Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования
«Горно-Алтайский государственный университет»

Н. Г. Кудрявцев, Д. В. Кудин, М. Ю. Беликова

РАБОТА С БАЗАМИ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ VBA MS EXCEL

(Учебное пособие)

Горно-Алтайск
РИО Горно-Алтайского госуниверситета
2015

Печатается по решению редакционно-издательского совета
Горно-Алтайского государственного университета

УДК 004
ББК 32.973.26-018.2я73
К 88

Кудрявцев Н.Г., Кудин Д. В., Беликова М. Ю.
Работа с базами данных с использованием VBA MS Excel: учебное пособие / Н. Г. Кудрявцев, Д. В. Кудин, М. Ю. Беликова – Горно-Алтайск: РИО ГАГУ, 2015. – 101 с.

Рецензенты :

Власов В.Н., к. ф.-м. н., доцент кафедры общей информатики факультета информационных технологий Новосибирского государственного университета.

Кречетова С. Ю., к. ф.-м. н., доцент, заведующий кафедрой математики и информатики Горно-Алтайского государственного университета.

Пособие состоит из двух частей. Первая, теоретическая часть, содержит элементарные сведения о реляционных базах данных и включает в себя обзор основных конструкций языка структурированных запросов SQL. Вторая, практическая часть, состоит из многочисленных примеров демонстрирующих возможности использования языка VBA MS Excel при реализации технологии взаимодействия с базами данных. Пособие может использоваться в качестве основной учебной литературы при изучении дисциплин «Информатика и программирование» и «Базы данных» (230700 «Прикладная информатика»), «Базы данных» и «Практикум на ЭВМ» (01.03.01 «Математика», 02.03.01 «Математика и компьютерные науки»)

ОГЛАВЛЕНИЕ

1. Введение

Ни одна серьезная организация, ни один серьезный проект не могут существовать без обработки, большого количества самой разнообразной информации. Это могут быть экономические показатели предприятия, ЖКХ, муниципального образования или целого региона. Это могут быть данные по бухгалтерии или массивы информации от научного оборудования, измеряющего состояние погоды, а могут быть и показания приборов учета расхода воды, тепла, электроэнергии. Все, что связано с хранением, обработкой, поиском больших объемов информации имеет прямое или косвенное отношение к базам данных или, точнее, к системам управления базами данных (СУБД). Исследованию баз данных, аппаратному и программному обеспечению, связанному с СУБД посвящено большое количество самой разнообразной литературы, начиная от фундаментальных исследований, заканчивая сравнительными обзорами и перспективными прогнозами.

Задача данного учебного пособия заключается в том, чтобы, дав студентам набор начальных знаний о теоретических основах реляционных базах данных, показать как на практике, используя такие общедоступные инструменты, как VBA и MS Excel, научиться автоматизировать процессы взаимодействия с базами данных. В данном учебном пособии на многочисленных примерах показано, как в программе на VBA создавать базы данных, создавать, удалять и редактировать таблицы, создавать и удалять связи между таблицами, работать с ключевым полями и индексами. В книге также приведены примеры программ, реализующих работу с базами данных.

2. Базы данных. СУБД

База данных (БД) – поименованная совокупность данных, относящаяся к определенной предметной области.

СУБД – комплекс программных и языковых средств, необходимых для создания, обработки и поддержании баз данных в актуальном состоянии. СУБД имеют средства для эффективного хранения информации, средства защиты данных от случайной потери/порчи, средства экономного использования ресурсов, средства быстрого поиска информации.

Примеры некоторых СУБД: MS Access, MS SQL Server, Oracle, IBM DB/2, Interbase, Informix, MySQL, SQLite.

Некоторая терминология баз данных:

Предме*тная область* – часть реального мира, информация о которой моделируется средствами баз данных.

Модель данных – это описание предметной области. При разработке базы данных рассматривают несколько моделей данных, но перед началом разработки всегда необходимо определить сущности, информация о которых будет храниться в базе данных, и их свойства (атрибуты). Сущностями обычно называют классы или совокупности объектов, имеющих одинаковое назначение, обладающих одинаковыми свойствами. Например, класс – Студенты; класс – Преподаватели; класс – Родители; класс – Автомобили; класс – Бытовая техника. У каждой сущности есть какие-то характерные признаки, по которым один экземпляр сущности можно отличить от другого экземпляра сущности, такие признаки или свойства называют атрибутами. Например, автомобили отличаются друг от друга по марке автомобиля, объему двигателя, заводу производителю, году выпуска, цвету и.т.п. Студенты могут характеризоваться (отличаться друг от друга), фамилией, именем, отчеством, годом рождения, специальностью, номером группы в которой обучаются, городом из которого приехали и т.п. Характерные или отличительные признаки могут выражаться простыми словами (цвет - красный), а могут являть собой и целые сущности (фирма производитель - Дженерал моторс, Тойота, Фольксваген), которые имеют свои наборы характеризующих их атрибутов. Поэтому модель данных (описание предметной области) должна содержать в себе кроме простого набора сущностей, относящихся к искомой области, еще и связи между некоторыми или всеми сущностями рассматриваемого набора.

Для предварительной проработки структуры связей между сущностями в базе данных используют так называемую модель ER-диаграмм (Entity-

Relationship) Сущность – связь. В некоторых источниках эту модель еще называют логической моделью данных. Концепция данной модели заключается в том, что сущности рассматриваются как дискретные объекты, между которыми устанавливаются логические связи

Например, для базы данных Бюро путешествий можно выделить следующие сущности: Туристы, Путевки, Сезон, Туры, Оплаты.

На следующем шаге определяют логические связи

Каждый турист может купить одну или несколько путевок

- Каждая путевка может иметь одну или несколько оплат (оплата в кредит)
- Каждый тур имеет несколько сезонов
- Путевки продаются на один тур одного сезона

В итоге получается схема, представленная на Рис.2.1:

Рис.2.1

Схема базы данных (в некоторых источниках ее называют физической моделью) – это перевод абстрактной (концептуальной или логической) модели данных на язык базы данных. Например, в реляционных базах данных каждой сущности соответствует таблица или представление, где столбцам соответствуют атрибуты сущности, а строкам - экземпляры сущности. В схему также входят связи между сущностями. В некоторых источниках физическую модель связывают с конкретной СУБД.

База данных – это совокупность схемы и самих данных.

Ядро базы данных (database engine) – это программный механизм, фактически реализующий СУБД. Данный механизм *обеспечивает физическое*

манипулирование данными: хранение на диске и извлечение по запросу; работу с базой данных для приложений и пользователей, например, ядро Jet, ядро SQL Server и т.п.

Объектные модели (ADO, ADO.NET, RDO, DAO и т.п.) – это наборы COM (Component Object Model) объектов, которые используются для упрощения доступа к данным в базах данных из приложений. Напрямую с базами данных через API (Application Programmin Interface – интерфейс прикладного программирования) работать неудобно, поэтому эти объектные модели используются очень широко.

COM (англ. Component Object Model) — объектная модель компонентов; технологический стандарт от компании Microsoft, предназначенный для создания программного обеспечения на основе взаимодействующих компонентов, каждый из которых может использоваться во многих программах одновременно. Стандарт воплощает в себе идеи полиморфизма и инкапсуляции объектно-ориентированного программирования.

Многие современные БД используют реляционные модели данных. Relation – отношения и состоят из набора таблиц. Реляционная модель – не единственная, которую можно использовать при работе с данными.

Существуют также иерархическая модель, сетевая модель, звездообразная модель, на современном этапе развития информационных технологий набирает популярность объектная модель. Однако реляционная модель оказалась на определенном этапе наиболее удобной и поэтому используется до сих пор наиболее широко. Перед тем как подробнее рассмотреть принципы реляционной модели, скажем несколько слов о других моделях представления данных.

Иерархическая модель данных — представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней.

Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка (объект более близкий к корню) к потомку (объект более низкого уровня), при этом возможна ситуация, когда объект-предок не имеет потомков или имеет их несколько, тогда как у объекта-потомка обязательно только один предок. Объекты, имеющие общего предка, называются близнецами (в программировании применительно к структуре данных дерево устоялось название

Например, если иерархическая база данных содержала информацию о покупателях и их заказах, то будет существовать объект «покупатель» (родитель) и объект «заказ» (дочерний). Объект «покупатель» будет иметь указатели от каждого заказчика к физическому расположению заказов покупателя в объект «заказ».

В этой модели запрос, направленный вниз по иерархии, прост (например: какие заказы принадлежат этому покупателю); однако запрос, направленный вверх по иерархии, более сложен (например, какой покупатель поместил этот заказ). Также, трудно представить не иерархические данные при использовании этой модели.

Иерархической базой данных является файловая система, состоящая из корневого каталога, в котором имеется иерархия подкаталогов и файлов.

Разница между иерархической моделью данных и сетевой состоит в том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься любое число предков.

Сетевая БД состоит из набора экземпляров определенного типа записи и набора экземпляров определенного типа связей между этими записями.

Сетевая модель была первым подходом, использовавшимся при создании баз данных в конце 50-ых – начале 60-ых годов. Активным пропагандистом этой модели был Чарльз Бахман. Главным конкурентом тогда у нее была иерархическая модель данных, представленная ведущим продуктом компании IBM в области баз данных – IBM IMS. В конце 60-ых годов Эдгаром Коддом была предложена реляционная модель данных и после долгих и упорных споров с Бахманом реляционная модель приобрела большую популярность и теперь является доминирующей на рынке СУБД.

Объектно-ориентированная база данных (ООБД) — база данных, в которой данные моделируются в виде объектов, их атрибутов, методов и классов. Объектно–ориентированные базы данных обычно рекомендованы для тех случаев, когда требуется высокопроизводительная обработка данных, имеющих сложную структуру.

Звездообразная модель состоит из центральной таблицы фактов (fact table), которая окружена несколькими таблицами измерений (dimension table). Физически таблица фактов часто представляет собой несколько секционированных таблиц. Отношения между таблицей фактов и таблицами измерений должны быть простыми, чтобы существовал только один

возможный путь соединения любых двух таблиц и чтобы смысл этого соединения был очевиден и хорошо понятен

3. Реляционные базы данных

Реляционная модель была разработана сотрудником фирмы IBM Эдгаром Франком Коддом (Edgar Frank Codd) и опубликована в 1970 г в работе «A Relational Model of Data for Large Shared Data Banks». Реляционная модель определяет способ представления данных (структуру данных), методы обеспечения целостности данных, и операции, которые можно выполнять с данными.

В начале 80-х годов прошлого столетия реляционная модель начала входить в моду. В 2002 журнал Forbes поместил реляционную модель данных в список важнейших инноваций последних 85 лет.

Основные принципы реляционных баз данных:

- каждой сущности в реляционной модели данных соответствует отношение (*relation*) – упорядоченная организация данных, определенная в виде строк и столбцов. Чаще вместо слова «отношение» – используют просто слово таблица («набор записей», или набор результатов – result set). Именно от этого и происходит термин «реляционные базы данных»;
- каждому экземпляру сущности в отношении (таблице) соответствует кортеж или запись (строка таблицы);
- мощность отношения – число кортежей в отношении (проще говоря, число записей или строк в таблице);
- каждому атрибуту сущности соответствует столбец или поле в отношении (таблице);
- размерность – это число атрибутов в отношении;
- каждое отношение можно разделить на две части – заголовок и тело. На простом языке заголовок отношения – это список (названия) столбцов (полей), а тело – это сами записи (кортежи);
- каждый атрибут характеризуется таким понятием как «домен» или набор данных (тип данных). Понятие Домен можно интерпретировать как множество из которого «черпают» свои значения атрибуты.

Для того, чтобы не возникла путаница, еще раз акцентируем внимание на понятиях

- *Relation* – Отношение
- *Relationship* – Схема данных (схема связей)

В теории реляционных баз данных вводят реляционную алгебру – набор операций над отношениями, в результате которых получаются новые отношения. Реляционная алгебра в данной работе рассматриваться не будет.

3.1. Связи между отношениями

Как мы уже отмечали выше, сущности, информация о которых хранится в базе данных, могут характеризовать другие сущности. Т.е. сущность автомобиль может характеризоваться сущностью завод производитель или сущностью компания поставщик. Другими словами, одни сущности должны быть связаны с другими сущностями. Поэтому появляется необходимость каким-то образом связывать таблицы, из которых состоит база данных. Такой способ виртуального связывания таблиц существует и обеспечивается наличием общих для связываемых таблиц ключей (ключевых полей). С одной стороны (для одной таблицы) это основные или первичные ключи (*primary key*), а с другой стороны – внешние ключи или (*foreign key*). Отметим, что основной ключ (*primary key*) обязательно должен быть уникальным, т.е. в этом поле не должно быть повторяющихся значений. Также следует отметить, что в качестве основного ключа может выступать некоторая совокупность простых полей.

Структуры связей между таблицами принято делить на связи 1:1, 1:M, M:1, M:M.

Связь между таблицами называют связью 1:1 в том случае, если одной записи в одной таблице соответствует 1 запись в другой таблице. В качестве примера можно привести связь таблицы «студенты» и таблицы «зачетная_книжка». Одному студенту однозначно соответствует одна зачетная книжка. Обычно такие типы связи на практике реализуются путем добавления к искомой таблице дополнительного поля. В некоторых случаях такую связь все же реализуют для автоматического приведения таблиц ко второй нормальной форме (речь о нормальных формах пойдет ниже).

Часто используемыми типами связей между таблицами являются 1:M – один ко многим или M:1 - многие к одному. Такой тип связи используется в том случае, когда одной записи в одной таблице соответствует много записей в другой таблице. Например, у одного студента может быть несколько телефонных номеров (телефоны разных операторов связи). В этом случае в таблице «Телефон» (Таблица «Tel_1») создается поле «Key_stud», которое является внешним ключом для данной таблицы. В это ключевое поле заносятся значения поля «Key_stud», первичного ключа таблицы «студенты» (Таблица «Stud_1»), соответствующие экземпляру сущности «студент», имеющему

данный телефонный номер. Таким образом следует помнить, что внешний ключ добавляется в ту таблицу, которая соответствует букве *M* в схеме связей.

Таблица «Stud_1»

key_stud	fio_stud
1	Петров В.В.
2	Сидоров А.А.
3	Иванов С.С.

Таблица «Tel_1»

key_tel	num_tel	komp_name_tel	key_stud
1	89139991145	МТС	1
2	89039191789	Билайн	1
3	89136911117	МТС	3

Другой пример аналогичной связи – связь между таблицей «города» (Таблица «Town») и таблицей «студенты» (Таблица «Stud_2»), показывающей из какого города приехал данный студент. В этом случае буква *M* в схеме связей относится к сущности «студент» (из одного города могут приехать много студентов) внешний ключ «key_town» добавляется в таблицу «студенты» и для каждого экземпляра сущности «студент» в него заносится значение поля «key_town», соответствующее городу, из которого приехал данный студент.

Таблица «Stud_2»

key_stud	fio_stud	key_town
1	Петров В.В.	1
2	Сидоров А.А.	2
3	Иванов С.С.	2

Таблица «Town»

key_town	name_town	reg_town
1	Анапа	Россия
2	Горно-Алтайск	Россия
3	Майма	Россия

Еще один распространенный тип связей – связь многие ко многим *M:M*. Т.е одной записи в таблице А может соответствовать несколько записей в таблице В и наоборот одной записи в таблице В может соответствовать несколько записей в таблице А. В качестве примера можно привести связи между сущностями «студенты» (Таблица «Stud_3») и «родители» (Таблица «Par»). Каждому экземпляру сущности «студенты» может соответствовать несколько экземпляров сущности «родители». Другими словами, у студента

может быть отец, мать, дедушки и бабушки. В то же время у отца или матери могут быть несколько детей, которые учатся в данном учебном заведении. Для реализации связи M:M используют внешнюю таблицу (Таблица «Stud_Par»), содержащую обычно три поля: первичный ключ самой таблицы и внешние ключи связываемых таблиц.

Таблица «Stud_3»

key_stud	fio_stud
1	Петров В.В.
2	Сидоров А.А.
3	Иванов С.С.
4	Петрова И.В.

Таблица «Par»

key_par	fio_par	status_par
1	Петров В.П.	Отец
2	Сидоров А.С.	Дед
3	Иванова С.Г.	Мать
4	Иванова И.П.	Бабушка

Таблица «Stud_Par»

key_stud_par	key_stud	key_par
1	1	1
2	4	1
3	2	2
4	3	3
5	3	4

3.2. Функциональная зависимость и нормальные формы

При создании реляционной базы данных стараются привести таблицы (отношения) к так называемым нормальным формам. Свойства нормальности — это свойства отношения, характеризующие его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение. Процесс преобразования отношений базы данных (БД) к виду, отвечающему нормальным формам, называют нормализацией. Для уточнения можно добавить, что отсутствие логической избыточности обеспечивается путем исключения различных функциональных зависимостей между атрибутами отношения.

О *функциональной зависимости* между атрибутами можно говорить в том случае, если значение одного атрибута определяет значение другого атрибута. Так как по определению первичный ключ (*primary key*) однозначно определяет экземпляр сущности, то от него зависят и конкретные значения атрибутов этой сущности (присущие данному экземпляру). Например у студента с первичным ключом 123 однозначно определены имя – Петр, отчество, Петрович, фамилия – Иванов и год рождения – 1997 (т.е. у одного студента не может быть двух имен, отчеств и дней рождения)

Но бывают случаи, когда какой-то атрибут уникально не определяет значение одного атрибута, но ограничивает его набором нескольких значений. Например, у одного студента может быть несколько телефонов. В этом случае говорят о многозначной зависимости между первичным ключом и номерами телефонов. Для того, чтобы исключить многозначную зависимость «многозначно–зависимый» атрибут выделяют в отдельную сущность и применяют связь один ко многим (буква М относится к выделенному атрибуту).

Также существует так называемая частичная зависимость. Данная зависимость возникает тогда, когда помимо или вместо основного ключа в отношении существует составной ключ – несколько атрибутов, которые однозначно определяют экземпляр сущности. Например, в таблице работник, состоящей из атрибутов: Основной_ключ_работника, ФИО, Должность, Зарплата, Наличие_компьютера, помимо Основного_ключа_работника однозначно определяют экземпляр сущности еще два атрибута ФИО и Должность. Говорят, что в отношении существует частичная зависимость, если какой-то атрибут зависит от части составного ключа. Так, в данном примере Наличие_компьютера зависит только от Должности (компьютер нужен бухгалтеру, но не техничке, вне зависимости от личности), однако Зарплата определяется как Должностью (всегда существует вилка окладов), так и ФИО (например, техничке тете Маше директор может назначить заплату несколько больше, чем техничке тете Вале). Поэтому в случае Зарплаты частичной зависимости нет, а в случае Наличия_компьютера – есть. Для исключения частичной зависимости «частично-зависимый» атрибут вместе с атрибутом – частью составного ключа, от которого он зависит, выделяют в отдельную сущность. В рассматриваемом примере должно получиться два отношения: Основной_ключ_работника, ФИО, Внешний_ключ_должности, Зарплата и Основной_ключ_должности, Должность, Наличие_компьютера. Связь М:1 - многие работники могут иметь должность с одним названием.

Еще один тип зависимости, который выделяют особо – это транзитивная зависимость. Данный тип зависимости проявляется, когда какой-то атрибут зависит от первичного ключа не напрямую а через другой атрибут. Например это случается, когда в базу данных вносят так называемые «вычисляемые атрибуты» т.е атрибуты, значения которых можно получить из других атрибутов путем некоторых преобразований. Другими словами, если изменение значения какого–то атрибута обязательно влечет необходимость изменения другого атрибута, то говорят от транзитивной зависимости. Например, иногда хранят Фамилию и Фамилию в родительном падеже в одной таблице. Исправление ошибки в фамилии должно повлечь и исправлении ошибки в фамилии в родительном падеже. Для того, чтобы убрать такой тип зависимости вычисление значений «вычисляемых атрибутов» выполняют либо в хранимых процедурах, либо в представлениях при помощи агрегирующих функций языка SQL.

Полное отсутствие или частичное исключение перечисленных выше зависимостей определяются так называемыми правилами нормировки или приведением к нормальным формам.

Итак, говорят, что таблица (отношение) находится в первой нормальной форме *(1NF),* если она соответствует следующим требованиям:

- Строки неупорядочены
- Столбцы неупорядочены
- Нет повторяющихся строк
- Значения ячеек (пересечений строк и столбцов) атомарны.

При этом, под атомарностью понимается отсутствие в данной ячейке каких-то структурированных данных, или, если разбиение значения ячейки на части ведет к потере смысла содержимого ячейки. Другими словами атрибут атомарен, если его значение теряет смысл при любом разбиении на части или переупорядочивании. Следовательно, если какой-либо способ разбиения на части не лишает атрибут смысла, то атрибут неатомарен. Хорошим способом принятия решения о необходимости разбиения атрибута на части является вопрос: «будут ли части атрибута использоваться по отдельности?». Если да, то атрибут следует разделить (но так, чтобы сохранились осмысленные части атрибута)

По большому счету, атрибут Ф.И.О. в наших примерах сложно назвать атомарным, но в последующих учебных примерах мы будем считать его

атомарным для экономии времени и ресурсов (в классических реальных БД вместо Ф.И.О. должны быть три атрибута Фамилия, Имя, Отчество).

Говорят, что таблица (отношение) находится во второй нормальной форме (*2NF*), если она соответствует требованиям первой нормальной формы и между ее атрибутами отсутствует частичная функциональная зависимость. Вторая нормальная форма по определению запрещает наличие неключевых атрибутов, которые вообще не зависят от потенциального ключа. Таким образом, вторая нормальная форма запрещает создавать отношения как несвязанные (хаотические, случайные) наборы атрибутов. Например в таблице Студент не должно быть поля количество осадков за день.

Говорят, что таблица (отношение) находится в третьей нормальной форме **(*3NF*)**, если она соответствует требованиям второй нормальной формы и между ее атрибутами отсутствует транзитивная функциональная зависимость. Другими словами, для обеспечения третьей нормальной формы желательно включать в таблицу только те поля, которые непосредственно (не через какую-то другую сущность) характеризуют искомый экземпляр сущности. Например, экземпляр сущности Студент непосредственно характеризуется своими ФИО, ключевым полем, указывающим на родителей, датой своего рождения, ключевым полем, указывающим на специальность по которой данный студент обучается. Однако, набор изучаемых дисциплин является уже характеристикой не студента, а специальности, которую студент для себя выбрал, а фамилии и имена преподавателей, являются характеристиками дисциплин, которые соответствуют специальности. Также профессия и доходы родителей не являются непосредственной характеристикой студента. Если преподавателей и названия дисциплин внести в таблицу Студенты, то данное отношение не будет соответствовать 3NF

Говорят, что таблица (отношение) находится в четвертой нормальной форме **(4*NF*)**, если она соответствует требованиям третьей нормальной формы и между ее атрибутами отсутствует многозначная зависимость.

4. Архитектура баз данных и Visual Basic

Приложение *Visual Basic* для работы с базами данных содержит три части, которые показаны на Рис.4.1

Интерфейс пользователя
Ядро базы данных
Хранилище данных

Рис.4.1

Повторим определение ядра базы данных, которое мы давали выше: *Ядро базы данных (database engine)* - это программный механизм, реализующий СУБД.

Ядро базы данных является прослойкой между пользовательской программой и физическими файлами базы данных. Это дает независимость от конкретной базы данных. Будь то собственная база данных *VB*, либо база, обладающая поддерживаемыми форматами, можно использовать одинаковые объекты для доступа к данным и одни и те же приемы программирования.

Интерфейс пользователя – прикладная программа. Это то, что пользователь видит: формы, позволяющие пользователю рассматривать данные, обновлять и удалять их. Программа управляет этими формами, используя объекты доступа к данным и методы их вызова. Запросы направляются не к физическому хранилищу данных, а к ядру базы данных Jet, которое выполняет эти запросы и возвращает результаты.

Ядро базы данных Microsoft Jet находится в нескольких файлах библиотек динамической компоновки *(dinamic link library - DLL*), которые связываются с программой на *VB, VBA* во время выполнения. Ядро переводит запросы приложения в физические действия над файлом *mdb* или другими хранилищами данных. Также ядро выполняет вспомогательные операции типа защиты данных, блокировки индексирования, поддержания целостности ссылок. В ядре же находится обработчик запросов на языке структурированных запросов SQL - (Structured Query Language). Там же находится обработчик результатов запросов.

Хранилище данных. Данные могут храниться в файле, файлах или просто на сырых разделах (partition) жесткого диска.

Локальные и удаленные базы данных. Все три указанных выше составляющих могут иметь различное местоположение – находится на одном компьютере и работать с однопользовательским приложением – этот случай есть локальная база данных. В другом случае составляющие могут размещаться на нескольких компьютерах, объединенных сетью. Например, хранилище может быть расположено на файловом сервере, а приложение – на нескольких пользовательских компьютерах.

Различия между удаленными и клиент–серверными базами данных показаны на Рис.4.2.

<table>
<tr><th>Клиент–сервер</th><th>Удаленная</th></tr>
<tr><td>Хранилище данных</td><td rowspan="2">Хранилище данных</td></tr>
<tr><td>Ядро базы данных</td></tr>
<tr><td></td><td></td></tr>
<tr><td rowspan="2">Интерфейс пользователя</td><td>Ядро базы данных</td></tr>
<tr><td>Интерфейс пользователя</td></tr>
</table>

Рис.4.2

В системе *Клиент–Сервер* ядро базы данных размещается на центральном сервере вместе с хранилищем данных. Ядро может обслуживать одновременно много клиентских приложений, управляя хранилищем данных и возвращая результаты запросов. В системе с удаленной базой ядро находится на одной машине с приложением и только хранилище данных размещается на удаленном компьютере. Ядро базы данных Jet является сервером. Если на нескольких машинах есть локальные копии приложений, то у каждой должна быть своя копия библиотеки Jet. С этим связаны вопросы производительности. В приложении на VB или VBA можно создавать приложения клиент-сервер, соединяясь с источником данных Open Database Connectivity (ODBC) или MS SQL Server

5. Язык структурированных запросов SQL

Язык структурированных запросов (*Structured Query Language - SQL*) является промышленным стандартом языка баз данных, используемым ядром базы данных Microsoft. Ранним предшественником *SQL* был язык *SEQUEL*.

SQL обеспечивает команды как языка определения данных (*data definition language - DDL*) так и языка управления данными (*data manipulation language - DML*). Команды *DDL (Таблица 5.1)* позволяют создавать и определять новые базы данных, поля и индексы, в то время как команды *DML (Таблица 5.2)* позволяют формировать запросы сортировки, фильтрации и извлечения данных.

Таблица 5.1 Команды DDL

Команда	Описание
CREATE	Используется для создания новых таблиц, полей и индексов
DROP	Используется для удаления таблиц и индексов из базы данных
ALTER	Используется для изменения таблицы

	путем добавления полей или изменения определенных полей

Таблица 5.2 Команды DML

Команда	Описание
SELECT	Используется для запроса из базы данных записей, удовлетворяющих определенным критериям
INSERT	Используется для загрузки пакета данных в базу данных за одну операцию
UPDATE	Используется для изменения значения определенных записей и полей
DELETE	Используется для удаления записей из таблицы базы данных

Предложения *SQL* *(Таблица 5.3)* используются для изменения условий, определяющих данные, которые необходимо выбрать или обработать

Таблица 5.3 Предложения SQL

Предложение	Описание
FROM	Используется для указания имени таблицы, из которой должны быть выбраны данные
WHERE	Используется для задания условия, которому должны удовлетворять выбранные записи
GROUP BY	Используется для разделения выбранных записей на определенные группы
HAVING	Используется, чтобы установить условие, которому должна удовлетворять каждая группа
ORDER BY	Используется для сортировки выбранных записей в определенном порядке

Логические операторы *AND, OR, NOT* вместе с операторами сравнения *(Таблица 5.4)* позволяют формировать сложные условия выборки данных. Операторы сравнения используются для сравнения относительных величин двух выражений, для того, чтобы определить действия, которые должны быть предприняты

```
Select * from Stud WHERE key_stud=5
```

Таблица 5.4 Логические операторы

Оператор	Значение\употребление
<	
<=	
>	
>=	
=	
<>	
BETWEEN	Для указания диапазона величин
LIKE	Для сравнения с шаблоном
IN	Для указания записей в базе данных

Статистические функции *(Таблица 5.5)* – используются в предложениях *SELECT* на группах записей, чтобы возвратить единственную величину, относящуюся к группе записей

Таблица 5.5 Статистические функции

Статистическая функция	Описание
AVG	Среднее значение указанного поля
COUNT	Число отобранных записей
SUM	Сумма всех величин в указанном поле
MAX	Наибольшая величина в указанном поле
MIN	Наименьшая величина в указанном поле

5.1. Запрос на выборку данных SELECT

Одним из основных типов запросов в языке *SQL* является запрос *SELECT* на выборку данных из базы данных. *SELECT* используется, чтобы извлечь записи из *БД* как некоторый набор и сохранить их в новом объекте типа

Recordset. Операторы *SELECT* не изменяют данные, а только извлекают их из базы данных.

Формат запроса:

SELECT	*список_полей*
FROM	*список_таблиц IN внешняя_база_данных*
WHERE	*условие_отбора*
GROUP_BY	*список_полей*
HAVING	*условие_группировки*
ORDER BY	*список_полей*

```
SELECT * FROM stud
```

– запрос на выборку всех данных из всех полей из таблицы stud

```
SELECT fio_stud FROM stud
```

– запрос на выборку всех данных из поля fio_stud из таблицы stud

```
SELECT birthday_stud  as рождение FROM stud
```

– создание псевдонимов столбцов

Если имя поля повторяется более чем в одной таблице предложения *FROM*, то при обращении к полю ставится имя таблицы, точка, имя поля:

```
SELECT stud.name, stud.surname, par.name, par.surname
FROM stud, par
```

Можно использовать алиасы (ссылки) для обращения к таблицам в громоздких конструкциях

```
SELECT s.name, s.surname, p.name, p.surname FROM stud
as s, par as p
```

Также допускается другая запись после *from:*

```
... FROM stud s, par p
```

Ниже показан пример использования переменных VBA при формировании запроса:

```
"SELECT name, birthday FROM stud WHERE name = '" &_
name.text & "'"
```

Далее покажем как происходим фильтрация и сортировка результатов запроса при помощи предикатов *DISTINCT* , *TOP* и оператора *ORDER BY*.

Предикат *DISTINCT* позволяет пропускать записи, которые содержат повторяющиеся данные в выбранных столбцах. Если мы хотим увидеть все разные имена, которые имеют студенты группы *325*, то можем выполнить следующий запрос

```
SELECT DISTINCT name FROM stud WHERE num_group=325
```

Предикат *TOP N* возвращает первые *N* записей (от начала набора записей) в порядке, определенном оператором *ORDER BY*

```
SELECT TOP 10 [name], [surname] FROM stud WHERE
year_out = 2002 ORDER BY [average_mark] DESC
```

– сортировать по уменьшению значения

Если не использовать *ORDER BY*, то будут возвращены 10 произвольных значений. Для того, чтобы возвратить процентное количество можно использовать предикат *PERCENT*

```
SELECT TOP 20 PERCENT [name], [surname]
FROM stud
WHERE  year_out = 2002
ORDER BY [average_mark] DESC
```

Предложение *WHERE* указывает какие записи из таблиц, указанных в предложении *FROM*, будут включены в результат оператора *SELECT*. Если предложение *WHERE* не указано, то возвращаются все записи таблицы.

Если в запросе указывается более чем одна таблица и не включается предложение *WHERE* и предложение *JOIN*, запрос построит декартово произведение таблиц.

Замечание: Для объединения таблиц рекомендуется использовать предложение *JOIN*, хотя предложение *WHERE* может выполнять аналогичные задачи. *WHERE* используется после *FROM*.

Примеры:

```
WHERE spec_group = 'физика'
WHERE stud_age BETWEEN 18 AND 25
```

Предложение *WHERE* может содержать *40* выражений, связанных логическими операторами *AND и OR*.

Если имя поля содержит пробел или знаки пунктуации, тогда оно должно быть окружено квадратными скобками.

```
SELECT [ID товара], [имеется на складе]
FROM Товары
WHERE [Имеется на складе] <= [Допустимый остаток]
```

Если в качестве условий выступает дата, то она должна быть записана в Американской нотации *MM/DD/YY* и заключена в литералы даты (#). Например, если вас интересуют товары с датой поставки 13 марта 2012 года, то запрос будет выглядеть так:

```
SELECT * FROM Заказы
WHERE [дата поставки] = #03/13/12#
```

Для того, чтобы учитывались параметры международной настройки, можно использовать функцию *DateValue*. Например для программы в России запрос будет выглядеть следующим образом:

```
SELECT * FROM Заказы
WHERE [дата поставки] = DateValue('13/3/12')
```

Пример для программы в США

```
SELECT * FROM Заказы
WHERE [дата поставки] = DateValue('3/13/12')
```

GROUP BY объединяет записи с идентичными величинами в указанном поле в одну запись. Если в оператор *SELECT* включена статистическая функция *SUM* или *COUNT*, то для каждой записи создается итоговое значение. Если статистической функции нет, то итоговые значения не создаются. Величины Null группируются и не опускаются.

Предложение *WHERE* используется для исключения строк, которые не нужно группировать, а предложение *HAVING* для фильтрации записей после того как они сгруппированы.

Поле в списке предложения *GROUP BY* может ссылаться на любое поле в предложении *FROM*, если оно не содержит данных типа *MEMO*, даже если поле не включено в оператор *SELECT*, при условии, что оператор *SELECT* содержит статистическую *SQL* функцию.

```
SELECT [Название товара], SUM([Имеется на складе])
FROM Товары
GROUP BY [Название товара]
```

Предложение *HAVING* определяет, какие записи показываются в операторе *SELECT* с предложение *GROUP BY* (подобно оператору *WHERE*)

Предложение *ORDER BY* определяет порядок сортировки записей, полученных в ходе выполнения запроса

```
SELECT * FROM stud
ORDER BY fio_stud ASC
```

– сортировка по возрастанию

```
SELECT [Фамилия], [Имя] FROM stud
ORDER BY [Фамилия]
```

– по умолчанию сортировка производится по возрастанию

Можно сортировать в порядке убывания по номеру столбца

```
SELECT [Фамилия], [Имя] FROM stud
ORDER BY 2 DESC
```

```
SELECT [Фамилия], Зарплата FROM Служащие
ORDER BY Зарплата DESC  Фамилия
```

Выбирается оклад жалования и сортируется в нисходящем порядке, служащие имеющие одинаковый оклад сортируются по возрастанию

Запрос на подсчет количества студентов, числящихся в группе 122:

```
SELECT COUNT (fio_stud) FROM stud WHERE
num_group_stud =122
```

Пример запроса на подсчет количества различных имен в таблице:

```
SELECT COUNT DISTINCT (name_stud)  FROM stud
```

Запрос на создание таблицы:

```
SELECT * INTO [Новые служащие] FROM Служащие
```

В данном случае создается новая таблица, а не набор записей

5.2. Запрос на удаление данных DELETE

DELETE – удаляет из одной и более таблиц записи, удовлетворяющие условию *WHERE*

Следующий запрос удаляет из таблицы stud запись, которой соответствует значение поля key_stud, равное 22

```
DELETE FROM stud WHERE key_stud = 22
```

5.3. Запрос на добавление данных INSERT INTO

Оператор *INSERT INTO* используется, чтобы добавить записи к таблице или создать запрос на добавление.

INSERT INTO приемник [IN внешняя база данных]

```
SELECT [источник. ] поле1[, поле2]
FROM выражение
```

Если надо добавить одну запись, то

```
INSERT INTO приемник [(Поле1[, Поле2])]
VALUES (Значение1[, Значение2])
```

Если в таблице – приемнике существует первичный ключ, то необходимо обеспечить его непустое неповторяющееся значение. Если тип ключа Count, то лучше его не включать в запрос. Если происходит добавление в связанные таблицы, то можно использовать либо триггер, либо хранимые процедуры, либо *SELECT* на получение основного ключа после добавления и добавление на следующем шаге внешнего ключа в связанную таблицу.

Следующий пример демонстрирует добавление информации в таблицу *town*

```
INSERT INTO town (name_town, region_town) VALUES
('Томск','Россия')
```

5.4. Запрос на обновление данных UPDATE

UPDATE *таблица*
SET *новое значение*
WHERE *критерий*

Запрос используется в методе *Execute*

Запрос из следующего примера позволяет изменить у всех экземпляров сущности Студент, хранящихся в таблице *stud*, имеющих значение поля *fio_stud*, похожее на "Петров В.А." на значение "Петров В.В."

```
UPDATE stud SET fio_stud='Петров В.В.' WHERE fio_stud
like 'Петров В.А.'
```

Обновление нескольких записей

```
UPDATE Заказы
SET [Сумма заказа]=[Сумма заказа]*1.1,
Фрахт = Фрахт * 1.03
WHERE [Страна поставки]='UK'
```

5.5. Составные запросы. Подзапросы

Для составления подзапроса используются следующие синтаксические формы:

```
сравнение [ANY | ALL | SOME] (SQL -оператор)
выражение [NOT] IN (SQL-оператор)
[NOT] EXISTS (SQL-оператор)
```

Подзапрос может быть вместо выражения в списке полей оператора *SELECT* или в предложения *WHERE и HAVING*

ANY, SOME – *выбираются записи, которые удовлетворяют сравнению с какой-либо записью*
ALL – удовлетворяют сравнению со всеми записями

```
SELECT * FROM Товары
WHERE [Цена] > ANY
(SELECT [Цена] FROM [Подробности заказа]
WHERE [Скидка]>=25)
```

Если условие поменять на *ALL*, то возвратит только те продукты, у которых цена выше чем любого продукта, проданного со скидкой 25 процентов или более. Если надо извлечь записи совпадающие по величине, то

*SELECT * FROM Товары*
WHERE [Product ID] IN
(SELECT [Product ID] FROM [Подробности заказа]

WHERE [Скидка]>= 25)

Чтобы связать таблицу в основном запросе с таблицей в подзапросе можно воспользоваться алиасом

SELECT [Фамилия], [Имя], Должность, Зарплата
FROM Служащие AS T1
WHERE Зарплата >=
(SELECT AVG(Зарплата)
FROM Служащие
WHERE T1.Должность=Служащие.Должность) ORDER BY должность

5.6. Соединения

INNER JOIN (внутреннее) – записи из обеих таблиц включаются в соединение только в случае совпадения заданного поля первой таблицы с заданным полем второй таблицы.

LEFT OUTER JOIN – включаются все записи первой таблицы и те записи второй таблицы, где заданные поля совпадают.

RIGHT OUTER JOIN – включаются все записи *второй* таблицы и те записи первой таблицы, где заданные поля совпадают.

... FROM table1 INNER JOIN table2 ON table1.field1=table2.field2
... FROM table1 [LEFT | RIGHT] JOIN ON table1.field1=table2.field2

Ниже показана конструкция сложного запроса

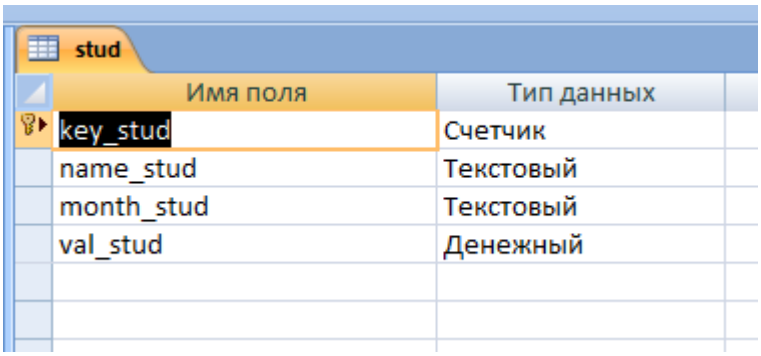
SELECT поля
FROM таблица1 INNER JOIN
(таблица2 INNER JOIN [(] таблица 3
[INNER JOIN [(] таблицаX [INNER JOIN...)]
ON таблица3.поле3=таблицаX.полеX])
ON таблица2.поле2=таблица3.поле3)
ON таблица1.поле1=таблица2.поле2

*SELECT DISTINCTROW sum([Цена]) * [Количество] AS [Объем сбыта],*
[Имя] & " " &
[Фамилия] AS Имя FROM Служащие
INNER JOIN (Заказы INNER JOIN [Подробности заказа]
ON Заказы.[ИД Заказа]=[Подробности заказа].[ИД Заказа])
ON Служащие.[ИД Служащего]=Orders.[ИД Служащего]
GROUP BY [Имя] & " " [Фамилия]

Приведем еще несколько полезных запросов. Допустим у вас есть набор некоторых ресурсов. Расход этих ресурсов распределен по месяцам. Для хранения такой информации достаточно одной таблицы, содержащей три поля: *название_ресурса, название_месяца и значение_расхода_ресурса*. Требуется создать SQL запрос, формирующий таблицу, в которой столбцы соответстствовали бы месяцам, а строки – ресурсам. В ячейках таблицы должно быть записано значение расхода ресурса, соответствующего строке за месяц, соответствующий столбцу. Покажем два запроса, позволяющих решить данную задачу.

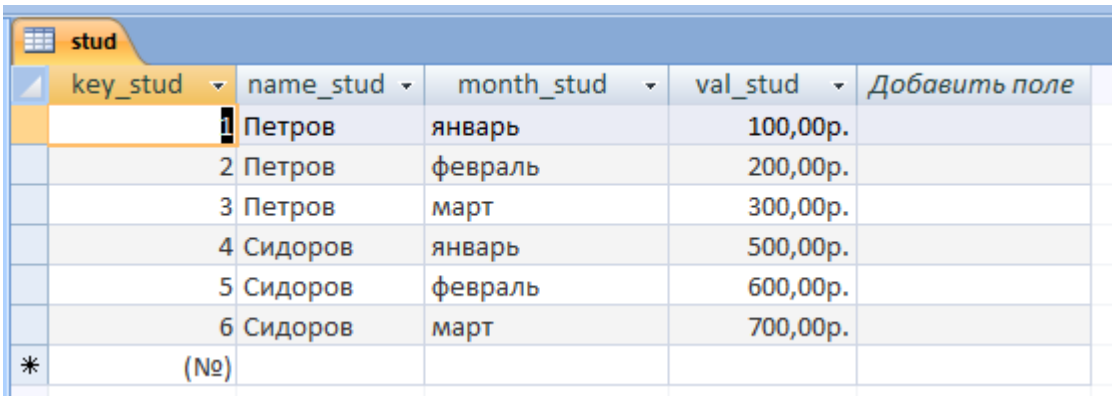
Например, студенты Петров и Сидоров в течение трех месяцев (Января, Февраля и Марта) питались в буфете. Необходимо написать запрос, который бы вывел распределение затрат обоих студентов на питание в буфете по месяцам.

Структура исходной таблицы показана на Рис.5.1, а хранящаяся информация – на Рис.5.2


stud

Имя поля	Тип данных
key_stud	Счетчик
name_stud	Текстовый
month_stud	Текстовый
val_stud	Денежный

Рис.5.1


stud

key_stud	name_stud	month_stud	val_stud	Добавить поле
1	Петров	январь	100,00р.	
2	Петров	февраль	200,00р.	
3	Петров	март	300,00р.	
4	Сидоров	январь	500,00р.	
5	Сидоров	февраль	600,00р.	
6	Сидоров	март	700,00р.	
(№)				

Рис.5.2

Первый вариант запроса использует группировку по полю *name_stud*, предварительно сформировав три дополнительных результирующих поля на базе поля *val_stud*, подставляя в каждое из них значение, соответствующее заданному месяцу при помощи конструкции SWITCH:

```
SELECT name_stud as ФИО, sum (switch (month_stud like 'январь', val_stud, true, 0)) AS январь, sum (switch (month_stud like 'февраль', val_stud, true, 0)) AS февраль, sum (switch (month_stud like 'март', val_stud, true, 0)) as март FROM stud GROUP BY name_stud;
```

Подобный результат может быть получен путем выполнения сложного запроса, формирующего дополнительные результирующие поля при помощи подчиненных запросов.

```
SELECT a.name_stud, a.val_stud, b.val_stud, c.val_stud FROM (select name_stud, val_stud from stud where month_stud like 'январь') AS a, (select name_stud, val_stud from stud where month_stud like 'февраль') AS b, (select name_stud, val_stud from stud where month_stud like 'март') AS c WHERE (((a.name_stud) = [b].[name_stud]) AND ((b.name_stud)=[c].[name_stud]));
```

В данном запросе мы намеренно оставили квадратные скобки, в которые заключены имена полей и алиасы таблиц (результатов работы подзапросов). Такие квадратные скобки автоматически проставляются программой MS Access при формировании запросов в конструкторе запросов. Они по умолчанию ствятся для того, чтобы можно было использовать имена полей или таблиц, состоящие из нескольких слов.

В результате получаем искомую табицу, показанную на Рис.5.3:

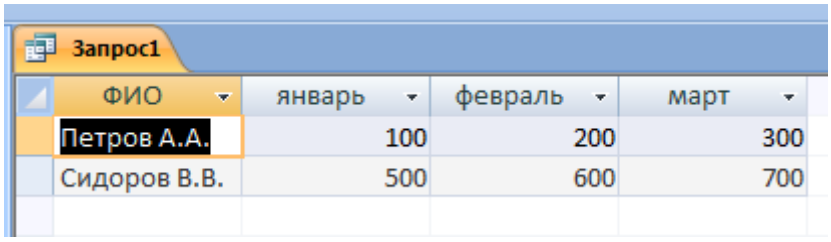


Рис 5.3

В заключение сделаем несколько замечаний о выполнении запросов в программах. Для того, чтобы выполнить любой запрос, который не возвращает записи, необходимо заключить оператор в двойные кавычки и использовать его как аргумент метода *Execute* объектов *Database* или *QueryDef*

```
MyDB.Execute "CREATE TABLE stud ([fio_stud] TEXT, [sex_stud] bool)"
```

Для того, чтобы использовать любую из команд, которые возвращают строки (*SELECT*), запрос можно использовать как источник метода *OpenRecordset*

```
MyDB.OpenRecordset("SELECT *from stud",dbOpenDynaset)
```

Далее на примерах программ рассмотрим практические особенности использования объектов VBA и языка SQL для работы с базами данных.

6. Модель объектов DAO

Модель объектов доступа к данным является объекто–ориентированым интерфейсом программирования ядра базы данных *Jet*. Это иерархия классов, которая соответствует логической структуре реляционной базы данных.

Для единообразия и простоты использования эти классы, а также объекты, которые создаются на базе этих классов, имеют такие же свойства как и другие классы и объекты в VB и VBA

6.1. Иерархия классов

В *VB* реализована иерархия объектов. Это значит, что одни объекты могут содержать другие объекты, которые, в свою очередь, могут включать и третьи. Это включение обеспечивается особым типом объекта, который называется коллекция. Единственным назначением объекта коллекция является содержание в себе других объектов. Коллекция включает в себя объекты одного типа.

Замечание: Элементы в иерархии *DAO* на самом деле являются классами, а не объектами. Это только «черновики» (прототипы) объектов, создаваемых при разработке приложения. В объектно–ориентированном программировании *(ООП)* класс похож на тип данных в том смысле, что описывает тип объекта, на который производится ссылка.

Например, `Dim MyWs As Workspace`

указывает, что *MyWs* - это переменная, которая используется в качестве объекта класса *Workspace*. Поэтому нужно понимать, что объект *Database* – это объект класса *Database.*

На вершине иерархии коллекций находится само ядро базы данных *Microsoft Jet – DBEngine*. Это единственный объект, который не является коллекцией. Объект *DBEngine* владеет коллекцией *Workspaces* (название коллекции всегда является множественным числом от названия объектов, которые в ней содержатся), которая содержит один или несколько объектов *Workspace* (рабочая область). Каждый объект *Workspace* владеет коллекцией *Databases*, в которой содержится один или несколько объектов *Database*. Объект *Database* содержит коллектцию *TableDefs*.

Обратиться к первому элементу коллекции *TableDefs* можно:

```
MyDatabase.TableDefs(0)
```

Объекты в иерархии обпределяются по полному «пути» через вложенные коллекции, которым они принадлежат. Ниже приведен пример использования для ссылки на объект коллекции не индекса а строки, соответствующей свойству *Name* объекта коллекции *Fields*

DBEngine.Workspaces(0).Databases(0).TableDefs(0).Fields("Заказчик")

Также для обращения к свойству объекта можно использовать восклицательный знак.

```
MyTableDef.Fields("Заказчик")
MyTableDef.Fields!Заказчик
```

Две приведенные выше строки равнозначны.

Коллекции по умолчанию. У многих объектов доступа к данным есть коллекции по умолчанию. Это позволяет упростить программу и сократить запись некоторых операторов. Например, коллекцией по умолчанию объекта *Recordset* является коллекция *Fields*. Чтобы получить значение объекта *Field* с названием Заказчик, можно написать:

```
Cust = MyRecordset!Заказчик
или Cust = MyRecordset.Fields!Заказчик
или Cust = MyRecordset.Fields("Заказчик")
```

6.2. Свойства и методы

Одна из основных идей *ООП* заключается в том, что все данные и процедуры, относящиеся к конкретному объекту, хранятся вместе с самим объектом. Эта идея носит название *Инкапсуляция*. В *VB* данные объекта (установки, атрибуты) называются свойствами, а процедуры работающие с объектом – его методами.

Программирование с использованием объектов доступа к данным состоит из создания объектных переменных и управления ими с помощью вызова их методов и установки свойств.

```
Dim MyDB As Database, MyWS as Workspace, _
MyRS As Recordset
Set MyWS = DBEngine.Workspaces(0)
Set MyDB = MyWS.OpenDatabase("MyDB.mdb")
Set MyRS = MyDB.OpenRecordset("Customers")
```

```
MyRS.Index = "IDCustomer"
```

1. Метод *OpenDatabase* объекта *MyWS* класса *Workspace* необходим для открытия базы данных *MyDB.mdb* и присвоения ссылки на нее объектной переменной *MyDB* класса *Database*.

2. Метод *OpenRecordset* объекта *MyDB* используем для создания набора записей (объект recordset – группа записей, представляющая таблицу базы данных или результат запроса), основанного на таблице *Customers* и назначения ссылки на него переменной *MyRS* класса *Recordset*)

3. Свойству Index объекта MyRS класса Recordset присваевается значение IDCustomer, что должно изменить порядок сортировки записей в объекте *MyRS.*

У большинства объектов можно определять новые свойства.

Теперь приступим к практической части работы с базами данных. Для работы с базой данных вначале ее необходимо создать. Создание новой базы данных, добавление в нее таблиц и т.п. можно несколькими способами:

- всю последовательность действий можно выполнять в ручном режиме;
- можно воспользоваться объектами и методами *DAO*
- можно использовать *SQL* запросы (как это было показано выше)

7. Создание базы данных в ручном режиме

1. Открыть Microsoft Access. Создать новую базу данных (Файл/Создать/База данных). В диалоговом окне указать путь сохранения диск D:\Личный_Каталог\ Базу данных назвать, например, Uch_Proc. Расширение файла будет зависеть от версии СУБД ACCESS (может быть **mdb**, может быть **accdb**.

2. В открывшемся окне созданной базы данных выбрать объект *Таблицы*. Два раза щелкнуть по пункту меню «Создание таблицы в режиме конструктора». Заполнить появившуюся таблицу так, как это указано в табличной схеме. При сохранении макета, на вопрос как назвать Таблицу, задать имя сущности или отношения. В некоторых версиях MS Access имя таблицы запрашивается сразу при переходе в режим конструктора.

Для Access 2007 - Создание\Таблица\Режим конструктора\

Таким же образом в режиме конструктора можно создать все остальные таблицы сущностей (отношений)

3. Создание связей по внешним ключам можно осуществить с использованием мастера подстановок. Для выполнения данной процедуры

необходимо выбрать искомую таблицу; перейти в режим конструктора; для поля являющегося внешним ключом выбрать тип данных - Мастер подстановок.

8. Создание базы данных при помощи DAO

1. Необходимо использовать оператор *Dim* для объявления новых объектных переменных для каждого объекта в базе данных. Нам понадобятся:

- один объект *Database*;
- один объект *TableDef* для каждой таблицы;
- один объект *Field* для каждого поля в каждой таблице;
- один объект *Index* для выбранных полей в определенных разработчиком таблицах.

```
Dim MyDB As Database
Dim MyWs As Workspace
Dim TownTd As TableDef, StudTd as TableDef
Dim TempFlds(2) as Field
Dim TownFlds(2) As Field, StudFlds(3) As Field
Dim TownIdx as Index, StudIdx(2) as Index
Dim StudTel as Relatioin
```

2. Необходимо использовать метод *CreateDatabase* объекта *Workspace* для создания новой базы данных. Для метода *CreateDatabase* желательно использовать три аргумента: *Название базы данных* (путь к файлам базы); тип национальных стандартов (кодировок); версия базы данных. Также можно использовать строку соединения (либо использовать готовое соединение). В этой строке указывается пароль, источник данных, версия *Microsoft Jet*. Примеры констант национальных стандартов приведены в Таблице 8.1

Таблица 8.1

Константы национальных стандартов	Описание
dbLangGeneral	*English, German, French, Portuguese, Italian, and Modern Spanish*
dbLangChineseSimplified	*Simplified Chinese*
dbLangCyrillic	*Russian*

Примеры констант типов ПО приведены в Таблице 8.2

Таблица 8.2

Константы типов ПО	Описание
DbEncrypt	*Создает "зашифрованную" базу данных*
dbVersion40	*Создает базу данных, которая использует формат файла Microsoft Jet database engine версии 4.0*
dbVersion120	*Создает базу данных, которая использует формат файла Microsoft Jet database engine версии 12.0*

```
Set MyWs = DBEngine.Workspaces(0)
Set MyDB = MyWs.CreateDatabase("DBPoba1.mdb", _
dbLangCyrillic, dbVersion40)
```

3. Необходимо использовать метод *CreateTableDef* объекта *Database* для создания новых объектов TableDef для каждой таблицы базы данных

```
Set StudTd = MyDB.CreateTableDef("stud")
Set TownTd = MyDB.CreateTableDef("town")
```

4. Необходимо использовать Метод *CreateField* для создания новых объектов *Field* для каждого поля в таблице и задания его свойств, определяющих размер, тип и другие атрибуты.

```
Set StudFlds(0) = StudTd.CreateField("key_stud",
dbLong)
```

'Делаем поле счетчиком

```
StudFlds(0).Attributes = dbAutoIncrField
Set StudFlds(1)= StudTd.CreateField("fio_stud", dbText)
TelFlds(1).Size = 255
Set StudFlds(2)= StudTd.CreateField("key_town", dbLong)
Set TownFlds(0)= TownTd.CreateField("key_town", dbLong)

TownFlds(0).Attributes = dbAutoIncrField
Set TownFlds(1)=TownTd.CreateField("name_town", dbText)
TownFlds(1).Size = 255
```

Необходимо использовать метод *Append* для присоединения каждого поля к своей таблице и каждой таблицы к базе данных

```
For i = 0 To 1
   StudTd.Fields.Append StudFlds(i)
   TownTd.Fields.Append TownFlds(i)
Next i
StudTd.Fields.Append StudFlds(2)

 MyDB.TableDefs.Append StudTd
 MyDB.TableDefs.Append TownTd

 For i = 0 To 1
   Set TownFlds(i) = Nothing
 Next i
Set StudFlds(2) = Nothing
MyDB.Close
```

8.1. Добавление индексов и отношений

Индексы – это специальные объекты, которые позволяют упорядочивать (сортировать) информацию в базе данных. Сам факт упорядочивания (создание индексов) существенно влияет на производительность *СУБД*. Индексы строятся по одному или нескольким полям таблицы, по которым обычно производится выборка или поиск информации. У одной таблицы может быть несколько индексов.

В данном случае под индексами будем понимать ключевые поля. Индексы будем строить по основным и внешним ключевым полям таблиц для создания отношений, которые, в свою очередь, будут обеспечивать целостность данных.

Чтобы добавить индексы к таблице:

1. Необходимо создать индексы для каждой таблицы с помощью метода *CreateIndex* объекта *TableDef* и установить их свойства. Количество создаваемых индексов будет соответствовать количеству ключевых полей.

```
Set StudIdx(0) = StudTd.CreateIndex("stud_key")
StudIdx(0).Primary = True
StudIdx(0).Unique=True

Set StudIdx(1) = StudTd.CreateIndex("key_town")
StudIdx(1).Primary = False
StudIdx(1).Unique=False

Set TownIdx = TownTd.CreateIndex("key_town")
```

```
TownIdx.Primary = True
TownIdx.Unique=True
```

2. Необходимо создать поле для каждого индексного объекта с помощью метода CreateField объекта I*ndex*

```
Set TempFlds(0)=StudIdx(0).CreateField("key_stud")
Set TempFlds(1)=StudIdx(1).CreateField("key_town")
```

Для TownIdx проделываем тоже самое

3.Необходимо присоединить поле к объекту *Index* и объект *Index* к объекту *TableDef*.

```
StudIdx(0).Fields.Append TempFlds(0)
StudIdx(1).Fields.Append TempFlds(1)

StudTd.Indexes.Append StudIdx(0)
StudTd.Indexes.Append StudIdx(1)
```

Для *TownIdx* проделываем тоже самое

Замечание: Поля, созданные методом *CreateField* объекта *Index*, не присоединяются к *TableDef*. Эти поля присоединяются к объекту *Index* причем с теми же именами, что и подлежащие индексированию поля *TableDef*.

8.2. Отношения и целостность ссылок

Создание объектов *TsbleDef* с первичными и внешними ключами позволяет связывать записи в одной таблице с соответствующими записями в другой таблице, используя равенство значений первичного и внешнего ключей. Чтобы эти отношения работали, необходимо поддерживать целостность ссылок при добавлении и удалении записей. Целостность ссылок означает, что внешний ключ в любой ссылающейся таблице всегда должен указывать на существующую запись в таблице, на которую производится ссылка.

Для поддержания целостности ссылок и предотвращения их повреждений в библиотеке *Jet* имеется объект *Relation*.

8.3. Добавление отношения в базу

Создаем объект *Relation* с помощью метода *CreateRelation* объекта Database и устанавливаем его свойства *Table* и *ForeignTable* (таблица и внешняя таблица)

```
Set stud_town_r = MyDB.CreateRelation("stud_town_r")
stud_town_r.Table = "town"
```

```
stud_town_r.ForeignTable = "stud"
```

Создаем поле для определения общего поля первичного и внешнего ключей в отношении с помощью метода *CreateField* объекта *Relation*

```
Set TempFlds(0)= stud_town_r.CreateField("key_town")
TempFlds(0).ForeignName = "key_town"
```

Используя метод *Append*, добавляем объект *Field* к его объекту *Relation* и объект *Relation* к *Database*

```
stud_town_r.Fields.Append TempFlds(0)
MyDB.Relations.Append stud_town_r
```

8.4. Установка дополнительных свойств

Если при использовании метода *Create* опущен один или несколько необязательных аргументов, можно использовать оператор присваивания, чтобы присвоить или сбросить соответствующее свойство до присоединения нового объекта к коллекции. Например,

```
Set MyFld= MyTableDef.CreateField("fio_stud", dbText, 30)
```

эквивалентно

```
Set MyFld = MyTableDef.CreateField()
MyFld.Name="fio_stud"
MyFld.Type= dbText
MyFld.Size = 30
```

Теперь рассмотрим пример программы, создающей базу данных. Напоминаем, что для корректной работы программы к проекту необходимо подключить библиотеку *Microsoft DAO 3.6.*

'Объявляем переменные для базы данных и рабочего пространства

```
Dim MyDB As Database
Dim MyWs As Workspace
```

'Объявляем переменные для таблиц, полей, индексов и отношений

```
Dim StudTd As TableDef, TownTd As TableDef
Diml TelTd As TableDef
Dim StudFlds(3) As Field, TownFlds(2) As Field
Dim TelFlds(3) As Field
Dim stud_town_r As Relation, tel_stud_r As Relation
Dim TempFlds(2) As Field
Dim StudIdx(2) As Index, TownIdx As Index
Dim TelIdx(2) As Index
Dim i As Integer
```

'Задаем переменную для рабочего пространства

```
Set MyWs = DBEngine.Workspaces(0)
If Dir("C:\DATE\DBProbaRel.mdb") <> "" Then
    Kill "C:\DATE\DBProbaRel.mdb"
End if
```

'Создаем базу данных

```
Set MyDB = MyWs.CreateDatabase("C:\
   DATE\DBProbaRel.mdb", dbLangCyrillic, dbVersion40)
```

'Создаем объекты: таблицы и поля

```
Set StudTd = MyDB.CreateTableDef("stud")
Set TownTd = MyDB.CreateTableDef("town")
Set TelTd = MyDB.CreateTableDef("tel")

Set StudFlds(0)= StudTd.CreateField("key_stud", dbLong)
StudFlds(0).Attributes = dbAutoIncrField
Set StudFlds(1)=StudTd.CreateField("fio_stud",dbText, 100)
Set StudFlds(2)= StudTd.CreateField("key_town", dbLong)
Set TownFlds(0)= TownTd.CreateField("key_town",dbLong)
TownFlds(0).Attributes = dbAutoIncrField
Set TownFlds(1) = TownTd.CreateField("name_town",
dbText,100)

Set TelFlds(0) = TelTd.CreateField("key_tel", dbLong)
TelFlds(0).Attributes = dbAutoIncrField
Set TelFlds(1) = TelTd.CreateField("num_tel", dbText,100)
Set TelFlds(2) = TelTd.CreateField("key_stud", dbLong)
```

'Добавляем созданные поля в коллекции полей каждой таблицы

```
For i = 0 To 2
   StudTd.Fields.Append StudFlds(i)
   TelTd.Fields.Append TelFlds(i)
Next i
 For i = 0 To 1
   TownTd.Fields.Append TownFlds(i)
 Next i
```

'Добавляем созданные таблицы в коллекцию таблиц

```
MyDB.TableDefs.Append StudTd
MyDB.TableDefs.Append TelTd
MyDB.TableDefs.Append TownTd
```

'Создаем индексы

```
 Set StudIdx(0) = StudTd.CreateIndex("key_stud")
 StudIdx(0).Primary = True
```

```
StudIdx(0).Unique = True
Set StudIdx(1) = StudTd.CreateIndex("key_town")
StudIdx(1).Primary = False
StudIdx(1).Unique = False
```

'Создаем поля для индексов

```
Set TempFlds(0) = StudIdx(0).CreateField("key_stud")
Set TempFlds(1) = StudIdx(1).CreateField("key_town")
```

'Добавляем созданные поля в коллекцию полей объекта *Index* и созданные 'индексы в коллекцию индексов каждой таблицы

```
For i = 0 To 1
  StudIdx(i).Fields.Append TempFlds(i)
  StudTd.Indexes.Append StudIdx(i)
Next i
Set TownIdx = TownTd.CreateIndex("key_town")
TownIdx.Primary = True
TownIdx.Unique = True
Set TempFlds(0) = TownIdx.CreateField("key_town")
TownIdx.Fields.Append TempFlds(0)
TownTd.Indexes.Append TownIdx

Set TelIdx(0) = TelTd.CreateIndex("key_tel")
TelIdx(0).Primary = True
TelIdx(0).Unique = True
Set TelIdx(1) = TelTd.CreateIndex("key_stud")
TelIdx(1).Primary = False
TelIdx(1).Unique = False
Set TempFlds(0) = TelIdx(0).CreateField("key_tel")
Set TempFlds(1) = TelIdx(1).CreateField("key_stud")
For i = 0 To 2
  TelIdx(i).Fields.Append TempFlds(i)
  TelTd.Indexes.Append TelIdx(i)
Next i
```

'Создаем два отношения: между таблицами *stud u town* и между таблицами *tel* 'и *stud*

```
Set stud_town_r = MyDB.CreateRelation("stud_town_r")
Set tel_stud_r = MyDB.CreateRelation("tel_stud_r")
```

'Указываем на основную и внешнюю таблицы отношения stud_town_r

```
stud_town_r.Table = "town"
```

```
stud_town_r.ForeignTable = "stud"
```

'Создаем поле для отношения

```
Set TempFlds(0) = stud_town_r.CreateField("key_town")
```

'Указываем внешний ключ для отношения

```
TempFlds(0).ForeignName = "key_town"
```

'Добавляем поле в коллекцию полей созданного отношения

```
stud_town_r.Fields.Append TempFlds(0)
```

'Добавляем отношение в коллекцию отношений базы данных

```
MyDB.Relations.Append stud_town_r
tel_stud_r.Table = "stud"
tel_stud_r.ForeignTable = "tel"
Set TempFlds(0) = tel_stud_r.CreateField("key_stud")
TempFlds(0).ForeignName = "key_stud"
tel_stud_r.Fields.Append TempFlds(0)
MyDB.Relations.Append tel_stud_r
```

‘Очищаем ссылки на объекты

```
For i = 0 To 2
   Set StudFlds(i) = Nothing
   Set TelFlds(i) = Nothing
Next i
 For i = 0 To 1
   Set TownFlds(i) = Nothing
 Next i
```

'Закрываем базу данных и завершаем программу

```
MyDB.Close
End Sub
```

8.5. Изменение Базы данных

Можно добавить к базе данных новые объекты *TableDef* или добавить к существующим таблицам новые объекты *Field и Index*. Можно также удалить объект *TableDef* из базы или удалить объект *Index* из *TableDef*. Есть несколько ограничений на удаление объекта *Field*

8.6. Добавление таблицы к базе данных

Чтобы добавить новую таблицу к базе данных, нужно просто включить в существующую коллекцию *TableDefs* новый объект *TableDef*.

```
Dim db as Database
Dim NewTd as TableDef
Dim NewFld as Field

Set db =
DBEngine.Workspaces(0).OpenDatabase("Test.mdb")
Set NewTd = db.CreateTableDef("new_table")
Set NewFld = NewTd.CreateField("new_fld", dbInteger)

NewTd.Fields.Append NewFld
db.TableDefs.Append NewTd
db.Close
```

8.7. Удаление таблицы

Для удаления таблицы используется метод *Delete* коллекции *TableDefs*. При удалении таблицы уничтожаются все данные хранящиеся в таблице, поля, индексы, поэтому необходимо быть очень внимательными перед тем как выполнять удаление.

```
db.TableDefs.Delete "Authors"
```

8.8. Добавление поля к таблице

Чтобы добавить поле к существующей таблице его необходимо присоединить к коллекции *Fields*

```
Dim db as Database
Dim Td as TableDef
Dim Fld as Field

Set db =
DBEngine.Workspaces(0).OpenDatabase("C:\proba.mdb")
Set Td = db.TableDefs("stud")
```

'Создаем новый объект поля

```
Set Fld = Td.CreateField("Address",dbText,40)
```

'Присоединяем созданный объект к коллекции

```
Td.Fields.Append Fld
```

'Используем переменную *Fld* для создания еще одного объекта поля

```
Set.Fld = Td.CreateField("Phone",dbText,25)
Td.Fields.Append Fld
db.Close
```

8.9. Изменение или удаление поля

После включения в коллекцию *TableDefs* отдельное поле не может быть изменено, удалить отдельное поле можно только в том случае, если оно не является частью объектов *Index* или *Relation*

Чтобы удалить поле, можно использовать метод *Delete* объекта *TableDef*. Для изменения отдельного поля нужно добавить отдельный объект *TableDef*, имеющий требуемые изменения в поле, затем переместить данные в новую таблицу, и после этого удалить старую.

8.10. Добавление индекса

Чтобы добавить индекс используется метод *Append*

```
Dim db as Database, Td as TableDef
Dim NewIdx as Index, NewFld as Field

Set db =
DBEngine.Workcpace(0).OpenDatabase("MyBase.mdb")
Set Td=db.TableDefs("stud")
Set NewIdx = Td.CreateIndex("fio_stud_index")
NewIdx.Unique = False
Set NewFld = NewIdx.CreateField("fio_stud")
NewIdx.Fields.Append NewFld
Td.Indexes.Append NewIdx
db.Close
```

8.11. Удаление индекса

```
db.TableDefs("stud").Indexes.Delete "fio_stud_index"
```

Необходимо отметить, что нельзя удалять индекс, поддерживающий отношение.

9. Использование объекта Recordset

Recordset - «Набор записей» представляет записи в таблице базы данных или записи, полученные при выполнении запроса. *Recordset* используется для управления данными в базе на уровне запроса. Объект *Field* (поле) используется для управления данными на уровне поля.

Есть пять типов *Recordset*

9.1. Табличный набор (table-type)

Табличный набор ссылается на локальную таблицу в текущей базе данных или внешней базе. При создании табличного набора ядро БД открывает таблицу и выполняет все последующие действия непосредственно с данными таблицы. Набор такого типа можно открыть только по одной таблице. Данный набор можно проиндексировать, используя индекс основной таблицы При поиске данных метод *Seek* работает быстрее чем метод *Find*

9.2. Динамический набор (dynaset)

Динамический набор записей ссылается либо на локальные или связанные таблицы, либо на результат выполнения запроса, возвращающего строки. На самом деле он является множеством ссылок на записи одной или более таблиц. С помощью динамического набора можно извлекать и обновлять данные более чем в одной таблице, включая связанные таблицы из других баз данных. Изменения записей в динамическом наборе также отражаются и в основных таблицах. Пока динамический набор открыт в нем также будут отражаться все текущие изменения основных таблиц. Динамический набор является наиболее гибким и мощным типом набора записей, хотя поиск и другие операции выполняются в нем медленнее, чем с остальными наборами.

9.3. Статический набор (snapshot)

Статический набор записей содержит постоянную копия данных по состоянию на момент ее создания. Данный набор записей нельзя обновлять. Некоторые наборы из источников данных, удовлетворяющих стандарту *ODBC*, можно обновлять в зависимости от возможностей, предоставляемых базой данных на другом конце соединения. Данный набор занимает меньше процессорного времени, чем динамический и табличный наборы, поэтому может обслуживать запросы и возвращать данные гораздо быстрее, особенно при работе с источниками *ODBC*.

Статический набор записей с последовательным доступам (*forward-only*) обеспечивает часть возможностей статического набора. Позволяет двигаться по данным только в прямом направлении, но обеспечивает максимальную скорость при работе с данными. Работает только с методами *Move* и *MoveNext*

Динамический набор записей *(Dynamic*) - Соответствует динамическому курсору *ODBC*. Записи, которые добавляют или удаляют другие пользователи также заносятся в данный набор.

Тип используемого набора зависит от целей пользователя. Если надо отсортировать данные и работать с индексами лучше работать с табличным набором. Табличные наборы также обеспечивают наиболее быстрый поиск. Если необходимо обновлять выбранные данные, то нужен динамический набор. Если нужно быстро просмотреть данные то можно пользоваться статическим набором с последовательным доступом.

Для создания переменной объекта *Recordset* используется метод *OpenRecordset*. Этот метод доступен для объектов *Database, Connection, TableDef, QueryDef* и уже существующих объектов Recordset

```
Set переменная = база_данных.OpenRecordset (источник
[, тип [, параметры, [блокировка]]])
```

- *база_данных – тип Database или Connection*
- *источник – SQL запрос, TableDef или QueryDef*
- тип –
  dbOpenTable
  dbOpenDynaset
  dbOpenSnapshot
  dbOpenForwardOnly
  dbOpenDynamic

DAO автоматически устанавливает тип набора по умолчанию, исходя из источника данных и способа его открытия

```
Set rstNew = dbs.OpenRecordset("Источник данных")
```

Если источник – локальная табица, то доступны наборы табличного, динамического, статического и динамического (*dynamic*) типов, по умолчанию – табличный тип

В противном случае по умолчанию тип *dynaset*

```
Set rstNew = tdfTableData.OpenRecordset
```

если локальная таблица, то табличный тип, если таблица ODBC, то *dynaset*

9.4. Параметры метода OpenRecordset

dbAppendOnly – позволяет добавлять новые записи к набору, но запрещает редактирование и удаление. Только для dynaset. Полезно в системах архивирования и сбора данных

dbReadOnly – Запрещает изменение набора

dbSeeChanges – порождает ошибку времени выполнения, если один пользователь пытается изменить запись, которую редактирует другой пользователь.

dbDenyWrite – Запрещает изменений или добавление записей

dbDenyRead – запрещает чтение другими пользователями данных в таблице.

dbForwardOnly

dbSQLPassThrough

dbConsistent – допускает только согласованные обновления

dbInconsystent – допускает несовместные обновления

Задание параметров блокировки набора

dbReadOnly – запрещает изменение набора

dbPessimistic – пессимистическая блокировка

dbOptimistic – оптимистическая блокировка

dbOptimisticValue – использует оптимистическую согласованность

Примеры:

```
Dim dbs As Database, rstCustomers As Recordset
Set dbs = OpenDatabase("MyDb.mdb")
Set rstCustomers = dbs.OpenRecordset("Студенты")
```

Создание набора из связанной таблицы с другим форматом базы данных

Т.к. для базы не *MS Jet* табличный тип недоступен, то при открытии используется следующий по эффективности тип.

```
Dim dbs as Database
Dim tdfNonJetLinked as TableDef
Dim rstTableData As Recordset
```

' Открыть базу данных и создать TableDef

```
set dbs = OpenDatabase("MyDb.mdb")
set tdfNonJetLinked = dbs.CreateTableDef("PDXAuthor")
```

' Соединиться с Paradox-таблицей Author в базе данных C:\PDX\Publish

```
tdfNonJetLinked.Connect = "Paradox 3.X; DATABASE=
C:\PDX\Publish "
tdfNonJetLinked.SourceTableName = "Author"
```

' Присоединить таблицу

```
dbs.TableDefs.Append tdfNonJetLinked
```

' Создать динамический набор для таблицы

```
Set rstTableData = tdfNonJetLinked.OpenRecordset()
```

Для источников данных *ODBCDirect* можно использовать метод *OpenRecordset* с объектом *Connection*

Чтобы упорядочить записи в табличном наборе, необходимо установить его свойство *Index*. Свойство *Index* должно быть установлено до начала использования метода *Seek*.

Можно создать набор записей при помощи хранимого запроса на выборку (типа *SELECT*)

```
Dim dbs As Database, rstProducts As Recordset
Set dbs = OpenDatabase("MyDb.mdb")
Set rstProducts = dbs.OpenRecordset("Текущий перечень
изделий")
```

Если хранимого запроса нет, то вместо имени запроса воспринимается строка SQL

```
Dim dbs As Database, rstProducts As Recordset
Set dbs = OpenDatabase("MyDb.mdb")
strQuerySQL = "SELECT * FROM Изделия " _
& "WHERE Прекращен = No " _
& "ORDER BY НазваниеИзделия ;"
Set rstProducts = dbs.OpenRecordset(StrQuerySQL)
```

10. Параметрический запрос

Недостаток рассмотренных выше запросов заключается в том, что строка запроса должна компилироваться при каждом исполнении. Ниже показан пример того, как избежать данной проблемы, используя параметрический запрос.

```
Sub Param1
Dim dbs As Database
Dim rstProducts As Recordset
Dim qdf As QueryDef

Set dbs = OpenDatabase("Northwind.mdb")
Set qdf = dbs.CreateQueryDef("SalesRepQuery")
qdf.SQL = "SELECT * FROM Служащие " _
& "WHERE Должность = 'Торговый агент' "
Set rstProducts = dbs.OpenRecordset(SalesRepQuery)
End Sub
```

Ниже показан еще один пример использования параметрического запроса.

```
Sub find_fio()
Dim dbs As Database
Dim qdf As QueryDef
Dim rst As Recordset
Dim path As String
Dim par(2) As String
Dim q_flag as Integer
par(0) = "Петров Владимир Иванович"
par(1) = "Иванов Иван Сергеевич"
par(2) = "Иванов Василий Сергеевич"

path = ActiveWorkbook.path + "\"
Set dbs = OpenDatabase(path + "DBproba2.mdb")
q_flag=0
If dbs.QueryDefs.Count > 0 Then
 If Not dbs.QueryDefs("FindFio").Name="FindFio" Then
    q_flag=1
 Else
    q_flag=2
  End If
Else
  q_flag=1
End If
Select case q_flag
 case 1: Set qdf = dbs.CreateQueryDef("FindFio")
 case 2: Set qdf = dbs.QueryDefs("FindFio")
End Select

strSQL = "PARAMETERS Param1 TEXT; "
```

```
strSQL = strSQL & "select in_stud, num_gr_stud from
                  stud where fio_stud like [Param1];"
qdf.Sql = strSQL
```

‘Открываем рекордсет с заведомо «ложным» значением параметра

```
qdf.Parameters("Param1") = "1"
Set rst = qdf.OpenRecordset()
```

‘Переоткрываем рекордсет с искомыми значениями параметра

```
For i = 0 To 2
 qdf.Parameters("Param1") = par(i)
 rst.Requery qdf
 If rst.EOF Then
  MsgBox "Nothing"
 Else
  Do Until rstStud.EOF
    MsgBox par(i) & " " & rst.fields(0) & " " & _
    rst.fields(1)
    rstStud.MoveNext
  Loop
 End If
Next i
dbs.QueryDefs.Delete "FindFio"
End Sub
```

11. Эксплуатация базы данных

Опыт использования баз данных показывает, что для эффективной работы с ними нужно стараться придерживаться некоторых «неписанных» правил:

- создавать резервные копии баз данных
- создавать (периодически) текущую схему базы данных для определения ее текущей структуры;
- сжимать базу данных для экономии места и улучшения производительности;
- восстанавливать базу данных в случае уничтожения или повреждения.

Для создания резервных копий обычно используются механизмы, реализованные в СУБД. Если таких механизмов нет, а база существует в виде файлов, то на одном из «скриптовых» языков пишется сценарий копирования и переименования нужных файлов в архивный каталог.

11.1.Схематизация базы данных

Для схематизации базы данных часто используется следующая конструкция:

```
For Each объект in коллекция
   'отображение свойства объекта
Next объект
```

Ниже приведем сокращенный вариант сбора информации о базе данных

```
Sub MapDatabase()
Dim MyDB As Database
Dim MyWs As Workspace
Dim path As String
Dim db As Database, Td As TableDef, fld As Field
Dim Idx As Index, Rel As Relation
Dim i As Integer, n As Integer
Dim ss As String
Dim hFOut As Long
pathx = ActiveWorkbook.path + "\"
Set db = DBEngine.Workspaces(0).OpenDatabase(pathx &_
"DBProbaRel.mdb")
ss = ""
```

'Снимаем схему со свойств Database

```
ss = ss + vbLf + "База данных DATABASE"
ss = ss + vbLf + "Название:" & db.Name
```

'Снимаем схему с коллекции TableDefs

```
ss = ss + vbLf + "Таблицы (TABLEDEFS)"
For Each Td In db.TableDefs
 ss = ss + vbLf + vbLf + "Название ТАБЛИЦЫ: "+Td.Name
 If Td.Updatable = True Then
  ss = ss + vbLf + "Обновляемая"
 Else
  ss = ss + vbLf + "Необновляемая"
 End If
```

'Атрибуты объекта TableDef

```
ss = ss + vbLf + "Аттрибуты:" + Hex$(Td.Attributes)
If (Td.Attributes And dbSystemObject) <> 0 Then
  ss = ss + vbLf + "Системный объект"
End If
```

'Снимаем схему семейств Fields для каждого объекта TableDef

```
 ss = ss + vbLf + vbLf + "Поля(Fields)"
For Each fld In Td.Fields
  ss = ss + vbLf + "Название поля: " + fld.Name
Next fld
```

'Снимаем схему коллекции Indexes для каждого TableDef

```
 ss = ss + vbLf + vbLf + "Индексы (INDEXES)"
 If (Td.Attributes And dbSystemObject) = 0 Then
   For Each Idx In Td.Indexes
```

'Установим переменную Index

```
    ss = ss+vbLf+vbLf+"Название индекса: "+Idx.Name
    If Idx.Foreign Then
      ss = ss + vbLf + "Foreign индекс"
    Else
     If Idx.Primary Then
       ss = ss + vbLf + "Primary индекс"
     End If
   End If
```

'Снимаем схему коллекции Fields объекта Index

```
   For Each fld In Idx.Fields
    ss = ss+vbLf+"Название поля индекса: " + fld.Name
   Next fld
 Next Idx
End If
ss = ss + vbLf + "----------------------------"
Next Td
```

'Снимаем схему *Relations*

```
 ss = ss + vbLf + vbLf + "Отношения (RELATIONS)"
 For Each Rel In db.Relations
  ss = ss+vbLf+vbLf+"Название Отношения:" + Rel.Name
```

'Снимаем схему полей *Fields* отношения

```
 For Each fld In Rel.Fields
  ss = ss+vbLf+"Название поля отношения:" + fld.Name
  ss = ss+vbLf+"Внеш.назв.поля отношения:" & _
       fld.ForeignName
 Next fld
Next Rel
db.Close

hFileOut = FreeFile
```

```
Open pathx+"FO.txt" For Output Access Write As hFOut
Print #hFOut, ss
Close hFOut
End Sub
```

В результате работы программы получается показанная ниже распечатка. Единственное отличие реальных результатов от листинга, приведенного ниже, вместо свойств объектов (*db.Name*) будут показаны действительные данные.

"База данных DATABASE"

"Название:", db.Name
"Строка подключения:", db.Connect
"Поддержка транзакций:", db.Transactions
"Обновляемая?:", db.Updatable
"Порядок сортировки:", db.CollatingOrder
"Время ожидания запроса:", db.QueryTimeout

"Таблицы (TABLEDEFS)"

"Название:", Td.Name
"Создана:", Td.DateCreated
"Обновлена:", Td.LastUpdated
Td.Updatable (True, False)
Hex$(Td.Attributes)
Td.Attributes And dbSystemObject (0 или не 0)
Td.Attributes And dbAttachedTable "Присоединенная таблица"
Td.Attributes And dbAttachedODBC "Присоединенная ODBC-таблица"
Td.Attributes And dbAttachExclusive "Присоединенная таблица открыта в монопольном режиме"

"Поля(Fields)"

"Название:", fld.Name
"Тип:", fld.Type
"Размер:", fld.Size
"Биты атрибутов:", Hex$(fld.Attributes)
"Порядок сортировки:", fld.CollatingOrder
"Порядковый номер:", fld.OrdinalPosition
"Поле источника", fld.SourceField
"Таблица источника", fld.SourceTable

"Индексы (INDEXES)"

"Название:", Idx.Name
"Группированный:", Idx.Clustered
"Внешний:", Idx.Foreign
"Игнорировать Null:", Idx.IgnoreNulls

"Первичный:", Idx.Primary
"Неповторяющийся:", Idx.Unique
"Обязательный:", Idx.Required

"Поля(Fields)"

"Название", fld.Name
"Внешнее название", fld.ForeignName

"Отношения (RELATIONS)"

"Название:", Rel.Name
"Атрибуты:", Rel.Attributes
"Таблица:", Rel.Table
"Внешняя таблица:", Rel.ForeignTable

"Поля(Fields)"

"Название:", fld.Name
"Внешнее название:", fld.ForeignName

11.2. Уплотнение базы данных

Для повышения быстродействия *Visual Basic* откладывает удаление ненужных (удаленных) страниц до тех пор, пока не будет закрыта база данных и не будет произведено уплотнение неиспользуемых страниц. Через некоторое время можно обнаружить, что файл *mdb* возрастает пропорционально удаляемой и добавляемой информации.

Метод *CompactDatabase* даем возможность в процессе уплотнения применить шифрование данных или отключить его, изменить версию и национальный стандарт.

Перед уплотнением базы данных она должна быть закрыта. Также нельзя использовать одинаковые имена для исходных и результирующих баз. Уплотнение нельзя выполнять внутри транзакции

```
DBEngine.CompactDatabase исх_имя,Рез_имя,стандарт,парам.
```

Стандарт и параметры - такие же как и у *CreateDatabase*

```
DBEngine.CompactDatabase "C:\temp\my_old_db.mdb",
"C:\work\ my_new_db.mdb", dbLangGeneral
```

11.3. Восстановление базы данных

Прежде всего, под восстановлением базы данных понимают восстановление базы из архивов. Наиболее актуальным бывает вопрос восстановления базы данных при выходе из строя аппаратной части сервера базы данных. В этом случае базу данных развертывают на другом сервере и восстанавливают последнюю архивную копию. Очевидно, что надо регулярно архивировать данные. Проблемы могут случиться также и с программным обеспечением. В этом случае можно восстановить последнюю архивную копию, а отсутствующие записи «накатить» по журналу транзакций.

Существует еще одна интерпретация термина «восстановление» базы данных (*работает в MS Office до версии 2003 включительно*). Восстановление может быть выполнено с использования метода *RepairDatabase*. Данный метод проверяет все страницы базы данных на правильность связей, проверяет системные таблицы и все индексы. После успешного завершения база данных считается «надежной», а страницы, которые не могут быть спасены (как недействительные ссылки на другие страницы) уничтожаются. У данного метода только один аргумент

```
DBEngine.RepairDatabase "MyBase.mdb"
```

После восстановления база данных может увеличиться в размерах, т.к. процесс создания индексов может оставить некоторые удаленные страницы в базе. Поэтому после выполнения операции восстановления необходимо выполнить *CompactDatabase* для уничтожения неиспользуемых страниц.

Классы и методы *DAO* для работы со структурой базы данных и для манипуляции самими данными во многих случаях полезны и удобны, но есть один недостаток. Не требуя дополнительных настроек, они работают только с базами данных *MS Access* и *MS SQLserver* Для того, чтобы добиться универсальности при работе программы желательно использовать язык SQL как для создания и модификации структуры базы данных, так и при работе с данными. Еще более универсальный способ – использовать ODBC соединение и SQL запросы. Все это будет показано ниже.

12. Создание и изменение таблицы при помощи SQL запроса

Для создания таблицы будем использовать переменную MyDB, имеющую тип Database, метод Execute, выполняющий SQL запрос и сам запрос.

```
MyDB.Execute "CREATE TABLE town ([key_town] LONG," &_
[name_town] TEXT (100), [region_town] TEXT (100))"
```

Добавить, удалить, изменить столбцы можно при помощи ALTER TABLE.

```
ALTER TABLE stud ADD COLUMN spec_stud TEXT (50)
```

Для удаления поля используем *DROP*

```
ALTER TABLE stud DROP COLUMN spec_stud
```

Используя *ALTER TABLE*, можно одновременно удалить или добавить только один столбец.

12.1. Создание и удаление индексов

По определению реляционной базы данных информация в таблицах (отношениях) хранится неупорядоченно (в том порядке, в котором она поступала). Этот факт существенно затрудняет поиск нужной нам информации. Своеобразным способом «наведения порядка» в таблице является создание индекса, другими словами «сортировка» записей по одному или нескольким индексируемым полям. Использование индексов при выполнении поисковых запросов может сократить время выполнения такого запроса на порядок, а то и больше!

Существует три различных способа создания индексов

- при создании таблицы с помощью команды *CREATE TABLE*;
- с помощью команды *CREATE INDEX;*
- с помощью команды *ALTER TABLE*

Если мы хотим добавить внешний ключ и поддерживать целостность ссылок, то необходимо использовать предложение *CONSTRAINT* в командах *CREATE TABLE и ALTER TABLE.*

Иногда можно сначала создать таблицу при помощи *CREATE TABLE*, а затем добавить индекс с помощью операторов *CREATE INDEX* или *ALTER TABLE*.

12.2. Создание индекса при помощи CREATE TABLE

Пример создания индекса по двум столбцам

```
 CREATE TABLE stud ([fio_stud] TEXT (50),
[birthday_stud] DATATIME, [key_town] LONG, CONSTRAINT
stud_index UNIQUE ([fio_stud], [birthday_stud]))
```

Чтобы проиндексировать только один столбец, предложение *CONSTRAINT* помещается в определение столбца и записывается после типа данных индексируемго столбца

```
CREATE TABLE stud ([key_stud] counter CONSTRAINT key_stud PRIMARY, [fio_stud] TEXT (50), [birthday_stud] DATATIME, [key_town] dbLong)
```

12.3. Создание индекса при помощи CREATE INDEX

```
CREATE UNIQUE INDEX MyIndex ON town ([key_town])
```

В необязательном предложении WITH можно устанавливать условия на значения:

- *PRIMARY* – столбец первичного индекса;
- *DISALLOW NULL* – столбец не может быть оставлен пустым
- *IGNORE NULL* – запись будет проигнорирована, если этот столбец пуст

```
CREATE UNIQUE INDEX MyIndex ON stud (fio_stud) WITH DISALLOW NULL
```

Запись не будет вставлена в таблицу, если поле *fio_stud* будет пустым

Можно использовать *CREATE INDEX*, чтобы создать индекс для присоединенной таблицы, которая его не имеет. Для этого не обязательно иметь права на доступ к таблице. Достаточно иметь права только на чтение.

12.4. Создание индекса при помощи ALTER TABLE

```
ALTER TABLE stud ADD CONSTRAINT key_stud PRIMARY (key_stud)
```

Можно добавить индекс по многим полям

```
ALTER TABLE stud ADD CONSTRAINT MyIndex UNIQUE (fio_stud, birthday_stud)
```

12.5. Предложение CONSTRAINT и целостность ссылок

Предложение *CONSTRAINT* позволяет определять первичные и внешние ключи, чтобы установить отношения и поддерживать целостность ссылок

12.6.Создание первичного ключа по одному полю

CONSTRAINT имя {PRIMARY KEY | UNIQUE | REFERENCES внешняя таблица [(внешнее поле1, внешнее поле2)]}

12.7.Создание первичного ключа по многим полям

CONSTRAINT имя {PRIMARY KEY (первич1 [, первич2]) | UNIQUE (уник1 [, уник2]) | FOREIGN KEY (ссылка1 [, ссылка2]) REFERENCES внешняя таблица [(внешнее поле1, внешнее поле2)]}

В таблице 12.1 приведены параметры, используемые при создании первичного ключа

Таблица 12.1

Аргумент	Описание
Имя	Наименование создаваемого ключа
первич1, первич2	Поля первичного ключа
уник1, уник2	Поля неповторяющегося ключа
ссылка1, ссылка 2	Наименование поля внешнего ключа или полей, которые ссылаются на поля в другой таблице
внешняя_таблица	Наименование внешней таблицы, содержащей поле или поля, определяемые параметром внешнее поле
внешнее_поле1	Наименование поля или полей в таблице внешняя_таблица, заданных ссылка1, ссылка2

Пример добавления внешнего ключа key_town к таблице stud для создания отношения (использование ключевых слов FOREIGN KEY) межу таблицей stud и таблицей town

ALTER TABLE stud ADD CONSTRAINT key_town FOREIGN KEY (key_town) REFERENCES town (key_town)

12.8.Ключевые слова

UNIQUE – определяет поле как неповторяющийся ключ. Если ключ определен по нескольким полям, то неповторяющейся должна быть комбинация полей

PRIMARY KEY – определяет набор полей как первичный ключ. В таблице может быть только один первичный ключ. Значения полей должны быть неповторяющиеся

FOREIGN KEY – определяет поле как внешний ключ. Если первичный ключ внешней таблицы состоит более чем из одного поля, необходимо использовать определение составного индекса, перечисляя все ссылающиеся поля, имя внешней таблицы и имена полей, на которые делается ссылка в внешней таблице в той же последовательности, в которой указаны ссылающиеся поля

Ниже приведен пример, в котором создается база данных, таблица, и первичный ключ

```
Sub proba_DB5()
Dim i As Integer
Dim strSQL As String
Dim path As String

path = ActiveWorkbook.path + "\"
Set MyWs = DBEngine.Workspaces(0)
If Dir(path + "DBproba2.mdb") <> "" Then Kill path +
"DBproba2.mdb"

Set MyDB = MyWs.CreateDatabase(path + "DBproba2.mdb", _
dbLangCyrillic, dbVersion40)

strSQL = "Create table stud ([key_stud] COUNTER
CONSTRAINT key_stud PRIMARY KEY, [fio_stud] TEXT (50),
[birthday_stud] DATETIME, [key_town] LONG )"
MyDB.Execute strSQL

strSQL = "Create table town ([key_town] COUNTER
CONSTRAINT key_town PRIMARY KEY, [name_town] TEXT
(50))"
MyDB.Execute strSQL
```

```
strSQL = "ALTER table stud ADD CONSTRAINT key_town
FOREIGN KEY(key_town) REFERENCES town (key_town)"
MyDB.Execute strSQL

MyDB.Close
End Sub
```

13. Работа с базами данных при помощи ADO

Говоря о доступе к базам данных, нельзя не упомянуть о возможностях, которые предоставляют объекты *ADO (ActiveX Data Object).* Если до сих пор мы работали с базой данных (*СУБД MS Access*), как с файлом, то сейчас мы покажем как выполнять обработку данных, используя так называемый источник данных. Для этого рассмотрим, что такое *ODBC.*

ODBC (Open Database Connectivity) источниках соединения с базой. Для работы с *ODBC* необходимо выполнить следующие действия:

1. Если это необходимо, то выполнить установку *ODBC* драйвера для вашей *СУБД.* Возможно для этого понадобится установка клиентского *ПО* на ваш персональный компьютер. Все работы выполняются с правами администратора.

2. В меню «Пуск/Настройка/Панель управления/Администрирование» открыть иконку Источники данных (ODBC) Открыть закладку «*Пользовательский DSN*» и нажать кнопку «*Добавить*». В появившемся окне выбрать драйвер Microsoft Access Driver (в зависимости от версии может быть – **.mdb, может быть *.mdb, *.accdb)* В поле имя источника данных набрать “con_weather”. В разделе «База данных» нажать кнопку выбрать и в диалоговом окне выбрать базу, например «*...\DBproba1.mdb*» (может быть расширение *accdb).* Завершив указанные выше операции, нажать кнопку ОК.

3. Перейти в режим редактирования *VBA.* В меню *Tools\ References* выбрать библиотеку *Microsoft ActiveX Data Objects 2.8 Library.*

4. Набрать и выполнить следующую программу

```
Sub db_stud()
Dim MyCon As New Connection
Dim StrSQL As String
Dim rs As Recordset

strSQL = "Select data_weather, temperature_weather
from weather"
MyCon.Open "con_weather"
Set rs = New Recordset
rs.Open StrSQL, MyCon, adOpenForwardOnly,
adLockReadOnly, adCmdText
```

```
ActiveWorkbook.Worksheets("Лист1").Activate
i = 1

Do Until rs.EOF()
  Cells(i, 1) = rs!data_weather
  Cells(i, 2) = rs!temperature_weather
  rs.MoveNext
  i = i + 1
Loop
rs.Close
MyCon.Close
End Sub
```

Если вы работаете через *ODBC* источник, то существует альтернативный рассмотренному выше вариант получения информации о структуре базы данных основанный на использовании метода *OpenSchema*, объекта «соединение»:

```
Набор записей = соединение.OpenSchema
                    (QueryType, Criteria, SchemaID)
```

Таблица 13.1 Параметры схемы таблиц

QueryType	*Criteria*
adSchemaTables	*TABLE_CATALOG* *TABLE_SCHEMA* *TABLE_NAME* *TABLE_TYPE*

Пример получения списка всех таблиц и запросов

```
Set rs = cn.OpenSchema(adSchemaTables)
 While Not rs.EOF
   Debug.Print rs!TABLE_NAME
   rs.MoveNext
Wend
```

Пример получения списка только таблиц

```
Set rs = cn.OpenSchema(adSchemaTables,
                  Array(Empty, Empty, Empty, "Table")
```

Для получения списка только таблиц в базе данных Pubs Microsoft SQL Server, используйте следующую команду:

```
Set rs = cn.OpenSchema(adSchemaTables,
                 Array("Pubs", Empty, Empty,"Table")
```

Для получения списка полей в таблице можно использовать *adSchemaColumns*

Таблица 13.2 Параметры схемы полей таблицы

QueryType	*Criteria*
adSchemaColumns	*TABLE_CATALOG* *TABLE_SCHEMA* *TABLE_NAME* *COLUMN_NAME*

```
Set rs = cn.OpenSchema(adSchemaColumns,Array(Empty,
Empty, "Employees")
While Not rs.EOF
     Debug.Print rs!COLUMN_NAME
     rs.MoveNext
Wend
```

Для получения списка имен индексов в таблице *adSchemaIndexes*

Таблица 13.3 Параметры схемы индексов

QueryType	*Criteria*
adSchemaIndexes	*TABLE_CATALOG* *TABLE_SCHEMA* *INDEX_NAME* *TYPE* *TABLE_NAME*

```
Set rs = cn.OpenSchema(adSchemaIndexes,
       Array(Empty, Empty, Empty, Empty, "Employees")
While Not rs.EOF
  Debug.Print rs!INDEX_NAME
  rs.MoveNext
Wend
```

Чтобы просмотреть список индексов в таблице *Authors* базы данных Pubs *SQL Server* с использованием конструкции *adSchemaIndexes*, можно воспользоваться следующей записью:

```
Set rs = cn.OpenSchema(adSchemaIndexes,
        Array("Pubs", "dbo", Empty, Empty, "Authors")
```

Ниже приведены примеры нескольких программ, работающих со структурой БД

'Открываем определенное соединение *connection*.

```
Dim cn As New ADODB.Connection
Dim rs As New ADODB.Recordset
Private Sub Command1_Click()
```

'Получаем информацию об определенных полях

```
Set rs = cn.OpenSchema(adSchemaColumns, Array("pubs",
"dbo", "authors"))
While Not rs.EOF
  Debug.Print rs!COLUMN_NAME
  rs.MoveNext
Wend
End Sub
Private Sub Command2_Click()
```

'Получаем информацию об основном ключе таблицы

```
Set rs = cn.OpenSchema(adSchemaPrimaryKeys,
                       Array("pubs","dbo", "authors"))
MsgBox rs!COLUMN_NAME
End Sub

Private Sub Command3_Click()
```

'Получаем информацию обо всех таблицах

```
Dim criteria(3) As Variant
criteria(0) = "pubs"
criteria(1) = Empty
criteria(2) = Empty
criteria(3) = "table"
Set rs = cn.OpenSchema(adSchemaTables, criteria)
While Not rs.EOF
   Debug.Print rs!TABLE_NAME
   rs.MoveNext
Wend
End Sub
```

Приведенный ниже пример программы позволяет получить список таблиц в базе данных

' Перед решением необходимо в *TOOLS->REFERENCES* выбрать Microsoft ' *ActiveX Data Objects 2.8*

```
Sub Test_STRUCT_table()
```

```
Dim MyCon As New Connection
MyCon.Open "testr"
Set rs = MyCon.OpenSchema(adSchemaTables)
Workbooks("Example_DB.xls").Worksheets("Table_Structu
re1").Activate
i = 2
   While Not rs.EOF
     Cells(i, 1).Value = rs!TABLE_NAME
     Cells(i, 2).Value = rs!TABLE_CATALOG
     Cells(i, 3).Value = rs!TABLE_SCHEMA
     Cells(i, 4).Value = rs!TABLE_TYPE
      i = i + 1
      rs.MoveNext
   Wend
MyCon.Close
Cells(1, 1).Value = i - 2
End Sub
```

Следующая программа показывает как получать информацию о полях таблиц.' Перед решением необходимо в TOOLS->REFERENCES Microsoft ActiveX Data Objects 2.8

```
Sub Test_STRUCT_columns()
Dim MyCon As New Connection
MyCon.Open "testr"
Workbooks("Example_DB.xls").Worksheets
                        ("Table_Structure1").Activate
For ii = 2 To Cells(1, 1).Value
  Set rs = MyCon.OpenSchema(adSchemaColumns,
               Array(Empty, Empty, Cells(ii, 1).Text)
  i = 2
  While Not rs.EOF
    Cells(ii, 5 + i).Value = rs!COLUMN_NAME
    i = i + 1
    rs.MoveNext
  Wend
Next ii
MyCon.Close
End Sub
```

14. Приложение 1. Проект «База данных Студент»

Предлагается разработать простой проект работы с базой данных. Данный проект демонстрирует взаимодействие пользователя с единственной таблицей, хранящей информацию о студентах. Интерфейс работы с базой данных реализован посредством Пользовательской формы. Пользовательский Интерфейс позволяет добавлять, удалять, изменять информацию в таблице, просматривать текущие записи и получать отчет о студентах с использованием поискового фильтра по Фамилии.

Для начала в базе данных «*MyDB1.mdb*», находящейся в том же рабочем каталоге, что и рабочая книга с разрабатываемы проектом, создадим таблицу «*stud*» имеющую следующую структуру (Рис.14.1):

Имя поля	Тип данных
key_stud	Счетчик
fio_stud	Текстовый
birthday_stud	Дата/время
sex_stud	Логический
date_in_stud	Дата/время
spec_stud	Числовой

Рис.14.1 Таблица «stud»

Далее переименуем рабочий лист нашей рабочей книги в *«INFO_OUT»*. Следующим шагом создадим специальный класс работы с базой данных. Методы данного класса будут реализовывать основные функции работы с базой данных (некоторые даже не будут использоваться в данном проекте), а также функции взаимодействия с основной пользовательской формой.

Для создания класса добавляем новый класс *«Stud_Info2»* в рабочий проект и пишем следующий код:

```
Dim myWS As Workspace
Dim fio As String
Dim bd As Date
Dim sex As Boolean
Dim date_in As Date
Dim spec As Integer
Public myDB As Database
```

‘Методы, отвечающие за установку свойств класса

```
Sub set_fio(ff As String)
 If ff = "" Then
   MsgBox "Введите правильную фамилию"
 Else
   fio = ff
 End If
End Sub
```

```
Sub set_birthday(dd As Date)
If Year(dd) = 1899 Then
  MsgBox "Введите правильную дату рождения"
Else
  bd = dd
End If
End Sub

Sub set_date_in(dd As Date)
If Year(dd) = 1899 Then
  MsgBox "Введите правильную дату поступления"
Else
  date_in = dd
End If
End Sub

Sub set_sex(ss As Boolean)
 sex = ss
End Sub
Sub set_spec(sp As Integer)
 If sp <= 0 Then
   MsgBox "Введите правильное значение"
 Else
  spec = sp
 End If
End Sub
```

‘Метод проверяет корректность начальных установок

```
Private Function Check_Ok() As Boolean
  Check_Ok = True
  If fio = "" Then Check_Ok = False
  If Year(bd) = 1899 Then Check_Ok = False
  If Year(date_in) = 1899 Then Check_Ok = False
  If spec = 0 Then Check_Ok = False
End Function
```

‘Метод выполняет роль «конструктора» класса

```
Private Sub Class_Initialize()
 fio = ""
 bd = 0
 sex = False
 date_in = 0
 spec = 0
 ok = False
End Sub
```

‘Метод проверяет наличие базы данных и открывает ее

```
Function Check_Open_DB(pathx As String, dbName As String) As
Boolean
```

```
Check_Open_DB = True
On Error GoTo errh

 Set myWS = DBEngine.Workspaces(0)
  If Dir(pathx + dbName) <> "" Then
   Set myDB = myWS.OpenDatabase(pathx + dbName)
   If myDB.Name <> pathx + dbName Then
     Check_Open_DB = False
     MsgBox "You HAVE some problems with DATABASE"
     Exit Function
   End If
 Else
   Check_Open_DB = False
 End If
Exit Function
errh:
 Check_Open_DB = False
End Function
```

‘Метод удаляет определенную базу данных

```
Function Kill_Database(pathx As String, dbName As String) As
Boolean
 If Dir(pathx + dbName) <> "" Then
     Kill pathx + dbName
     Kill_Database = True
 Else
     MsgBox "Could not find DATABASE " & pathx + dbName
     Kill_Database = False
End If

End Function
```

‘Метод создает базу данных с определенным именем и по определеггному пути

```
Function Create_Database(pathx As String, dbName As String) As
Boolean
  If Dir(pathx + dbName) = "" Then
    Set myWS = DBEngine.Workspaces(0)
    Set myDB = myWS.CreateDatabase(pathx + dbName,
dbLangCyrillic, dbVersion40)
    Create_Database = True
  Else
    MsgBox "DATABASE " & pathx + dbName & " already exists"
    Create_Database = False
  End If
  End Function
```

‘Метод проверяет наличие определенной таблицы в открытой базе данных

```
Function Check_Table(tableName As String) As Boolean
 Dim td As TableDef
 Check_Table = False
 For Each td In myDB.TableDefs
```

```
    If tableName = td.Name Then
      Check_Table = True
      Exit Function
    End If
  Next
End Function
```

‘Метод удаляет определенную таблицу в базе данных

```
Function Drop_table(tableName As String) As Boolean
 Dim strSQL As String
 If Check_Table(tableName) = True Then
    strSQL = "drop table " & tableName
    myDB.Execute strSQL
    Drop_table = True
 Else
    MsgBox "There is no table " & tableName & " in the DATABASE"
    Drop_table = False
 End If
End Function
```

‘Метод создает определенную таблицу в базе данных

```
Function Create_Table(tableName As String) As Boolean

 Dim strSQL As String
 If Check_Table(tableName) = False Then
    strSQL = "Create table " & tableName & " ([key_stud] COUNTER
CONSTRAINT key_stud PRIMARY KEY," & _
           " [fio_stud] TEXT (50), [birthday_stud] DATETIME,
[sex_stud] BIT, [date_in_stud] DATETIME, " & _
           "[spec_stud] LONG)"
    myDB.Execute strSQL
    Create_Table = True
  Else
    MsgBox "The table " & tableName & " in the DATABASE"
    Create_Table = False
  End If
End Function
```

‘Метод проверяет наличие в базе данных (в таблице Stud) записи с
‘определенными характеристиками

```
Function Check_InsDB() As Boolean
  Check_InsDB = True
  Dim strSQL As String
  Dim rst As Recordset
  If Check_Ok() = True Then
    strSQL = "select * from stud where fio_stud like '" & fio &
"' and birthday_stud = " & Format(bd, "\#mm\/dd\/yyyy\#") & "and
sex_stud = " & sex & " and  date_in_stud = " & Format(date_in,
"\#mm\/dd\/yyyy\#") & " and spec_stud = " & spec

Set rst = myDB.OpenRecordset(strSQL, dbOpenSnapshot)
```

```
    If rst.RecordCount = 0 Then
         Check_InsDB = True
         'MsgBox "OK"
     Else
         Check_InsDB = False
         MsgBox "Information ALREADY in the dataBASE"
     End If
     rst.Close
   Else
     Check_InsDB = False
   End If

End Function
```

‘Метод реализует добавление информации в базу данных (в таблицу Stud)

```
Function Exec_InsDB() As Boolean
Dim strSQL As String
If Check_InsDB() = True Then
   strSQL = "insert into stud (fio_stud, birthday_stud,sex_stud,
date_in_stud, spec_stud) values ('" + fio & "', " & Format(bd,
"\#mm\ /dd\ /yyyy\#") & ", " & sex & ", " & Format(date_in,
"\#mm\/dd\/yyyy\#") & ", " & spec & ")"
   myDB.Execute strSQL
   Exec_InsDB = True
 Else
   Exec_InsDB = False
 End If
End Function
```

‘Метод реализует добавление информации в базу данных (в таблицу Stud)

```
Function Exec_UpdDB(key As Integer) As Boolean
Dim strSQL As String
If Check_InsDB() = True Then
  strSQL = "update stud set fio_stud = '" & fio & "',
  birthday_stud = " & Format(bd, "\#mm\/dd\/yyyy\#") & _
  ", sex_stud = " & sex & ", date_in_stud = " & Format(date_in,
  "\#mm\/dd\/yyyy\#") & ", spec_stud = " & spec & " where
  key_stud = " & key
  MsgBox strSQL
  myDB.Execute strSQL
  Exec_UpdDB = True
 Else
       Exec_UpdDB = False
 End If
End Function
```

‘Метод реализует удаление информации из базы данных (таблицы Stud)

```
Function Exec_DelDB(key As Integer) As Boolean
  Dim strSQL As String
  If key <> 0 Then
    strSQL = "Delete from stud where key_stud = " & key
```

```
    myDB.Execute strSQL
    Exec_DelDB = True
  Else
    MsgBox "Could not delete key=" & key
    Exec_DelDB = False
  End If
End Function
```

Далее создаем пользовательские формы для взаимодействия с базой данных. Это форма *UF_DATABASE (Рис.14.2)*, форма *UF_REPORT* (Рис.14.3), форма *UF_CALENDAR* (Рис.14.4).

Ниже изображения каждой формы приведен текст обработчиков событий элементов управления, связанных с соответствующей формой. Отметим, что данные формы практически не содержат программного кода, т.к. они в данном проекте предназначены только для отображения, ввода и изменения текущей информации, а вся обработка и взаимодействие с базой данных происходит в процедурах модуля (посредством выполнения методов разработанного класса). В форме только устанавливается флаг, соответствующий нажатой кнопке, и определяющий действие, которое надо выполнить в текущий момент.

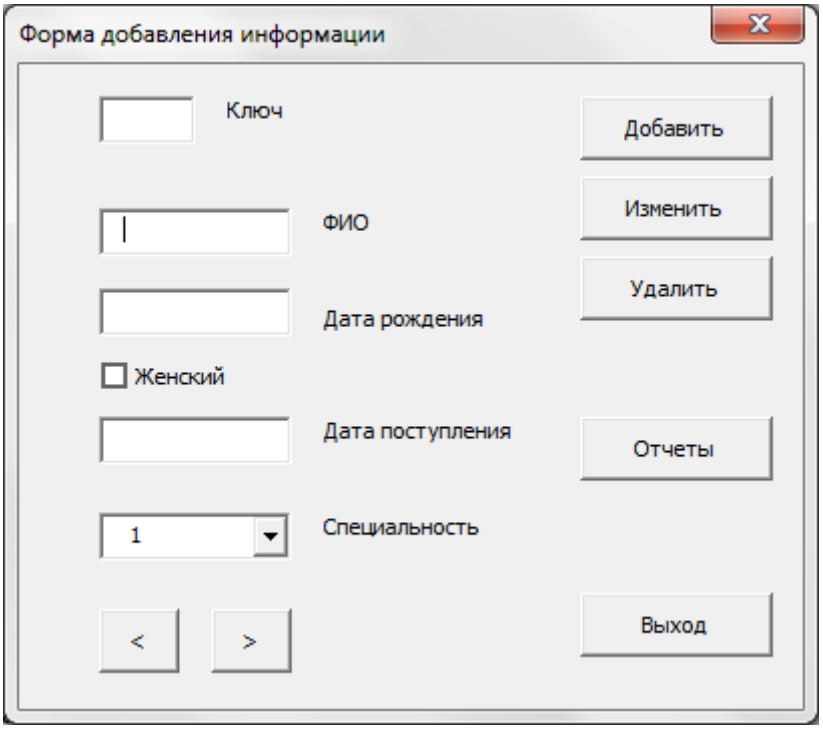


Рис.14.2

```
Public flagtb As Boolean
Public flagAll As Integer

Private Sub CheckBox1_Click()
If CheckBox1.Value = True Then
   CheckBox1.Caption = "Мужской"
Else
```

```
   CheckBox1.Caption = "Женский"
End If
End Sub

Private Sub CommandButton1_Click()
  flagAll = 4
  Hide
End Sub

Private Sub CommandButton2_Click()
  flagAll = 1
  Hide
End Sub

Private Sub CommandButton3_Click()
flagAll = 3
Hide
End Sub

Private Sub CommandButton4_Click()
flagAll = 2
Hide
End Sub

Private Sub CommandButton5_Click()
flagAll = 5
Hide
End Sub

Private Sub CommandButton6_Click()
flagAll = 6
Hide
End Sub

Private Sub CommandButton7_Click()
UF_REPORT.Show
Hide
End Sub

Private Sub TextBox2_MouseDown(ByVal Button As Integer, ByVal
Shift As Integer, ByVal X As Single, ByVal Y As Single)
 flagtb = False
 UF_CALENDAR.Caption = "Выбор даты рождения"
 If TextBox2.Text <> "" Then
    UF_CALENDAR.Calendar1.Value = TextBox2.Text
 End If
 UF_CALENDAR.Show
End Sub

Private Sub TextBox3_MouseDown(ByVal Button As Integer, ByVal
Shift As Integer, ByVal X As Single, ByVal Y As Single)
 flagtb = True
 UF_CALENDAR.Caption = "Выбор даты поступления"
If TextBox3.Text <> "" Then
    UF_CALENDAR.Calendar1.Value = TextBox3.Text
 End If
```

```
 UF_CALENDAR.Show
End Sub
```

```
Private Sub UserForm_Initialize()
  CheckBox1.Value = False
  CheckBox1.Caption = "Женский"
  flagtb = False
  For i = 1 To 5
    ComboBox1.AddItem (Str(i))
  Next i
  ComboBox1.ListIndex = 0
  flagAll = 0
  End Sub
```

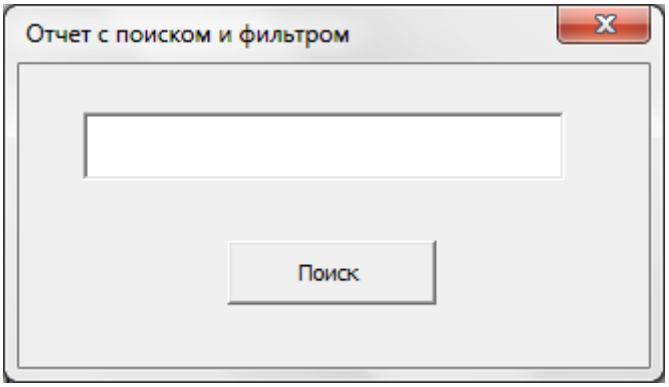


Рис.14.3

```
Private Sub CommandButton1_Click()
 UF_DATABASE.flagAll = 7
 Hide
End Sub
```

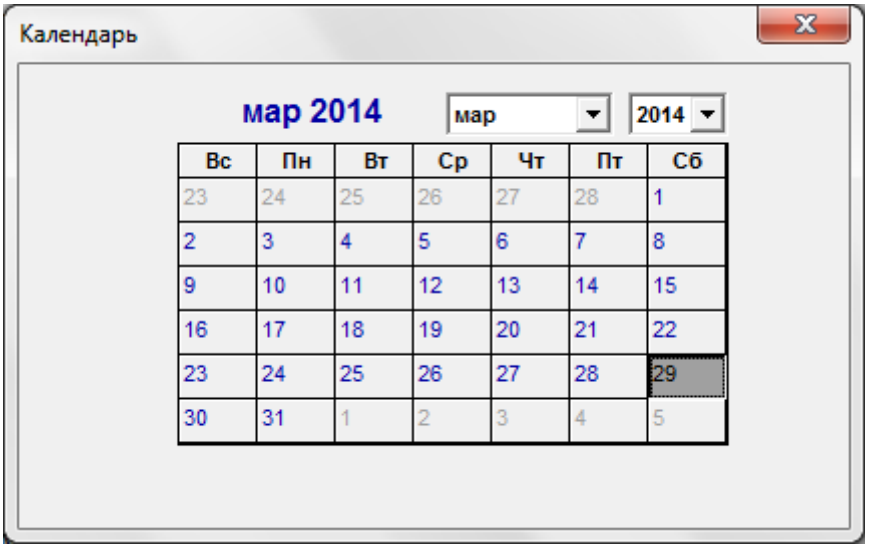


Рис.14.4

```
Private Sub Calendar1_DblClick()
If UF_DATABASE.flagtb = False Then
    UF_DATABASE.TextBox2.Value = Str(Calendar1.Value)
```

```
 Else
    UF_DATABASE.TextBox3.Value = Str(Calendar1.Value)
 End If
 Hide
 End Sub
```

Ниже приведена основная процедура модуля, которая практически выполняет все взаимодействия с пользователем и базой данных

```
Sub proba_x2()

Dim dbName As String, pathx As String, tableName As String
Dim wsheetName As String
Dim studi As New Stud_Info2
Dim rst As Recordset

wsheetName = "INFO_OUT"
dbName = "MyDB1.mdb"
tableName = "stud"
pathx = ActiveWorkbook.path + "\"

If studi.Check_Open_DB(pathx, dbName) = False Then
  MsgBox "You have some problems with DATABASE " & dbName & vbLf
& vbLf & "          Good Bye"
  Exit Sub
End If

If studi.Check_Table(tableName) = False Then
  MsgBox "You have some problems with DATABASE " & dbName & vbLf
& vbLf & "          Good Bye"
  Exit Sub
End If

 Set rst = studi.myDB.OpenRecordset("Select * from stud",
dbOpenSnapshot)
 If rst.RecordCount <> 0 Then
   rst.MoveLast
   rst.MoveFirst
   Call UF_SET(rst)
 End If

 UF_DATABASE.flagAll = 0
 Do Until UF_DATABASE.flagAll = 1
 '*** До тех пор пока не нажата кнопка выход

   UF_DATABASE.Show (1)
   Select Case (UF_DATABASE.flagAll)

   Case 4:
   '*** Добавление информации в Таблицу

      Call CLASS_SET(studi)
```

```
      If studi.Exec_InsDB() = True Then
        strBuf = UF_DATABASE.TextBox1 & " " &
UF_DATABASE.TextBox2
        MsgBox strBuf & vbLf & " Information has been ADDED into
the DATABASE"
      End If
      Set rst = studi.myDB.OpenRecordset("Select * from stud",
dbOpenSnapshot)
      rst.MoveLast
      rst.MoveFirst
      Call UF_SET(rst)
   Case 5:
   '*** Изменение информации в Таблице

      Call CLASS_SET(studi)
      If studi.Exec_UpdDB(rst.Fields("key_stud").Value) = True
Then
        strBuf = UF_DATABASE.TextBox1 & " " &
UF_DATABASE.TextBox2
        MsgBox strBuf & vbLf & " Information has been UPDATED in
the DATABASE"
      End If
      Set rst = studi.myDB.OpenRecordset("Select * from stud",
dbOpenSnapshot)
      rst.MoveLast
      rst.MoveFirst
      Call UF_SET(rst)

  Case 2:
   '*** Переход к предыдущей записи

      rst.MovePrevious
      If rst.BOF = True Then
       rst.MoveLast
      End If
      Call UF_SET(rst)
  Case 3:
  '*** Переход к следующей записи

      rst.MoveNext
      If rst.EOF = True Then
        rst.MoveFirst
      End If
      Call UF_SET(rst)

  Case 6:
  '*** Удаление информации из Таблицы

      If studi.Exec_DelDB(rst.Fields("key_stud").Value) = True
Then
        strBuf = UF_DATABASE.TextBox1 & " " &
UF_DATABASE.TextBox2
```

```
        MsgBox strBuf & vbLf & " Information has been DELETED
from the DATABASE"
      End If
      Set rst = studi.myDB.OpenRecordset("Select * from stud",
dbOpenSnapshot)
      rst.MoveLast
      rst.MoveFirst
      Call UF_SET(rst)
  Case 7:
  '*** Получение отчета по запросу ("с параметром")

    ActiveWorkbook.Worksheets(wsheetName).Activate
    strSQL = "Select * from stud where fio_stud like '" &
UF_REPORT.TextBox1.Value & _
    "' order by fio_stud"
    Set rst = studi.myDB.OpenRecordset(strSQL, dbOpenSnapshot)

    fldc = rst.Fields.Count
    Range("A:F").ClearContents

    Range("A1").Value = "Ключ"
    Range("B1").Value = "Фамилия И.О."
    Range("C1").Value = "Дата Рождения"
    Range("D1").Value = "Пол"
    Range("E1").Value = "Дата поступления"
    Range("F1").Value = "Специальность"

    i = 2
    Do Until rst.EOF
        For j = 0 To fldc - 1
            If j = 3 Then
                If rst.Fields(j) = True Then
                    Cells(i, j + 1).Value = "М"
                Else
                    Cells(i, j + 1).Value = "Ж"
                End If
            Else
                Cells(i, j + 1).Value = rst.Fields(j)
            End If
        Next j
        i = i + 1
        rst.MoveNext
    Loop
    If rst.RecordCount <> 0 Then
        rst.MoveFirst
        Call UF_SET(rst)
     Else
        Call UF_CLR
     End If
 End Select
 Loop
```

```
 MsgBox "GOOD bye"


End Sub
Sub UF_SET(rst As Recordset)
  '*** Заполнение полей пользовательской формы данными из
рекордсета

   With UF_DATABASE
    .Caption = "Всего " & rst.RecordCount & " записей"
    .TextBox1.Value = rst.Fields("fio_stud")
    .TextBox2.Value = Str(rst.Fields("birthday_stud"))
    .CheckBox1.Value = rst.Fields("sex_stud")
    .TextBox3.Value = Str(rst.Fields("date_in_stud"))
    .ComboBox1.Value = rst.Fields("spec_stud")
    .TextBox4.Value = rst.Fields("key_stud")
   End With
End Sub

Sub UF_CLR()
    '*** Заполнение полей пользовательской формы

   With UF_DATABASE
   .Caption = "Всего 0 записей"
    .TextBox1.Value = ""
    .TextBox2.Value = ""
    .CheckBox1.Value = False
    .TextBox3.Value = ""
    .ComboBox1.Value = 0
    .TextBox4.Value = 0
   End With

End Sub

Sub CLASS_SET(ByRef studii As Stud_Info2)
'*** Заполнение свойств класса данными из пользовательской формы

    With UF_DATABASE
      studii.set_fio (.TextBox1.Value)
      studii.set_birthday (.TextBox2.Value)
      studii.set_date_in (.TextBox3.Value)
      studii.set_sex (.CheckBox1.Value)
      studii.set_spec (.ComboBox1.Value)
    End With
End Sub
```

Для запуска проекта надо запустить процедуру proba_x2()

15. Приложение 2. Проект «База данных Поставки»

Данный проект несколько сложнее чем предыдущий. Он предполагает написание программы, которая взаимодействует с базой данных через объект ADO, посредством ODBC источника. Данная программа должна подключиться к базе данных, проверить существуют ли в ней исходные таблицы. Если таблицы существуют, то необходимо сначала удалить связи между отношениями, затем удалить сами таблицы (т.к. существующие таблицы могут носить имена таблиц проекта, но их структура может быть для нас не подходящей). Далее программа должна:

- создать нужные нам таблицы заново;
- создать необходимые связи между отношениями.
- загрузить информацию о товарах, базах хранения товаров и товарных накладных в созданную нами базу данных из текстовых файлов.

Базовая часть проекта должна обеспечивать пользователю возможность получать информацию о товарах, находящихся на складах товарных баз и о накладных, по которым данные товары отгружались. Отчеты о товарах должны выводится в Лист MS Excel, документ Word, список пользовательской формы, текстовый файл по выбору пользователя. Также пользователь должен иметь возможность выводить в окно списка (ListBox) пользовательской формы информацию о накладных.

Структура таблиц, которые необходимо создать показана на рисунках (Рис.15.1- Рис.15.6). Структура связей представлена на Рис.15.7.

Пользовательская форма показана на Рис.15.8. Ниже каждой формы приведен код, обработки событий, связанных с элементами управления, размещенными на форме. Далее представлен текст основных процедур проекта

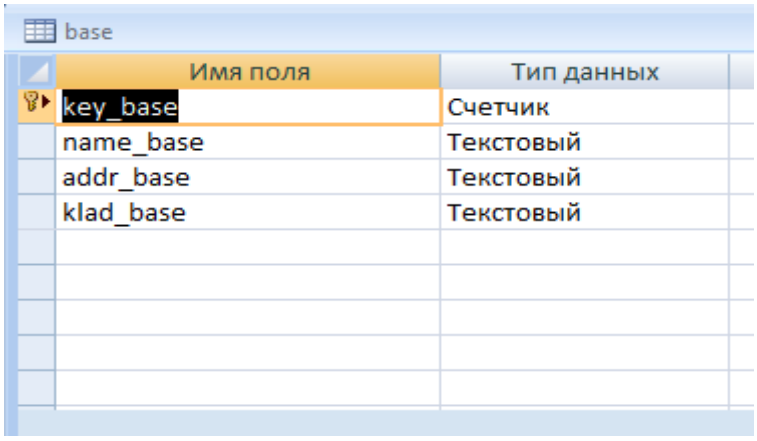


Рис.15.1

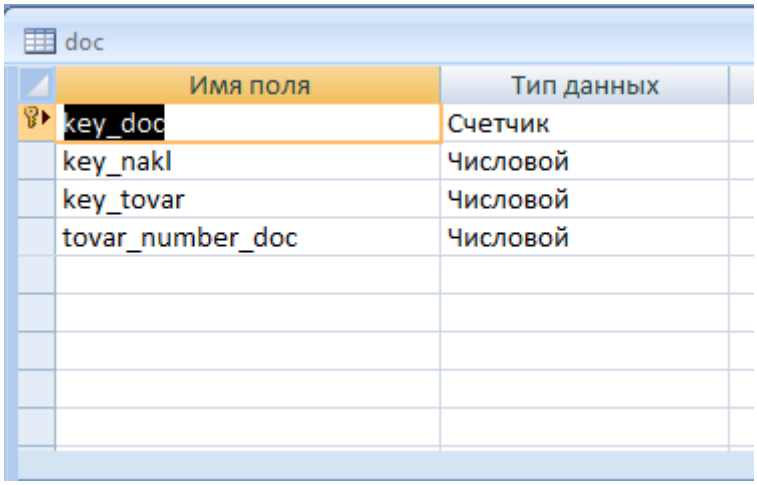


Рис.15.2

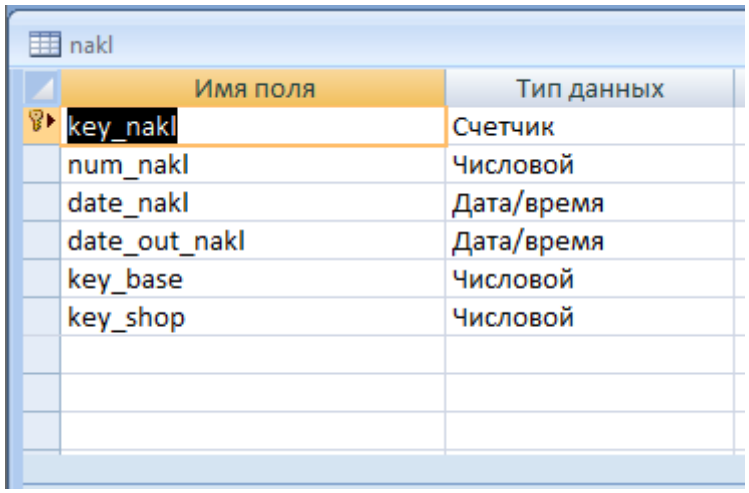


Рис.15.3

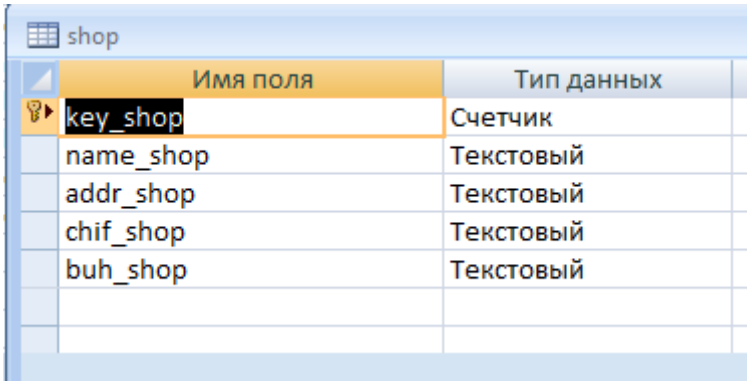


Рис.15.4

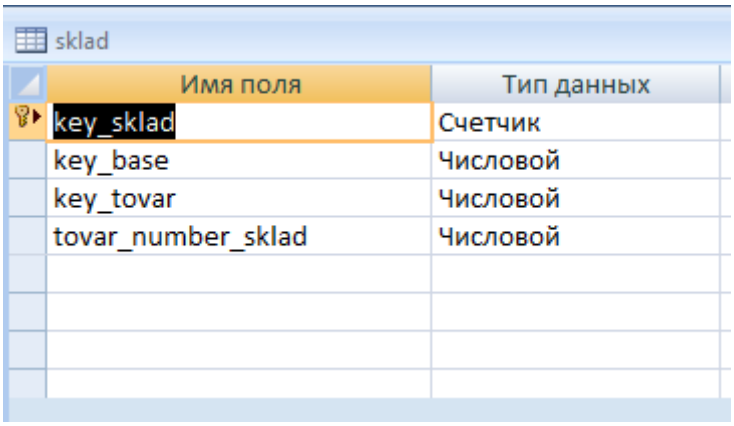


Рис.15.5

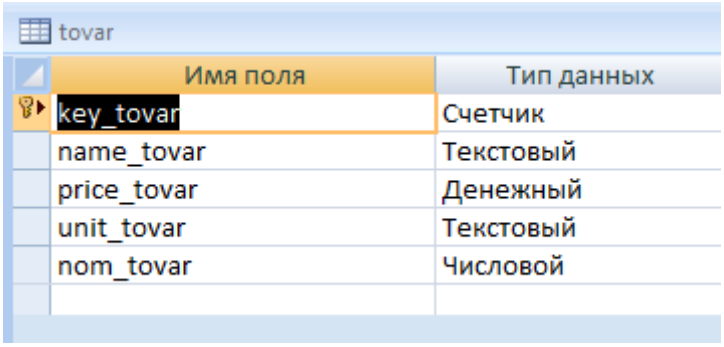


Рис.15.6

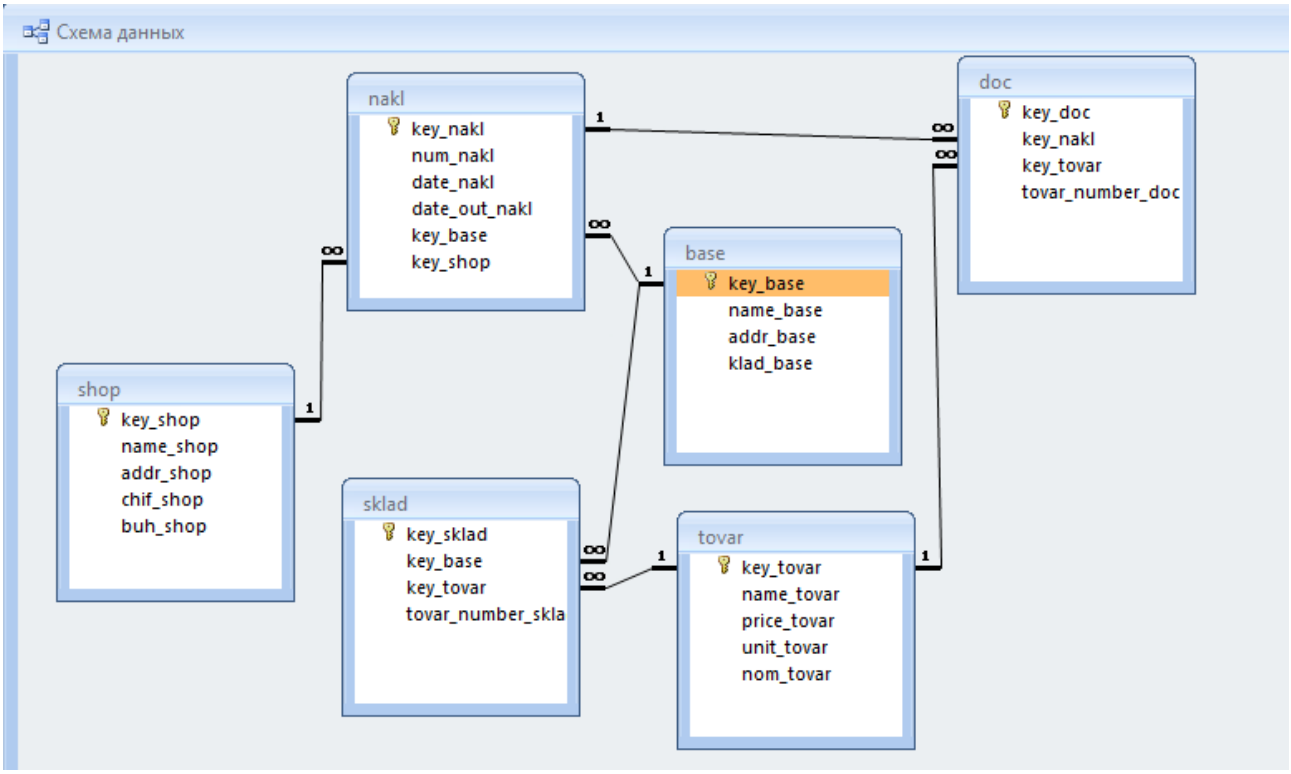


Рис.15.7

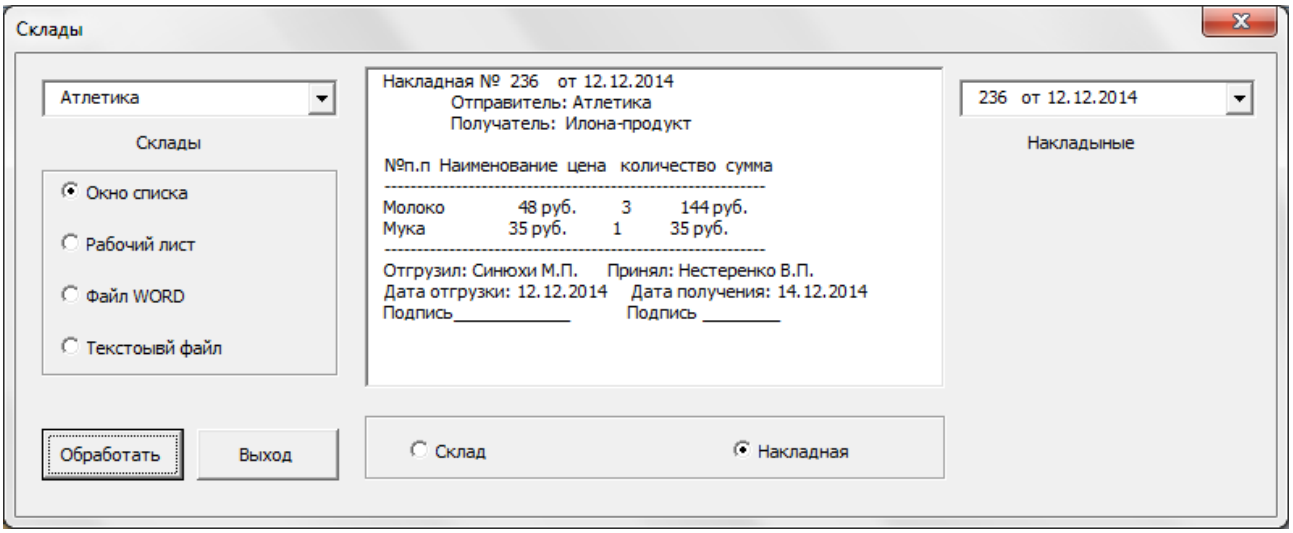


Рис.15.8

```
Private Sub CommandButton1_Click()
  Call Module12.show_sklad
End Sub

Private Sub CommandButton2_Click()
  Unload UF_SKLAD
End Sub

Private Sub OptionButton1_Click()
  Module12.radio_but = "win"
End Sub

Private Sub OptionButton2_Click()
  Module12.radio_but = "wsheet"
End Sub

Private Sub OptionButton3_Click()
  Module12.radio_but = "word"
End Sub

Private Sub OptionButton4_Click()
  Module12.radio_but = "file"
End Sub

Private Sub OptionButton5_Click()
  Module12.radio_but_obj = "nakl"
End Sub

Private Sub OptionButton6_Click()
  Module12.radio_but_obj = "sklad"
End Sub

Private Sub UserForm_Initialize()
  Call Module12.clr_wsheet
End Sub
```

Процедуры модуля будут взаимодействовать с формой через глобальные переменные, также для работы будет нужен пользовательский тип данных. Все эти переменные и структура должны быть объявлены в начале модуля.

```
Public radio_but As String
Public radio_but_obj As String

Type nakl_struct
  key As Integer
  num As Integer
  datex As Date
  date_outx As Date
  klad As String
  buh As String
  shop As String
  base As String
End Type
```

Далее приводим код удаления старых таблиц и добавления новых. Для удобства работы и отладки программы все действия будем протоколировать в текстовый файл.

```
Sub db3004_creation()
Dim mCON As New Connection
Dim strSQL As String
Dim hFileOut As Long
Dim path_x As String
Dim fOutName As String
fOutName = "ProtocolCreation.txt"
path_x = ActiveWorkbook.path + "\"

mCON.Open ("db_practic2")
```

'Проверка и удаление таблиц

```
' **** Check (drop) TABLES *****
```

'Открываем текстовый файл для протоколирования действий

```
hFileOut = FreeFile
If Dir(path_x + fOutName) = "" Then
    Open path_x + fOutName For Output Access Write As #hFileOut
Else
   Open path_x + fOutName For Append As #hFileOut
End If
Print #hFileOut,vbLf & "*START WORKING at " & Now() & "*" & vbLf
On Error GoTo err1
```

'Сначала удаляем связи

```
strSQL = "alter table nakl drop constraint key_base"
mCON.Execute strSQL
Print #hFileOut, strSQL & " -- " & Now()
```

```
strSQL = "alter table nakl drop constraint key_shop"
mCON.Execute strSQL
Print #hFileOut, strSQL & " -- " & Now()

strSQL = "alter table doc drop constraint key_nakl"
mCON.Execute strSQL
Print #hFileOut, strSQL & " -- " & Now()

strSQL = "alter table doc drop constraint key_tovar"
mCON.Execute strSQL
Print #hFileOut, strSQL & " -- " & Now()

strSQL = "alter table sklad drop constraint key_base_sklad"
mCON.Execute strSQL
Print #hFileOut, strSQL & " -- " & Now()

strSQL = "alter table sklad drop constraint key_tovar_sklad"
mCON.Execute strSQL
Print #hFileOut, strSQL & " -- " & Now()
```

'Открываем схему таблиц, для удаления только тех таблиц, которые в схеме

```
Set rs = mCON.OpenSchema(adSchemaTables)
   While Not rs.EOF
      Debug.Print rs!table_name

      strSQL = ""
       Select Case rs!table_name
       Case "shop"
         strSQL = "drop table shop"
       Case "base"
         strSQL = "drop table base"
       Case "tovar"
         strSQL = "drop table tovar"
       Case "nakl"
         strSQL = "drop table nakl"
       Case "doc"
         strSQL = "drop table doc"
       Case "sklad"
         strSQL = "drop table sklad"
      End Select

      If strSQL <> "" Then
         mCON.Execute strSQL
         Print #hFileOut, strSQL & " -- " & Now()

      End If
      rs.MoveNext
   Wend
```

```
' ***** create  TABLES *******
```

'Создаем нужные нам таблицы

```
strSQL = "CREATE TABLE shop ([key_shop] COUNTER CONSTRAINT
key_shop PRIMARY KEY, " & _
"[name_shop] TEXT (50), [addr_shop] TEXT (50), [chif_shop] TEXT
(50), [buh_shop] TEXT (50))"

mCON.Execute strSQL
MsgBox "Table shop created"
Print #hFileOut, "Table shop created " & " -- " & Now()

strSQL = "CREATE TABLE base ([key_base] COUNTER CONSTRAINT
key_base PRIMARY KEY, " & _
"[name_base] TEXT (50), [addr_base] TEXT (50), [klad_base] TEXT
(50))"

mCON.Execute strSQL
MsgBox "Table base created"
Print #hFileOut, "Table base created " & " -- " & Now()

strSQL = "CREATE TABLE tovar ([key_tovar] COUNTER CONSTRAINT
key_tovar PRIMARY KEY, " & _
"[name_tovar] TEXT (50), [price_tovar] CURRENCY, [unit_tovar]
TEXT (50), [nom_tovar] LONG)"

mCON.Execute strSQL
MsgBox "Table tovar created "
Print #hFileOut, "Table tovar created " & " -- " & Now()

strSQL = "CREATE TABLE nakl ([key_nakl] COUNTER CONSTRAINT
key_nakl PRIMARY KEY, " & _
"[num_nakl] LONG, [date_nakl] DATETIME, [date_out_nakl]
DATETIME, [key_base] LONG, [key_shop] LONG)"

mCON.Execute strSQL
MsgBox "Table nakl created "
Print #hFileOut, "Table nakl created " & " -- " & Now()

strSQL = "CREATE TABLE doc ([key_doc] COUNTER CONSTRAINT key_doc
PRIMARY KEY, " & _
"[key_nakl] LONG, [key_tovar] LONG, [tovar_number_doc] LONG )"

mCON.Execute strSQL
MsgBox "Table doc created"
Print #hFileOut, "Table doc created " & " -- " & Now()

strSQL = "CREATE TABLE sklad ([key_sklad] COUNTER CONSTRAINT
key_sklad PRIMARY KEY, " & _
"[key_base] LONG, [key_tovar] LONG, [tovar_number_sklad] LONG)"

mCON.Execute strSQL
```

```
MsgBox "Table sklad created"
Print #hFileOut, "Table sklad created " & " -- " & Now()
```

'Создаем связи между таблицами

```
strSQL = "ALTER TABLE nakl ADD CONSTRAINT key_base FOREIGN KEY
(key_base) REFERENCES base (key_base)"
mCON.Execute strSQL
MsgBox "CONSTRAINT nakl <-> base added"
Print #hFileOut, "CONSTRAINT nakl <-> base added"

strSQL = "ALTER TABLE nakl ADD CONSTRAINT key_shop FOREIGN KEY
(key_shop) REFERENCES shop (key_shop)"
mCON.Execute strSQL
MsgBox "CONSTRAINT nakl <-> shop added"
Print #hFileOut, "CONSTRAINT nakl <-> shop added " & " -- " &
Now()

strSQL = "ALTER TABLE doc ADD CONSTRAINT key_nakl FOREIGN KEY
(key_nakl) REFERENCES nakl (key_nakl)"
mCON.Execute strSQL
MsgBox "CONSTRAINT doc <-> nakl added"
Print #hFileOut, "CONSTRAINT doc <-> nakl added " & " -- " &
Now()

strSQL = "ALTER TABLE doc ADD CONSTRAINT key_tovar FOREIGN KEY
(key_tovar) REFERENCES tovar (key_tovar)"
mCON.Execute strSQL
MsgBox "CONSTRAINT doc <-> tovar added"
Print #hFileOut, "CONSTRAINT doc <-> tovar added " & " -- " &
Now()

strSQL = "ALTER TABLE sklad ADD CONSTRAINT key_base_sklad
FOREIGN KEY (key_base) REFERENCES base (key_base)"
mCON.Execute strSQL
MsgBox "CONSTRAINT sklad <-> base added"
Print #hFileOut, "CONSTRAINT sklad <-> base added " & " -- " &
Now()

strSQL = "ALTER TABLE sklad ADD CONSTRAINT key_tovar_sklad
FOREIGN KEY (key_tovar) REFERENCES tovar (key_tovar)"
mCON.Execute strSQL
MsgBox "CONSTRAINT sklad <-> tovar added"
Print #hFileOut, "CONSTRAINT sklad <-> tovar added " & " -- " &
Now()

Print #hFileOut, vbLf & " *** WORKING FINISH at" & " -- " & Now
& " *** "
Close (hFileOut)
mCON.Close

Exit Sub
```

'Обработка ошибок (вывод описания ошибки) и продолжени работы

```
err1:
MsgBox Err.Description
Print #hFileOut, Err.Description & " -- " & Now()
Resume Next

End Sub
```

Далее напишем процедуру, которая загружает данные в созданные таблицы из текстовых файлов. Эта же процедура может пригодится, если есть необходимость подгружать данные, полученные из внешних источников

```
Sub db3004_load()
Dim mCON As New Connection
Dim strSQL As String
Dim mas() As String
Dim strBuf As String
Dim fName As String
Dim hFile As Long
Dim path_x As String
Dim strMSG As String
```

'Получение пути к рабочему каталогу

```
Dim fOutName As String
path_x = ActiveWorkbook.path + "\"

fOutName = "ProtocolLoading.txt"
mCON.Open ("db_practic2")

hFileOut = FreeFile
```

'Открываем файл протокола для записи

```
If Dir(path_x + fOutName) = "" Then
    Open path_x + fOutName For Output Access Write As #hFileOut
Else
   Open path_x + fOutName For Append As #hFileOut
End If
Print #hFileOut,vbLf & "*START WORKING at " & Now() & "*" & vbLf
```

'Загрузка информации о магазинах из текстового файла

```
'1 ************************************************************
fName = "shop.txt"
If Dir(path_x + fName) = "" Then
  strMSG = "File " & fName & " not found!"
  MsgBox strMSG
  Print #hFileOut, strMSG & " -- " & Now()
  Close (hFileOut)
  Exit Sub
End If
```

```
hFile = FreeFile
Open path_x + fName For Input Access Read As #hFile
i = 0
k = 0
Do Until EOF(hFile)
    Line Input #hFile, strBuf
    mas = Split(strBuf, ";")
 If UBound(mas) = 3 Then
    k = k + 1
    strSQL = "select * from shop where name_shop like '" &
Trim(mas(0)) & "'"
    Set rst = New Recordset
    rst.Open strSQL, mCON, adOpenForwardOnly, adLockReadOnly,
adCmdText
    If rst.EOF = True Then
      strSQL = "INSERT into shop (name_shop, addr_shop,
chif_shop, buh_shop) values ('" & Trim(mas(0)) & "', '" & _
Trim(mas(1)) & "', '" & Trim(mas(2)) & "', '" & Trim(mas(3)) &
"')"
      mCON.Execute strSQL
      i = i + 1
    End If
    rst.Close
 End If
Loop
Close (hFile)
strMSG = i & " records from " & k & " added into shop"
MsgBox strMSG
Print #hFileOut, strMSG & " -- " & Now()
```

'Загрузка информации о базах из текстового файла

```
'2 ************************************************************

fName = "base.txt"
If Dir(path_x + fName) = "" Then
  strMSG = "File " & fName & " not found!"
  MsgBox strMSG
  Print #hFileOut, strMSG & " -- " & Now()
  Close (hFileOut)
  Exit Sub
End If

hFile = FreeFile
Open path_x + fName For Input Access Read As #hFile
i = 0
k = 0
Do Until EOF(hFile)
   Line Input #hFile, strBuf
   mas = Split(strBuf, ";")
   If UBound(mas) = 2 Then
     k = k + 1
```

```
    strSQL = "select * from base where name_base like '" &
Trim(mas(0)) & "'"
    Set rst = New Recordset
    rst.Open strSQL, mCON, adOpenForwardOnly, adLockReadOnly,
adCmdText
    If rst.EOF = True Then
        strSQL = "INSERT into base (name_base, addr_base,
klad_base) values ('" & Trim(mas(0)) & "', '" & _
Trim(mas(1)) & "', '" & Trim(mas(2)) & "')"
        mCON.Execute strSQL
        i = i + 1
     End If
     rst.Close
 End If
Loop
Close (hFile)
strMSG = i & " records from " & k & " added into base"
MsgBox strMSG
Print #hFileOut, strMSG & " -- " & Now()
```

'Загрузка информации о товарах из текстового файла

```
'3 ************************************************************
fName = "tovar.txt"

If Dir(path_x + fName) = "" Then
  strMSG = "File " & fName & " not found!"
  MsgBox strMSG
  Print #hFileOut, strMSG & " -- " & Now
  Close (hFileOut)
  Exit Sub
End If

hFile = FreeFile
Open path_x + fName For Input Access Read As #hFile
i = 0
k = 0
Do Until EOF(hFile)
    Line Input #hFile, strBuf
    mas = Split(strBuf, ";")
  If UBound(mas) = 3 Then
    k = k + 1
    strSQL = "select * from tovar where name_tovar like '" &
Trim(mas(0)) & "' and nom_tovar = " & Trim(mas(3))
    Set rst = New Recordset
    rst.Open strSQL, mCON, adOpenForwardOnly, adLockReadOnly,
adCmdText
    If rst.EOF = True Then
       strSQL = "INSERT into tovar (name_tovar, price_tovar,
unit_tovar, nom_tovar) values ('" & Trim(mas(0)) & "', " & _
Trim(mas(1)) & ", '" & Trim(mas(2)) & "', " & Trim(mas(3)) & ")"
```

```
        mCON.Execute strSQL
        i = i + 1
    End If
    rst.Close
  End If
Loop
Close (hFile)

strMSG = i & " records from " & k & " added into tovar"
MsgBox strMSG
Print #hFileOut, strMSG & " -- " & Now()
```

'Загрузка информации о накладных (заголовки) из текстового файла

```
' 4 ******************************************************
fName = "nakl.txt"

If Dir(path_x + fName) = "" Then
  strMSG = "File " & fName & " not found!"
  MsgBox strMSG
  Print #hFileOut, strMSG & " -- " & Now
  Close (hFileOut)
  Exit Sub
End If

hFile = FreeFile
Open path_x + fName For Input Access Read As #hFile
i = 0
k = 0
Do Until EOF(hFile)
    Line Input #hFile, strBuf
    mas = Split(strBuf, ";")
  If UBound(mas) = 4 Then
    k = k + 1
    strSQL = "select * from nakl where num_nakl = " &
Trim(mas(0)) & " and date_nakl = " & Format(Trim(mas(3)),
"\#mm\/dd\/yyyy\#")

    Set rst = New Recordset
    rst.Open strSQL, mCON, adOpenForwardOnly, adLockReadOnly,
adCmdText
    If rst.EOF = True Then
      strSQL = "INSERT into nakl (num_nakl, date_nakl,
date_out_nakl, key_base, key_shop) values (" & Trim(mas(0)) & ",
" & Format(Trim(mas(1)), "\#mm\/dd\/yyyy\#") & ", " &
Format(Trim(mas(2)), "\#mm\/dd\/yyyy\#") & ", " & Trim(mas(3)) &
_ "," & Trim(mas(4)) & ")"

     mCON.Execute strSQL
     i = i + 1
    End If
    rst.Close
```

```
 End If
Loop
Close (hFile)
strMSG = i & " records from " & k & " added into nakl"
MsgBox strMSG
Print #hFileOut, strMSG & " -- " & Now
```

'Загрузка информации о накладных (товары) из текстового файла

```
' 5 ************************************************************

fName = "doc.txt"
If Dir(path_x + fName) = "" Then
  strMSG = "File " & fName & " not found!"
  MsgBox strMSG
  Print #hFileOut, strMSG & " -- " & Now
  Close (hFileOut)
  Exit Sub
End If

hFile = FreeFile
Open path_x + fName For Input Access Read As #hFile
i = 0
k = 0
Do Until EOF(hFile)
    Line Input #hFile, strBuf
    mas = Split(strBuf, ";")
  If UBound(mas) = 2 Then
    k = k + 1
    strSQL = "select * from doc where key_nakl = " &
Trim(mas(0)) & " and key_tovar = " & Trim(mas(1))
    Set rst = New Recordset
    rst.Open strSQL, mCON, adOpenForwardOnly, adLockReadOnly,
adCmdText
    If rst.EOF = True Then
      strSQL = "INSERT into doc (key_nakl, key_tovar,
tovar_number_doc) values (" & Trim(mas(0)) & ", " & _
Trim(mas(1)) & ", " & Trim(mas(2)) & ")"

        mCON.Execute strSQL
        i = i + 1
    End If
    rst.Close
  End If
Loop
Close (hFile)
strMSG = i & " records from " & k & " added into doc"
MsgBox strMSG
Print #hFileOut, strMSG & " -- " & Now()
```

'Загрузка информации о складах из текстового файла

```
' 6 ********************************************************

fName = "sklad.txt"
If Dir(path_x + fName) = "" Then
  strMSG = "File " & fName & " not found!"
  MsgBox strMSG
  Print #hFileOut, strMSG & " -- " & Now
  Close (hFileOut)
  Exit Sub
End If

hFile = FreeFile
Open path_x + fName For Input Access Read As #hFile
i = 0
k = 0
Do Until EOF(hFile)
    Line Input #hFile, strBuf
    mas = Split(strBuf, ";")
  If UBound(mas) = 2 Then
    k = k + 1
    strSQL = "select * from sklad where key_base = " &
Trim(mas(0)) & " and key_tovar = " & Trim(mas(1))
    Set rst = New Recordset
    rst.Open strSQL, mCON, adOpenForwardOnly, adLockReadOnly,
adCmdText
    If rst.EOF = True Then
        strSQL = "INSERT into sklad (key_base, key_tovar,
tovar_number_sklad) values (" & Trim(mas(0)) & _
", " & Trim(mas(1)) & ", " & Trim(mas(2)) & ")"

        mCON.Execute strSQL
        i = i + 1

    End If
    rst.Close
  End If
Loop
Close (hFile)
strMSG = i & " records from " & k & " added into sklad"
MsgBox strMSG
Print #hFileOut, strMSG & " -- " & Now()

Print #hFileOut, vbLf & " *** WORKING FINISH at" & " -- " & Now
& " *** "
Close (hFileOut)
mCON.Close
Exit Sub
errh:
 MsgBox "SOMTHING NOT GOOD"
End Sub
```

Следующим шагом создаем процедуру, работающую с основной формой в немодальном режиме

```
Sub FORM_PREPARE()

Dim mCON As New Connection
Dim strSQL As String
Dim rst As Recordset

Set rst = New Recordset
mCON.Open ("db_practic2")

strSQL = "select name_base from base"
rst.Open strSQL, mCON, adOpenForwardOnly, adLockReadOnly,
adCmdText
UF_SKLAD.ComboBox1.Clear
Do Until rst.EOF
    UF_SKLAD.ComboBox1.AddItem (rst!name_base)
    rst.MoveNext
Loop
UF_SKLAD.ComboBox1.ListIndex = 0
rst.Close

strSQL = "select num_nakl, date_nakl from nakl"
rst.Open strSQL, mCON, adOpenForwardOnly, adLockReadOnly,
adCmdText
UF_SKLAD.ComboBox2.Clear
Do Until rst.EOF
  UF_SKLAD.ComboBox2.AddItem (rst!num_nakl & "   от " &
Str(rst!Date_nakl))
  rst.MoveNext
Loop

UF_SKLAD.ComboBox2.ListIndex = 0
radio_but = "win"
radio_but_obj = "sklad"
UF_SKLAD.OptionButton1 = True
UF_SKLAD.OptionButton6 = True

UF_SKLAD.Show (0)
mCON.Close
End Sub
```

Далее приведен код процедуры отвечающей за вывод информации о товарах на складах и информацию о накладных. Даная процедура выступает в роли своеобразного диспетчера, позволяющего выбрать способ представления информации

```
Sub show_sklad()
Dim mCON As New Connection
Dim strSQL As String
Dim rst As Recordset
Dim ss As String
Dim nn As Integer
Dim mas() As String
Dim ns As nakl_struct

Set rst = New Recordset
mCON.Open ("db_practic2")

Select Case radio_but_obj
 Case "sklad":
```

'Готовим запрос по складам

```
   strSQL = "select b.name_base, t.name_tovar,
s.tovar_number_sklad as tn, t.unit_tovar, " & _
" t.price_tovar as tp, tn*tp  from base as b " & _
"inner join (sklad as s inner join (tovar as t) on
t.key_tovar=s.key_tovar ) on b.key_base=s.key_base where
b.name_base like '" & UF_SKLAD.ComboBox1.Value & "'"

 Case "nakl"
    ss = UF_SKLAD.ComboBox2.Value
    mas = Split(ss, " ")
    nn = Val(mas(0))
```

'Получаем информацию о накладных из представления *nakl_q*

```
'select t.name_tovar, t.price_tovar as pt, d.tovar_number_doc as
'nt, pt*nt as sum_tovar from nakl as n inner join (doc as d inner
'join tovar as t on t.key_tovar=d.key_tovar) on n.key_nakl =
'd.key_nakl where n.key_nakl = " & ns.key

    strSQL = "select * from nakl_q where num_nakl = " & nn
    rst.Open strSQL, mCON, adOpenForwardOnly, adLockReadOnly,
adCmdText
    If rst.EOF <> True Then

     ns.key = rst!key_nakl
     ns.num = rst!num_nakl
     ns.base = rst!name_base
     ns.klad = rst!klad_base
     ns.date_outx = rst!date_out_nakl
     ns.datex = rst!Date_nakl
     ns.shop = rst!name_shop
     ns.buh = rst!buh_shop
    End If
    rst.Close
 End Select
```

```
rst.Open strSQL, mCON, adOpenForwardOnly, adLockReadOnly,
adCmdText
UF_SKLAD.ListBox1.Clear

Call clr_wsheet
Select Case radio_but

 Case "win":
```

'Вывод информации в окно ListBox

```
    Select Case radio_but_obj
    Case "sklad":
```

'Вывод информации по складу

```
      Do Until rst.EOF
        ss = rst.Fields(1) & ": " & rst.Fields(2) & " " &
rst.Fields(3) & "  по  " & rst.Fields(4) & " руб.  всего на  " &
rst.Fields(5) & " руб."
        UF_SKLAD.ListBox1.AddItem (ss)
        rst.MoveNext
      Loop
     Case "nakl"
```

'Вывод информации по накладной

```
        ss = "Накладная №  " & ns.num & "   от " & ns.datex
        UF_SKLAD.ListBox1.AddItem (ss)
        'ss = ""
        'UF_SKLAD.ListBox1.AddItem (ss)
        ss = "              Отправитель: " & ns.base
        UF_SKLAD.ListBox1.AddItem (ss)
        ss = "              Получатель:  " & ns.shop
        UF_SKLAD.ListBox1.AddItem (ss)
        ss = ""
        UF_SKLAD.ListBox1.AddItem (ss)
        'UF_SKLAD.ListBox1.AddItem (ss)
        ss = "№п.п  Наименование  цена   количество  сумма"
        UF_SKLAD.ListBox1.AddItem (ss)
        ss = "-------------------------------------------------"
        UF_SKLAD.ListBox1.AddItem (ss)
        i = 1
        Do Until rst.EOF
         ss = rst.Fields(0) & mstr(20 - Len(rst.Fields(0))) &
rst.Fields(1) & " руб." & mstr(10 - Len(rst.Fields(1))) &
rst.Fields(2) & mstr(10 - Len(rst.Fields(2))) & rst.Fields(3) &
" руб."
         UF_SKLAD.ListBox1.AddItem (ss)
         i = i + 1
         rst.MoveNext
        Loop
        ss = "-------------------------------------------------"
```

```
        UF_SKLAD.ListBox1.AddItem (ss)
        ss = "Отгрузил: " & ns.klad & "      Принял: " & ns.buh
        UF_SKLAD.ListBox1.AddItem (ss)
        ss = "Дата отгрузки: " & ns.datex & "     Дата получения:
" & ns.date_outx
        UF_SKLAD.ListBox1.AddItem (ss)
        ss = "Подпись_____________            Подпись ________ "
        UF_SKLAD.ListBox1.AddItem (ss)

    End Select
 Case "wsheet":
```

'Вывод информации в рабочий лист

```
 Cells(1, 1).Value = UF_SKLAD.ComboBox1.Value
 i = 2
  Do Until rst.EOF
   For j = 1 To 5
     Cells(i, j).Value = rst.Fields(j).Value
   Next j
   rst.MoveNext
   i = i + 1
 Loop

 Case "word":
```

'Вывод информации в документ MS Word

```
    Call word_exp("rep1.doc", UF_SKLAD.ComboBox1.Value, rst)

 End Select
 'MsgBox "Information DONE"
mCON.Close

End Sub
```

Далее приведен код вспомогательной функции, создающей строку из пробелов нужной длины. Эта функция полезна при формировани строк для вывода в окно списка польовательской формы

```
Function  mstr(l As Integer) As String

  Dim ss As String
  ss = ""
  If l > 0 Then
    For i = 0 To l
      ss = ss + " "
    Next
  Else
    ss = ""
  End If
  mstr = ss
End Function
```

```
Sub clr_wsheet()
ActiveWorkbook.Worksheets("info_out").Activate
 Range("A:F").Select
 Selection.ClearContents
End Sub
```

Следующая процедура формирует докумен Word для вывода информации о товарах, находящихся на складе

```
Sub word_exp(fn As String, cn As String, rstx As Recordset)

Dim oApp As New Word.Application
Dim oDoc1 As Word.Document
Dim TableWord As Word.Table
Dim pathx As String

pathx = ActiveWorkbook.path + "\"

MsgBox "Приступаем к сохранению отчета по фирме" & vbLf & cn &
vbLf & "в файл " & fn

Set oDoc1 = oApp.Documents.Add
oApp.Visible = True
oDoc1.Activate

oApp.CentimetersToPoints(2)
oDoc1.Application.Selection.PageSetup.LeftMargin =
oApp.CentimetersToPoints(2)
oDoc1.Application.Selection.PageSetup.RightMargin =
oApp.CentimetersToPoints(1)
oDoc1.Application.Selection.PageSetup.TopMargin =
oApp.CentimetersToPoints(2.5)
oDoc1.Application.Selection.PageSetup.BottomMargin =
oApp.CentimetersToPoints(1.5)

With oDoc1.Application.Selection
.ParagraphFormat.Alignment = 3
.Font.Bold = True
.Font.Size = 13
.Font.Name = "Times New Roman"
End With

oDoc1.Application.Selection.TypeText ("Отчет от " & Now())
With oDoc1.Application.Selection
.ParagraphFormat.Space15
.TypeText (vbLf & vbLf)
.EndOf
End With
With oDoc1.Application.Selection
.ParagraphFormat.Alignment = 1
```

```
.Font.Bold = True
.Font.Size = 20
.Font.Name = "Times New Roman"
.InsertAfter "ООО"
.InsertAfter " "
.InsertAfter """"
.InsertAfter cn
.InsertAfter """"
.EndOf
.InsertParagraphAfter
.InsertAfter "Перечень товаров, имеющихся на складе"
.EndOf
End With

With oDoc1.Application.Selection
.ParagraphFormat.Alignment = 1
.Font.Bold = True
.Font.Size = 13
.Font.Name = "Times New Roman"
End With

Set TableWord =
oDoc1.Tables.Add(oDoc1.Application.Selection.Range, 1, 5)

'TableWord.bo
TableWord.Borders(wdBorderTop).Visible = True
TableWord.Borders(wdBorderLeft).Visible = True
TableWord.Borders(wdBorderVertical).Visible = True
TableWord.Borders(wdBorderRight).Visible = True
TableWord.Borders(wdBorderHorizontal).Visible = True
TableWord.Borders(wdBorderBottom).Visible = True
TableWord.Cell(1, 1).Range.Text = "Наименование"
TableWord.Cell(1, 2).Range.Text = "Кол-во"
TableWord.Cell(1, 3).Range.Text = "Единица"
TableWord.Cell(1, 4).Range.Text = "Цена (руб)"
TableWord.Cell(1, 5).Range.Text = "Сумма (руб)"

  i = 0
 Do Until rstx.EOF
   oDoc1.Application.Selection.InsertRowsBelow
   For j = 1 To 5
     If j = 4 Or j = 5 Then
        TableWord.Cell(i + 2, j).Range.Text =
                  Format(rstx.Fields(j).Value, "##,##0.00")
     Else
       TableWord.Cell(i + 2, j).Range.Text =
                   rstx.Fields(j).Value
     End If
   Next j
   rstx.MoveNext
   i = i + 1
 Loop
```

```
oDoc1.Application.Selection.EndOf

TableWord.Columns(1).Width = oApp.CentimetersToPoints(5)
TableWord.Columns(2).Width = oApp.CentimetersToPoints(3)
TableWord.Columns(3).Width = oApp.CentimetersToPoints(3)
TableWord.Columns(4).Width = oApp.CentimetersToPoints(3)
TableWord.Columns(5).Width = oApp.CentimetersToPoints(3)

Dim fName As String
Do
 fName = Application.GetSaveAsFilename()
Loop Until fName <> "False"
oDoc1.SaveAs fName

MsgBox "Формирование отчета завершено!"
oDoc1.Close
oApp.Quit

End Sub
```

В заключение заметим, что первые две процедуры являются «разовыми» нужны только в начале выполнения проекта, поэтому они запускаются по вызову из среды разработки. Сама форма взаимодействия с пользователем запускается из процедуры FORM_PREPARE.

16. Приложение 3. Перечень вопросов экспресс проверки

Что такое база данных?
Что такое СУБД?
Привести примеры СУБД
Что такое предметная область БД?
Что такое модель данных?
Что такое сущность?

Привести примеры сущностей
Что такое атрибут?
Привести примеры атрибутов
Что такое ER-диаграмма?
Привести пример физических моделей данных
В каком году была предложена реляционная модель данных?

Что соответствует сущности в реляционной модели данных?
Что соответствует экземпляру сущности в отношении?
Что соответствует атрибуту в отношении?
Что такое первичный ключ?
Что такое размерность отношения?
Что такое мощность отношения?

Назовите возможные типы связей между отношениями
Что такое внешний ключ?
Что такое Relation?
Что такое Relationship?
Что такое Entity?
Что такое функциональная зависимость в отношении?

Что такое многозначная функциональная зависимость?
Что значит, значение ячейки должно быть атомарным?
Что такое составной ключ?
Что такое частичная функциональная зависимость?
Что такое транзитивная функциональная зависимость?
Что такое нормализация отношения?

Что значит привести отношение к первой нормальной форме?
Что значит привести отношение ко второй нормальной форме?
Что значит привести отношение к третьей нормальной форме?
Что значит привести отношение к четвертой нормальной форме?

Таблица экспресс проверки

1. Фамилия:	*Вопрос 1: Что такое база данных?*
	Вопрос 2: Что такое сущность?
2. Фамилия:	*Вопрос 1: Привести примеры сущностей*
	Вопрос 2: Что такое СУБД?
3. Фамилия:	*Вопрос 1: Что такое предметная область БД?*
	Вопрос 2: Что такое атрибут?
4. Фамилия:	*Вопрос 1: Что такое модель данных?*
	Вопрос 2: Что такое ER-диаграмма?
5. Фамилия:	*Вопрос 1: Привести примеры СУБД*
	Вопрос 2: Что соответствует сущности в реляционной модели данных?
6. Фамилия:	*Вопрос 1: Привести примеры атрибутов*
	Вопрос 2: В каком году была предложена реляционная модель данных?
7. Фамилия:	*Вопрос 1: Что соответствует экземпляру сущности в отношении?*
	Вопрос 2: Привести пример физических моделей данных

8. Фамилия:	*Вопрос 1: Назовите возможные типы связей между отношениями*
	Вопрос 2: Что значит привести отношение к четвертой нормальной форме?
9. Фамилия:	*Вопрос 1: Что соответствует атрибуту в отношении?*
	Вопрос 2: Что такое внешний ключ?
10. Фамилия:	*Вопрос 1: Что такое первичный ключ?*
	Вопрос 2: Что такое мощность отношения?
11. Фамилия:	*Вопрос 1: Привести примеры атрибутов*
	Вопрос 2: Что такое Relation?
12. Фамилия:	*Вопрос 1: Что такое функциональная зависимость в отношении*
	Вопрос 2: Что такое Entity?
13. Фамилия:	*Вопрос 1: Что такое Relationship?*
	Вопрос 2: Что такое атрибут?
14. Фамилия:	*Вопрос 1: Привести примеры сущностей*
	Вопрос 2: Что значит привести отношение ко второй нормальной форме?
15. Фамилия:	*Вопрос 1: В каком году была предложена реляционная модель данных?*
	Вопрос 2: Привести пример физических моделей данных
16. Фамилия:	*Вопрос 1: Что такое многозначная функциональная зависимость?*

	Вопрос 2: Что соответствует сущности в реляционной модели данных?
17. Фамилия:	*Вопрос 1: Что значит, значение ячейки должно быть атомарным?*
	Вопрос 2: Что такое СУБД?
18. Фамилия:	*Вопрос 1: Что такое составной ключ?*
	Вопрос 2: Что такое сущность?
19. Фамилия:	*Вопрос 1: Что такое частичная функциональная зависимость?*
	Вопрос 2: Что соответствует экземпляру сущности в отношении?
20. Фамилия:	*Вопрос 1: Что такое транзитивная функциональная зависимость?*
	Вопрос 2: Что такое нормализация отношения?
21. Фамилия:	*Вопрос 1: Что такое предметная область БД?*
	Вопрос 2: Что значит привести отношение к первой нормальной форме?
22. Фамилия:	*Вопрос 1: Что значит привести отношение к третьей нормальной форме?*
	Вопрос 2: Что такое атрибут?

17. Приложение 4. Контрольные вопросы для самопроверки

1. Базы данных. Модель данных. Предметная область. Сущности и атрибуты. Типы баз данных.

2. Реляционная модель данных. История развития. Основные принципы реляционных баз данных. Поля, записи, сущности, экземпляры сущностей. Нормальные формы

3. Связи между таблицами. Типы связей 1:1, 1:M, M:1, M:M.Способы реализации Основные и внешние ключи. Привести примеры использования.

4. СУБД - Системы управления базами данных. Примеры реализаций СУБД разными фирмами производителями

5. Создание базы данных в MS Access. Создание таблиц, создание представлений. Мастер подстановок и связи между таблицами. Схема данных. Создание форм доступа к таблицам. Создание отчетов.

6. Архитектура баз данных Visual Basic. Интерфейс, ядро, хранилище. Локальные и удаленные базы данных.

7. Создание базы данных при помощи DAO. Объекты Workspace, Database, TableDef, Field, Index, Relation

8. Работа с базами данных на VBA. Подключение библиотек. Объявление необходимых переменных. Подготовка запроса к базе. Операторы выполнения запросов.

9. Работа с базами данных на VBA. Программное получение информации о структуре базы данных.

10. Уплотнение базы данных. Зачем нужна операция уплотнения. Способы выполнения данной операции для MS Access. Восстановление базы данных.

11. SQL – структурированный язык запросов. Запросы на выборку данных. Условия выборки. Объединение таблиц. Структура и описание запроса.

12. SQL – структурированный язык запросов. Запросы на изменение данных. Структура и описание запроса.

13. SQL – структурированный язык запросов. Запросы на добавление данных. Структура и описание запроса.

14. SQL – структурированный язык запросов. Запросы на удаление данных. Структура и описание запроса.

15. SQL – структурированный язык запросов. Запросы создание и удаление таблиц. Структура и описание запроса.

16. SQL – структурированный язык запросов. Запросы создание и удаление таблиц. Структура и описание запроса.

17. Работа с объектом “Recordset”. Основные типы объекта Recordset. Типы по умолчанию. Доступ к полям выборки. Возможность изменения данных

18. Хранимый запрос или представление. Параметрический запрос. Выполнение параметрических запросов из программы. Отличия хранимого запроса от запроса, формируемого в программе.

18. Литература

1. Михеев Р. VBA и программирование в MS Office для пользователей / Михеев Р. – СПб.:БХВ-Петербург, 2006.– 369 с.

2. Visual Basic Руководство по объектам доступа к данным / Корпорация Microsoft . – 1997 г. – 331 с.

3. Водовозов В.М. Управление базами данных Access на VBA. [Электронный ресурс] / В.М. Водовозов. – Санкт-Петербург.– 2003.– 32 с. – Режим доступа: http://edrive.narod.ru/AccessVBA.pdf