формы и установите галочку напротив элемента **Set Main**.

После этого нажмите кнопку **Build** и укажите тип приложения **Application** (app). Система построит приложение с расширением **app**.

Можете запустить это приложение на выполнение, щелкнув дважды на имени proj1.app. К сожалению, приложение запустится вместе с окном среды Visual FoxPro. Чтобы среда не отображалась, запрограммируем некоторые события для формы. Прежде всего начнем с события **Load**. Откройте окно свойств формы и вкладку **Methods**.

Дважды щелкните по элементу **Load** и наберите в окне редактора кода следующую строку

```
_Screen.Visible=.F.
```

Задайте для формы свойство **ShowWindow** в окне свойств, как показано на рисунке 2.2 (**as Top Level Form**).

Теперь при запуске приложения системное окно пропадет. При этом отметим, что приложения типа **.app** тем не менее требуют наличия установленного в системе Visual FoxPro.

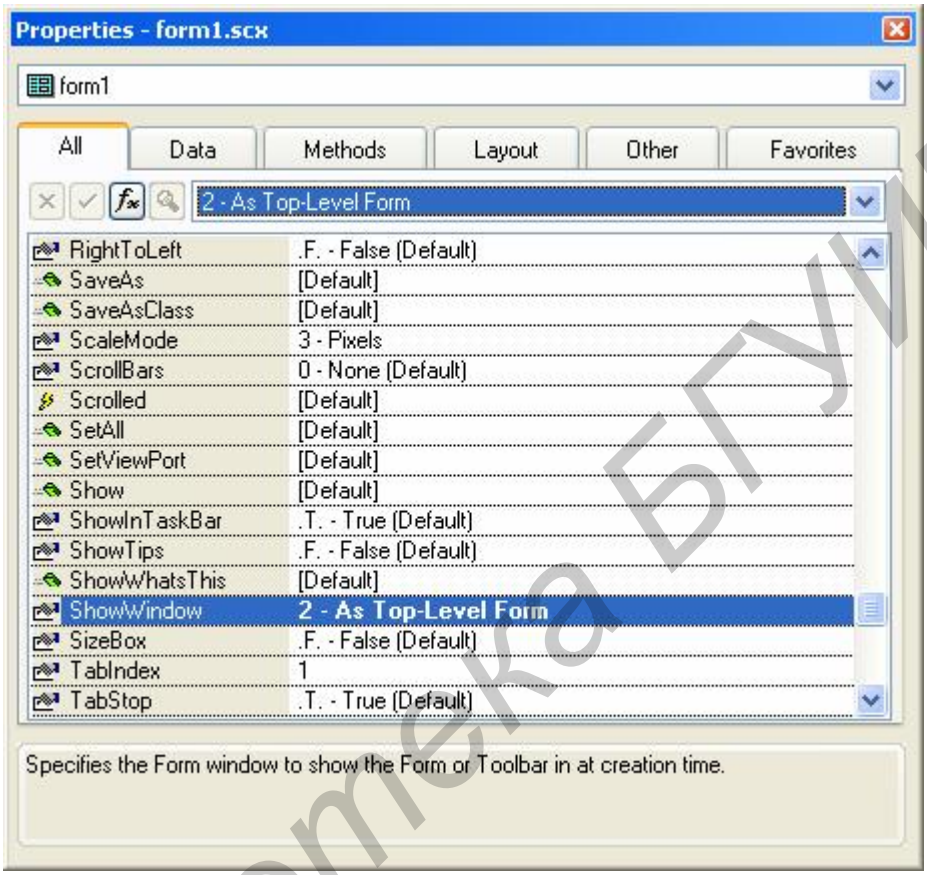


Рисунок 2.2 – Окно свойств

Перекомпилируйте приложение заново, используя кнопку **Build**, и запустите его на выполнение.

Теперь запрограммируем события от кнопок. Щелкнем на кнопке дважды. Откроется редактор кода, в который и следует поместить программный код. Начнем с кнопки открытия таблицы. Код нашего обработчика имеет следующий вид:

```
SELECT * FROM sclad
GO top
thisform.text1.value=sclad.title
thisform.text2.value=sclad.price
thisform.text3.value=sclad.amount
```

Проблема с такой реализацией состоит в том, что команда **Select** открывает окно браузера, в котором нет необходимости. Поэтому в таких ситуациях применяют курсоры. Изменим обработчик следующим образом:

PUBLIC opened

Open DataBase mydb
CREATE CURSOR mycur (kod int,title c(20),price int, amount int)
SELECT * FROM sclad INTO CURSOR mycur
GO top

thisform.text1.value=mycur.title
thisform.text2.value=mycur.price
thisform.text3.value=mycur.amount
opened=.T.

Теперь запрограммируем кнопку «Следующая запись»:

IF not(EOF("a")) then
SKIP
thisform.text1.value=mycur.title
thisform.text2.value=mycur.price
thisform.text3.value=mycur.amount
ELSE
GO top
thisform.text1.value=mycur.title
thisform.text2.value=mycur.price
thisform.text3.value=mycur.amount
ENDIF

2.2.11 Создание горизонтального меню на форме

Важным элементом интерфейса является меню. Рассмотрим, как реализовать меню на форме.

Создадим форму, на которой будет представлено меню. При выборе опций меню будут выполняться программные (**.prg**) файлы. Эти prg-файлы заменят нам кнопки. Разработка горизонтального меню будет нами выполнена вручную. Создадим текстовый файл, например, с помощью FAR или Norton Commander, или с помощью программы БЛОКНОТ. Имя этого файла – menu1.mpr. В этом файле будет определена структура меню. Для отображения меню на форме необходимо:

1) запрограммировать событие **Init** формы следующим образом:

Do d:\german\методичка-бибд(лаб)\mymenu.mpr with thisform

Эта команда запускает меню из файла **menu1.mpr**, который мы подготовим заранее. Событие **init** наступает после загрузки формы. Обычно это событие используют для инициализации переменных и объектов, порождаемых на основании описаний классов;

2) запрограммируем событие **Load** для формы так:

_SCREEN.Visible=.F.

Эта команда делает невидимым главное окно Visual FoxPro;

3) установим свойство **ShowWindow** формы в значение 2– As Top Level Window (значит, что наша форма является окном верхнего уровня).

Чтобы выполнить указанные три действия, нужно воспользоваться окном свойств формы. Для этого нужно выделить форму щелчком мыши, затем активизировать контекстное меню щелчком правой кнопки мыши. Далее следует выбрать закладку **All** (в окне свойств) и найти события Init, Load. В строке События дважды щелкнуть мышью. Откроется окно Редактора кода, куда и надлежит вписать приведенные строки.

Файл **menu1.mpr** должен иметь такой вид:

Parameters f
c="mymenu"
m.f.Name=c

Define Menu c in (m.f.Name) Bar
Define Pad a of c Prompt “Открыть таблицу”
Define Pad b of c Prompt “Перейти на следующую”

On Selection PAD a of c do d:\german\методичка-БиБД(лаб)\mainger1.prg
On Selection PAD b of c do d:\german\методичка-БиБД(лаб)\mainger2.prg

Activate menu c

Этот файл содержит команды FoxPro, описывающие меню. Этот файл можно было бы создать автоматически с помощью следующей схемы:

- описать структуру меню в окне Дизайнера меню;
- скомпилировать полученное описание, хранящееся в файле с расширением **.mpr**;
- отредактировать файл с расширением .mpr нужным образом.

Из приведенных пунктов, описываемый здесь способ выглядит более простым. Поясним смысл команд файла Меню.

Строка **Define Menu** c **in (m.f.Name) Bar**

задает название меню – в переменной **c**; место открытия меню – (**m.f.Name**); тип меню – **Bar**. Тип Bar соотвествует горизонтальному меню. Кроме горизонтального, имеется еще всплывающее (**PopUp**) меню.

Строка **Define Pad** a **of** c **Prompt “Открыть таблицу”**

определяет первый пункт меню. Ему присваивается программное имя **a** и текст соответствующего пункта меню – «Открыть таблицу».

Строка **Define Pad** b **of** c **Prompt “Перейти на следующую”**

определяет второй пункт меню. Ему присваивается программное имя **b** и текст соответствующего пункта меню – «Перейти на следующую запись».

Действия, реализуемые при выборе пунктов меню, определяют строки

On Selection PAD a **of** c **do mainger.prg**
On Selection PAD b **of** c **do mainger2.prg**

Первая строка определяет действие, выполняемое системой при выборе первого пункта меню, вторая строка – при выборе второго пункта меню. При выборе первого пункта меню выполняется команда **do mainger.prg**, т.е. запускается программный файл с именем **mainger.prg**. При выборе второго пункта меню запускается программный файл с именем **mainger2.prg**.

Теперь рассмотрим начальные строки файла меню:

```
Parameters f
c=”mymenu”
m.f.Name=c
```

Файл меню рассматривается как исполняемый. В него при вызове передается один параметр

```
Do menu1.mpr with thisform
```

А именно: создается переменная в памяти, связанная с объектом **thisform**, представляющим форму. FoxPro требует перед именем таких переменных указывать букву **m**. В результате получаем достаточно странное имя **m.f**. Однако у формы есть имя (**Name**). Поэтому имя формы должно определяться как **m.f.Name**.

Итак, меню создано и показано, как его запустить на форме. Теперь напишем программные файлы mainger1.prg и mainger2.prg. Содержимым файла **mainger1.prg** будет:

```
PUBLIC opened

Open DataBase mydb
CREATE CURSOR mycur (kod int,title c(20),price int, amount int)
SELECT * FROM sclad INTO CURSOR mycur
GO top
 form1.text1.value=mycur.title
 form1.text2.value=mycur.price
 form1.text3.value=mycur.amount
 opened=.T.
```

Содержимым файла **mainger2**.prg пусть будут строки

```
IF not(EOF()) then
SKIP
 form1.text1.value=mycur.title
 form1.text2.value=mycur.price
 form1.text3.value=mycur.amount
ELSE
GO top
 form1.text1.value=mycur.title
 form1.text2.value=mycur.price
 form1.text3.value=mycur.amount
ENDIF
```

Важное замечание. В отличие от кнопок на форму теперь ссылаемся не как **thisform**, а как **form1** (form1 – это программное имя формы).

2.3 ЗАДАНИЕ

На основе представленной частичной реализации создать кнопки для добавления, поиска, изменения и удаления записей из таблицы. Реализовать представление картинок на форме.

Добавить в проект вторую форму для реализации отображения таблицы Производители, связанной с таблицей SCLAD. Реализовать взаимные переходы между формами с помощью меню.

2.4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1 С помощью какого инструмента можно быстро создать шаблон формы?
2 Для чего нужна настройка среды окружения?
3 Из последовательности каких действий состоит размещение элементов управления на форме?
4 Какие свойства объектов обычно устанавливаются для связи с таблицами?
5 Для каких полей обычно используется поле редактирования?
6 Чем отличаются элементы управления List Box и Combo Box?
7 Как запустить форму на выполнение?

ЛАБОРАТОРНАЯ РАБОТА №3

ОСНОВЫ MS SQL Server

3.1 ЦЕЛЬ РАБОТЫ

1) изучение основных средств MS SQL Server;
2) получение практических навыков работы в среде MS SQL Server.

3.2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

3.2.1 Утилита Enterprise Manager

3.2.1.1 Основы

Данная утилита является одной из базовых инструментальных средств MS SQL Server. С ее помощью можно

- создавать базы данных;
- управлять запуском, остановом и конфигурированием служб MS SQL Server;
- управлять системой безопасности и др.

После инсталляции сервера можно через меню Пуск -> Программы -> MS SQL Server запустить программу Enterprise Manager, стартовое окно которой показано на рисунке 3.1.

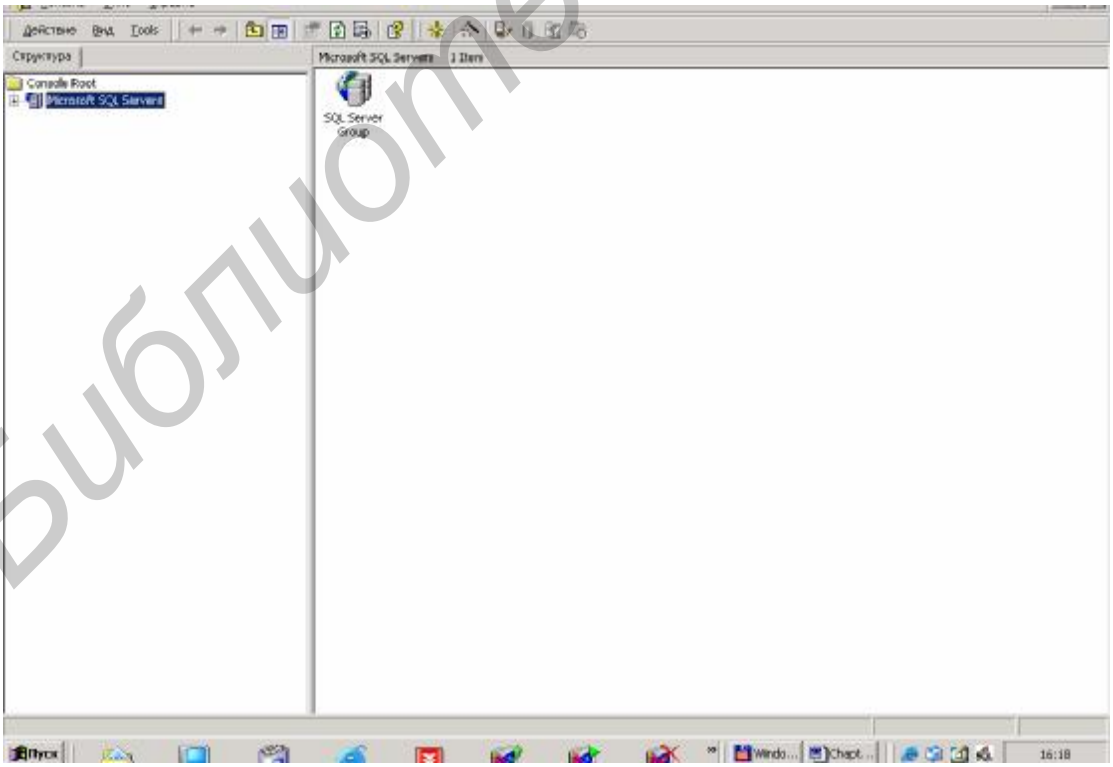


Рисунок 3.1 – Стартовое окно Enterprise Manager

Для создания базы данных (БД) откройте папку Microsoft SQL Servers в левом окне, выберите сначала свой сервер и при необходимости подсоединитесь к нему через опцию Connect. Откройте папку DataBases, активизируйте контекстное меню щелчком правой кнопки мыши (рисунок 3.2).

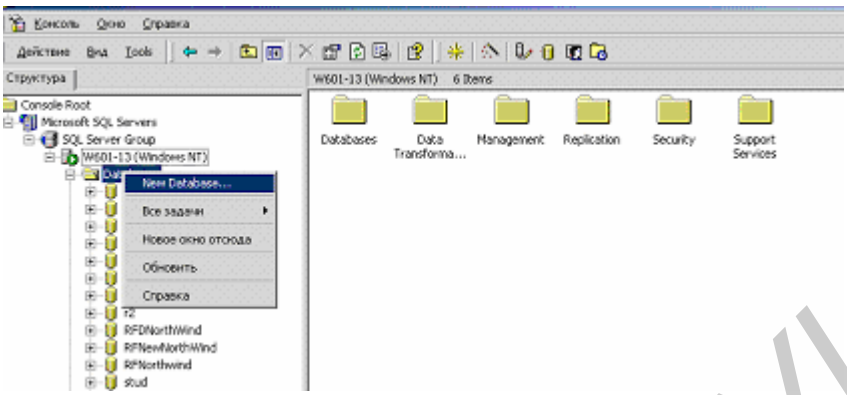


Рисунок 3.2 – Активация контекстного меню

Используйте пункт «New DataBase». В открывшемся окне укажите имя БД. Все действия по созданию БД так или иначе должны подготовить инструкцию SQL для создания БД. Выберите вкладку **DataFiles** и в открывшемся окне (рисунок 3.3) введите, если требуется, информацию о файлах, на которых физически будет размещена создаваемая БД.

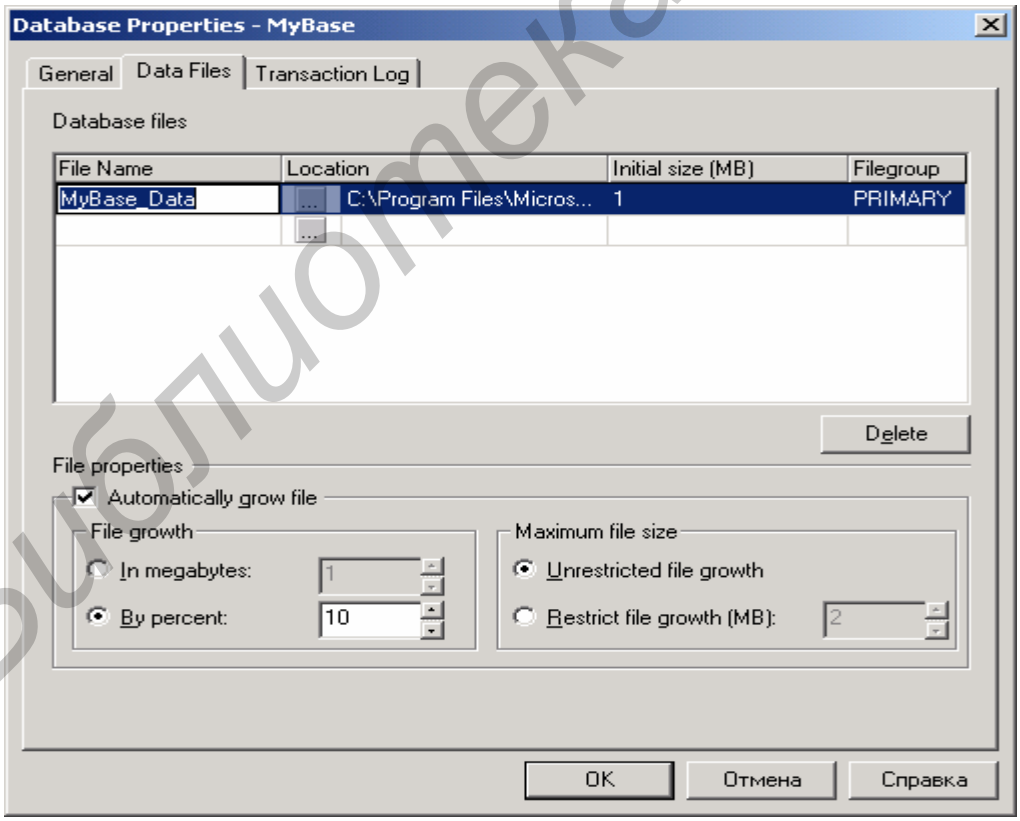


Рисунок 3.3 – Окно свойств БД

Теперь можно выбрать вкладку **Transaction Log** для управления файлами журнала транзакций.

После ввода всех данных нажмите кнопку **OK** – система построит пустую БД. Откройте пустую БД. В правом окне увидите значки компонентов, которые ее образуют (рисунок 3.4).

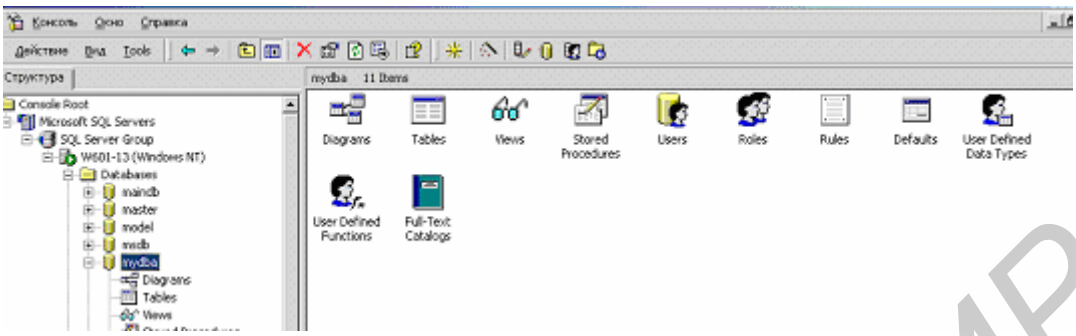


Рисунок 3.4 – Сформированная БД

Такими компонентами являются:

– таблицы;
– представления;
– хранимые процедуры;
– учетные записи пользователей;
– роли;
– правила;
– пользовательские типы и др.

3.2.1.2 Добавление пользователей

Прежде чем добавить какого-либо пользователя к той или иной БД, нужно создать учетную запись пользователя. Учетная запись администратора (sa) добавляется автоматически при создании БД. Для создания учетной записи выберем элемент дерева в левом окне с именем **Security**, затем – **Logins**. Появится окно, представленное на рисунке 3.5

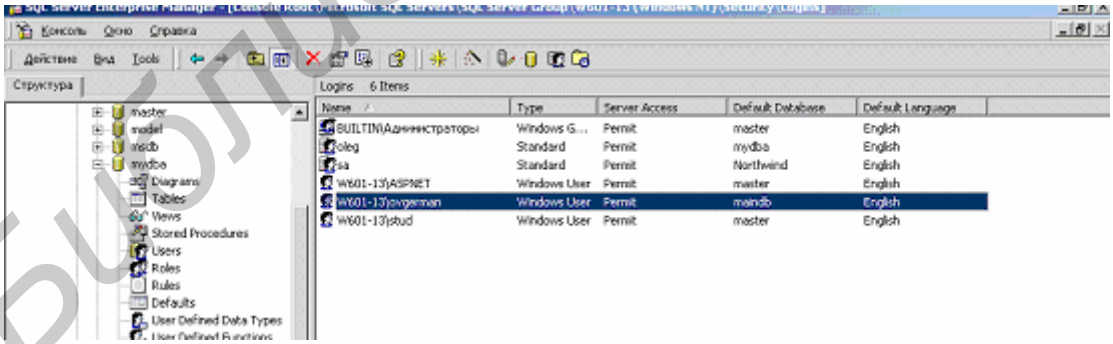


Рисунок 3.5 – Добавление пользователей

В колонке Type определен тип пользователя:

– Windows User – одиночный пользователь Windows NT;
– Windows Group – групповой пользователь Windows NT;
– Standard – пользователь SQL Server.

Остальные колонки определяют имя пользователя (**Name**); имя БД – (**Default Database**) и наличие разрешения доступа к серверу (**Server Access**).

Для создания новой учетной записи следует активизировать контекстное меню и нажать кнопку **New Login**. В появившемся окне следует ввести требуемую информацию о пользователе. Вариант заполнения полей формы приведен на рисунке 3.6.

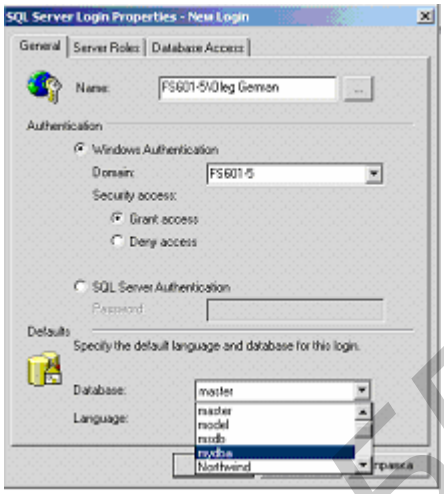


Рисунок 3.6 – Ввод информации о пользователе

Следует обратить внимание на следующее. Во-первых, имеется два варианта аутентификации пользователя. Первый использует пароль и имя, вводимые при регистрации в Windows, второй использует пароль и имя для регистрации в SQL Server. Если использовать первый вариант аутентификации, то необходимо дополнительно указать имя домена Windows, к которому принадлежит пользователь. Если использовать второй вариант аутентификации, то при подключении к SQL Server придется каждый раз вводить имя и пароль. В нижнем списке на рисунке 3.6 выбирается имя БД, к которой по умолчанию подключается пользователь. Далее на вкладке **Server Roles** следует указать, какую ролевую группу в рамках пользователей SQL Server будет представлять пользователь. Та или иная ролевая группа имеет определенные возможности по **администрированию** сервера, а именно:

– Sysadmin – все возможности по администрированию;

– SetUpAdmin – этой группе представлены права управления связанными серверами, а также конфигурирование хранимых процедур;

– ServerAdmin – это администраторы сервера;

– SecurityAdmin – имеют право на создание новых учетных записей и предоставление прав для работы с БД;

– Dbcreator – могут создавать новые БД и др.

На вкладке **Permissions** содержится список прав доступа, присвоенных выбранной роли пользователя (**Role**). Остается рассмотреть вкладку **DataBaseAccess** (рисунок 3.7). В верхнем окне представлены имена БД, размещенных на сервере. Пользователь может быть назначен той или иной БД, как пока-

зано на рисунке 3.7. Причем его можно назначить одновременно нескольким БД. В нижнем окне указывается, каким правами наделяется пользователь в рамках той или иной роли БД. Таким ролями являются:

– Db_owner – эта роль предполагает любые действия над БД;

– Db_dataWriter – члены этой группы могут выполнять изменение содержимого любой таблицы или представления;

– Db_dataReader – могут выполнять чтение из таблиц и представлений;

– Db_ddlAdmin – могут создавать, изменять и удалять объекты БД;

– Db_securityAdmin – могут управлять правами доступа других пользователей к БД и др.

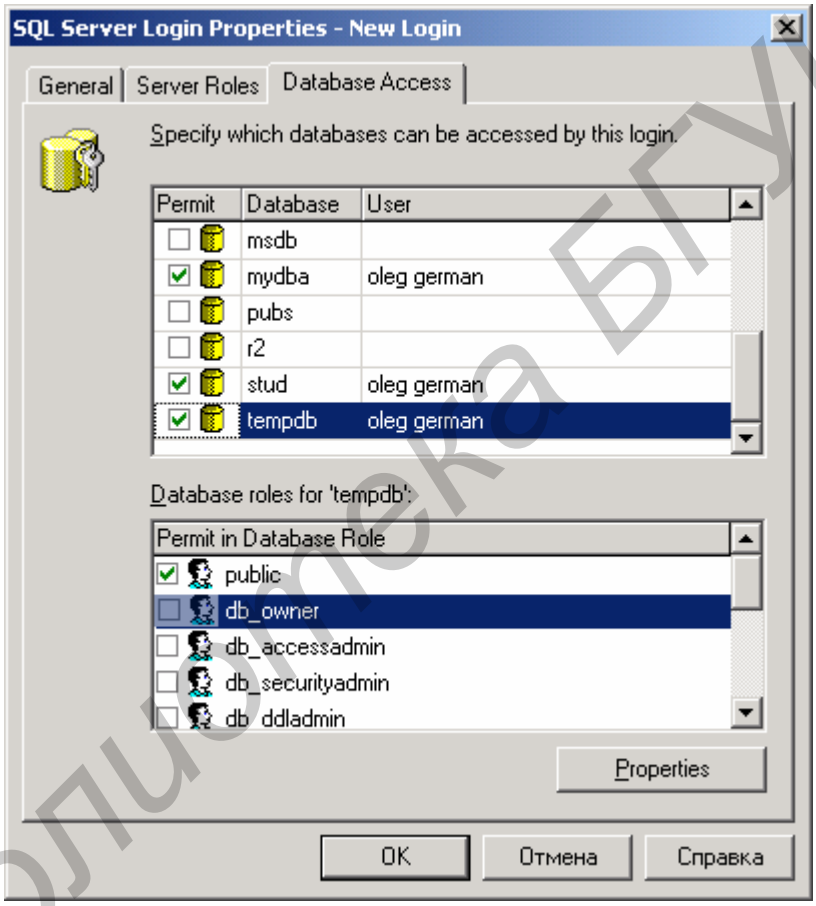


Рисунок 3.7 – Информация о пользователе. Вкладка DataBase Access

3.2.2 Создание и связывание таблиц

Для создания таблицы в открытой БД нужно активизировать контекстное меню щелчком правой кнопки на поле Tables и выбрать пункт «Add Tables». Откроется окно, показанное на рисунке 3.8.

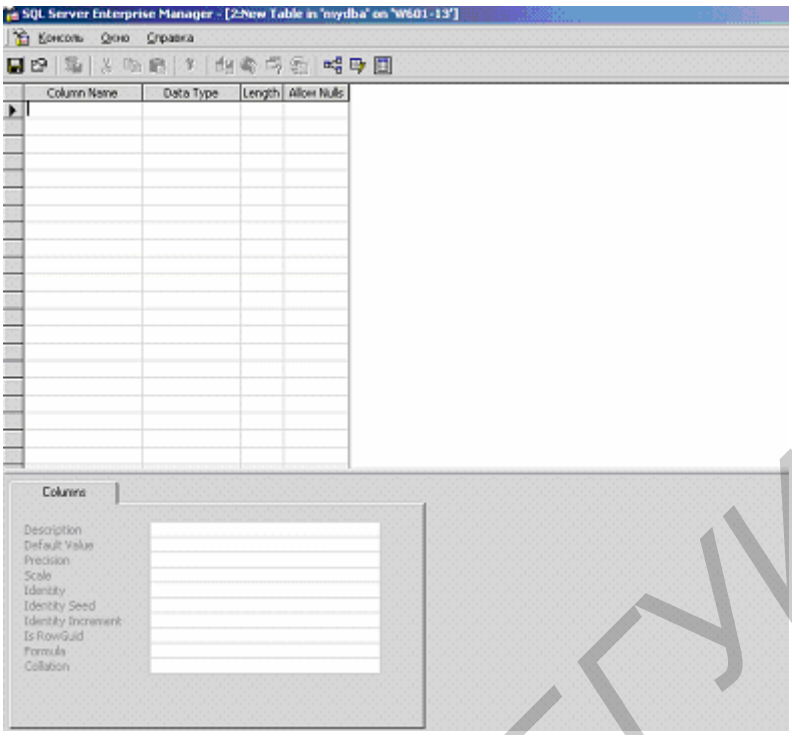


Рисунок 3.8 – Создание таблицы

В верхнем окне следует ввести следующую информацию:

– Column Name – имя столбца;
– Data Type – тип столбца (тип SQL);
– Length – ширина поля;
– Allow Nulls – разрешение не вводить данные в столбец.

С помощью иконки с изображением ключа столбец можно сделать ключевым (символ ключа размещается слева от имени столбца). Повторный щелчок мышью на иконке со знаком ключа отменяет ключевое поле.

В нижнем окне можно указать следующую информацию:

Description – описание столбца (можно ввести произвольную информацию, характеризующую столбец);

Default Value – значение в столбце, вводимое по умолчанию (если пользователь не ввел данные в столбец);

Precision – максимальное число в представлении числа (0 – ограничено только типом столбца);

Scale – число символов после запятой (дробная часть);

Identity – задает столбец-счетчик (содержит номер записей);

Identity Seed – начальное значение счетчика записей;

Identity Increment – приращение счетчика при введении номера очередной записи;

IsRowGuide – столбец, куда помещается уникальное число, присваиваемое записи;

Formula – выражение для вычисления содержимого столбца. В этом случае столбец является вычисляемым полем.

Введенную таблицу следует сохранить, используя для этого кнопку с изображением дискеты и введя имя таблицы (рисунок 3.9).

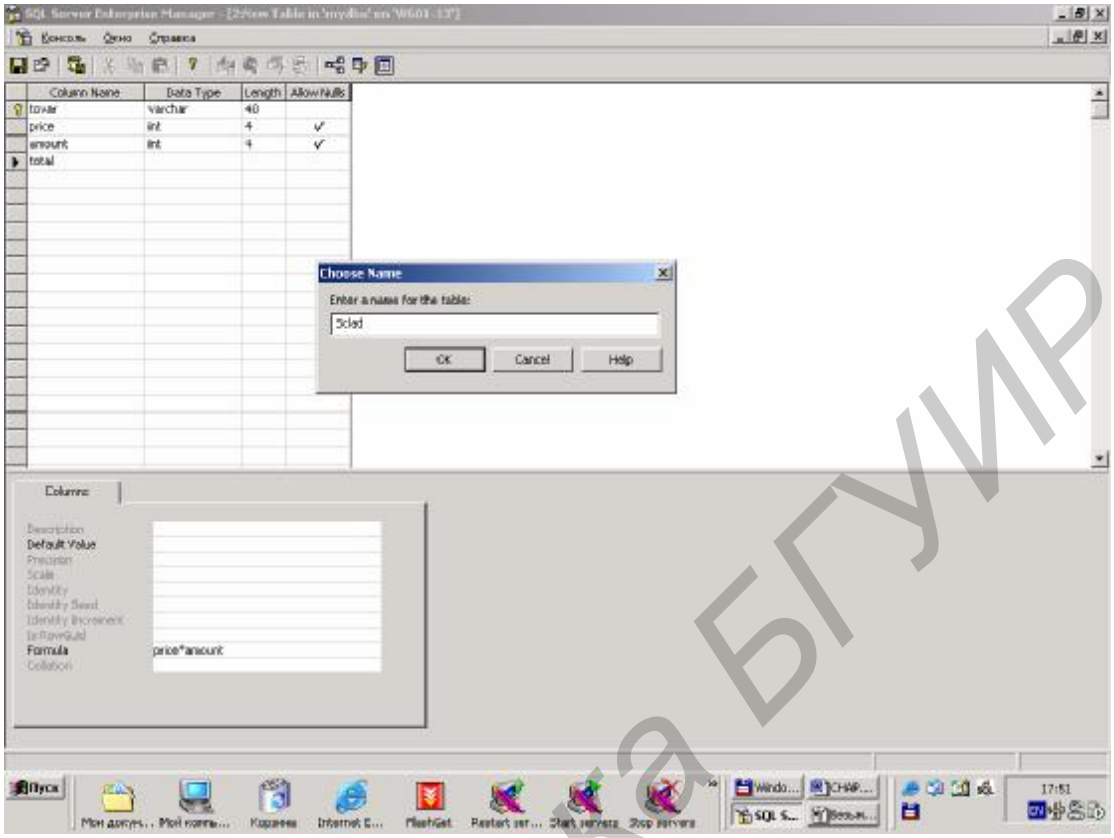


Рисунок 3.9 – Сохранение созданной таблицы

В нашем примере создали таблицу **Sclad** с полями

tovar

price

amount

total

Последнее поле является вычисляемым по формуле **total=price*amount**.

Создадим еще одну таблицу **Firms**, столбцами которой будут **title**, **tovar**, **address**. После этого свяжем таблицы по полю **tovar**. Для связывания двух таблиц откроем конструктор таблицы **Firms** (она будет зависимой). На панели инструментов конструктора таблиц найдем иконку **Table and Index Properties**. Откроем окно щелчком мыши на этой иконке (рисунок 3.10). На вкладке **Tables** указываем выбранную для работы таблицу. Затем перейдем на вкладку **RelationShips** (рисунок 3.11).

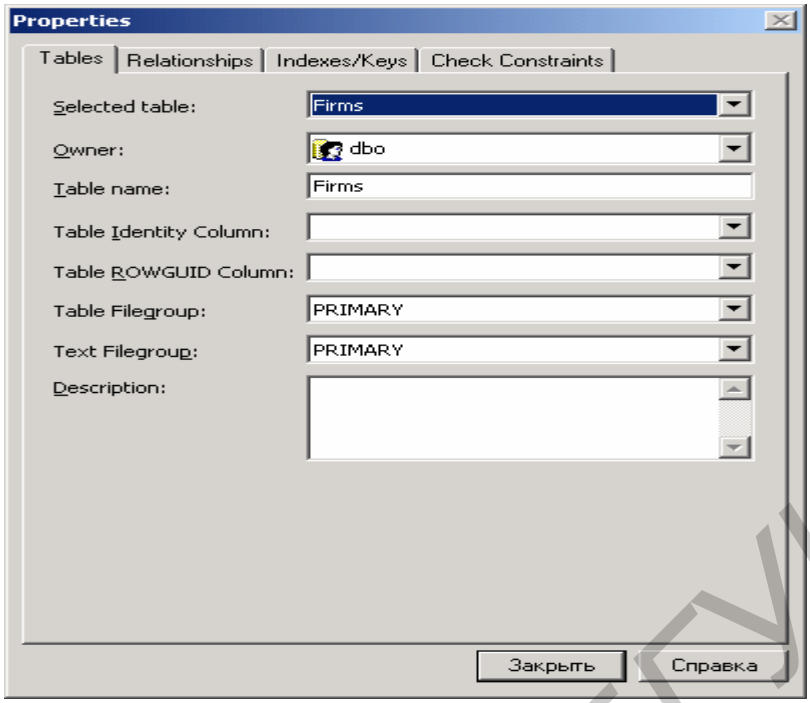


Рисунок 3.10 – Окно свойств таблицы. Вкладка Tables

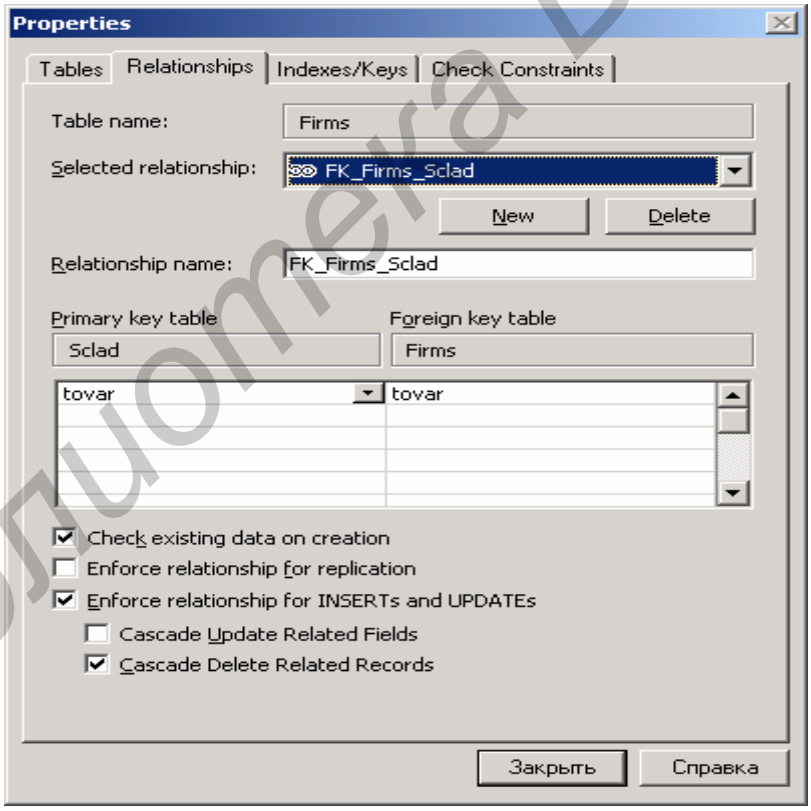


Рисунок 3.11 – Окно свойств таблицы. Вкладка Relation

Для создания связи к таблице **Firms** нажмем кнопку **New**. Затем в поле **Primary key table** выберем имя родительской таблицы **Sclad**. В поле **Foreign Key Table** выберем имя зависимой таблицы, т. е. **Firms**. В левой колонке таблицы зададим имя поля родительской таблицы, с которым связывается поле зависимой таблицы. В нашем случае это поле называется одинаково для обеих

таблиц – **tovar**. Выберем тип ограничения целостности – **Cascade Delete Related Records**. Этот вариант ограничения целостности по связи означает, что при удалении записи из родительской таблицы удаляются связанные с ней по внешнему ключу записи дочерней таблицы. Другой вариант ограничения целостности – **Cascade Update Related Fields** означает, что при изменении внешнего ключа в родительской таблице соответствующие изменения автоматически вносятся в дочернюю таблицу.

Теперь выберем вкладку **Indexes/Keys** (рисунок 3.12)

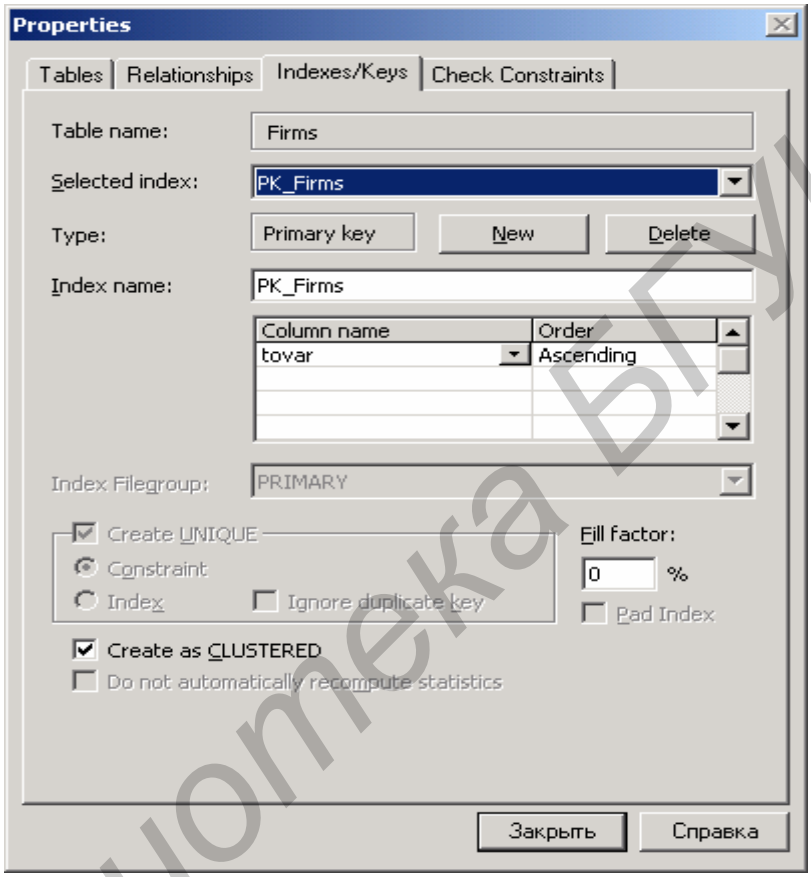


Рисунок 3.12 – Окно свойств таблицы. Вкладка Indexes/Keys

Здесь можно создавать новые индексы, но ограничимся созданным индексом для столбца **tovar**. Имя индекса – **PK_Firms**. PK определяет аббревиатуру для Primary Keys. Флажок **Create As Clustered** предписывает создать кластерный индекс. В этом случае записи в таблице размещаются в порядке следования индексов. Значение индекса можно объявить как уникальное с помощью флажка **Unique**.

Теперь можно рассмотреть вкладку **Check Constarints**, с помощью которой можно создавать, удалять и модифицировать ограничения для таблицы. После выбора этой вкладки и нажатия кнопки **New** будет создано новое ограничение. Имя этого ограничения следует ввести в поле **Constraint Name**. В поле **Constraint expression** следует указать условие, проверяемое ограничением. Например, для поля price (цена) выражение можно было бы записать как price > 0.

Флажок **Check Existing Data on Creation** предписывает выполнять контроль ограничений при создании записей. Флажок **Enforse Constraint for Inserts and Updates** включает проверку ограничения при вставке новых записей или обновлении данных.

Теперь можно ввести данные в таблицы. Для этой цели следует активизировать контекстное меню на имени таблицы и выбрать сначала пункт **Open Table**, а затем пункт **Return All Rows** (показать все строки). Для добавления новой строки используйте клавишу **Enter**.

3.2.3 ЯЗЫК TRANSACT-SQL

3.2.3.1 Основы программирования

Программы пишут на основе хранимых процедур. Программа состоит из части объявлений и части кода. В части объявлений объявляются переменные и курсоры.

Пример:

DECLARE @a int, @s varchar(40)
Select @a=1, @s=’hello’

Строка объявлений начинается со слова DECLARE. Имени переменной предшествует знак @. После имени переменной указывается ее тип. Присваивание значений переменным выполняется либо командой SET, *например*:

SET @a=2,

либо, как показано выше, с помощью команды SELECT.

Для вывода значения переменной на экран используют команду Print:

Print @a

Наряду с переменными используют временные таблицы. Именам таких таблиц предшествует символ #. Временные таблицы хранятся в системной базе данных с именем Tempdb. Работа с временными таблицами реализуется с помощью языка T-SQL уже знакомым нам образом.

Логические условия проверяют с помощью оператора IF, *например*:

IF price >=0 **AND** price <=1000 PRINT ‘Valid’ ELSE PRINT ‘INVALID’

При работе со строками важную роль играет оператор LIKE. Этот оператор проверяет совпадение строки с шаблоном. В качестве одного из символов шаблона выступает знак ”%”.

Пример:

SELECT Tovar from Sclad where Tovar Like “B%”

Данный оператор выбирает из таблицы Sclad значения из поля tovar, которые начинаются на букву «B»

Другие символы шаблона:

– «_» – соответствует одному единственному символу;

– «[]» – в квадратных скобках записываются подряд без разделителей символы; оригинальный символ должен соответствовать любому одному из списка.

Пример:

Select Tovar from Sclad where Tovar Like “[Bb]%”

В отличие от предыдущего примера название выбираемого товара в этом случае может начинаться как с заглавной, так и с малой литеры «В».

– ”[^…]” – в квадратных скобках записываются подряд без разделителей символы; оригинальный символ **не должен** соответствовать ни одному из списка.

Команда «**IF** условие *команда1* **ELSE** *команда2*» проверяет **условие** и выполняет *команду1*, если условие истинно, в противном случае выполняет *команду2*. В качестве команды может использоваться блок команд, рассматриваемый ниже.

Пример:

IF (SELECT min(price) from Sclad)>1000
PRINT ‘Rich things’
ELSE
PRINT ‘Cheep things’

Подмножеством команды **IF** является команда **IF EXISTS**, которая проверяет наличие хотя бы одной записи в результирующем наборе.

Пример:

IF EXISTS (SELECT tovar from Sclad where price>3000)
Print ‘You have rich things’
ELSE
Print ‘You have no rich things’

Операторы могут объединяться в блоки **BEGIN** … **END**. Такой блок рассматривается как один (например транзакция). Кроме того, блоки **BEGIN** … **END** используются в теле цикла.

Пример:

DECLARE @cod int, @sum decimal(10,2),@price
Set @sum=0
Set @cod=1
WHILE 1=1
BEGIN
@price=-1
SELECT @price=price from SClad where cod=@cod
IF @price=-1
BREAK ELSE
BEGIN
SET @sum=@sum+@price

```
SET @cod=@cod+1
END
CONTINUE
END
PRINT @sum
```

В этом примере подсчитывается сумма цен товаров в переменной **@sum**. Выборка товаров производится в цикле по полю **cod**. Предполагается, что значения в поле **cod** начинаются с 1 и возрастают последовательно на 1. Таким образом, если полю **cod** не соответствует ни одна запись, то оператор **SELECT** не изменит значения переменной **@price** и будет выполнен выход из цикла по команде **BREAK**. Если запись находится, то выполняется блок команд

```
BEGIN
SET @sum=@sum+@price
SET @cod=@cod+1
CONTINUE
END
```

Оператор **CONTINUE** выполняет переход на очередной шаг цикла **WHILE**. Структура цикла While такова:

```
WHILE условие
Блок
CONTINUE
```

Условие определяет возможность вхождения в тело цикла. Тело цикла состоит из единственной команды или блока **Begin … End**. Ключевое слово **CONTINUE** завершает цикл. Преждевременный выход из цикла реализует команда **BREAK**. Итерации цикла повторяются до тех пор, пока условие цикла истинно. Например, истинным условием будет 1=1 (как в примере выше). В качестве истинного можно записать и другие условия, например 1<2.

Создадим программный код в теле хранимой процедуры и затем выполним его. Для этого в открытой базе данных выберем элемент **Stored Procedures** (рисунок 3.13), а затем – подпункт **New Stored Procedure**. Введем простой текст в окне редактирования кода процедуры:

```
CREATE PROCEDURE [dbo].st_a AS
Select * from Sclad
```

Проверим синтаксис кнопкой **Check Syntax** и нажмем **OK**.

Для того чтобы посмотреть процедуру в действии, из меню Пуск-> Программы->MicroSoft SQL Server -> Query Analyzer запустим утилиту **Query Analyzer** (рисунок 3.14) и введем команды:

```
use mydba
exec st_a
```

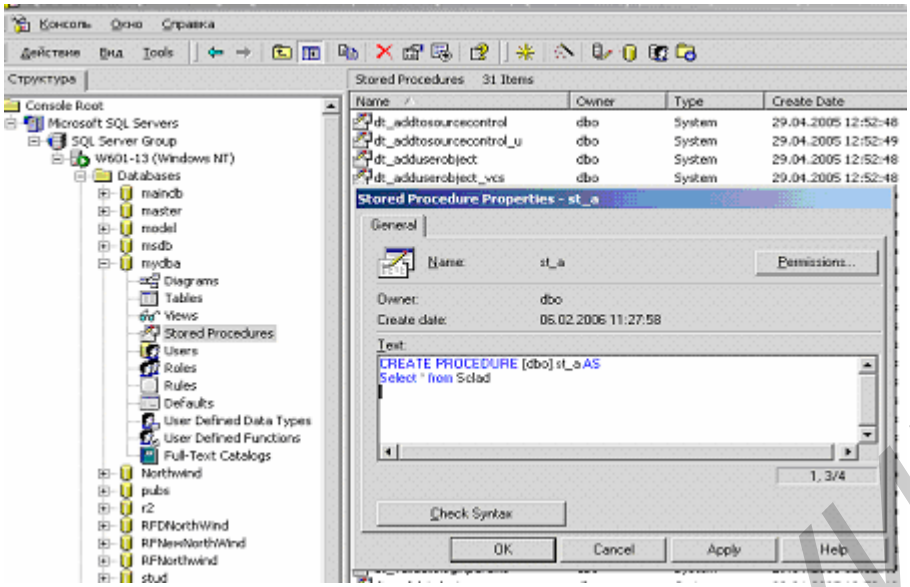


Рисунок 3.13 – Элемент Stored Procedures

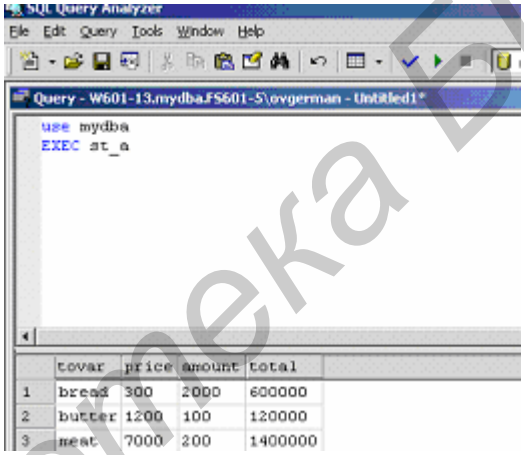


Рисунок 3.14 – Утилита Query Analyzer

Команда **use mydba** открывает БД и делает тем самым доступными все ее элементы – таблицы, запросы, представления, хранимые процедуры. Команда **EXEC st_a** выполняет созданную хранимую процедуру и отображает результат ее действия в форме таблицы в нижнем окне на рисунке 3.14.

В языке **T-SQL** имеется оператор **GOTO**. Он используется для безусловного перехода на метку. Меткой является имя, за которым следует двоеточие.

Пример:

```
DECLARE @cod int, @sum decimal(10,2),@price
Set @sum=0
Set @cod=1
Cycl:
BEGIN
@price=-1
SELECT @price=price from Sclad where cod=@cod
IF @price=-1
```

```
GOTO EX ELSE
BEGIN
SET @sum=@sum+@price
SET @cod=@cod+1
GOTO Cycl
END
END
EX:
PRINT @sum
```

Здесь реализован ранее рассмотренный пример с циклом While. Теперь управление итерациями построено на основе оператора **IF** и команды **GOTO**.

Полезной функцией является функция **CASE**. Эта функция позволяет возвращать одно выражение из множества выражений в зависимости от результата проверки условия.

Пример:

```
SELECT tovar, 'Price is' +
    CASE
        WHEN price<1000 THEN 'Low'
        WHEN price>1000 AND price<2000 THEN 'MIDDLE'
            ELSE 'Too High'
     END
From Sclad
```

Конструкция **CASE** … **END** включает строки

WHEN условие **THEN** результат

Таких строк может быть несколько. Если условие истинно, то возвращается значение, указанное за словом THEN. В разбираемом примере выбирается название товара (tovar) из таблицы Sclad. К названию товара добавляется строка «Price is» , а к этой строке в свою очередь добавляется значение, возвращаемое командой CASE.

3.2.3.2 Функции

Рассмотрим важные встроенные функции языка T-SQL. Некоторые из функций представляют собой глобальные значения (переменные), именам которых предшествует два знака @.

@@CURSOR_ROWS – задает количество записей, выбранных в курсор.

@@FETCH_STATUS – устанавливает результат последней выполненной команды **FETCH**:

0 – успех;

-1 – неудача, связанная с попыткой выборки строки, выходящей за пределы курсора;

-2 – неудача, связанная с попыткой выборки недоступной строки.

ABS(числовое_выражение) – возвращает абсолютную величину аргумента.

CEILING(числовое_выражение) – определяет наименьшее целое, большее либо равное аргументу.

CHAR(целое_выражение) – получает символ ASCII по его номеру.

CHARINDEX(str1,str,num) – возвращает номер подстроки str1 в строке str, начиная с позиции с номером num. Если подстрока не найдена, результатом является 0.

COALESCE(s1,s2,…,sn) – возвращает первый из аргументов, не равный NULL.

COL_NAME(имя_таблицы, номер_столбца) – имя столбца в таблице.

DAY(выражение_дата) – формирует целое число, определяющее день месяца.

FLOOR(выражение) – наибольшее целое, меньшее либо равное заданной величине.

GETDATE() – системная дата и время.

ISDATE(выражение) – возвращает 1, если выражение корректно определяет дату.

ISNUMERIC(выражение) – возвращает 1, если выражение является числовым.

ISNULL(выражение1, выражение2) – возвращает в качестве значения выражения1 выражение2 при условии, что выражение1 имеет текущее значение NULL. В качестве выражения1 обычно используют имя столбца таблицы.

LEFT(str,num) – возвращает строку, полученную из левых num символов строки str.

LEN(str) – длина строки str без учета завершающих пробелов.

LOWER(str) – возвращает строку, полученную из строки str преобразованием в нижний регистр.

LTRIM(str) – удаляет ведущие пробелы в строке str.

MONTH(дата) – определяет номер месяца по дате в виде целого числа.

RAND() – получает случайное число от 0 до 1. Можно задавать числовой аргумент, используемый для инициализации генератора случайных чисел.

REPLACE(str1,str2,str3) – заменяет в строке str1 все вхождения строки str2 на строку str3.

REVERSE(str) – размещает строку в обратном порядке.

RIGHT(str,n) – возвращает подстроку строки str, состоящую из n последних символов строки str1.

ROUND(x,n1,n2) – округляет число x, оставляя в нем n1 цифр. Если n2=0, то ROUND не округляет, а отбрасывает цифры.

RTRIM(str) – удаляет завершающие пробелы в строке str.

SPACE(n) формирует строку, состоящую из n пробелов.

STR(числовое_выражение, n,m) – формирует строковое выражение из заданного числового выражения, используя n в качестве общего числа цифр и m – в качестве числа цифр в дробной части. Значения n и m можно не указывать.

STUFF(str1,n,m,str2) – m символов строки str1, начиная с символа с номером n, заменяются на строку str2.

SUBSTRING(str,n,m) – возвращает подстроку строки str, начиная с символа n и включая m последовательно стоящих символов.

UPPER(str) – возвращает строку, полученную из str переводом ее символов в верхний регистр.

YEAR(дата) – определяет по дате значение года.

3.2.3.3 Хранимые процедуры

Хранимые процедуры являются объектами базы данных. Для создания хранимой процедуры используют следующий общий синтаксис:

CREATE PROCEDURE имя_процедуры
[@параметр тип [VARYING] [=*значение*] [OUTPUT]
[FOR REPLICATION]
AS
Команды_SQL
[RETURN [код_возврата]]

В квадратных скобках записывают части, которые могут отсутствовать. Здесь **@параметр** представляет параметр, передаваемый или возвращаемый из процедуры. Количество параметров ограничено величиной 1024. Ключевое слово **VARYING** означает, что параметр представляет итоговый набор записей. В этом случае его типом должен быть **CURSOR**. Параметру может изначально (по умолчанию) присваиваться *значение*. Ключевое слово **OUTPUT** указывает, что параметр является выходным. Параметры типа CURSOR обязаны быть выходными. Ключевые слова **FOR REPLICATION** указывают на то, что процедура участвует в репликации. Под репликацией понимают копирование БД или ее частей на другие сервера. Тело хранимой процедуры записывается после ключевого слова **AS**. Команда **RETURN** может использоваться для передачи кода завершения процедуры, по которому можно определить характер ее выполнения. Как правило, код возврата 0 считают успешным.

Пример:

CREATE PROCEDURE pr1
@mycur CURSOR VARYING OUTPUT
AS
SET @mycur=CURSOR SCROLL FOR
SELECT * from Sclad
OPEN @mycur
Return 0

Для запуска процедуры следует ввести команду **EXECUTE** в следующем виде:

EXECUTE [@код_возврата=] имя_процедуры [список_параметров] [**WITH RECOMPILE**]

Здесь **@код_возврата** (если указан) получает значение, указываемое в команде Return вызываемой процедуры.

Список параметров задается в виде **имя_параметра=значение** либо перечислением имен параметров. Однако при наличии задания в форме **параметр=значение** нельзя использовать альтернативный вариант.

WITH RECOMPILE – определяют необходимость повторной компиляции процедуры при вызове.

В качестве *примера* создадим процедуру, которая возвращает цену товара, название которого передается при вызове процедуры.

```
CREATE PROCEDURE pr2
@tovar varchar(40),
 @price int OUTPUT
AS
    BEGIN
    Use myDb
    SELECT @price=price from Sclad where tovar=@tovar
    Return 0
    END
```

Непосредственно под строкой

CREATE PROCEDURE pr2

указываются параметры процедуры, записываемые в строке вызова процедуры. Один из этих параметров – **@price** – определен как выходной (OUTPUT). Выходной параметр получает значение внутри процедуры.

Вызов этой процедуры может быть оформлен в следующем виде:

```
DECLARE @tovar varchar(40)
DECLARE @price int
SET @tovar='milk'
EXECUTE pr2 @tovar, @price OUTPUT
Print @price
```

В строке вызова процедуры перечисляются параметры, разделяемые запятой.

3.2.3.4 Использование курсоров

Для работы с курсором его необходимо прежде всего объявить.

Пример:

```
DECLARE mycur CURSOR FOR
SELECT * FROM [dbo].Sclad WHERE price>2000
FOR READ ONLY
```

Это объявление курсора запрещает изменять данные (FOR READ ONLY). Если курсор допускает обновление данных в оригинальной таблице, то вместо слов FOR READ ONLY можно использовать FOR UPDATE (эти слова можно не указывать, поскольку они действуют по умолчанию). Однако можно объявить курсор для обновления конкретных столбцов.

Пример:

DECLARE mycur CURSOR FOR
SELECT * **FROM** [dbo].Sclad **WHERE** price>2000
FOR UPDATE OF tovar, price

Здесь указываются в части названия полей, которые допускают изменение:
FOR UPDATE OF tovar, price
Открытие курсора выполняет команда
OPEN mycur.

После того как курсор открыт, можно выполнять выборку записей курсора и перемещение по ним. Выборка записей из курсора реализуется командой **FETCH**. Эта команда имеет следующий синтаксис:

FETCH [направление FROM] имя_курсора **INTO** список_переменных

Значение направления можно указывать так:
NEXT – выбирается следующая запись;
PRIOR – выбирается предыдущая запись;
FIRST – переход на первую запись;
LAST – переход на последнюю запись;
ABSOLUTE n/-n/0 – Если n > 0, то выполняется переход на запись с номером n от начала;

Если n<0, то выполняется переход на n-ую запись с конца набора; если n=0, то возвращается текущая запись.

RELATIVE n/-n. Производится переход на n записей вперед, если n > 0, или на n записей назад, если n < 0.

Пример:
CREATE PROCEDURE [dbo].st_a AS
DECLARE mycur **CURSOR FOR**
SELECT tovar, price **FROM** [dbo].Sclad **WHERE** price>2000
FOR READ ONLY
OPEN mycur
DECLARE @tovar VARCHAR(40)
DECLARE @price int
FETCH NEXT FROM mycur **INTO** @tovar, @price
PRINT @tovar +str(@price)
CLOSE mycur
DEALLOCATE mycur

Команда **CLOSE mycur** закрывает курсор. Команда **DEALLOCATE mycur** освобождает память, занятую курсором. Если курсор закрыт и находится в памяти, то его можно открыть повторно.

Теперь рассмотрим более сложную хранимую процедуру:
CREATE PROCEDURE [dbo].st_a AS
DECLARE @n int
DECLARE mycur CURSOR

```
SCROLL
FOR
SELECT tovar, price FROM [dbo].Sclad
FOR READ ONLY
OPEN mycur
Print 'selected rows='+str(@@CURSOR_ROWS)
DECLARE @tovar VARCHAR(40)
DECLARE @price int
SET @n=0
WHILE (@n<@@CURSOR_ROWS)
BEGIN
  SET @n=@n+1
  Print @n
  FETCH ABSOLUTE @n FROM mycur INTO @tovar, @price
  PRINT @tovar +str(@price)
END
CLOSE mycur
DEALLOCATE mycur
```

Для понимания этой процедуры следует иметь в виду, что при открытии курсора количество записей, выбираемых в него, определяется в переменной **@@CURSOR_ROWS**. Нумерация записей начинается с 1 (а не с 0!). Указатель записей курсора устанавливается на последнюю запись курсора. Для того чтобы курсор можно было просматривать в любом направлении, ему следует задать свойство SCROLL. Просмотр записей ведется в цикле. Переход к очередной записи выполняется по команде

```
FETCH ABSOLUTE @n FROM mycur INTO @tovar, @price
```

Опция ABSOLUTE ссылается на абсолютный адрес записи.

Выполнение и просмотр хранимой процедуры осуществляется с помощью утилиты QUERY ANALIZER, входящей в состав MS SQL Server 2000.

Приведем альтернативный вариант процедуры:

```
CREATE PROCEDURE [dbo].st_a  AS
DECLARE @n int
DECLARE mycur CURSOR
SCROLL
FOR
SELECT tovar, price FROM [dbo].Sclad
FOR READ ONLY

OPEN mycur
Print 'selected rows='+str(@@CURSOR_ROWS)
DECLARE @tovar VARCHAR(40)
DECLARE @price int
SET @n=0
WHILE (@n<@@CURSOR_ROWS)
```

```
BEGIN
  SET @n=@n+1
  Print @n
  FETCH NEXT FROM mycur INTO @tovar, @price
  PRINT @tovar +str(@price)
END
CLOSE mycur
DEALLOCATE mycur
```

Результат будет тем же.

В хранимой процедуре можно только открыть курсор и объявить его выходным параметром процедуры, а обработку курсора перенести по месту вызова. Текст процедуры в этом случае можно записать таким образом:

```
CREATE PROCEDURE [dbo].st_a AS
@mycur CURSOR VARYING OUTPUT
AS
  BEGIN
    SET @mycur= CURSOR STATIC FOR
    SELECT * FROM Sclad
    OPEN @mycur
    RETURN
END
```

Ключевое слово **STATIC** в объявлении курсора означает, что курсор создается в виде временной таблицы во временной системной БД (tempdb), так что действия с курсором не отражаются над состоянием оригинала. Статический курсор также не отслеживает изменения данных в источнике.

Для вызова этой процедуры можно использовать следующий код:

```
DECLARE @cur CURSOR
DECLARE @tovar varchar(40), @price int, @amount int

EXECUTE [dbo].st @cur OUTPUT
FETCH FIRST FROM @cur INTO @tovar, @price, @amount
Print @tovar+str(@price)+':'+str(@amount)
CLOSE @cur
```

В курсорах можно производить изменение данных. Для этой цели используется SQL-команда следующего вида:

```
UPDATE имя_таблицы_оригинала SET столбец=значение … столбец=значение
    WHERE CURRENT OF имя_курсора
```

В сравнении с формой представления обычной SQL-команды здесь добавлена строка

```
WHERE CURRENT OF имя_курсора
```

которая указывает, что изменение производится для текущей строки курсора. Перемещение по записям курсора по-прежнему реализуется с помощью команды **FETCH**.

Аналогичным образом производится удаление записи из таблицы и курсора. Используем команду

DELETE FROM имя_таблицы WHERE CURRENT OF имя_курсора

Итак, в **статических курсорах**, хранимых во временных таблицах, изменение в исходных таблицах никак не отражается. В **динамических курсорах**, напротив, изменение исходных таблиц приводит к изменению содержимого курсора. Аналогичным образом, изменение данных оригинала также проводится в динамический курсор. Наряду с динамическим и статическим курсорами имеются еще **курсоры, представляющие собой ключи записей**. В самом деле, имея ключ, можно прочитать и саму запись. Таким образом, нет необходимости размещать всю запись, а только ключевое поле.

3.2.4 Утилита QUERY ANALYSER

Данная утилита запускается через меню Пуск –> Программы–> SQL Server–> Query Analyser.

В окне редактора кода можно формировать и выполнять SQL-команды, создавать и запускать хранимые процедуры, триггеры, представления и пр. Например, рассмотрим результат выполнения запроса:

Use mydba

Select * from Sclad

Для выполнения этой группы SQL-команд следует набрать их в окне редактора и нажать на кнопку Execute Query или нажать клавишу F5.

Через меню Tools можно получить доступ к объектам и индексам БД. Например, чтобы открыть таблицу, следует выбрать пункт Tools –>Object Browser–>Show/Hide, а затем в дереве конфигурации можно открыть и просмотреть интересующую таблицу.

Для синтаксической проверки корректности записи SQL-команд в окне редактора следует использовать меню Query–> Parse или нажать клавиши **CTRL+F5** на клавиатуре.

С помощью утилиты Query Analyzer можно писать сценарии (**scripts**). Сценарии представляет несколько пакетов SQL-команды, разделенных командой **GO**. *Пример* сценария:

SELECT * **FROM** Склад

GO

SELECT * **FROM** Фирмы

В результате выполнения этого сценария будут выбраны сразу две результирующие таблицы.

Скрипты можно писать в обычных текстовых файлах, а затем грузить в окно редактора кода утилиты **QUERY ANALYZER**.

3.3 ЗАДАНИЕ

Написать хранимые процедуры для следующих видов запросов для таблицы Sclad со структурой, определенной в лабораторной работе №1. Предварительно построить базу данных и создать таблицу:

1) определение цены товара по переданному в процедуру названию товара;
2) определение числа товаров, цена которых не превосходит указанную в качестве параметра вызова процедуры;
3) вывести названия и цену товара, причем названия товара упорядочить по алфавиту;
4) определить самый дорогой товар;
5) вычислить налоги на выручку. Для вычисления налога использовать функцию IIF (налог считать так: если выручка от продажи товара больше 10000, то налог равен 20% от выручки, иначе – 13% от выручки);
6) написать процедуру, возвращающую названия всех товаров, выручка от реализации которых превосходит указанное значение передаваемого в процедуру параметра;
7) написать процедуру, изменяющую структуру таблицы Sclad путем добавления нового столбца, соответствующего дате поставки. В качестве даты использовать текущую дату для всех записей;
8) написать процедуру, изменяющую цену указанного товара на новую цену, также указываемую в качестве параметра;
9) вывести название фирмы, производящей самый дешевый товар;
10) продемонстрировать использование курсора для подсчета суммарной выручки от реализации всех товаров.

3.4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Каково назначение утилиты Enterprise Manager?
2 Каково назначение утилиты Query Analyzer?
3 Определите синтаксис хранимой процедуры в MS SQL Server?
4 Из чего состоит база данных в SQL Server?
5 Как объявляются курсоры в SQL Server?
6 Для чего создаются и используются учетные записи в MS SQL Server?
7 Объясните основные команды T-SQL (по выбору преподавателя).

ЛАБОРАТОРНАЯ РАБОТА №4

СВЯЗЬ С ВНЕШНЕЙ БАЗОЙ ДАННЫХ

4.1 ЦЕЛЬ РАБОТЫ

1) изучение возможности работы с внешними базами из среды программирования;
2) изучение возможностей работы с базой данных из VBA.

4.2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Вопрос о работе с внешней базой данных из какой-нибудь среды программирования по-прежнему актуален. В данной работе в качестве среды программирования используем VBA. Связь с базой осуществим в среде MicroSoft Word. Создадим новый документ Word и разместим на нем элементы управления: три текстовых поля и три кнопки (рисунок 4.1)

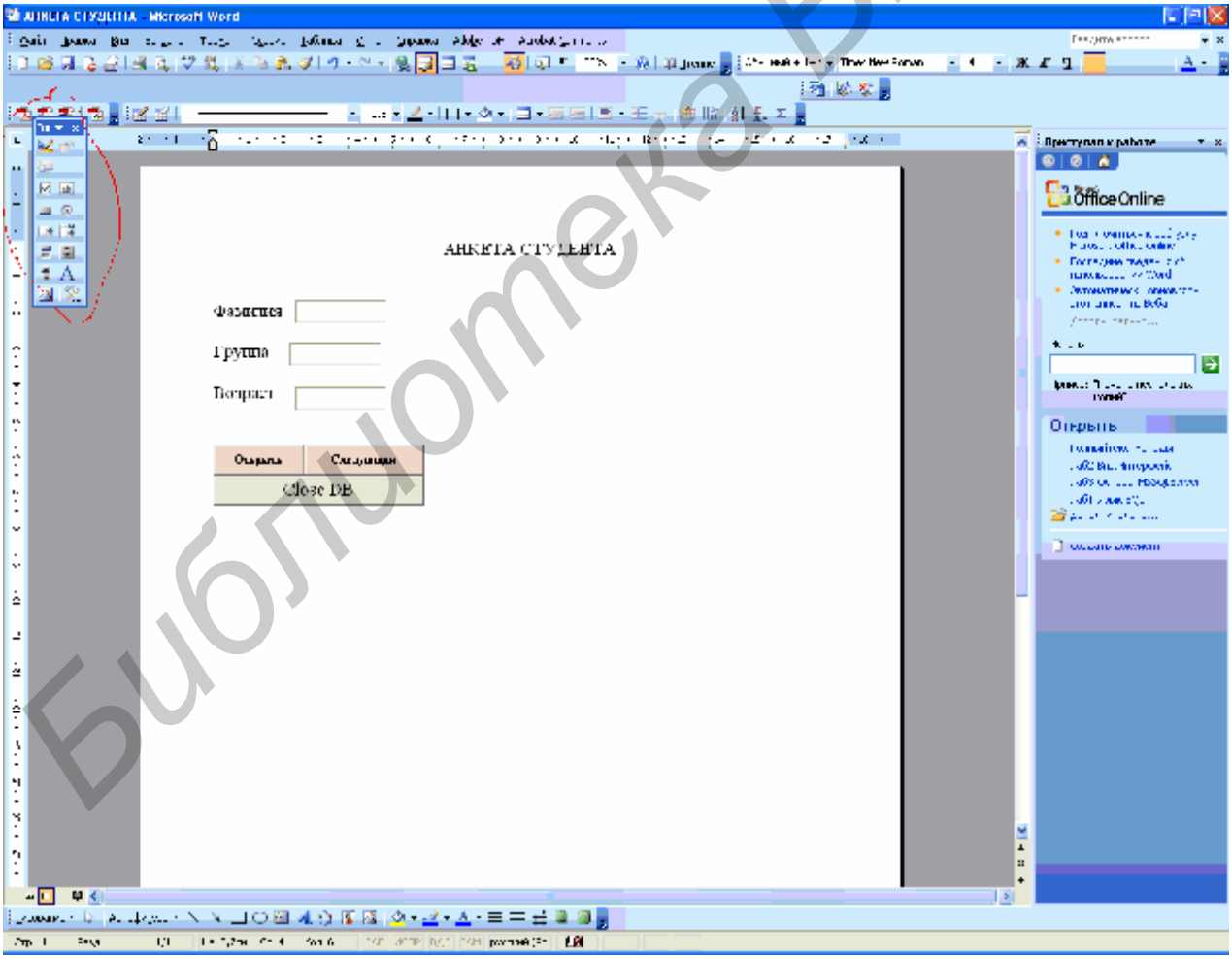


Рисунок 4.1 – Работа БД с помощью VBA

Для того чтобы разместить элементы управления – кнопки и текстовые поля, откроем соответствующую панель элементов через основное меню:

Вид –> Панель инструментов –> Элементы управления. Эта панель отмечена на рисунке 4.1 красной линией. Для того чтобы задать свойства элементов, щелкните правой кнопкой мыши на элементе и выберите в контекстном меню пункт Свойства. Для кнопок нужно задать их названия (поле **Caption** в окне свойств). Для текстовый полей нужно очистить содержимое – задать пустое значение в поле **Text**. Кроме того, можно отрегулировать тип и размер шрифта, используя свойство **Font**. Важно научиться свободно позиционировать элементы в документе. Для этого выделите элемент щелчком правой кнопки мыши и выберите из контекстного меню пункта Формат объекта. Откроется следующее окно (рисунок 4.2). Нажмите кнопку Дополнительно и выберите вариант сквозного размещения.

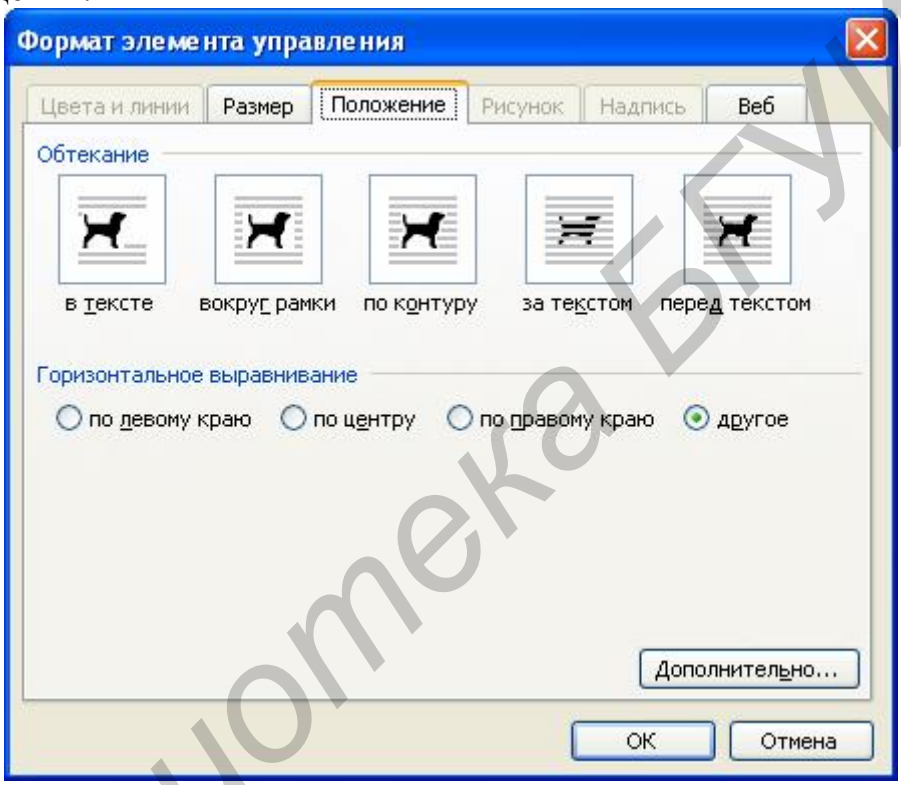


Рисунок 4.2 – Окно редактирования формата объекта

Теперь перейдем к программированию кнопок. Щелкните по кнопке «Открыть» дважды. Откроется окно редактора кода VBA. Введите следующий код:

```
Dim bd As DataBase
Dim rs As RecordSet
Dim opened As Boolean
Private Sub CommandButton1_Click()
If opened = True Then
 MsgBox "База уже открыта"
  Else
   'Set wrk = CreateWorkspace(Name:="", UseType:=dbUseJet)
    Set bd = OpenDatabase(Name:="d:\german\studdb.mdb", Options:=True)
    Set rs = bd.OpenRecordset("stud", dbOpenDynaset)
    s = rs.Fields("name").Value
```

```
    Documents(1).TextBox2.Value = s
    s = rs.Fields("group").Value
    Documents(1).TextBox1.Value = s
    s = rs.Fields("age").Value
    Documents(1).TextBox3.Value = s
    opened = True
End If
End Sub
```

Переменные в VBA объявляются в форме строк типа **Dim rs As RecordSet**. Здесь **rs** – переменная, а **RecordSet** – ее тип. Для работы с базами данных в VBA имеется несколько библиотечных классов. Класс нам еще предстоит подключить, так что объявления

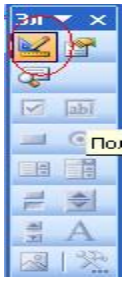
```
Dim bd As DataBase
Dim rs As RecordSet
Dim opened As Boolean
```

пока будут не понятны системе. Нам требуется объект «база данных» (тип DataBase), объект «Набор Записей» (тип RecordSet). Объект DataBase позволяет подключиться к существующей базе, в качестве которой в данной работе мы используем базу данных Access с именем studdb.mdb. Открытие базы выполняется в строке

```
Set bd = OpenDatabase(Name:="d:\german\studdb.mdb", Options:=True)
```

Открытие набора записей (читай – таблицы) выполняется в строке

```
Set rs = bd.OpenRecordset("stud", dbOpenDynaset)
```



Также используется переменная **opened**, чтобы проверять, открыта база или нет. Итак, подключим нужную библиотеку классов. Это производится через окно редактора VBA (в это окно, заходят через двойной щелчок на кнопке). **Внимание!** Для входа в окно VBA надо перевести элемент управления в режим конструирования щелком на элементе управления, изображающем линейку (рисунок 4.3).

Рисунок 4.3 – Панель инструментов

В окне VBA выберите пункт меню **Tools – References – MicroSoft DAO 3.6**. Пометьте последний элемент галочкой и нажмите OK. DAO и есть нужный нам класс для работы с базой данных. Другим важным классом является OLE DB, но здесь он не рассматривается.

При открытии базы данных нам нужно поместить в текстовые поля фамилию студента, его группу и возраст. Например, фамилия заносится следующим образом:

```
    s = rs.Fields("name").Value
    Documents(1).TextBox2.Value = s
```

Сначала переменная s получает значение непосредственно из текущей записи из поля name (обратите внимание на то, как мы получаем значение этого поля. Данный способ является общим для других полей также). Строка

Documents(1).TextBox2.Value = s показывает, как занести значение из поля в элемент управления с именем TextBox2. Обратим внимание на то, что значение присваивается свойству Value. Кроме того, перед именем элемента указываем документ, к которому данный элемент принадлежит, – Documents(1) (нумерация в семействе начинается с 1, а не с 0).

Можно произвести выборку в базу с помощью запроса SQL. Это делается следующим образом:

```
Private Sub CommandButton11_Click()
If opened = True Then
 MsgBox "База уже открыта"
  Else
   'Set wrk = CreateWorkspace(Name:="", UseType:=dbUseJet)
    Set bd = OpenDatabase(Name:="d:\german\studdb.mdb", Options:=True)
    Set rs = bd.OpenRecordset("Select * from stud", dbOpenDynaset)
    s = rs.Fields("name").Value
    Documents(1).TextBox21.Value = s
    s = rs.Fields("group").Value
    Documents(1).TextBox11.Value = s
    s = rs.Fields("age").Value
    Documents(1).TextBox31.Value = s
    s = rs.Fields("foto").Value
    Documents(1).Image1.Picture = LoadPicture("d:\german\" + s, 100, 100, Color)
    opened = True
End If
End Sub
```

Мы изменили только одну команду

```
   Set rs = bd.OpenRecordset("Select * from stud", dbOpenDynaset)
```

Итак, работа кнопки «Открыть» подробно рассмотрена.

Перейдем теперь к кнопке «Следующая». Ее код помещен ниже:

```
Private Sub CommandButton2_Click()
If opened = True Then
  If rs.EOF() = False Then
     rs.MoveNext
     Else
     rs.MoveFirst
     End If
    If rs.EOF() = False Then
    s = rs.Fields("name").Value
    Documents(1).TextBox2.Value = s
    s = rs.Fields("group").Value
    Documents(1).TextBox1.Value = s
    s = rs.Fields("age").Value
```

```
   Documents(1).TextBox3.Value = s
  Else
    rs.MoveFirst
    End If
  Else
  MsgBox "База закрыта"
  End If
End Sub
```

Переход на следующую запись выполняет команда

```
rs.MoveNext
```

Однако прежде чем переходить, необходимо убедиться, что не достигнута последняя запись. Для этой цели используем проверку:

```
If rs.EOF() = False Then ….
```

Если достигли последней записи, то переходим на первую запись:

```
 Else
    rs.MoveFirst
```

Обратим внимание на то, что требуется вторая проверка достижения конца, такова специфика VBA.

Наконец, кнопка **Закрыть** (Close) запрограммирована следующим образом:

```
Private Sub CommandButton3_Click()
If opened = True Then
   rs.Clone
   bd.Close
   Set rs = Nothing
   opened = False
Else
End If
   Documents(1).TextBox1.Value = ""
   Documents(1).TextBox2.Value = ""
   Documents(1).TextBox3.Value = ""
End Sub
```

В заключение рассмотрим отображение рисунков. Пусть в таблице stud имеется для каждого студента адрес рисунка (JPEG). Рисунки подготовим в PaintBrush. Структура таблицы показана на рисунке 4.4.

stud : таблица

	name	group	age	foto
	petrov	2	18	st1.jpg
	ivanov	1	21	st2.jpg
	sidorov	2	19	st3.jpg
	iovik	1	19	st1.jpg
*		0	0	

Рисунок 4.4 – Окно таблицы Stud

Разместим на документе элемент **Image** (изображение) – рисунок 4.5.

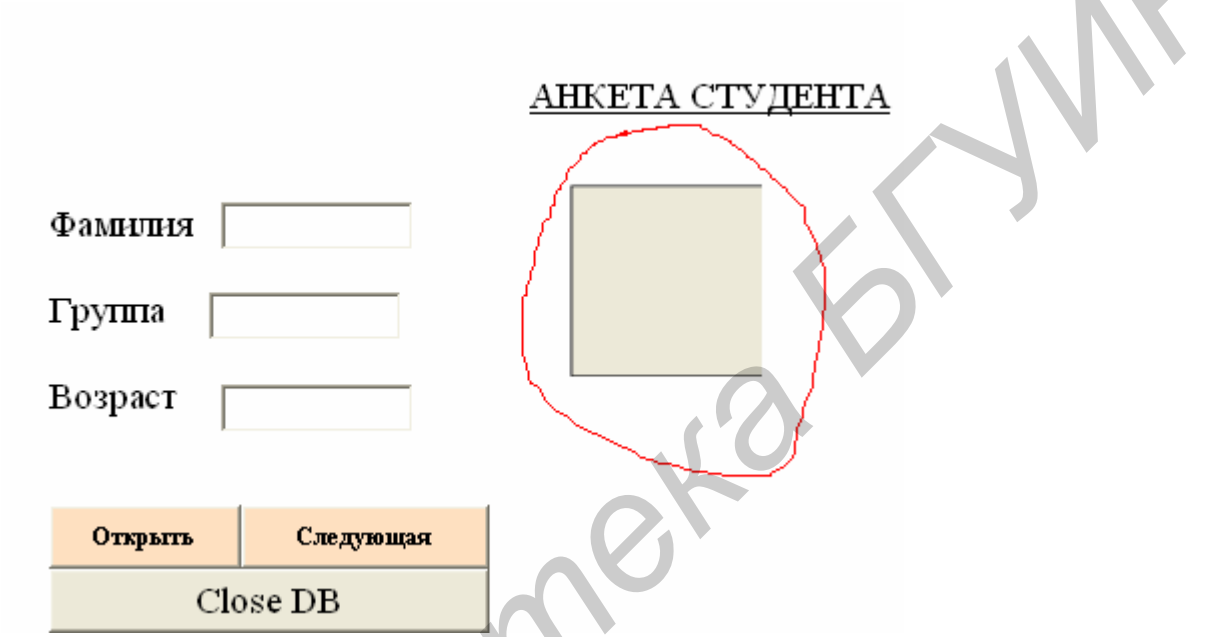


Рисунок 4.5 – Отображение рисунков

При навигации по базе данных требуется загружать в этот элемент указанный в базе файл с рисунком. В связи с этим код для загрузки базы (аналогично для перемещения по базе) изменяем следующим образом:

```
Private Sub CommandButton1_Click()
If opened = True Then
 MsgBox "База открыта"
  Else
    Set bd = OpenDatabase(Name:="d:\german\studdb.mdb", Options:=True)
   Set rs = bd.OpenRecordset("stud", dbOpenDynaset)
   s = rs.Fields("name").Value
   Documents(1).TextBox21.Value = s
   s = rs.Fields("group").Value
   Documents(1).TextBox11.Value = s
```

```
    s = rs.Fields("age").Value
    Documents(1).TextBox31.Value = s
    s = rs.Fields("foto").Value
    Documents(1).Image1.Picture = LoadPicture("d:\german\" + s)
     opened = True
End If
End Sub
```

Результат загрузки показан на рисунке 4.6.

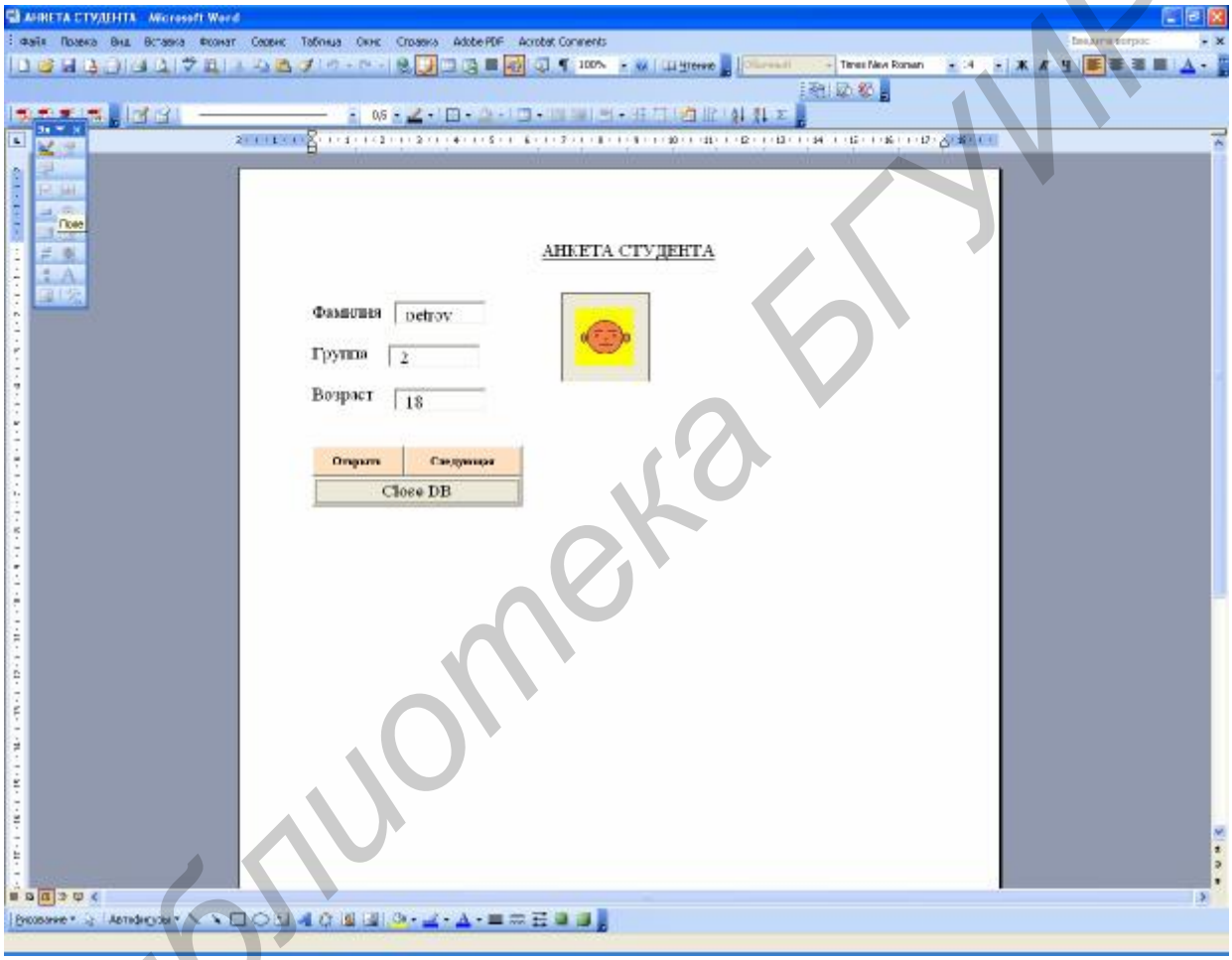


Рисунок 4.6 – Результат загрузки рисунка

Поиск записи выполняется с помощью инструкции вида

```
rs.FindFirst "name='" + Trim(Documents(1).TextBox1.Text) + "'"
  If rs.NoMatch = True Then
  rs.MoveFirst
  End If
```

Здесь выполняется поиск по имени студента. В команде поиска нужно задать значение имени так:

name='petrov'

Внимание! Имя должно быть взято в одиночные кавычки.

Успешность поиска проверяют команды

```
If rs.NoMatch = True Then
  rs.MoveFirst
  End If
```

Теперь выясним, как добавить новую запись или отредактировать имеющуюся.

Добавление новой записи выполняем так:

```
  rs.AddNew
  rs.Fields("age").Value=  Documents(1).TextBox2.Value
  rs.Update
```

Здесь многоточие заменяет однотипные команды для записи значений в поля таблицы.

Редактирование текущей записи выполняем таким образом:

```
rs.Edit
 rs.Fields("age").Value=  Documents(1).TextBox2.Value
  rs.Update
```

Команда rs.Update отправляет изменения в существующую базу на диске.
Удаление текущей записи выполняется по команде

```
rs.Delete
```

4.3 ЗАДАНИЕ

Расширить возможности программы, добавив кнопки для поиска, изменения, добавления и сохранения записи.

Реализовать SQL запросы для выборки студентов по возрасту, группе, успеваемости (в последнем случае ввести в таблицу поле успеваемости).

Добавить в базу вторую таблицу – Библиотека, в которой отразить литературу, взятую студентами из библиотеки. Выполнить поиск из двух таблиц. Для этого нужно реализовать SQL-запрос к двум таблицам.

4.4 КОНТРОЛЬНЫЕ ВОПРОСЫ

1 Какие объекты VBA используются для работы с базами данных?
2 Как выполняется навигация по базе данных?
3 Как открыть базу данных с помощью SQL-запроса?
4 Как выполнить изменение, добавление новых записей?

Св. план 2009, поз.45

Учебное издание

Герман Олег Витольдович
Тиханович Татьяна Викторовна

БАЗЫ И БАНКИ ДАННЫХ

Лабораторный практикум для студентов специальности
«Автоматизированные системы обработки информации»
дневной и дистанционной форм обучения

Редактор Т. П. Андрейченко
Корректор Е. Н. Батурчик
Компьютерная верстка Е. С. Чайковская

Подписано в печать 15.04.2009. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 4,07.
Уч.-изд. л. 3,8. Тираж 100 экз. Заказ 162.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0494371 от 16.03.2009. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6