Министерство общего и профессионального образования
Российской федерации
Владимирский государственный университет

Кафедра компьютерной графики

# СУБД И РАСПРЕДЕЛЕННАЯ ОБРАБОТКА ИНФОРМАЦИИ

Методические указания к лабораторным работам

Составитель
И.Е. Жигалов

Владимир 1999

СУБД Microsoft Access 7.0 - 32-разрядная система управления реляционной базой данных, работающая в среде Windows 95. Диспетчером данных, выполняющим загрузку и сохранение данных в пользовательских и системных БД, является ядро БД Microsoft Jet 3.0. Практический минимум для работы Access - процессор 486DX2/66 и 8Мб оперативной памяти.

C++Builder - система программирования общего назначения, которая может использоваться для быстрой разработки любых приложений, в том числе СУБД. Визуальное программирование существенно упрощает построение интерфейса пользователя с таблицей базы данных и организации SQL запросов к локальному или удаленному серверу БД.

Совместное использование систем Microsoft Access и Borland C++Builder для создания приложений пользователя, связанных с обработкой данных (в том числе распределенных), позволяет повысить эффективность процесса разработки такого сложного программного продукта, как СУБД.

# I. Использование среды Access для построения баз данных

## ЛАБОРАТОРНАЯ РАБОТА №1.

Проектирование и создание баз данных в среде Access

**Цель работы**. Изучение интерфейса пользователя среды Access и приобретение навыка работы с ней. Ознакомление с подходами к разработке БД в среде Access.

### 1. Введение

Access поддерживает связи между реляционными таблицами БД типа 1:1 и 1:M. Объекты баз данных, с которыми работает Access - таблицы (для хранения данных), запросы (для выборки данных из таблиц), формы (для ввода и просмотра данных на экране), отчеты (для формирования выходного документа), макросы (для объединения операций обработки данных), модули (для реализации нестандартных процедур на языке Visual Basic).

Для автоматизации процесса создания отдельных объектов используются специализированные диалоговые средства - Конструкторы. С помощью конструктора можно быстро создать и модифицировать таблицу, форму, запрос, и другие объекты. Помощь пользователю оказывают также Мастера по анализу таблиц, по созданию форм и отчетов, подстановок, по импорту/экспорту, защите, разделению базы данных, созданию полей, списков, диаграмм.

Access поддерживает работу с многопользовательскими базами данных и репликацию БД, а также работать в локальной сети, поддерживающей концепцию «клиент-сервер».

В процессе создания БД Access сначала выполняется конструирование таблиц, соответствующих информационным объектам модели данных. Далее может создаваться схема данных, в которой фиксируются существующие логические связи между таблицами, соответствующие связям информационных объектов, что обеспечивает целостность БД.

### 2. Построение информационно-логической модели данных

Информационно-логическая модель (ИЛМ) отображает данные предметной области в виде совокупности информационных объектов (ИО) и связей между ними. ИО - это информационное описание некоторой сущности предметной области, имеющее уникальное имя, например, ТОВАР, СОТРУДНИК, КАФЕДРА и т.п. ИО имеет множество реализаций - экземпляров. Экземпляр образуется совокупностью

конкретных значений полей и должен однозначно определяться значением ключа ИО, состоящего из одного или нескольких ключевых полей.

Первый этап построения ИЛМ - выделение ИО предметной области. Затем между выделенными объектами устанавливаются связи, что приводит к построению логической структуры БД.

### 3. Создание новой базы данных

Создание новой БД Access выполняется в соответствии с ее структурой, полученной в результате проектирования. Структура БД определяется составом таблиц и их взаимосвязями, реализуемыми через ключ связи (входящий в состав полей таблиц). При этом БД может создаваться поэтапно и в любой момент ее можно дополнять новыми таблицами.

Access хранит все таблицы БД в одном файле, который необходимо создать командой **Файл/Создать**, обратившись затем к вкладке **Общие** появившегося окна **Создание** и выбрать значок **Новая база данных**. В появившемся окне **Файл новой базы данных** нужно выбрать папку, в которой будет размещен файл, задать имя файла новой базы данных и нажать кнопку **Создать**. Тип файла по умолчанию имеет значение **Базы данных** и расширение .mdb. В результате выполнения команды **Создать** открывается окно **<имя БД>: база данных**, причем **<имя БД>** соответствует заданному в окне **Файл новой базы данных**.

Для создания новой таблицы надо в окне базы данных выбрать закладку **Таблицы,** нажать кнопку **Создать** и в открывшемся окне **Новая таблица** выбрать один из режимов создания таблицы. Строка **Конструктор** в окне **Новая таблица** определяет выбор основного способа создания новой таблицы, при котором создание таблицы начинается с определения ее структуры в режиме конструктора таблиц. Этот режим позволяет пользователю самому указать параметры всех элементов структуры таблицы.

При выборе режима конструктора таблиц появляется окно **Таблица1:таблица,** в котором определяется структура таблицы базы данных. Для определения поля в окне **таблица** задаются **Имя поля, Тип данных, Описание** - краткий комментарий, а также свойства поля в разделе **Свойства поля**. Общие свойства - на закладке **Общие** и **Тип элемента управления** на закладке **Подстановка**.

Каждая таблица в реляционной базе данных должна иметь уникальный первичный ключ, который может быть простым или составным, включающем несколько полей. Для определения ключа выделяются поля, составляющие ключ, и на панели инструментов нажимается кнопка **Ключевое поле** или выполняется команда **Правка/Ключевое поле**. Для ключевого поля автоматически строится индекс. Убедиться в этом можно, просмотрев информацию об индексах таблицы. Окно **Индексы** вызывается щелчком на кнопке просмотра и редактирования индексов **Индексы** или выполнением команды **Вид/Индексы**. В этом окне индекс первичного ключа имеет имя **PrimaryKey**, в столбце **Поле** перечисляются имена полей, составляющие ин-

декс. После определения структуры таблицы ее надо сохранить командой **Файл/Сохранить**. В окне **Сохранение** вводится имя таблицы.

Создание новой таблицы в режиме таблицы выполняется выбором строки **Режим таблицы** в окне **Новая таблица**. Этот режим позволяет создать таблицу, не определяя предварительно ее структуры; здесь сразу открывается пустая таблица, в которую можно ввести данные. При сохранении этой таблицы Access автоматически создаст ее структуру. Имена полей таблицы, принятые по умолчанию могут быть изменены командой **Формат/Переименовать**; вставить новый столбец можно командой **Вставка/Столбец**, удалить столбец - командой **Правка/Удалить столбец**.

Вне зависимости от способа создания таблицы режим конструктора позволяет в любой момент изменить структуру таблицы.

**Мастер таблиц**, выбранный в окне **Новая таблица**, автоматически создает таблицу по одному из шаблонов.

После определения структуры таблицы можно приступить ко второму этапу создания таблицы - вводу данных. Непосредственно данные в таблицу вводятся в **Режиме таблицы**; переход в него из окна базы данных - кнопкой **Открыть**. Переход в этот режим из режима конструктора таблиц - командой **Вид/Режим таблицы**. Просмотреть и изменить права доступа пользователя к объектам базы данных можно командой **Сервис/Защита/Права доступа**.

## 4. Схема данных в Access

Создание схемы данных начинается в окне базы данных с выполнения команды **Сервис/Схема данных**. Открывается окно **Добавление таблицы**, в котором можно выбрать таблицы и запросы, включаемые в схему данных; после чего необходимо нажать кнопку **Закрыть**. Далее можно приступать к определению связей между таблицами.

Устанавливая связи между парой таблиц в схеме данных, надо выделить в главной таблице уникальное ключевое поле, по которому устанавливается связь. Далее при нажатой кнопке мыши продвинуть курсор в соответствующее поле подчиненной таблицы. Тогда откроется окно **Связи**, где задаются параметры типа отношения, обеспечения целостности данных, обновления и удаления связанных полей и записей.

## 5. Модификация и анализ структуры базы данных

Все изменения структуры таблиц выполняются в режиме конструктора таблиц. В заполненной таблице могут быть изменены имена полей, добавлены новые поля, удалены поля, изменены последовательности полей. При изменении типа данных, размера и других свойств выполняется преобразование данных.

При изменении схемы данных выполняется изменение состава ее таблиц - удаление, добавление таблиц и изменение связей. Для этого необходимо закрыть все таблицы и выполнить команду **Сервис/Схема данных**. Удаление таблицы из схемы данных осуществляется переходом в окно **Схема Данных**, где надо удалить связи таблицы и при отмеченной таблице выполнить команду **Правка/удалить**. Связь удаляется командой **Удалить связь**, а изменение ее параметров выполняется командой **Связи/Изменить связь**.

С помощью мастера Access можно выполнить анализ таблицы с избыточностью и разделить ее на взаимосвязанные таблицы, в которых данные не будут дублироваться. **Мастер анализа таблиц** устраняет дублирование данных и автоматически создает схему данных с определением параметров обеспечения целостности.

## 6. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением и принципами использования Access.

2. Разработать информационно-логическую модель базы данных согласно варианту задания: выполнить анализ предметной области, определить необходимые для хранения поля, сформировать словарь данных (обязательно должны присутствовать числовые, текстовые, счетные поля, поле даты), определить функциональные зависимости между полями, определить набор таблиц (не менее трех) и распределить все атрибуты по таблицам, определить первичные ключи и связи между таблицами.

3. Реализовать в среде Access базу данных согласно варианту задания: создать новую базу данных, создать главную таблицу в режиме конструктора, создать подчиненные таблицы в режиме таблиц, создать вторую и последующие подчиненные таблицы в режиме мастера таблиц, ввести данные в таблицы (не менее четырех строк в каждую), создать схему данных (установив связи между таблицами), провести анализ таблиц с помощью мастера.

## 7. Содержание отчета по лабораторной работе

1. Постановка задачи (индивидуальное задание).

2. Последовательность разработки информационно-логической модели БД.

3. Состав таблиц с типами полей.

4. Схема базы данных.

5. Последовательность команд и установок параметров, используемых для разработки базы данных.

6. Результирующие заполненные таблицы.

### 8. Контрольные вопросы

1. Назначение и использование полей, записей, ключа, степени связи, схемы данных.
2. Технические характеристики Access.
3. Этапы проектирования БД.
4. Разработка новой БД в Access.
5. Модификация и анализ БД.

## ЛАБОРАТОРНАЯ РАБОТА №2.

Разработка форм для загрузки, просмотра и корректировки данных в среде Access

**Цель работы**. Изучение принципов разработки форм в среде Access.

### 1. Введение

Один из важнейших элементов работы с БД - формы ввода/вывода, позволяющие загружать записи в таблицы, просматривать их, корректировать данные. При разработке форм, обеспечивающих загрузку взаимосвязанных таблиц БД, следует придерживаться общепринятых требований к последовательности их загрузки в соответствии со схемой данных (с учетом связей между полями).

### 2. Разработка однотабличных форм

Создание форм выполняется в режиме конструктора форм, который позволяет оформлять **Область заголовка**, **Область данных** и **Область примечаний формы**. Области формы наполняются различными графическими объектами, в частности элементами управления (предназначенными для отображения данных - полями и списками) и статическими надписями. Свойства элементов формы (вид, размер, положение и др.) можно просматривать и корректировать командой **Вид/Свойства**. Выделить форму можно командой **Правка/Выделить форму**.

В форму можно включать кнопки управления для работы с записями (добавить, сохранить и др.) и формой (открыть, обновить и др.). При конструировании форм используется **Панель элементов** из меню **Вид**, позволяющая создавать необходимые элементы управления.

Форма на основе одной таблицы может быть построена как самостоятельная для загрузки, просмотра и корректировки таблиц, а также как вспомогательная для включения в составную форму. Для ее создания целесообразно использовать **Мастер форм** или команду **Автоформа**. Эти возможности доступны в диалоговом окне **Новая форма**, которое можно вызвать из закладки **Формы** кнопкой **Создать**. Команды **Автоформа** отличаются от **Мастера форм** стандартным оформлением; они включают в формы все поля таблицы.

## 3. Разработка многотабличных форм

Составная многотабличная форма создается для работы с несколькими взаимосвязанными таблицами. Многотабличная форма может быть составной (состоять из основной части и подключенных включаемых форм) или нет (в одну форму включаются поля их разных таблиц). Обычно она создается с помощью Мастера формы с последующей доработкой в Конструкторе.

Мастер использует различные способы построения многотабличной формы - с явным включением подчиненной формы, с вызовом связанной формы по кнопке, в виде формы на основе запросов и без подчиненных и связанных форм. Для создания формы в окне базы данных надо выбрать закладку **Форма** и нажать клавишу **Создать**. В окне **Новая форма** выбрать режим **Мастера форм** и можно в качестве источника данных основной части формы выбрать из списка таблицу или запрос. В открывшемся диалоговом окне **Создание форм** выбираются таблицы и из них поля, включаемые в форму, стиль оформления, заголовки.

При последующем выборе пункта **Изменение макета формы** включается режим конструктора, позволяющий доработать форму - выполнить перемещение полей, изменение их свойств и заголовков и др. Для добавления полей используется команда **Вид/Список полей**; для защиты данных поля от изменений - закладка **Данные/Блокировка** в окне свойств; для установки ограничений на корректировку записей через форму - свойства формы **Разрешить...**; для добавления подчиненной формы - кнопка панели элементов **Подчиненная форма/Отчет**.

## 4. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением и принципами разработки форм в среде Access.

2. Разработать в среде Access однотабличную форму для главной таблицы базы данных в режиме конструктора. Добавить элементы управления записями и формами (кнопки). Добавить в таблицу несколько записей (не менее двух).

3. Разработать в среде Access однотабличную форму для одной из подчиненных таблиц базы данных согласно варианту задания с помощью мастера форм. Добавить в соответствующую таблицу несколько записей (не менее двух).

4. Разработать в среде Access однотабличную форму для второй подчиненной таблицы с помощью команды автоформа. Добавить в таблицу несколько записей (не менее двух).

5. Разработать в среде Access составную многотабличную форму (основная часть с подключением включающих форм) из полей главной и всех подчиненных таблиц с помощью мастера формы.

6. Разработать в среде Access многотабличную форму с включением полей из разных таблиц для загрузки и просмотра данных БД согласно варианту задания.

### 5. Содержание отчета по лабораторной работе

1. Постановка задачи (индивидуальное задание).

2. Последовательность команд и установок параметров, используемых для разработки заданных форм.

3. Все разработанные формы в режиме ввода данных.

### 6. Контрольные вопросы

1. Назначение форм.
2. Использование форм.

## ЛАБОРАТОРНАЯ РАБОТА №3.

Обработка данных и разработка отчетов в среде Access

**Цель работы**. Изучение принципов обработки данных и разработки отчетов в среде Access.

### 1. Обработка данных в режиме таблицы и формы

Обработка данных в режиме таблицы и формы включает просмотр (с сортировкой и фильтрацией данных), добавление и удаление записей, обновление полей. В режиме таблицы данные вводятся в формате строк и столбцов с возможной настройкой макета таблицы, установкой состава, размера и размещения столбцов и других параметров.

Для поиска записей таблицы по заданному значению поля используется команда **Правка/Найти** и в окне **Поиск в поле: <Имя поля>** устанавливаются необходимые параметры. Сортировка записей выполняется командой **Записи/Сортировка**. Отбор записей с помощью фильтра может быть сделан по выделенному полю или его части (команда **Записи/Фильтр/Фильтр по выделенному**) или с помощью обычного (**Записи/Фильтр/Изменить фильтр**) и расширенного фильтров (**Записи/Фильтр/Расширенный фильтр**).

В режиме таблицы выполняется корректировка данных в таблицах БД. Добавление записи в таблицу начинается с заполнения пустой строки, размещенной в конце таблицы и помеченной звездочкой. Переход к этой записи выполняется по команде **Правка/Перейти/Новая запись**; сохранение новой записи происходит после перехода к другой записи или выполнения команды **Записи/Сохранить** запись. Для удаления записей используется команда **Правка/Удалить запись**.

## 2. Разработка запросов

В Access существует графическое средство формирования запроса по образцу QBE (Query By Example), с помощью которого можно построить сложный запрос на основе одной или нескольких таблиц. Запрос QBE содержит схему данных (включающую используемые таблицы) и бланк запроса. Он позволяет выбрать необходимые данные из одной или нескольких взаимосвязанных таблиц, произвести вычисления и получить результат в виде таблицы. Через запрос можно обновлять данные в таблицах, добавлять и удалять записи. В Access может быть создано несколько видов запроса: - на выборку (выбирает данные из взаимосвязанных таблиц и других запросов); - на создание таблицы (результат сохраняется в новой таблице); - на обновление, добавление, удаление (изменяющие данные в таблицах).

Запрос разрабатывается в режиме **Конструктора запросов**. Для создания запроса в окне базы данных выбирается закладка **Запрос** и кнопкой **Создать** открывается окно **Новый запрос**, где выбирается пункт **Конструктор запросов**. Далее в окне **Добавление таблицы** необходимо выбрать используемые таблицы, после чего появится окно конструктора запросов **<Имя запроса>: запрос на выборку**. В окне конструктора содержится схема данных запроса и бланк запроса по образцу, который необходимо заполнить (занести имена полей, условие отбора, порядок сортировки результата).

Условие отбора - выражение, состоящее из операндов и операторов сравнения; оно может быть построено с помощью **Построителя выражений** (команда **Построить**).

Запросы на обновление, добавление и удаление первоначально создаются как запрос на выборку, а затем в окне конструктора запросов преобразуются командой меню **Запрос/{Обновление|Добавление|Удаление}**.

Простейшие запросы некоторых видов могут быть созданы с помощью мастеров Access, вызываемых в окне диалога **Новый запрос**, а именно: простой запрос на

выборку, запрос для поиска повторяющихся записей, запрос для поиска записей, не имеющих подчиненных и перекрестный запрос из нескольких полей.

## 3. Разработка отчетов

Средства Access по разработке отчетов предназначены для создания макета отчета, по которому данные из таблиц выводятся в виде печатного документа. В процессе конструирования отчета формируется состав и содержание его разделов, размещение в нем данных из полей таблиц, оформляются заголовки, наименования реквизитов, размещаются вычисляемые реквизиты.

Отчет создается с помощью мастера или в режиме конструктора отчета. Обычно используется мастер, а затем созданный отчет дорабатывается в режиме конструктора. При необходимости вывода в отчете данных из многих таблиц в качестве основы для отчета может быть использован многотабличный запрос.

Для создания отчета в режиме конструктора необходимо в окне базы данных выбрать закладку **Отчеты** и после нажатия кнопки **Создать** в появившемся окне **Новый отчет** выбрать пункт **Конструктор**. В окне отчета первоначально отображаются пустые разделы заголовка, колонтитулов, области данных и примечания. Наличие этих разделов, а также из добавление или удаление определяется командами меню **Вид/Колонтитулы** и **Вид/Заголовок/Примечание отчета**.

Мастер отчетов позволяет построить многотабличный отчет для взаимосвязанных таблиц, выбрать из них нужные поля в заданной последовательности и указать, какая таблица будет записеобразующей. Кроме того, мастер может определить группировку и сортировку записей отчета по различным полям, подсчитать итоговые значения. **Мастер отчетов** вызывается из окна **Новый отчет**.

## 4. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением и принципами разработки запросов и отчетов в среде Access.

2. Выполнить в среде Access обработку данных: сортировка, отбор по фильтру, поиск записи, добавление и удаление записей.

3. Разработать в среде Access следующие запросы в режиме конструктора (все - с условиями):

- на выборку из двух взаимосвязанных таблиц;
- на создание таблицы из двух взаимосвязанных таблиц;
- на добавление записи в главную таблицу;
- на обновление данных в одной из подчиненных таблиц;
- на удаление данных из одной из подчиненных таблиц.

4. Разработать в среде Access следующие запросы с помощью мастеров запросов из главной таблицы:

- простой запрос на выборку;

- запрос для поиска повторяющихся записей;
- запрос для поиска записей, не имеющих подчиненных;
- перекрестный запрос из нескольких полей.

5. Разработать в среде Access следующие отчеты в режиме конструктора:
- из главной таблицы;
- из двух таблиц (главной и подчиненной);
- из двух подчиненных таблиц.

6. Разработать в среде Access с помощью мастера отчет из двух взаимосвязанных таблиц (главной и подчиненной) с группировкой и сортировкой записей по одному из ключевых полей и подсчетом итоговых значений по крайней мере по одному числовому полю.

### 5. Содержание отчета по лабораторной работе

1. Постановка задачи (индивидуальное задание).
2. Последовательность команд и установок параметров, используемых для разработки заданных запросов и отчетов.
3. Вид всех разработанных запросов.
4. Вид всех разработанных отчетов.

### 6. Контрольные вопросы

1. Назначение и использование запросов.
2. Назначение и использование отчетов.

## ЛАБОРАТОРНАЯ РАБОТА №4.
Разработка приложений пользователя в среде Access

**Цель работы**. Изучение принципов разработки макросов и приложений пользователя в среде Access.

### 1. Проектирование задач приложения

Приложение пользователя образуется обычно объединением некоторого множества задач предметной области. Наряду с такими объектами, как формы, запросы и отчеты для реализации практических задач используется язык макросов. Макросы

оперируют этими объектами и объединяют разрозненные действия в единую задачу пользователя.

Язык макросов является языком программирования, который позволяет реализовать задачи пользователя, выполняя необходимые действия над объектами Access и их элементами. Макрос - программа, состоящая из последовательности макрокоманд. Макрокоманда - это инструкция, ориентированная на выполнение определенного действия. Например, макрокомандой можно открыть форму, отчет, напечатать отчет, запустить на выполнение запрос, применить фильтр, присвоить значение, создать свое меню для формы или отчета, выполнить любую заданную команду меню. В Access более 40 макрокоманд.

Язык макросов, являющихся надстройкой над Visual Basic, обеспечивает решение большинства задач без использования программирования на этом языке. Имеются средства, обеспечивающие взаимодействие макросов с объектами на основе событий. События наступают при выполнении определенных действий, к которым относятся в том числе действия пользователя (что позволяет управлять выполнением программы извне).

Макросы создаются в диалоговом режиме записью в окне макроса последовательности макрокоманд с заданием параметров. Создание макроса начинается в окне базы данных (закладка **Макрос**, кнопка **Создать**). В появившемся окне макроса в соответствующие столбцы вводятся макрокоманды, их аргументы, условия их выполнения (логические выражения) и комментарии. Макрос сохраняется командой меню **Файл/Сохранить**, а выполняется по команде меню **Запуск**. Для просмотра и редактирования существующего макроса в окне базы данных выбирается его имя и нажимается кнопка **Конструктор**.

Макрокоманды можно создавать переносом объектов из окна базы данных в окно макроса с помощью мыши. Могут быть созданы группы макросов, связанных решением одной задачи; макросы могут вызываться из других макросов.

## 2. Создание приложения пользователя

Для организации эффективной работы пользователя нужно создать целостное приложение данной предметной области, в котором все компоненты приложения должны быть сгруппированы по функциональному назначению, и обеспечить удобный графический интерфейс пользователя. Приложение должно позволить пользователю решать задачи, затрачивая меньше усилий, чем при работе с разрозненными объектами.

При создании приложений особую роль играют формы, так как они являются основным диалоговым средством работы пользователя. Формы построены так, что любое действие пользователя автоматически вызывает реакцию системы, т.е. воспринимаются как событие, в зависимости от которого могут выполняться необходимые действия. Для выполнения этих действий используются макросы.

С помощью **Диспетчера кнопочной формы** можно создать форму стандартного вида. По команде **Сервис/Надстройки/Диспетчер кнопочных форм** вызывается

окно диспетчера, со строкой **Главная кнопочная форма**, в котором формируется список форм разных уровней (для этого используются кнопки окна **Создать** и **Изменить**). После завершения работы с диспетчером кнопочных форм в списке форм базы данных появляется форма с именем **Кнопочная форма**. Главная кнопочная форма будет запускаться при открытии базы данных; параметры запуска можно просмотреть, выполнив команду **Сервис/Параметры запуска**.

В стандартную кнопочную форму можно вставить рисунок, например из Microsoft Paint. Для создания кнопочной формы необходимо в окне базы данных на закладке **Формы** нажать кнопку **Создать** и в окне **Новая форма** выбрать **Конструктор** (при этом не должен указываться источник данных). Процесс построения кнопки ускоряет **Мастер**, который позволяет создавать кнопки более 30 типов и создает процедуру обработки события на Visual Basic. Эта процедура связывается с событием **Нажатие кнопки** и выполняет выбранное действие.

Если на рабочем столе Windows создать ярлык для запуска Access с автоматическим открытием базы данных, то с его помощью пользователь сразу попадает в среду своего приложения.

## 3. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением и принципами разработки макросов в среде Access.

2. Разработать в среде Access макросы (не менее трех), включающие макрокоманды (в том числе использующие условия): открытие таблицы, открытие формы, открытие отчета, запуск запроса, применение фильтра, присвоение значения, создание меню для формы, выполнение команды меню Access, вызов другого макроса, вывод сообщения, выдача сигнала.

3. Разработать главную кнопочную форму стандартного вида с помощью диспетчера кнопочной формы, содержащую кнопки:

- вызов всех разработанных форм;
- просмотр отчетов;
- открытие запросов;
- запуск макросов;
- изменение кнопочной формы;
- выход из приложения.

Добавить в кнопочную форму стандартный рисунок.

4. Разработать в среде Access приложение для базы данных согласно варианту задания. Создать ярлык на рабочем столе Windows для запуска Access с автоматическим открытием базы данных.

## 4. Содержание отчета по лабораторной работе

1. Постановка задачи (индивидуальное задание).

2. Последовательность команд и установок параметров, используемых для разработки макросов и заданного приложения.

3. Тексты разработанных макросов.

4. Вид кнопочной формы.

### 5. Контрольные вопросы

1. Назначение и использование макросов.
2. Назначение и создание в Access приложений пользователей.

## ЛАБОРАТОРНАЯ РАБОТА №5.

Разработка приложений в среде Access с использованием Visual Basic

**Цель работы**. Изучение принципов разработки модулей на языке Visual Basic в среде Access.

### 1. Использование модулей на Visual Basic в Access

Объектно-ориентированный язык Visual Basic (VB), встроенный в Access, позволяет автоматизировать работу пользователя с базой данных и создавать приложения. Основными объектами являются таблицы, формы, запросы, отчеты, модули, макросы. Они строятся на основе более мелких объектов - элементов управления - и объединяются в более крупные объекты - семейства. Все объекты имеют сохраняемый набор свойств, изменяя которые можно управлять объектом. С каждым объектом связывается ряд методов (исполняемых модулей для работы с данными). Программа на VB не требует запуска; ее выполнение инициируется наступлением какого-либо события или действием пользователя.

По сравнению с макросами VB предоставляет больше возможностей по управлению выполнением приложений и контролю над их выполнением. Его целесообразно использовать, например, для создания функций, выполняющих сложные вычисления, организации сложного ввода дополнительных данных во время работы программы, обработки определенным образом каждой записи набора, исполнения условного цикла, обработки ошибок и др. В простых приложениях применение программ VB ограничивается созданием процедур обработки событий и несложных функций.

### 2. Структура программ на Visual Basic в Access

Основа программ на VB - **процедуры**, состоящие из инструкций, выполняющих необходимые действия и вычисления. Программы на VB, входящие в состав приложения, хранятся в базе данных в **модулях**. Модули подразделяются на:

- *стандартные модули* - отдельные объекты БД, представленные в окне БД и хранящие процедуры, доступные из любых других объектов БД;

- *модули форм и отчетов* - создаются автоматически при создании формы или отчета и являются частью их описания; процедуры модулей связываются с событиями этих объектов.

Модули содержат раздел описаний (определения переменных и констант) и набор процедур двух типов:

- Sub - *процедура-подпрограмма*, не возвращающая значений;

- Function - *процедура-функция*, возвращающая значение.

Кроме того, модули форм и отчетов содержат *процедуры обработки событий* - Sub-процедуры, связанные с определенными событиями объекта.

Формат процедуры-подпрограммы:

[**Private**|**Public**][**Static**] **Sub** имя_процедуры [(список_аргументов)]
[инструкции]
[**Exit Sub**] ‘Немедленное завершение процедуры
[инструкции]
**End Sub**

Формат процедуры-функции:

[**Private**|**Public**][**Static**] **Function** имя_процедуры [(список_аргументов)]
[**As** тип_данных]
[инструкции]
[Имя_процедуры=выражение]
[**Exit Function**] ‘Немедленное завершение процедуры
[инструкции]
[Имя_процедуры=выражение]
**End Function**

Процедура обработки события создается пользователем добавлением программы в стандартные шаблоны процедур в модулях форм и отчетов. Для создания процедуры обработки события следует:

- открыть форму или отчет в режиме конструктора;
- выделить элемент управления, для которого создается процедура;
- нажать кнопку Свойства на панели инструментов;
- в окне свойств формы, отчета или элемента управления выбрать закладку События;
- выбрать событие, запускающее процедуру обработки события;
- в строке события выбрать [процедура обработки события];
- для открытия окна модуля формы или отчета нажать кнопку построителя (или правую кнопку мыши);
- ввести инструкции процедуры обработки события между строками Sub и End Sub в открывшемся окне шаблона процедуры;
- нажать на панели инструментов кнопку Компилировать загруженные модули;

- закрыть окно модуля; процедура сохраняется автоматически.

Создавать процедуры Function или процедуры Sub можно в стандартном модуле (выбрать закладку Модули в окне БД и нажать кнопку Создать), в модуле формы или в модуле отчета (открыть форму или отчет в режиме конструктора и нажать на панели инструментов кнопку Программа).

Инструкции описания переменных модуля имеют формат:

**Dim** Имя_переменной [**As** Тип_данных]

Например, инструкция создания текстовой переменной X:

**Dim** X **As** String

Основные типы данных: Boolean, Byte, Integer, Long, Currency, Single, Double, Date, String, Variant, Object. В процедурах могут применяться прямые ссылки на объекты БД и их элементы, а также использоваться объектные переменные, представляющие объекты: а) объектные типы данных Access - Application, Form, Report, Screen, Control; б) объектные типы данных DAO (Data Access Objects - объекты доступа к данным) - Database, TableDef, Field, Recordset, QueryDef, Relation и др.

Например:

**Dim** БД **As** Database, Таблица1 **As** TableDef, Поле1 **As** Field

Инструкция присвоения объектной переменной нового значения:

**Set** имя_объектной_переменной = **[New] объектное_выражение | Nothing**

Например, создание поля в таблице и присвоение его объектной переменной:

**Set** Поле1 = Таблица1.CreateField(″Начисление1″,dbText)

Для описания константы и присвоения ей значения испоьщуется инструкция **Const**:

**Public Const** Con1 **As** Integer = 50

Существуют системные константы True, False, Null. Списки встроенных констант можно просмотреть в окне просмотра объектов по кнопке **Просмотр объектов** панели инструментов при работе в окне модуля.

Запуск программ VB можно выполнить следующими способами:

- включением программы в процедуру обработки события;
- вызовом процедуры-функции в выражении;
- вызовом процедуры в другой процедуре;
- выполнением макрокоманды **ЗапускПрограммы** в макросе.

Перед выполнением программы целесообразно ее откомпилировать, выполнив в открытых формах, отчетах или модулях команду **Запуск/Компилировать загруженные модули**.

### 3. Объекты, используемые в VB

Объекты и семейства (наборы однотипных объектов), управляемые из VB, образуют иерархическую структуру и разделяются на классы.

К классу Microsoft Access относится объект Application, включающий семейства Forms и Reports (включающие объект Control) и объекты Screen и DoCmd. Объект Screen и его свойства используются для ссылки на форму, отчет или элемент управ-

ления активного окна. Объект DoCmd располагает методами, которые позволяют выполнять макрокоманды Access из программ VB.

К классу DAO относится объект DBEngine, содержащий объекты рабочих областей Workspace, включающие по несколько объектов БД Database. Каждый объект Database содержит семейство из одного или нескольких объектов - таблиц, запросов, связей, индексов. Объекты DAO используются кроме того для доступа к таким БД, как FosPro, dBase, Paradox, таблицам Excel, текстовым файлам, серверу Microsoft SQL Server и другим БД, поддерживающим драйвер ODBC.

Объект DoCmd через макрокоманды Access помогает выполнять такие действия, как закрытие окон, открытие форм (метод OpenForm), присвоение значений элементам управления, переход на некоторую запись таблицы, запроса или формы (метод GoToRecord). Ссылка на методы объекта DoCmd имеет вид:

[**Application**.]**DoCmd**.имя_метода[аргументы...],

где аргументы совпадают с аргументами макрокоманд и разделяются запятыми. Например, открытие формы Студенты из текущей БД и отбор в нее отличников:

**DoCmd.OpenForm** ″Студенты″,,,″[Оценка]=’5’″

Сделать текущей новую запись в открытой форме ″Студенты″:

**DoCmd.GoToRecord** acForm,″Студенты″,acNewRec,

где кроме acForm типом объекта может быть acQuery или acTable, а запись задается одной из встроенных констант: acPrevious, acNext, acFirst, acLast, acGoTo (с номером), acNewRec.

## 4. Пример создания процедуры обработки события

Пусть в некоторой форме необходимо создать кнопку для получения справок о наборах данных. Последовательность действий:

- открыть форму в режиме конструктора и создать кнопку, нажав на панели инструментов **Кнопка**;
- изменить подпись кнопки, открыв контекстно-зависимое меню кнопки, выбрав **Свойства** и на закладке **Макет** изменив значение в строке **Подпись**;
- изменить на закладке **Другие** имя кнопки;
- открыть процедуру обработки события *Нажатие кнопки* , выбрав команду **Обработка событий** в контекстно-зависимом меню кнопки;
- ввести в окне модуля текст процедуры между строками Sub и End Sub.

## 5. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением и принципами разработки модулей на Visual Basic в среде Access.

2. Разработать в среде Access стандартные модули на VB, включающие: процедуру-подпрограмму, выполняющую вычисление значений по полям таблиц БД

(суммарные, разностные, средние значения по полям разных таблиц, отклонения текущего значения поля от среднего, минимального, максимального и др.), процедуру-функцию (вычисление среднего значения по группе полей из разных таблиц, минимальные и максимальные значения и др.).

3. Разработать процедуры обработки событий для одной из форм и одного из отчетов, изменяющие реакцию приложения на события открытия отчета и обращения к элементу формы (вывод сообщения MsgBox и сигнала при нажатии на кнопку, организация диалога с пользователем InputBox, открытие следующей формы OpenForm и др.).

4. Использовать разработанные модули в объектах БД:

- включить программы в процедуры обработки событий открытия элемента формы, нажатия кнопки на кнопочной форме, открытия отчета;
- включить процедуру-функцию в выражение, используемое в запросе;
- включить программу в макрос.

### 5. Содержание отчета по лабораторной работе

1. Постановка задачи (индивидуальное задание).
2. Последовательность команд и установок параметров, используемых для разработки модулей.
3. Листинги разработанных программных модулей.

### 6. Контрольные вопросы

1. Назначение и использование модулей на Visual Basic.
2. Основные операторы языка Visual Basic.
3. Порядок разработки модулей на Visual Basic.

## ЛАБОРАТОРНАЯ РАБОТА №6.

Разработка сетевых приложений в среде Access

**Цель работы**. Изучение языка SQL и принципов разработки сетевых приложений пользователя в среде Access.

### 1. Введение

Язык SQL (Structured Query Language) используется для создании запросов и управления реляционными базами данных, такими как Microsoft Access. Когда пользователь создает запрос в режиме конструктора запроса, Access автоматически создает эквивалентную инструкцию SQL. Можно просматривать и изменять инструкции SQL; для этого необходимо нажать на панели инструментов кнопку раскрытия списка рядом с кнопкой "Представление запроса" и выбрать Режим SQL. Изменения, внесенные в запрос в режиме SQL, приведут к соответствующим изменениям в бланке запроса в режиме конструктора. Некоторые типы запросов, такие как запросы к серверу, управляющие запросы и запросы на объединение, не могут быть определены в бланке запроса. Для их создания требуется ввести инструкцию SQL непосредственно в окно запроса в режиме SQL.

Инструкции SQL могут быть использованы в Access в тех ситуациях, когда требуется указать имя таблицы, запроса или поля. Например, при создании формы или отчета с помощью мастера Access автоматически создает инструкцию SQL, которая становится значением свойства "Источник записей". При создании списка или поля со списком с помощью мастера также создается инструкция SQL, которая становится значением свойства "Источник строк" списка или поля со списком. Кроме того, с помощью инструкций SQL определяются подчиненные запросы в бланке запроса, аргументы макрокоманды "Запуск Запроса SQL", а также конструкции в программах.

Для определения статистических данных на основе наборов числовых значений в инструкциях SQL могут быть использованы статистические функции Avg, Count, Min, Max, StDev, StDevP, Sum, Var, VarP.

## 2. Запросы SQL и их применение

Примерами запросов SQL являются:

*Запрос на объединение***.** Объединяет столбцы одной или нескольких таблиц или запросов в один столбец в результирующем наборе записей; затем можно использовать запрос на создание таблицы, основанный на запросе на объединение.

*Запрос к серверу***.** Передает команды непосредственно в базу данных ODBC, например, Microsoft SQL Server; применяется для загрузки или изменения данных.

*Управляющий запрос***.** Создает или изменяет объекты базы данных, такие как таблицы Access или Microsoft SQL Server.

*Подчиненный запрос***.** Состоит из инструкции SQL SELECT, находящейся внутри другого запроса на выборку или изменение. Эти инструкции вводятся в строку "Поле" бланка запроса для определения нового поля или в строку "Условие отбора" для определения условия отбора поля. Подчиненные запросы используются, например, для проверки существования некоторых результатов (с помощью зарезервированных слов EXISTS, NOT EXISTS) или для поиска в главном запросе значений, сравнимых с возвращаемыми в подчиненном запросе (с помощью зарезервированных слов ANY, IN, ALL).

## 3. Выполнение типовых операций с использованием инструкций SQL

1. Просмотр и изменение инструкции SQL для существующего запроса при работе в окне запроса в режиме SQL:

- создать или открыть существующий запрос в режиме конструктора;
- нажать на панели инструментов кнопку раскрытия списка рядом с кнопкой "Представление запроса" и выбрать режим SQL; Access выводит эквивалентную инструкцию SQL для запроса, созданного в режиме конструктора;
- при необходимости, ввести изменения в инструкцию SQL;
- для просмотра соответствующих изменений в режиме конструктора запроса нажать кнопку раскрытия списка "Представление запроса" на панели инструментов и выбрать Конструктор.

2. Сохранение в виде запроса инструкции SQL, определяющей свойство "Источник записей" (затем этот запрос можно выполнять независимо или создавать на его основе новые формы и отчеты):

- если окно свойств еще не открыто, выбрать в режиме конструктора формы или отчета команду Свойства в меню Вид;
- для вывода запроса в режиме конструктора нажать кнопку построителя рядом с ячейкой свойства "Источник записей";
- на панели инструментов нажать кнопку сохранения;
- в окне диалога Сохранение ввести имя и нажать кнопку "OK"; сохраненный запрос добавляется на вкладку "Запросы" в окне базы данных.

3. Создание, удаление или изменение таблиц или индексов с помощью управляющих запросов SQL.

- в окне базы данных выбрать вкладку "Запросы" и нажать кнопку "Создать";
- в окне диалога Новый запрос выбрать в списке "Конструктор";
- не добавляя таблицы или запросы, нажать кнопку "Закрыть" в окне диалога Добавление таблицы;
- в меню Запрос выбрать команду Запрос SQL и подкоманду Управление;
- ввести инструкцию SQL управляющего запроса (CREATE TABLE создает таблицу, ALTER TABLE добавляет новое поле или индекс в существующую таблицу, DROP удаляет таблицу из базы данных или удаляет индекс, определенный для поля или группы полей, CREATE INDEX создает индекс для поля или группы полей);
- для запуска запроса нажать кнопку Запуск на панели инструментов.

4. Определение поля или условий для поля с помощью подчиненного запроса. Подчиненный запрос используется как выражение, помещаемое в ячейку строки "Поле" в бланке запроса, или для определения условий для поля.

- в окне базы данных выбрать вкладку "Запросы" и нажать кнопку "Создать";
- в окне диалога Новый запрос выбрать в списке "Конструктор";
- добавить в запрос таблицы или запросы;
- добавить поля в бланк запроса, в том числе те, для которых определяется подчиненный запрос.

- если подчиненный запрос используется для определения условий для поля, ввести инструкцию SELECT (в круглых скобках) в ячейку строки "Условие отбора" в столбце этого поля, если он используется для определения поля, ввести инструкцию SELECT в ячейку строки "Поле";

- для просмотра результатов нажать кнопку Вид на панели инструментов.

5. Комбинирование в полях данных из двух или нескольких таблиц с помощью запроса на объединение. При запуске запроса на объединение возвращаются записи из соответствующих полей всех включенных в запрос таблиц или запросов.

- в окне базы данных выбрать вкладку "Запросы" и нажать кнопку "Создать";

- в окне диалога Новый запрос выберать в списке "Конструктор";

- не добавляя таблицы или запросы, нажать кнопку "Закрыть" в окне диалога Добавление таблицы;

- в меню Запрос выбрать команду Запрос SQL и подкоманду Объединение;

- ввести инструкции SQL SELECT, которые комбинируются с помощью операции UNION, если не требуется возвращать повторяющиеся записи, или с помощью операции UNION ALL для возвращения всех записей;

- для выполнения сортировки в запросе на объединение добавить вслед за последней инструкцией SELECT предложение ORDER BY.

6. Использование запроса в режиме конструктора для создания инструкции SQL, предназначенной для вставки в другом месте базы данных.

Инструкции SQL и предложения SQL используются в различных выражениях, в аргументах процедур и в значениях свойств. Например, инструкция SQL, указанная в свойстве "Источник строк", определяет создание списка. При создании запросов в режиме конструктора создается эквивалентная инструкция SQL, которую можно просматривать в окне запроса в режиме SQL, копировать и вставлять в другие места в базе данных.

- в режиме конструктора создать запрос - источник инструкции SQL;

- нажать на панели инструментов кнопку раскрытия списка рядом с кнопкой "Представление запроса" и выбрать Режим SQL;

- выделить всю или часть инструкции SQL и нажать клавиши CTRL+C для копирования выделенного текста в буфер обмена;

- перевести курсор в позицию, в которую требуется вставить инструкцию SQL, например, в окне модуля или в ячейке свойства;

- нажмать клавиши CTRL+V.

7. Изменение базовой таблицы, запроса или инструкции SQL формы или отчета.

- в режиме конструктора формы или отчета установть указатель на область выделения формы или область выделения отчета и открыть окно свойств двойным нажатием кнопки мыши;

- выполнить одно из следующих действий: а) если форма или отчет имеют базовый запрос или инструкцию SQL, и требуется изменить инструкцию SQL или структуру запроса, нажать кнопку построителя рядом с ячейкой свойства "Источник записей"; запрос будет открыт в режиме конструктора (в инструкцию SQL изменения вносятся тем же способом, что и в запрос; достаточно перетащить в нее нужные поля и задать в бланке условия отбора и порядок сортировки); б) для замены табли-

цы или запроса, нажать кнопку раскрытия списка в ячейке свойства "Источник записей" и выбрать из списка нужную таблицу или запрос.

## 4. Использование выражений в инструкциях SQL

Пользователь вводит выражения в инструкции SQL при изменении запроса в режиме SQL, а также при определении значений некоторых свойств и аргументов. Например, с помощью инструкции SQL определяется значение свойства "Источник строк" при заполнении списка.

*Ввод выражений в режиме SQL.* Для ввода выражения в режиме SQL нажмите на панели инструментов кнопку раскрытия списка рядом с кнопкой "Представление запроса" и выберите Режим SQL. Будет открыта инструкция SQL, соответствующая активному запросу. Допускается ввод выражений в инструкцию SELECT, а также в предложения WHERE, ORDER BY, GROUP BY или HAVING.

*Ввод выражений SQL в значения свойств и аргументов.* Значения некоторых свойств и аргументов могут быть определены с помощью вводящейся пользователем инструкции SQL. Проще всего скопировать и вставить выражение из окна режима SQL. Например, выражения SQL используются в качестве аргумента макрокоманд "ОткрытьФорму" и "ПрименитьФильтр"; в качестве значения свойства "Источник строк"; в аргументах, определяющих подмножество или условия в статистических функциях по подмножеству, и в качестве значения свойства "Источник записей".

## 5. Особенности подчиненных запросов SQL

Подчиненный запрос - это инструкция SELECT, вложенная в инструкцию SELECT, SELECT...INTO, INSERT...INTO, DELETE или UPDATE или в другой подчиненный запрос. Подчиненный запрос можно использовать вместо выражения в списке полей инструкции SELECT или в предложениях WHERE или HAVING. Инструкция SELECT используется в подчиненном запросе для задания набора конкретных значений, вычисляемых в выражениях предложений WHERE или HAVING. Рекомендации:

- используйте предикаты ANY или SOME, которые являются синонимами, для отбора записей в главном запросе, которые удовлетворяют сравнению со всеми записями, отобранными в подчиненном запросе;
- используйте предикат ALL для отбора в главном запросе только тех записей, которые удовлетворяют сравнению со всеми записями, отобранными в подчиненном запросе;
- используйте предикат IN для отбора в главном запросе только тех записей, которые содержат значения, совпадающие с одним из отобранных подчиненным запросом (и предикат NOT IN для отбора в главном запросе только тех записей, кото-

рые содержат значения, несовпадающие ни с одним из отобранных подчиненным запросом);

- используйте предикат EXISTS (с необязательным зарезервированным словом NOT) в логическом выражении для определения того, должен ли подчиненный запрос возвращать какие-либо записи.

В подчиненном запросе можно использовать псевдонимы имен таблиц для ссылки на таблицы, перечисленные в предложении FROM, расположенном вне подчиненного запроса.

## 6. Формат основных инструкций SQL

1. DELETE. Создает запрос на удаление записей, предназначенный для удаления записей из одной или нескольких таблиц, перечисленных в предложении FROM, которые удовлетворяют предложению WHERE:

DELETE [таблица.*]
FROM таблица
WHERE условие_отбора

2. INSERT INTO. Добавляет запись или записи в таблицу. Эта инструкция образует запрос на добавление:

*Запрос на добавление нескольких записей:*

INSERT INTO назначение [IN внешняя_база_данных] [(поле1[,поле2[, ...]])]
SELECT [источник.]поле1[,поле2[, ...]
FROM выражение

*Запрос на добавление одной записи:*

INSERT INTO назначение [(поле1[,поле2[, ...]])]
VALUES (значение1[, значение2[, ...])

3. SELECT. По этой инструкции ядро базы данных Microsoft Jet представляет данные из базы данных в виде набора записей:

SELECT [предикат] { * | таблица.* | [таблица.]поле1 [AS псевдоним1] [, [таблица.]поле2 [AS псевдоним2] [, ...]]}
FROM выражение [, ...] [IN внешняя_база_данных]
[WHERE... ]
[GROUP BY... ]
[HAVING... ]
[ORDER BY... ]
[WITH OWNERACCESS OPTION]

Возиожные предикаты отбора: ALL, DISTINCT, DISTINCTROW или TOP.

4. SELECT...INTO. Создает запрос на создание таблицы:

SELECT поле1[,поле2[, ...]] INTO новая_таблица[IN внешняя_база_данных]
FROM источник

5. UPDATE.

Создает запрос на обновление записей, который изменяет значения полей указанной таблицы на основе заданного условия отбора.

Синтаксис

UPDATE таблица

SET новое_значение

WHERE условие_отбора;

6. Операция UNION. Создает запрос на объединение, который объединяет результаты выполнения нескольких независимых запросов или таблиц:

[TABLE] запрос1 UNION [ALL] [TABLE] запрос2 [UNION [ALL] [TABLE] запросn [ ... ]]

В следующем примере объединяется существующая таблица с именем "Счета" и результат выполнения инструкции SELECT:

TABLE [Счета] UNION ALL

SELECT *

FROM Клиенты

WHERE [Стоимость заказа] > 1000.

## 7. Примеры управляющих запросов

В следующем управляющем запросе создается таблица "Справочник". Приведенная инструкция определяет имена и типы полей таблицы и создает для поля "Код" индекс, назначающий это поле ключевым.

CREATE TABLE Справочник

[Код] integer,

[Фамилия] text,

[Имя] text,

[ДатаРождения] date,

[Телефон] text,

[Примечания] memo,

CONSTRAINT [Индекс1] PRIMARY KEY ([Код]));

Следующая управляющая инструкция создает составной индекс по полям "Фамилия" и "Имя".

CREATE INDEX НовыйИндекс

ON Справочник ([Фамилия], [Имя]);

## 8. Работа с приложением пользователя в сети

Access позволяет работать с БД в многопользовательском режиме (в сети с общей базой данных). Наиболее распространенная структура работы в сети с БД предполагает выделение одного компьютера в качестве файлового сервера, в то время как все остальные являются рабочими станциями, имеющими доступ к его ресурсам. При коллективном доступе к данным несколько пользователей одновременно смогут просматривать и обновлять БД, решая задачи приложения на своей рабочей станции.

Прежде чем открывать БД, размещенную на файловом сервере, пользователь должен зарегистрироваться в сети и иметь доступ к папке с файлом БД. Работа с приложением начинается с запуска на рабочей станции СУБД Access, которая может быть установлена на сервере (программы СУБД загружаются в оперативную память компьютера рабочей станции).

Для установки режима доступа к базе данных необходимо открыть БД (команда **Файл/Открыть**) и в окне **Открытие файла базы данных** изменить флажок **Монопольно**. Режим монопольного доступа запрещает другим пользователям и программам доступ к этой БД. Режим общего доступа обеспечивает коллективное использование объектов базы данных как в режиме чтения, так и в режиме обновления. Можно открыть БД в режиме только для чтения, если в окне **Открытие файла базы данных** нажать кнопку **Команды и режимы** и выбрать режим **Открыть только для чтения.** Режим доступа, используемый при открытии базы данных по умолчанию, устанавливается в окне **Параметры** (открывается командой меню **Сервис/Параметры**) на закладке **Другие.**

В режиме общего доступа данные будут доступны пользователям для одновременного просмотра, ввода и обновления. Access вносит изменения в БД сразу же после перехода к другой записи или при выполнении команды сохранения. Однако попытки одновременного изменения одних и тех же данных разными пользователями могут привести к конфликту. Для предотвращения подобных ситуаций используются методы блокировки данных, которые запрещают другим пользователям изменять записи или таблицы, но разрешает читать данные.

Параметры блокировки записей устанавливаются пользователем, макросом или программой на Visual Basic в свойствах форм, отчетов и запросов. Это свойство представлено строкой **Блокировка записей** в окне свойств формы на закладке **Данные**, в окне свойств запроса на закладке **Общие**, в окне свойств отчета на закладке **Другие**; то есть оно воздействует на формы, отчеты и запросы. Свойство **Блокировка записей** может принимать одно из трех значений: **Отсутствует** (по умолчанию - записи не блокируются, могут возникать конфликты), **Всех записей** (блокируются все записи в базовой таблице или запросе при открытии формы в режиме формы или в режиме таблицы, во время просмотра или печати отчета и при выполнении запроса - до завершения пользователем соответствующих действий), **Изменяемой записи** (только в формах и запросах - блокируется страница записей, с которой работает пользователь). В режиме формы и таблицы в области выделения каждой заблокированной записи изображается символ блокировки ∅.

Для задания режима блокировки записей для конкретной формы, запроса или отчета необходимо открыть их в режиме конструктора, вывести окно свойств и установить требуемое значение свойства **Блокировка записей**. Параметры блокировки записей, используемые по умолчанию, можно просмотреть и изменить в окне **Параметры** на закладке **Другие** (открывается командой **Сервис/Параметры**).

Чтобы обеспечить использование последней версии данных, следует обновлять отображаемые на экране данные командой **Записи/Обновить**. Если за время до обновления отображения какая-либо из выведенных записей была удалена другим

пользователем, Access выведет в каждом из полей этой записи значение ошибки - **#Удалено**. При обновлении отображения данных Access не изменяет порядок записей, не добавляет и не удаляет их; чтобы увидеть такие изменения, необходимо закрыть и повторно открыть форму или другой объект в режиме таблицы (можно клавишей Shift|F9).

В Access предусматривается автоматическое обновление через определенные промежутки времени отображения записей. Частота обновления определяется настройкой **Периода обновления** (меню **Сервис/Параметры**). Параметры, регулирующие процесс обновления: **Период обновления** (от 1 до 32766 сек), **Период повтора обновления** (от 0 до 100 мс - по истечении которых Access автоматически пытается сохранить измененную запись, заблокированную другим пользователем), **Число повторов обновления** (от 0 до 10 - число попыток сохранения изменений записи, заблокированной другим пользователем).

При одновременной работе нескольких пользователей с одной базой данных каждый из них получает свою копию всех форм, отчетов, модулей и других объектов. Объект в режиме конструктора могут одновременно редактировать несколько пользователей, но при попытке сохранения возможны конфликты. Исключением являются таблицы, которые невозможно открыть в режиме конструктора, если они используются другими пользователями (таблица доступна только для чтения). При внесении существенных изменений в объект рекомендуется исключить доступ к БД других пользователей, установив монопольный режим. Особенно это относится к макросам, так как операция сохранения измененного макроса во время его выполнения другим пользователем может привести к ошибкам.

Возможен вариант отдельного хранения таблиц БД на сервере в одном файле, а других объектов (формы, отчеты, запросы, макросы и модули) - в другом файле, как на сервере, так и у каждого пользователя на рабочей станции. Такое разделение выполняет команда меню **Сервис/Надстройки/Разделение баз данных**. Все необходимые действия выполняет мастер по разделению баз данных в диалоге с пользователем.

## 9. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением и принципами разработки SQL - запросов в среде Access.

2. Разработать в среде Access следующие SQL- запросы для базы данных согласно варианту задания:

- запрос на выборку из нескольких таблиц БД с использованием выражений;
- управляющие запросы на создание, изменение и удаление таблицы БД;
- управляющие запросы на создание, изменение и удаление индекса таблицы;
- подчиненный запрос со всеми основными предикатами, приведенными в п.5;
- запрос на объединение таблиц, таблицы и запроса.

Исправить существующий запрос, созданный в конструкторе запросов, добавлением статистических функций в режиме SQL.

3. Использовать разработанные SQL- запросы в объектах Access:

- в форме;
- в макросе (макрокоманда ЗапускЗапросаSQL);
- в кнопочной форме;
- в модуле на Visual Basic (инструкция DoCmd.RunSQL).

4. Выполнить настройки созданного приложения для работы в сети. Дважды запустить на выполнение Access в Windows с открытием одной и той же базы данных. Установить параметры доступа к базе данных: монопольный/общий доступ, для чтения/записи, блокировка записей, параметры обновления, разделение БД.

### 10. Содержание отчета по лабораторной работе

1. Постановка задачи (индивидуальное задание).
2. Последовательность команд и установок параметров, используемых для разработки SQL - запросов.
3. Тексты и результаты выполнения разработанных SQL - запросов.
4. Последовательность команд и установок параметров, используемых для установки сетевого приложения.

### 11. Контрольные вопросы

1. Назначение и использование SQL - запросов.
2. Порядок работы с базой данных в сети.

## II. Введение в разработку приложений СУБД в среде BCB

### Введение

Borland C++ Builder (BCB)- одна из лучших систем быстрой разработки приложений RAD. Основа системы - **Палитра компонент**, разделенная на несколько функциональных групп. BCB содержит **Библиотеку** из более 100 повторно используемых визуальных компонент - элементов управления Windows, средств поддержки диалога, обслуживания баз данных и др. **Инспектор объектов** устанавливает свойства компонент и определяет коды обработки событий.

Вариант BCB Client/Server Suite включает функции языка запросов SQL для баз данных и средства связи с Internet. BCB поддерживает связь с различными БД трех

групп: 1) dBASE, Paradox; 2) Sybase, Oracle, InterBase, Informix; 3) Excel, Access, FoxPro, Btrieve. Механизм BDE (Borland Database Engine) обслуживает связи с БД, а проводник Database Explorer позволяет изображать связи и объекты БД графически. Система поддерживает работу с актуальными данными (live data), при которой данные из БД подключаются еще на стадии проектирования приложения. BCB предназначена для операционных систем Windows 95 и NT.

Проектирование формы приложения СУБД в среде C++Builder в простейшем случае требует выполнения всего лишь трех или четырех действий:

1. Перенести на форму компоненту набора данных (TTable или TQuery) из вкладки Палитры **Data Access** и установить ее свойства в соответствии со специфическими требованиями выбранной базы данных.

2. Перенести на форму компоненту источника данных TDataSource и в свойстве DataSet указать ссылку на объект набора данных (например, Table1 или Query1).

3. Перенести на форму нужные компоненты отображения и редактирования данных из вкладки **Data Controls** и в их свойстве DataSource задать источник данных (например, DataSource1). Определить отображаемое поле набора данных в свойстве DataField.

4. Если на предыдущем шаге была выбрана компонента сетки TDBGrid, то желательно использовать ее совместно с компонентой навигатора TDBNavigator.

Далее можно скомпилировать и собрать полностью работоспособное приложение, которое будет поддерживать отображение, просмотр и редактирование данных.

## ЛАБОРАТОРНАЯ РАБОТА №5

Особенности разработки приложений СУБД в среде BCB

**Цель работы**. Изучение интерфейса пользователя системы BCB и приобретение навыка работы с ней. Ознакомление с подходами к разработке приложений СУБД в среде BCB.

### 1. Введение

Основными инструментами интегрированной среды BCB являются **Палитра компонент** (содержит библиотеку повторно используемых компонент разработки приложений), **Редактор форм** (предназначен для создания интерфейса программы с пользователем), **Редактор кода** (служит для написания кодов обработки событий), **Инспектор объектов** (позволяет визуально устанавливать свойства объектов), **Хранилище объектов** (содержит типовые формы и модули данных).

Визуальное проектирование в среде BCB заключается в перенесении необходимых приложению компонент из палитры на форму проектирования и их настройке с помощью инспектора объектов и редактора форм.

Быстрая разработка приложений подразумевает поддержку свойств, методов и событий компонент в рамках объектно-ориентированного программирования. Свойства позволяют устанавливать характеристики компонент (названия, подсказки, источники данных и др.), методы выполняют операции над объектом, а события связывают воздействие пользователя на компоненты (нажатие кнопок, ввод данных и др.) с кодами реакции на эти воздействия.

## 2. Палитра компонент

В палитру компонент BCB входят вкладки:

- Standard - типовые интерфейсные элементы (меню, кнопки, списки и др.);
- Win95 - дополнительные интерфейсные элементы Windows 95 (поля ввода, горячие клавиши и др.);
- Additional - специализированные графические интерфейсные элементы;
- Data Access - элементы доступа к базам данных;
- Data Controls - элементы управления базами данных;
- Dialogs - стандартные диалоговые элементы Windows 95;
- System - системные элементы управления (файлы, устройства мультимедиа и др.).

Основные средства связи приложения с базами данных содержатся во вкладках Data Access и Data Controls.

## 3. Пример обработки существующей БД

Пример просмотра демонстрационной базы данных BCDEMOS поясняет работу компонент доступа TDBGrid, TDBText, TDBImage, TDataSource, Ttable, TDBMemo, TSaveDialog. Последовательность действий:

- открыть диалог выбора проектов File/Open Project;
- войти в каталог \...\Cbuilder\Examples\Dbtasks\FishFact;
- открыть проектный файл Fishfact;
- командой главного меню Run/Run скомпилировать и запустить приложение.

## 4. Пример создания приложения обработки БД

Реализация различных способов доступа к элементам баз данных сосредоточена в компонентах вкладки Data Access Палитры компонент. Здесь определяются спецификации конкретной базы данных, таблиц, запросов и ссылок к полям записи. Чтобы создать на основе готового модуля данных приложение «Заказчики» с данными о заказчиках товара некоторой компании, необходимо выполнить действия:

- выполнить команду File/New основного меню;
- на вкладке Data Modules выбрать Customer Data;
- выбрать таблицу Customers модуля данных;
- выбрать нужные поля из списка полей реляционной таблицы, перетащив из мышью на форму;
- перетащить компоненту навигатора базы данных DBNavigator из вкладки Data Controls Палитры и в ее свойстве DataSource указать источник данных CustomerData->CustomerSource; теперь можно просматривать актуальные данные всех записей таблицы уже на данной стадии проектирования приложения, манипулируя кнопками списка полей;
- с помощью Инспектора объектов модифицировать свойства компонент полей (например, Caption);
- скомпилировать и собрать приложение.

## 5. Пример использования мастера форм

Для использования Мастера форм при создании приложения необходимо выполнить действия:

- вызвать Мастер форм командой Database/Form Wizard главного меню;
- на первой странице указать, что создается простая форма (...simple form) на основе компонентных объектов таблицы БД (...using TTable objects);
- на второй странице отметить обрабатываемую таблицу БД (например, файл client.dbf из демонстрационной БД BCDEMOS);
- на третьей странице выбрать необходимые для приложения поля таблицы из списка Available Fields;
- на четвертой странице установить расположение выбранных полей в форме (вертикальное - горизонтальное - сеткой);
- на пятой странице задать положение названий относительно полей (слева - вверху);
- на шестой странице можно потребовать создание для новой формы нового модуля данных;

- воспользовавшись Инспектором объектов, адаптировать компоненты формы (п.4);

- скомпилировать и собрать приложение.

### 6. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением и принципом работы системы BCB, ее составом и режимами работы.

2. Вызвать BCB и ознакомиться с командами основного меню системы.

3. Просмотреть Словарь баз данных, выполнив команду главного меню Database/Explorer; убедиться в наличии созданной ранее в среде Access базы данных согласно варианту задания.

4. Разработать приложение, обрабатывающее базу данных из среды Access.

### 7. Содержание отчета по лабораторной работе

1. Постановка задачи (индивидуальное задание).

2. Последовательность команд и установок параметров, используемых для разработки заданного приложения.

3. Листинги разработанных программных модулей.

4. Результирующая форма.

### 8. Контрольные вопросы

1. Функции и структура пакета BCB.

2. Способы разработки приложений баз данных в среде BCB.

3. Состав среды BCB.

4. Состав палитры компонент BCB.

5. Использование мастера форм при разработке приложений в среде BCB.

## ЛАБОРАТОРНАЯ РАБОТА №6

Использование палитры компонент при визуальной разработке СУБД

**Цель работы**. Изучение палитры компонент системы BCB и приобретение навыка работы с ней. Ознакомление с подходами к визуальной разработке приложений СУБД в среде BCB с использованием палитры компонент.

# 1. Инструменты визуальной разработки в среде BCB

## 1.1. Администратор проекта

Администратор проекта (Project Manager) предназначен для манипуляций с текущим проектным файлом. Окно администратора открывается командой View/Project Manager и содержит список всех файлов проекта. Контекстное меню администратора (вызывается Alt+F10) содержит опции сохранения проекта, добавления проектного шаблона к хранилищу объектов, создания новых модулей и форм, добавления и удаления файлов проекта просмотра модулей и форм, установки опций, редактирования проекта.

## 1.2. Редактор форм

Форма - это окно с управляющими компонентами, которые переносятся программистом на стадии проектирования или создаются динамически в процессе работы программы. Открыть окно Редактора форм можно одним из способов:

- командой File/New Application создать новое приложение;
- командой File/New Form создать новую форму;
- командой File/Open Project открыть существующий проект.

Добавление компоненты к форме выполняется из вкладок Палитры компонент. По умолчанию новые имена формы, модуля и проекта при их сохранении - Form1, Unit1, Project1.

## 1.3 Инспектор объектов

Инспектор объектов связывает визуальные компоненты управления разрабатываемого приложения с программным кодом, который обеспечивает его работу. Инспектор объектов имеет две вкладки: Свойства (Properties) и События (Events). Вкладка свойств дает возможность манипулировать свойствами компонент, помещаемых на форму, а вкладка событий содержит список возможных программных событий и позволяет соединять с ними компоненты с помощью Редактора кода. В верхней части Инспектора расположено поле селектора объектов, которое отображает объектный тип выбранной компоненты или формы.

Контекстное меню Инспектора объектов (Alt+F10) содержит опции его управления и установок.

## 1.4. Хранилище объектов

Хранилище объектов обеспечивает возможность разделения или повторного использования имеющихся в нем объектов (форм, проектов, модулей данных, шаблонов). Добавляя формы, диалоги и модули данных к хранилищу, программист делает их доступными другим проектам.

Разделение объектов выполняется между проектами и внутри проекта. Например, некоторому приложению СУБД может понадобиться несколько форм, отображающих одни и те же данные, но с разными кнопками управления. Тогда к хранилищу добавляется единая форма с общими элементами отображения данных, а затем из этой шаблонной формы производятся формы с разными кнопками.

Существует три способа включения в проект объектов из хранилища: их копирование, наследование и прямое использование. Разработка нового проекта может начинаться не с пустого проекта Project1, а с некоторого шаблона из хранилища объектов, для чего выполняются действия:

- выполняется команда File/New, открывающая диалог New Items;
- выбирается вкладка Projects и указывается нужный шаблон;
- указывается каталог для хранения файлов нового проекта.

Чтобы добавить новый объект к хранилищу, выполняются действия:

- выполняется команда Project/Add to Repository, открывающая окно диалога;
- вводится название объекта Tile, его описание Description, имя автора Author и выбирается имя страницы Page;
- выбирается пиктограмма объекта (кнопка Browse), сохраняется текущий объект (кнопка OK).

Чтобы добавить к хранилищу проект или форму в виде шаблона, выполняется команда Tools/Rpository, а затем на вкладке Project Templates или Form Templates - команда Add, открывающая то же окно диалога.

Проект по умолчанию для нового приложения открывается командой File/New Application, создающей пустой проект с пустой формой.

## 1.5. Редактор кода

Редактор кода программ - средство для просмотра и редактирования текста программного модуля, независимо компилируемого в объектный файл и состоящего из *.h. и *.cpp - файлов. Для редактирования текста модуля его имя указывается после

вызова команды View/Units. BCB автоматически открывает в окне редактора новую вкладку с текстом модуля при создании новых приложения, формы, модуля.

Особенностью BCB является автоматическая генерация строк программы. Когда к форме добавляется компонента, в тексте файла Unit1.h появляется объявление переменной экземпляра класса данной компоненты. Например, перенос на пустую форму компоненты кнопки TButton сгенерирует объявление объекта Button1, а определение события OnClick - объявление метода Button1Click обработчики этого события.

Контекстное меню Редактора кода (Alt+F10) содержит опции для просмотра исходного текста и его отладки.

### 1.6. Дизайнер меню

Дизайнер меню облегчает процесс создания меню для текущей формы, позволяя легко добавлять, вычеркивать и переупорядочивать команды меню непосредственно в окне дизайнера. Для создания меню необходимо:

- поместить значок компоненты MainMenu или PopUpMenu из вкладки стандартных компонент на форму;
- открыть окно дизайнера меню (опция Menu Designer из контекстного меню компоненты);
- ввести имя меню (по умолчанию - MainMenu1).

С помощью дизайнера меню можно добавлять и удалять команды, создавать вложенные подменю, перемещать элементы меню. Редактировать меню можно и в окне инспектора объектов.

## 2. Элементы палитры компонент для обработки БД

Библиотека визуальных компонент VCL BCB включает набор стандартных интерфейсных объектов графического интерфейса пользователя Windows (области редактируемого ввода, простые и комбинированные списки и др.), а также компоненты для управления реляционными базами данных. BCB позволяет пользователю создавать новые визуальные компоненты, отличающиеся от исходных расширенными функциональными возможностями, видом, поведением. Создание компонент базируется на наследовании от существующего компонентного типа, определении новых свойств, методов и событий, регистрации созданной компоненты.

## 2.1. Компоненты доступа к базам данных

Невидимые компоненты вкладки Data Access палитры компонент обеспечивают соединения с базами данных. Доступ к БД поддерживает его основа - механизм BDE (Borland Database Engine). В число компонент вкладки входят:

- TDataSource - интерфейс между компонентами доступа к наборам данных и видимыми компонентами управления, размещенными на форме; для организации связи таблиц master-detail; установка AutoEdit/false запрещает редактирование данных;

- TTable - интерфейс между BDE и TDataSource; обеспечивает доступ на этапе проектирования к актуальным данным;

- Tquery - интерфейс между сервером локальной (или удаленной) базы данных и компонентой TDataSource; BCB передает запросы BDE на языке SQL серверу, который интерпретирует их и возвращает приложению результирующий набор;

- TStoredProc - разрешает приложению клиента выполнять процедуры, хранимые на удаленном сервере БД с передачей результатов клиенту;

- TDatabase - представляет возможность одиночного соединения клиент/сервер; используется для выдачи специальных команд управления БД;

- TSession - представляет средства глобального обслуживания групповых соединений с несколькими БД;

- TBatchMove - разрешает приложению выполнять пакетные операции над группами записей или целыми таблицами (добавление, удаление, копирование групп записей с обновлением или созданием таблицы назначения); Source - имя таблицы источника, Destination - имя таблицы назначения;

- TUpdateSQL - позволяет использовать механизм кэшируемых обновлений для обслуживания результирующих наборов с атрибутом «только для чтения»; ускоряет отклик SQL-сервера за счет уменьшения общего числа сетевых обменов с клиентами (упакованные множественные коммуникации проявляются как одиночные транзакции).

## 2.2. Компоненты управления данными

Компоненты вкладка Data Controls палитры компонент обеспечивает взаимодействие пользователя с источниками данных приложения. В число элементов управления визуализацией и редактированием данных входят:

- TDBGrid - реализует отображение и редактирование записей, представляемых на регулярной сетке;

- TDBNavigator - навигатор БД, используется для перемещений по записям набора данных и выполнения операций по их просмотру и редактированию;

- TDBText - отображает поле текущей записи в наборе данных в виде названия - статического текста, который нельзя редактировать;

- TDBEdit - создает однострочную прямоугольную область для отображения и редактирования короткого поля текущей записи в наборе данных;

- TDBMemo - создает многострочную прямоугольную область с линейкой прокрутки для отображения и редактирования длинного поля текущей записи;

- TDBImage - создает контейнер для представления графического изображения, которое хранится в текущей записи набора данных в виде бинарного объекта (BLOB);

- TDBListBox - создает список, выбранный элемент которого становится новым значением поля текущей записи в наборе данных;

- TDBComboBox - создает комбинацию области редактирования и выпадающего списка текстовых вариантов для выбора;

- TDBCheckBox - создает область выбора состояния, связанного с определенной записью в БД;

- TDBRadioGroup - создает контейнер для группы логически взаимоисключающих кнопок, связанных с полями записей в наборе данных;

- TDBLookupListBox - создает таблицу ссылок для заполнения полей информацией из другого набора данных; таблица ссылок содержит коды разрешенных значений, которые может принимать некоторое поле;

- TDBLookupComboBox - создает комбинацию области редактирования и выпадающей таблицы ссылок для заполнения полей информацией из другого набора.

## 3. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением и принципом работы инструментов визуальной разработки приложений BCB.

2. Ознакомиться с составом и назначением вкладок палитры компонент по обработке баз данных.

3. Вызвать BCB и ознакомиться с инструментами визуальной разработки приложений и элементами палитры компонент для обработки баз данных.

4. Просмотреть Словарь баз данных, выполнив команду главного меню Database/Explorer; убедиться в наличии созданной ранее в среде Access базы данных согласно варианту задания.

5. Разработать приложение, обрабатывающее базу данных из среды Access, с использованием инструментов визуальной разработки приложений и основных компонент вкладок Data Access и Data Controls Палитры.

### 7. Содержание отчета по лабораторной работе

1. Постановка задачи (индивидуальное задание).
2. Последовательность команд и установок параметров, используемых для разработки заданного приложения.
3. Листинги разработанных программных модулей.
4. Результирующая форма.

### 4. Контрольные вопросы

1. Состав и функции инструментов визуальной разработки приложений BCB.
2. Состав и функции элементов Палитры компонент для обработки баз данных.
3. Использование инструментов визуальной разработки приложений BCB для работы с БД

## III. Построение программ обработки локальных БД в среде BCB

### Введение

Разработка Систем Управления Базами Данных (СУБД) раньше всегда была очень трудоемким и медленным процессом вследствие необходимости учитывать массу специфических деталей подсистем обслуживания различных баз данных на низком уровне. C++Builder принимает на себя выполнение этих рутинных операций позволяя сосредоточиться на решении основной задачи.

Все приложения СУБД, создаваемые в среде C++Builder, являются клиентами в архитектуре программного взаимодействия клиент/сервер. *Клиент* выдает запросы к серверу базы данных на получение или передачу информации. *Сервер* обрабатывает запросы от множества клиентов одновременно, координируя доступ к данным и их обновление.

Все приложения СУБД, создаваемые в среде C++Builder, основаны на компонентах пользовательского интерфейса с некоторой базой данных, которые предоставляют легкие в использовании средства разработки специальных приложений.

Процесс разработки заключается при этом в основном в визуальной установке свойств выбранных компонент.

## ЛАБОРАТОРНАЯ РАБОТА №7

Организация доступа к базам данных с использованием механизма BDE

**Цель работы**. Изучение организации доступа к базам данных различных типов в системе BCB с использованием механизма BDE.

### 1. Механизм BDE

Ключевой механизм BDE (Borland Database Engine), обеспечивающий работу визуальных компонент баз данных, действует как интерфейс между приложением и базой данных. Взаимодействие компонентных объектов с BDE никак не специфицирует конкретную базу данных и не зависит от реализации обмена информацией на нижнем уровне иерархии. Именно BDE обращается к драйверам, специфическим для базы данных указанного типа, возвращая приложению запрошенные фактические данные. BDE играет роль, аналогичную контроллеру драйверов ODBC (Open Database Connectivity) производства фирмы Microsoft, изолируя приложения от нижнего уровня взаимодействия с базой данных.

Унифицированная технология BDE применяется во всех продуктах Borland: C++Builder, Borland C++, Delphi, IntraBuilder и JBuilder. Чтобы получить доступ к содержимому базы данных, приложению необходимо знать только идентификатор ее *псевдонима (alias)*.

При добавлении компонент баз данных к форме приложения соединение с BDE происходит автоматически - никакого программирования не требуется. Во время выполнения программы BDE делает необходимые запросы и получает данные, заказанные свойствами каждой используемой компоненты.

На рисунке приведена иерархическая структура взаимодействия приложения с базами данных.

## 2. Конфигурация BDE

При запуске утилиты конфигурации из основного меню BCB, открывается окно, оформленное в виде блокнота с рядом страниц, выбираемых соответствующими закладками в нижней части окна. При первом запуске утилиты устанавливаются значения параметров конфигурации по умолчанию, а изменения будут иметь эффект при следующем запуске приложения баз данных.

Страница Drivers используется для модификации установок, используемых драйверами BDE при создании, сортировке и обслуживании таблиц базы данных. В графе Driver Name перечисляются типы инсталлированных драйверов,. Драйверы STANDARD обеспечивают доступ к базам данных Paradox и dBASE, а прочие драйверы - соединения с серверами SQL и ODBC. В графе Parameters перечислены параметры выбранного типа драйвера вместе с их текущими установками.

Чтобы получить доступ к ODBC (например, к базе данных Microsoft Access), надо сначала создать соответствующий источник данных, и только потом вызвать утилиту конфигурации BDE для подключения к этому источнику. Кнопка New ODBC Driver открывает диалог добавления соединения ODBC к имеющемуся списку. Кнопка Delete ODBC Driver разрешает вычеркнуть выбранный драйвер.

Страница Aliases используется для выполнения операций вычеркивания и модификации псевдонимов баз данных типа STANDARD, SQL или ODBC. Alias Names перечисляет все имеющиеся псевдонимы. Они используются для указания имени нужной базы данных в свойстве DatabaseName компонент таблицы TTable или запроса TQuery. Графа Parameters содержит, в частности, тип сервера и полный путь к каталогу, содержащему нужные таблицы.

Кнопка New Alias открывает диалог добавления нового псевдонима выбранного типа к имеющемуся списку. Кнопка Delete Alias разрешает вычеркнуть выбранный псевдоним.

Страница System используется для модификации установок системных и сетевых параметров, которые BDE использует при запуске приложения. Эта информация хранится в регистрационном файле Windows.

Страница Date используется для установки формата отображения даты, а страница Time - для установки формата отображения времени. Страница Number используется для установки формата отображения числовых величин.

## 3. Актуальные данные

Основная задача быстрой разработки приложений управления базами данных состоит в том, чтобы оперативно обнаруживать проблемы в экранном представлении базы данных и ошибки в запросах к ней. Большинство систем программирования реляционных баз данных требуют написания специализированных процедур для выборки данных из таблиц и заполнения полей в соответствующих диалогах. Чтобы удостовериться, что программа производит ожидаемые результаты, раньше приходилось многократно проходить по циклу редактирование => компиляция => сборка.

Компоненты же визуальной среды C++Builder поддерживают поступление актуальных (“живых”) данных. Когда на форму помещены компоненты доступа и управления и определены их свойства, отвечающие за связь с базой данных, данные будут представлены точно так же, как их увидит пользователь. C++Builder объединяет три этапа разработки в единый процесс. Форма приложения будет выглядеть одинаково на стадии проектирования и во время работы программы.

### 4. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением и принципом работы BDE.

2. Вызвать конфигуратор BDE и ознакомиться его текущими установками.

3. Просмотреть Словарь баз данных, выполнив команду главного меню Database/Explorer; убедиться в наличии созданной ранее в среде Access базы данных согласно варианту задания.

4. Настроить BDE на свою базу данных, используя конфигуратор.

5. Разработать приложение, обрабатывающее базу данных из среды Access, с использованием актуальных данных и битовой картинки.

### 5. Контрольные вопросы

1. Назначение механизма BDE.
2. Использование актуальных данных при разработке приложений БД.
3. Использование визуальных компонент при разработке приложений БД.

## ЛАБОРАТОРНАЯ РАБОТА №8

Разработка приложений БД с использованием таблиц

**Цель работы**. Изучение методов разработки приложений БД в среде BCB с использованием таблиц.

## 1. Использование визуальных компонент

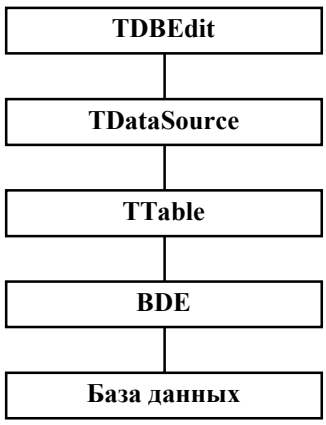
Одним из важнейших достоинств интегрированной среды C++Builder является наличие удобных средств быстрой визуальной разработки приложений СУБД - специализированных компонент баз данных. Только очень опытные программисты способны создать программу подобного уровня, используя исключительно прямые обращения к соответствующим функциям Windows API. При таком подходе даже простое приложение требует написания непомерного по объему кода.

C++Builder предоставляет разработчикам интерфейсные элементы баз данных из Библиотеки Визуальных Компонент на двух вкладках Палитры компонент: компоненты управления данными Data Control обеспечивают отображение и редактирование записей на форме приложения; компоненты доступа к данным Data Access адресуют фактические данные, хранящиеся в файле базы данных, а компонента источника TDataSource служит как интерфейс межкомпонентной связи.

Для работы с базами данных необходимо проанализировать и правильно установить значения ключевых свойств компонент доступа и управления.

C++Builder поддерживает “трехступенчатую” модель разработки приложения баз данных. В этой модели компонента управления связана с компонентой источника, а та, в свою очередь получает фактические данные таблицы или запроса, посредством механизма BDE (рис).

Среднее звено, компонента TDataSource, позволяет менять фактическую таблицу на стадии проектирования формы без перепрограммирования самого приложения - все отображаемые элементы связаны с источником, а не с питающими его таблицей или запросом. Кроме того, компоненты источников берут на себя задачу синхронизации обмена данными между парами таблиц по принципу master-detail.




Отображает область редактируемого ввода текста.

Служит как интерфейс между компонентами управления и доступа

Содержит все записи данной таблицы базы данных.

Изолирует приложение от специфики обслуживания базы данных конкретного типа.

## 2. Источники данных

Невидимая компонента TDataSource действует как интерфейс между некоторым объектом набора данных (таблица, запрос) и визуальной компонентой управления. Все наборы данных должны быть ассоциированы с некоторым источником, так же как и каждая компонента управления. Все компоненты управления имеют свойство DataSource, значение которого фиксирует такую связь.

Основные свойства компоненты TDataSource:

- AutoEdit - разрешает или запрещает режим редактирования записей, вводимых в поля компонент управления.

- DataSet - определяет имя конкретного набора данных (таблицы или запроса), который питает данный источник. Можно переключаться с одного набора данных на другой во время выполнения программы. Следующий простой код реализует попеременное подключение объекта источника DataSource1 к таблице заказчиков “Заказчики” или к таблице ”Заказы”:

```
if  (DataSource1->DataSet == “Заказчики”)
    DataSource1->DataSet = “Заказы”;
else
    DataSource1->DataSet = “Заказчики”;
```

Чтобы синхронизировать работу компонент управления на двух формах, достаточно установить свойство DataSet на один и тот же набор данных:

```
void __fastcall TForm2::FormCreate (TObject *Sender)
{ DataSource1->Dataset = Form1->Table1; }
```

- Name - определяет содержательное название данной компоненты, отличающее ее от других источников данных приложения.

С компонентой TDataSource связаны три события:

*OnDataChange* возникает при перемещении курсора на новую запись и используется при необходимости “ручной” синхронизации поведения компонент управления.

*OnStateChange* возникает при изменении свойства State наборов данных. Например, приведенный ниже обработчик события будет отслеживать изменения состояния таблицы MyTable, выводя на форму соответствующие текстовые отметки.

```
void __fastcall TForm1::StateChange(TObject *Sender)
 {
char S[20];
  switch (MyTable->State)
  {
    case dsInactive:
            strcpy(S,“Таблица неактивна“);
            break;
    case dsBrowse:
            strcpy(S, “Идет просмотр“);
            break;
    case dsEdit:
            strcpy(S, “Идет редактирование“);
```

```
            break;
    case dsInsert:
            strcpy(S, “Идет вставка записи“);
            break;
  }
  // Вывод текстовой строки S
  . . . . .
}
```

*OnUpdateData* возникает перед обновлением текущей записи и используется при необходимости синхронизации поведения обычных компонент с изменениями некоторого набора данных.

### 3. Таблицы

Компонента таблицы TTable устанавливает прямую связь с таблицей базы данных посредством BDE, причем все записи или столбцы этой таблицы становятся доступными для приложения - как раздельно, так и внутри определенного интервала адресов. Основные свойства компоненты:

- Active - разрешает или запрещает (true/false) режим просмотра актуальных данных таблицы на этапе проектирования;
- DatabaseName - содержит псевдоним базы данных или полный путь к ее каталогу; использование псевдонима всегда предпочтительнее, так как это позволяет переназначить физический носитель данных, например, заменив локальный дисковод на сетевой (перекомпиляция приложения не требуется – следует просто изменить путь на вкладке Aliases в утилите конфигурации BDE);
- TableName - позволяет выбрать фактическое имя таблицы из выпадающего списка с именами всех таблиц в адресуемой базе данных;
- Exclusive - разрешает или запрещает другому приложению обращаться к таблице пока работает текущее приложение;
- IndexFiles - открывает диалог выбора индексного файла для таблицы;
- IndexName - задает правило сортировки отображаемых данных, отличное от упорядочивания по первичному ключу;
- Filter - позволяет устанавливать критерий фильтрации, в соответствии с которым адресуется некоторое подмножество записей таблицы;
- ReadOnly - управляет правами доступа в процессе выполнения программы;
- MasterFields и MasterSource - участвуют в образовании связи двух таблиц (ведущей и ведомой) по принципу master-detail.

Методы Locate и Lookup используются для поиска указанных записей как в индексных таблицах, так в таблицах с ключами: эти методы реализуют самый быстрый из возможных способов поиска в данной таблице. Если столбцы для поиска индексированы, и индекс совместим с указанными опциями, используется способ ин-

дексного поиска. В противном случае методы создают для BDE соответствующий фильтр.

Locate производит поиск специфической записи и позиционирует курсор на нее. В простейшем варианте вы передаете методу название столбца для поиска, искомое значение ключа записи и флаг опций. Ниже приведен фрагмент кода, обеспечивающего поиск в столбце “Имя” таблицы MyTable первой записи со значением “Иван”. Если поиск завершился успешно, Locate возвращает значение true и найденная запись становится текущей. Если искомая запись не найдена, Locate возвращает значение false и позиция курсора не меняется.

```
{
bool Success;
TLocateOptions Options;
Options << loPartialKey;
  Success = MyTable->Locate(“Имя”, “Иван”, Options);
}
```

Возможности метода проявляются при поиске вариантных значений записи в нескольких столбцах таблицы. Обобщенный синтаксис описания метода имеет следующий вид:

```
bool __fastcall Locate(const AnsiString KeyFields,
                       const Variant &KeyValues,
                       TLocateOptions Options);
```

Названия столбцов для поиска разделяются в текстовой строке параметра KeyFields символом точка с запятой.

Lookup выбирает значения столбца той записи, которая удовлетворяет заданным значениям поиска. Позиция курсора не меняется. В простейшем варианте методу следует передать название столбца для поиска, искомое значение ключа записи и возвращаемые поля этой записи. Ниже приведен фрагмент кода, обеспечивающего поиск в таблице MyTable первой записи, у которой в столбце “Фирма” имеется значение “Borland”. Если поиск завершился успешно, Lookup возвращает в массиве типа Variant название фирмы, фамилию ее представителя и номер телефона. Если искомая запись не найдена, Lookup возвращает значение Null.

```
{
Variant Results;
  Results = MyTable->Lookup(“Фирма”, “Borland”,
                            “Фирма;Представитель;Телефон”);
}
```

Возможности метода проявляются при поиске вариантных значений полей записи в нескольких столбцах таблицы. Обобщенный синтаксис описания метода имеет следующий вид:

```
Variant __fastcall Lookup(const AnsiString KeyFields,
                          const Variant &KeyValues,
                          const AnsiString ResultFields);
```

Названия столбцов для поиска разделяются в текстовой строке параметра KeyFields символом точка с запятой.

С компонентой TTable также связаны следующие методы:

- GotoCurrent синхронизирует перемещения курсора по нескольким табличным компонентам, ассоциированным с одной и той же фактической таблицей базы данных;

- First, Next, Prior, Last и MoveBy используются для навигации по данным таблицы;

- SetKey, FindKey, FindNearest, GotoKey и GotoNearest используются для поиска по специфическим значениям ключей;

- Append, Insert, AppendRecord и InsertRecord добавляют новую запись к таблице, Delete вычеркивает текущую запись, Edit разрешает приложению модифицировать записи, а Post вызывает фактическое изменение содержимого базы данных.

- EditRangeStart, EditRangeEnd, SetRangeStart, SetRangeEnd, ApplyRange и SetRange устанавливают границы интервала адресов записей, возвращаемых приложению при групповом доступе.

## 4. Пример использования таблиц

Следующая процедура иллюстрирует процесс создания простой формы для демонстрационной базы данных BCDEMOS, в которой пользователь может прокручивать записи таблицы заказчиков CUSTOMER с отображением всех заказов таблицы ORDERS, сделанных текущим заказчиком. Необходимые действия:

1. Выполнить команду главного меню File | New Data Module, чтобы открыть контейнер нового модуля данных DataModule2. В этот модуль поместить две пары компонент доступа к базам данных TTable и TDataSource.

2. Установить свойства объекта ведущей таблицы Table1
   DatabaseName = BCDEMOS
   TableName = CUSTOMER.DB
   Name = CustomersTable

3. Установить свойства объекта ведомой таблицы Table2
   DatabaseName = BCDEMOS
   TableName = ORDERS.DB
   Name = OrdersTable

4. Установить свойства объекта источника DataSource1
   DataSet = CustomersTable
   Name = CustomersSource

5. Установить свойства объекта источника DataSource2
   DataSet = OrdersTable
   Name = OrdersSource

6. Поместить на форму пару компонент управления сеткой TDBGrid.

7. Выполнить команду File | Include Unit Hdr, чтобы указать, что данная форма должна использовать созданный модуль данных.

8. Установить свойство объекта первой сетки DBGrid1
   DataSource = DataModule2->CustomersSource

и свойство объекта второй сетки DBGrid2

DataSource = DataModule2->OrdersSource

9. Активизировать таблицу OrdersTable модуля данных и установить свойство MasterSource = CustomersSource, чтобы связать ведущую таблицу CUSTOMER с ведомой таблицей ORDERS.

10. Дважды щелкнуть мышью в графе значений свойства MasterFields, и в открывшемся окне дизайнера связи полей:

- выберать номер заказчика CustNo (связующее поле таблиц) из выпадающего списка Available Indexes;
- задать CustNo в списках Detail Fields и Master Fields;
- нажать кнопку Add, чтобы добавить в список Joined Fields соединение CustNo -> CustNo, затем кнопку OK.

11. Установить свойство Active = true для таблиц CustomersTable и OrdersTable с тем, чтобы сразу же отобразить актуальные данные в сетках на форме.

12. Скомпилировать и запустить приложение. При прокрутке записей таблицы заказчиков, будут видны только те записи в таблице заказов, которые относятся к текущему заказчику.

## 5. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с компонентами доступа к данным TDataSource и таблицы TTable - их назначением и свойстами.

2. Выполнить пример п.4.

3. Просмотреть Словарь баз данных, выполнив команду главного меню Database/Explorer; убедиться в наличии созданной ранее в среде Access базы данных согласно варианту задания.

4. Настроить BDE на свою базу данных, используя конфигуратор.

5. Разработать приложение, обрабатывающее базу данных из среды Access, с использованием таблиц.

## 6. Контрольные вопросы

1. Назначение компонент источника данных и таблицы.

2. Использование связанных таблиц master/detal.

# IV. Построение программ обработки БД в среде BCB для архитектуры клиент-сервер

## Введение

Инструменты визуальной разработки приложений BCB обеспечивают создание в этой среде программ обработки баз данных для архитектуры клиент-сервер. В частности компонента TQuery обеспечивает интерфейс с сервером локальной или удаленной БД, обеспечивая доступ на этапе проектирования к актуальным данным из одной или нескольких таблиц. Благодаря командам на языке структурированных запросов SQL приложение получает групповой доступ к таблице. Компонента TStoredProc разрешает приложению клиента выполнять процедуры, хранимые на удаленном сервере базы данных с передачей результатов клиенту.

## ЛАБОРАТОРНАЯ РАБОТА №9

Использование запросов SQL при разработке приложений баз данных.

Компонента запросов TQuery.

**Цель работы**. Изучение методов разработки приложений БД в среде BCB с использованием запросов SQL.

### 1. Компонента запросов TQuery

Компоненты таблиц являются компонентами доступа, достаточными для многих приложений СУБД. TTable возвращает все строки и столбцы единственной таблицы, если доступ не ограничивается установкой интервалов и фильтров. Компоненты запросов предоставляют разработчикам альтернативные возможности. TQuery обеспечивает доступ к нескольким таблицам одновременно и способна адресовать некоторое подмножество записей. Вид *возвращаемого набора данных (result set)* зависит от формы запроса, который может быть либо статическим, когда все параметры запроса задаются на стадии проектирования, или динамическим, когда параметры определяются во время выполнения программы.

Указанные действия записываются и реализуются на стандартизованном языке структурированных запросов SQL (Structured Query Language), принятом большинством удаленных серверов реляционных баз данных, таких как Sybase, Oracle, InterBase и SQL Server. На SQL можно сформулировать весьма сложные запросы к базам данных. C++Builder передает запросы серверу, который интерпретирует их и возвращает результаты приложению.

Основные свойства компоненты запроса в окне Инспектора объектов:

- Active - разрешает или запрещает режим просмотра “живых данных”, возвращаемых запросом на этапе проектирования. По умолчанию устанавливается false.

- DatabaseName - содержит псевдоним базы данных или полный путь к ее каталогу, необходимые для разрешения запроса.

- RequestLive - разрешает или запрещает BDE сделать попытку возвратить “живой” результирующий набор. Значение false (устанавливается по умолчанию) означает, что результаты запроса нельзя модифицировать. Значение true гарантирует возврат редактируемого результирующего набор только при условии, что синтаксис команды SELECT согласуется с требованиями запрашиваемых данных.

- SQL - используется для ввода команды SQL посредством строчного редактора списка, который открывается двойным щелчком мышью в графе значений этого свойства. Локальные и удаленные серверы баз данных обеспечивают выполнение четырех базовых команд, которые поддерживаются всеми версиями стандарта SQL: SELECT - для выбора существующих данных из таблиц; INSERT - для добавления новых данных в таблицы; UPDATE - для модификации данных таблиц; DELETE -для удаления данных из таблиц. Результаты обработки запроса возвращаются приложению клиента.

Аналогично таблице, компонента запроса также инкапсулирует следующие методы:

First, Next, Prior, Last и MoveBy используются для навигации по результатам динамического запроса.

Append, Insert, AppendRecord и InsertRecord добавляют новую запись к таблице, Delete вычеркивает текущую запись, Edit разрешает приложению модифицировать записи, а Post вызывает фактическое изменение содержимого базы данных.

## 2. Пример использования запроса

Следующая процедура иллюстрирует процесс создания формы со статическим запросом к таблице EMPLOYEE всей информации о служащих, зарплата которых превышает заданную величину. Действия:

1. Поместить компоненту TQuery на форму.

2. Установить псевдоним адресуемой базы данных сервера в свойстве DatabaseName. В примере используется псевдоним BCDEMOS локальной демонстрационный базы данных, содержащей, в частности, таблицу служащих некоторого предприятия.

3. Открыть строчный редактор списка, ввести команду SQL «SELECT * FROM EMPLOYEE WHERE Salary>40000» и нажать кнопку OK.

4. Поместить на форму компоненту TDataSource и установить ее свойство DataSet = Query1.

5. Поместить на форму компоненту управления сеткой TDBGrid и установить ее свойство DataSource = DataSource1.

6. Установите свойство Active = true для запроса Query1 с тем, чтобы сразу же отобразить актуальные данные в сетке.

### 3. Пример ввода и использования команды SQL

Свойство SQL имеет объектный тип TStrings и заключает список текстовых строк наподобие массива. Ниже приведен код обработчика события Button1Click, реализующий ввод пользователем запроса при нажатии кнопки на форме. Введенная команда SQL записывается в строчный массив (того же типа TStrings) свойства Memo1->Lines компоненты редактирования TMemo. Результаты запроса можно, как и в предыдущем примере, отобразить на сетке.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  // Проверить, введена ли какая-то строка в Memo1
  if (strcmp(Memo1->Lines->Strings[0].c_str(), "") == 0)
  {
   MessageBox(0, "No SQL Statement Entered", "Error", MB_OK);
   return;
  }
  else
  {
   // Деактивировать предыдущий запрос, если он имел место
   Query1->Close();
   // Очистить свойство SQL от предыдущего запроса
   Query1->SQL->Clear();
   // Присвоить введенный в Memo1 текст свойству SQL
   Query1->SQL->Add(Memo1->Lines->Strings[0].c_str());
   try
   {
    Query1->Open();   // выполнить команду SQL
   }
   catch(EDBEngineError* dbError)   // обработка ошибок BDE
   {
    for (int i=0; i<dbError->ErrorCount; i++)
     MessageBox(0, dbError[i].Message.c_str(), "SQL Error", MB_OK);
   }
  }
}
```

### 4. Спецификация параметров запроса

Свойство Params компоненты запроса TQuery позволяет специфицировать имена, типы и начальные значения параметров запроса. C++Builder дает возможность конструировать команду SQL динамического запроса с параметрами. Чтобы указать нужный параметр динамического запроса, следует использовать символ двоеточия перед именем этого параметра. Например, параметр номера служащего в таблице EMPLOYEE идентифицируется следующей командой SQL:

SELECT * FROM EMPLOYEE WHERE EmpNo = :EmpNo.

Увидеть или поменять атрибуты выбранного параметра можно посредством диалогового редактора (Form1->Query1 Parameters), который открывается двойным щелчком мышью в графе значений этого свойства. Нажатие кнопки OK подготавливает SQL сервер к запросу и вызывает попытку его выполнения на стадии проектирования приложения.

## 5. Пример использования параметра

Свойство Params содержит указатель на объект типа TParams. Поэтому изменить значение параметра во время выполнения программы можно по индексу в массиве Items объекта типа TParams:

Query1->Params->Items[0]->AsInteger = 4;

или по имени параметра, посредством метода ParamByName:

Query1->ParamByName("FirstName")->AsString = "John";

Ниже приведен законченный пример использования метода ParamByName во время исполнения программы. Параметр имени служащего FirstName идентифицируется следующей командой SQL:

SELECT * FROM EMPLOYEE WHERE FirstName = :FirstName

Обработчик события первым делом обращается к методу подготовки запроса Prepare, посылая команду SQL серверу. Сервер выделяет ресурсы и оптимизирует динамический запрос только однажды, до первого исполнения. Все, что необходимо теперь - это подставлять новые значения параметров и выполнять команду запроса с помощью метода Open.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 // Деактивировать предыдущий запрос, если он имел место
 Query1->Close();
 if (!Query1->Prepared)
   Query1->Prepare();   // подготовить запрос
 // Заменить значение параметра на введенное пользователем
 Query1->ParamByName("FirstName")->AsString =
                          Edit1->Text.c_str();
 try
 {
   Query1->Open();   // выполнить команду SQL
```

```
  }
  catch(EDBEngineError* dbError)   // обработка ошибок BDE
  {
    for (int i=0; i<dbError->ErrorCount; i++)
      MessageBox(0, dbError[i].Message.c_str(), "SQL Error", MB_OK);
  }
}
```

Работа примера: пользователь вводит значение параметра FirstName и в результате выполнения запроса получает список всех служащих с указанным именем.

### 6. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением принципами использования компоненты запросов TQuery.
2. Просмотреть Словарь баз данных, выполнив команду главного меню Database/Explorer; убедиться в наличии созданной ранее в среде Access базы данных согласно варианту задания.
3. Разработать приложение, обрабатывающее базу данных из среды Access, с использованием компоненты запросов.

### 7. Контрольные вопросы

1. Назначение компоненты запросов.
2. Использование компоненты запросов.

## ЛАБОРАТОРНАЯ РАБОТА №10

Использование запросов SQL при разработке приложений баз данных.
Свойство DataSource компоненты запросов TQuery.

**Цель работы**. Изучение методов разработки приложений БД в среде BCB с использованием запросов SQL.

### 1. Свойство DataSource компоненты запросов TQuery

DataSource указывает на источник другого набора данных, отличный от источника данного запроса, из которого необходимо выбрать значения текущего поля.

Объект запроса будет сравнивать имя параметра в команде SQL с именами полей дочернего набора данных. Когда имена совпадают, параметр автоматически приобретает значение соответствующего поля. Следующая процедура иллюстрирует процесс создания формы, в которой запрос к таблице заказчиков CUSTOMER соединяется с таблицей заказов ORDERS по номеру заказчика CustNo:

1. Поместить компоненты TQuery и TTable на форму и установить псевдоним BCDEMOS в их свойствах DatabaseName.

2. Поместить на форму две компоненты TDataSource и установить свойство DataSet = Query1 для источника Datasource1 и свойство DataSet = Table1 для источника Datasource2.

3. Идентифицировать параметр номера заказчика CustNo следующей командой SQL: SELECT * FROM CUSTOMER WHERE CustNo =: CustNo.

4. Поместить на форму две компоненты управления сеткой TDBGrid и установить свойство DataSource = DataSource1 для сетки DBGrid1 и DataSource = DataSource2 для сетки DBGrid2.

5. Установить свойства DataSource = Datasource2 и Active = true для запроса Query1 с тем, чтобы сразу же отобразить актуальные данные запроса и таблицы в сетках. Прокручивая нижнюю таблицу, пользователь увидит атрибуты заказчика, сделавшего текущий заказ, в верхней таблице. Такой способ соединения запроса с таблицей через свойство DataSource вообще не требует написания кода.

**2. Соединение запроса с таблицей по событию источника.**

Приведенный ниже листинг иллюстрирует другой способ подсоединения запроса к таблице во время выполнения программы посредством обработки события, возникающего при изменении данных источника таблицы. В этом примере обработчик события *OnDataChange* компоненты источника обеспечивает выборку значения параметра запроса CustNo из соответствующего поля таблицы Table1->Fields[1].

```
void __fastcall TForm1::DataSource2DataChange(Tobject
          *Sender, TField *Field)
{
 // Связь запроса с таблицей ORDERS установлена?
 if (Query1->DataSource != NULL)
   return;   // да, возврат
 // Деактивировать предыдущий запрос, если он имел место
 Query1->Close();
 if (!Query1->Prepared)
   Query1->Prepare();   // подготовить запрос
 // Выбрать значение параметра запроса из поля таблицы
 Query1->ParamByName("CustNo")->AsInteger =
                    Table1->Fields[1]->AsInteger;
 try
```

```
  {
    Query1->Open();   // выполнить команду SQL
  }
  catch(EDBEngineError* dbError)   // обработка ошибок BDE
  {
    for (int i=0; i<dbError->ErrorCount; i++)
      MessageBox(0, dbError[i].Message.c_str(), "SQL Error", MB_OK);
  }
}
```

### 3. Формирование команды SQL с помощью функции sprintf

Для динамического формирования текста командной строки SQL во время исполнения программы удобно использовать стандартную функцию sprintf. Эта функция замещает параметры форматирования (%s, %d, %n и т.д.) передаваемыми значениями, например, в результате подстановки значений параметров форматирования:

```
tblName = "EMPLOYEE";
fldName = "EmpNo";
fldValue = 3;
sprintf(sqls, "SELECT * FROM %s WHERE %s = %d",
        tblName, fldName, fldValue)
```

символьный массив sqls будет содержать следующую команду:

```
"SELECT * FROM EMPLOYEE WHERE EmpNo = 3"
```

Ниже приведен листинг, который иллюстрирует применение функции sprintf для формирования команды SELECT динамического запроса к таблице EMPLOYEE. Методы Clear и Add используются для занесения этой команды в свойство SQL. Поскольку подготовленный запрос использует ресурсы сервера и нет никакой гарантии, что новый запрос будет работать с данными одной и той же таблицы, C++Builder снимает готовность при любом изменении свойства SQL (т.е. устанавливает значение false свойства Prepared). При очередном исполнении запроса готовность автоматически восстанавливается.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
char sqls[250];   // массив для хранения команды SQL
char fmts[50];   // массив для значения зарплаты
  // Присвоить fmts значение, введенное пользователем
  if (!(strcmp(SalaryEdit->Text.c_str(),"") == 0))
    strcpy(fmts, SalaryEdit->Text.c_str());
  else
    strcpy(fmts, "100000");            // максимальная зарплата
  // Деактивировать предыдущий запрос, если он имел место
  Query1->Close();
```

```
  // Очистить свойство SQL от предыдущего запроса
  Query1->SQL->Clear();
  // Построить команду SELECT с помощью функции sprintf
  sprintf(sqls, "SELECT * FROM EMPLOYEE WHERE Salary<%s", fmts);
  // Присвоить сформированную команду SELECT свойству SQL
  Query1->SQL->Add(sqls);
  try
  {
   Query1->Open();   // выполнить команду SELECT
  }
  catch(EDBEngineError* dbError)   // обработка ошибок BDE
  {
   for (int i=0; i<dbError->ErrorCount; i++)
    MessageBox(0, dbError[i].Message.c_str(), "SQL Error", MB_OK);
  }
}
```

В результате выполнения сформированного запроса пользователь получает список всех служащих, зарплата которых (выбранная из столбца Salary таблицы) оказывается меньше введенного значения.

### 4. Использование методов Open и ExecSQL

Методы Open и ExecSQL предназначены для передачи серверу команды SQL для исполнения. В предыдущих примерах все запросы выдавали единственную команду SELECT. Результат запроса по команде SELECT рассматривается как набор данных, точно так же, как при работе с таблицей. Существуют другие команды SQL, например, команда UPDATE, которая обновляет содержимое некоторой записи, но не возвращает какой бы то ни было результат. Для исполнения сервером таких запросов, следует использовать метод ExecSQL вместо метода Open.

Приведенный ниже листинг представляет собой некоторое обобщение всех рассмотренных ранее операций с динамическими запросами и их параметрами. Внимание: Данное приложение предназначено для управляемой модификации данных столбца Salary таблицы служащих EMPLOYEE из демонстрационной базы данных BCDEMOS. Изменения, внесенные командой UPDATE необратимы, поэтому перед запуском собранного приложения позаботьтесь о том, чтобы сохранить копию оригинальной таблицы. По умолчанию C++Builder помещает все таблицы в каталоге инсталляции \…\CBuilder\Examples\Data.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
char sqls[250];   // массив для хранения команды SELECT
```

```
char fmts[50];    // массив для зарплаты и надбавки
int  mins, adds;   // десятичные эквиваленты
 // Присвоить fmts зарплату, введенную пользователем
 strcpy(fmts, SalaryEdit->Text.c_str());
 mins = atoi(fmts);
 // Деактивировать предыдущий запрос и очистить SQL
 Query1->Close();
 Query1->SQL->Clear();
 sprintf(sqls, "SELECT * FROM EMPLOYEE WHERE Salary<%s", fmts);
 // Присвоить сформированную команду SELECT свойству SQL
 Query1->SQL->Add(sqls);
 try
 {
  Query1->Open();   // выполнить команду SELECT
 }
 catch(EDBEngineError* dbError)   // обработка ошибок BDE
 {
  for (int i=0; i<dbError->ErrorCount; i++)
   MessageBox(0, dbError[i].Message.c_str(), "SQL Error", MB_OK);
 }
 // Присвоить fmts надбавку [в %], введенную пользователем
 strcpy(fmts, AddEdit->Text.c_str());
 adds = atoi(fmts);
 // Деактивировать предыдущий запрос и очистить SQL
 Query1->Close();
 Query1->SQL->Clear();
 // Присвоить команду UPDATE свойству SQL
 Query1->SQL->Add("UPDATE EMPLOYEE set Salary =
    (Salary+((Salary*:adds))/100) WHERE (Salary < :mins)");
 Query1->ParamByName("mins")->AsInteger = mins;
 Query1->ParamByName("adds")->AsInteger = adds;
 try
 {
  Query1->ExecSQL();   // выполнить команду UPDATE
 }
 catch(EDBEngineError* dbError)   // обработка ошибок BDE
 {
  for (int i=0; i<dbError->ErrorCount; i++)
   MessageBox(0, dbError[i].Message.c_str(), "SQL Error", MB_OK);
 }
 // Деактивировать предыдущий запрос и очистить SQL
 Query1->Close();
 Query1->SQL->Clear();
 // Восстановить команду SELECT
```

```
 Query1->SQL->Add(sqls);
 try
 {
  Query1->Open();   // выполнить команду SELECT
 }
 catch(EDBEngineError* dbError)   // обработка ошибок BDE
 {
  for (int i=0; i<dbError->ErrorCount; i++)
   MessageBox(0, dbError[i].Message.c_str(), "SQL Error", MB_OK);
 }
}
```

Приложение в примере вызывает повышение заданной параметром mins минимальной зарплаты на величину процентной надбавки, определяемой параметром adds. Приложение представляет пользователю список служащих, зарплата которых осталась меньше установленного минимума.

### 5. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением принципами использования свойства DataSource компоненты запросов TQuery.

2. Просмотреть Словарь баз данных, выполнив команду главного меню Database/Explorer; убедиться в наличии созданной ранее в среде Access базы данных согласно варианту задания.

3. Разработать приложение, обрабатывающее базу данных из среды Access, с использованием свойства DataSource компоненты запросов.

### 6. Контрольные вопросы

1. Назначение компоненты запросов.
2. Использование компоненты запросов.

## ЛАБОРАТОРНАЯ РАБОТА №11

Разработка СУБД с использованием хранимых процедур

**Цель работы**. Изучение методов разработки приложений БД в среде BCB с использованием хранимых процедур.

## 1. Хранимые процедуры

Хранимая процедура представляет собой инкапсуляцию команд и данных (таблиц, индексов, областей значений) в некотором мета-объекте базы данных удаленного сервера. Компонента TStoredProc позволяет выполнить часто повторяющуюся процедуру, хранимую на сервере, и передать результаты приложению клиента. Типичное применение хранимых процедур - операции над большими группами строк в таблице базы данных, агрегатные или математические функции. Перемещая на мощный сервер такие повторяющиеся задачи с интенсивными вычислениями, можно заметно улучшить производительность приложения. Общая загруженность сети при этом снижается, поскольку обработка происходит там же, где находятся сами данные.

Для примера рассмотрим приложение, задачей которого является вычисление единственной величины - среднеквадратичного отклонения (СКО) значений по большой выборке записей. Для реализации этой функции приложение должно получить по сети от сервера все значения, участвующие в вычислении, а затем произвести подсчет СКО. Результат работы приложения в виде единственного числа можно было бы получить гораздо более эффективно с помощью хранимой на сервере процедуры, которая считывает данные “на месте” и передает только конечное значение, которое требовалось приложением.

Свойства компоненты хранимой процедуры в окне Инспектора объектов:

- Active - разрешает или запрещает (false - по умолчанию) режим просмотра актуальных данных, возвращаемых процедурой на этапе проектирования;
- DatabaseName - содержит псевдоним адресуемого сервера базы данных;
- StoredProcName - позволяет выбрать имя нужной процедуры из выпадающего списка имен процедур, хранимых на данном сервере;
- ParamBindMode - задает метод, по которому фактические параметры ставятся в соответствие формальным параметрам в описании хранимой процедуры. Значение pbByName (по умолчанию) определяет соответствие по именам, а значение pbByNumber - по порядку перечисления в процедуре.
- Params - используется для ввода параметров хранимой процедуры (если таковые имеются).

## 2. Редактор параметров хранимой процедуры

Редактор параметров, который открывается двойным щелчком мышью в графе значений свойства Params, обращается к серверу для выборки информации о входных и выходных параметрах. Для некоторых серверов полная информация, необходимая для запуска хранимой процедуры, может оказаться недоступной. В таком случае необходимо самостоятельно определить тип каждого параметра (Input, Output, Result), тип его данных и, возможно, значения входных параметров. Редактор параметров отображает параметры в том порядке, в котором они перечислены в описании данной хранимой процедуры. Нажатие кнопки OK подготавливает сервер

и вызывает запуск хранимой процедуры на стадии проектирования приложения. Только база данных Oracle позволяет добавлять, вычеркивать или удалять все параметры в определении хранимой на сервере процедуры, производя таким образом ее перегрузку. Отвечающие за такие действия кнопки редактора параметров Add, Delete и Clear обычно запрещены, поскольку приложение клиента не может модифицировать хранимые процедуры других серверов.

Аналогично запросу, свойство Params содержит указатель на массив объектного типа TParams. Поэтому изменить значение параметра во время выполнения программы можно по индексу в массиве Items объектов типа TParams:

```
StoredProc1->Params[0]->Items[1]->AsString = Edit1->Text;
```

или по имени параметра, посредством метода ParamByName:

```
StoredProc1->ParamByName("ЧИСЛО_СЛУЖАЩИХ")->AsInteger++;
```

а затем подготовить сервер методом Prepare и выполнить процедуру методом ExecProc:

```
StoredProc1->Prepare();   StoredProc1->ExecProc().
```

Хранимая процедура возвращает результаты через выходные параметры или через результирующий набор. Доступ к выходным параметрам во время выполнения программы (как и модификация значений входных параметров перед запуском хранимой процедуры) осуществляется по индексам в массиве Items объектов типа TParams:

```
Edit1->Text = StoredProc1->Params[0]->Items[0]->AsString;
```

или по имени параметра, посредством метода ParamByName:

```
Edit2->Text =   StoredProc1->ParamByName("ЧИСЛО_СЛУЖАЩИХ")->AsInteger.
```

## 3. Процедура проектирования формы приложения с компонентой TStoredProc

Хранимые процедуры некоторых серверов (например, Sybase), возвращают, подобно запросу, результирующий набор, для отображения которого надо использовать подходящую компоненту управления - чаще всего сетку. Процедура проектирования формы приложения с компонентой TStoredProc аналогична той, которая используется для отображения результатов запроса:

1. Установить псевдоним адресуемой базы данных сервера в свойстве DatabaseName.
2. Поместить на форму компоненту TDataSource и установить ее свойство DataSet = StoredProc1.
3. Поместить на форму компоненту управления сеткой TDBGrid и установить ее свойство DataSource = DataSource1.
4. Поместить компоненту TStoredProc на форму.
5. Указать имя процедуры в свойстве StoredProcName.
6. Установить свойство Active = true для процедуры StoredProc1 с тем, чтобы сразу же отобразить результаты в сетке.

7. Открыть редактор параметров, ввести (если надо) их значения и нажать кнопку OK.

### 4. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением и принципами использования хранимых процедур.

2. Просмотреть Словарь баз данных, выполнив команду главного меню Database/Explorer; убедиться в наличии созданной ранее в среде Access базы данных согласно варианту задания.

3. Разработать приложение, обрабатывающее базу данных из среды Access, с использованием хранимых процедур.

### 5. Контрольные вопросы

1. Назначение хранимых процедур.
2. Использование хранимых процедур.

## ЛАБОРАТОРНАЯ РАБОТА №12

Соединения с базой данных и транзакции

**Цель работы**. Изучение методов разработки приложений БД в среде BCB с использованием транзакций.

### 1. Соединения с базой данных и транзакции

Компонента TDatabase позволяет создавать в приложении локальный BDE псевдоним базы данных, таким образом не требуя его наличия в конфигурационном файле BDE. Этот локальный псевдоним может использоваться другими компонентами доступа. Кроме того, с помощью TDatabase можно разработать оригинальный процесс первого соединения с сервером (login), подавляя некоторые подсказки и автоматически подставляя значения необходимых параметров. Наконец, TDatabase способна поддерживать одиночное соединение с базой данных, концентрируя в себе все необходимые операции для поддержания *транзакций*.

Классическим примером транзакции является перевод денежных средств банковских счетов. Такая транзакция обычно состоит в добавлении определенной суммы перевода к новому счету и вычитании этой суммы из исходящего счета. Если выполнение любой из этих операций терпит неудачу, весь трансферт считается незавершенным. SQL-серверы дают возможность “прокручивать назад” команды при возникновении ошибки, не производя никаких изменений в базе данных. Именно управление транзакциями является функцией компоненты TDatabase. Как правило транзакция содержит несколько команд, поэтому начало транзакции надо отметить методом StartTransaction. Как только транзакция началась, все ее исполняемые команды находятся во временном состоянии, до тех пор, пока один из методов Commit или Rollback не отметят конец транзакции. Вызов Commit фактически модифицирует данные, а вызов Rollback отменяет всякие изменения.

Свойства компоненты соединения с базой данных в окне Инспектора объектов:

- AliasName - содержит псевдоним существующей базы данных, определенный утилитой конфигурации BDE. Указание этого свойства является альтернативой значения DriverName;
- DatabaseName - позволяет создать локальный псевдоним базы данных в дополнение к значениям AliasName или DriverName;
- DriverName - содержит имя драйвера BDE при создании локального псевдонима по значению DatabaseName; указание этого свойства является альтернативой значения AliasName;
- Params - содержит строчный массив параметров одиночного соединения.

## 2. Пример выполнения транзакции

Приведенный ниже листинг реализует транзакцию по изменению адреса фирмы на примере связанных таблиц CUSTOMER и ORDERS. Старый адрес, введенный пользователем в область редактирования EditOld заменяется на новый, введенный в область редактирования EditNew. В этом примере компонентный объект Database1 использовался для одиночного соединения с базой данных, поддерживающего выполнение одиночной транзакции. Этот объект необходимо каким-то образом связать с псевдонимом базы данных - или установкой соответствующих свойств компоненты, или определив параметры соединения (такие как тип драйвера, имя сервера, имя пользователя, пароль) во время выполнения программы. Сначала приведем первый способ соединения на стадии проектирования формы приложения, установив соответствующие значения свойств компоненты.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
char sqls[250];   // массив для хранения команды SQL
  try
  {
   Database1->StartTransaction();
```

```
  Query1->SQL->Clear();
  // Изменить EditOld на EditNew в таблице CUSTOMER
  sprintf(sqls, "UPDATE CUSTOMER set Addr1 =
        \"%s\" WHERE (Addr1 = \"%s\")",
        EditNew->Text.c_str(), EditOld->Text.c_str());
  Query1->SQL->Add(sqls);
  Query1->ExecSQL();
  Query1->SQL->Clear();
  // Изменить EditOld на EditNew в таблице ORDERS
  sprintf(sqls, "UPDATE ORDERS set ShipToAddr1 =
        \"%s\" WHERE (ShipToAddr1 = \"%s\")",
        EditNew->Text.c_str(), EditOld->Text.c_str());
   Query1->SQL->Add(sqls);
   Query1->ExecSQL();
  // Внести все изменения, сделанные до этого момента
  Database1->Commit();
  Table1->Refresh();
  Table2->Refresh();
 }
 catch(EDBEngineError* dbError)   // обработка ошибок BDE
 {
  for (int i=0; i<dbError->ErrorCount; i++)
   MessageBox(0, dbError[i].Message.c_str(), "SQL Error", MB_OK);
  Database1->Rollback();
  return;
 }
 catch (Exception* exception)   // обработка исключений
 {
  MessageBox(0, exception->Message.c_str(), "Error", MB_OK);
  Database1->Rollback();
  return;
 }
}
```

### 3. Пример соединения с сервером без псевдонима

Следующий листинг показывает как соединиться с сервером базы данных во время выполнения программы, не создавая ее псевдонима. Ключевые моменты заключаются в том, чтобы указать DriverName и заполнить массив параметров информацией, необходимой для первого соединения (в данном примере пользователь вводит свое имя и пароль в объекты компонент редактирования Edit1 и Edit2). Здесь определены только те параметры соединения, которые не установлены для данного драйвера в утилите конфигурации BDE. Значение параметра SQLPASSTHRU

MODE определяет один из трех возможных способов взаимодействия табличных методов Add, Append и Insert с компонентами запросов, которые соединены с той же базой данных. Использованное в примере значение NOT SHARED означает, что табличные методы и запросы используют два раздельных соединения с сервером. Сервер рассматривает их как соединения с двумя разными пользователями. До тех пор, пока транзакция не завершится, табличные методы не применяются, хотя результаты выполнения запросов могут менять содержимое базы данных, раздельно от действий активной транзакции. Два других значения SHARED NOAUTOCOMMIT и SHARED AUTOCOMMIT указывают, что табличные методы и запросы разделяют одно общее соединение с сервером. Если нужно включить табличные методы в транзакцию, следует использовать способы SHARED NOAUTOCOMMIT или NOT SHARED.

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
char name[20];   // буфер для имени пользователя
char pass[20];    // буфер для пароля
  try
  {
  // Закрыть базу данных и установить параметры
  Database1->Close();
  Database1->DriverName = "STANDARD";
  Database1->KeepConnection = true;
  Database1->LoginPrompt = false;
  Database1->Params->Add("SERVER NAME=
          ...\\CBuilder\\EXAMPLES\\DATA\\EMPLOYEE.DB");
  Database1->Params->Add("SCHEMA CACHE=8");
  Database1->Params->Add("OPEN MODE=READ/WRITE");
  Database1->Params->Add("SQLPASSTHRU MODE=NOT SHARED");
  sprintf(name, "USER NAME=%s", Edit1->Text.c_str());
        Database1->Params->Add(name);
  sprintf(pass, "PASSWORD=%s", Edit2->Text.c_str());
        Database1->Params->Add(pass);
  // Снова открыть базу данных и указанную таблицу
  Database1->Open();
  Table1->Open();
  }
  catch(EDBEngineError* dbError)     // обработка ошибок BDE
  {
  for (int i=0; i<dbError->ErrorCount; i++)
   MessageBox(0, dbError[i].Message.c_str(), "SQL Error", MB_OK);
  }
}
```

Первое вхождение в локальную демонстрационную базу данных BCDEMOS не требует задания имени пользователя и пароля, однако, используя такую методику можно стандартизовать процесс соединения с обычно защищенными базами данных удаленных серверов.

## 4. Порядок выполнения лабораторной работы

1. Ознакомиться по методическим указаниям и литературе с назначением и принципами использования транзакций.

2. Просмотреть Словарь баз данных, выполнив команду главного меню Database/Explorer; убедиться в наличии созданной ранее в среде Access базы данных согласно варианту задания.

3. Разработать приложение, обрабатывающее базу данных из среды Access, с использованием транзакций.

## 5. Контрольные вопросы

1. Назначение транзакций.
2. Использование транзакций.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бекаревич Ю.Б., Пушкина Н.В. СУБД Access для Windows 95 в примерах. - СПб.: BHV, 1997, 400с.

1. Шамис В.А. Borland C++ Builder. Программирование на C++ без проблем. - М.: Нолидж, 1997, 266с.

# ПРИЛОЖЕНИЕ

Варианты заданий (предметная область и **примерный** перечень полей, входящих в состав всех таблиц базы данных)

1. Аэропорт (тип самолета, рейс, аэропорт назначения, экипаж, время, цена).
2. Склад (наименование товара, код товара, количество, цена, поставщик, дата).
3. Пациент (номер, имя, адрес, дата, диагноз, лекарства, доктор, температура).
4. Хирург (номер, имя, дата операции, вид операции, пациент, возраст).
5. Вуз (название, город, студентов, факультетов, преподавателей, категория).
6. Экзамены (предмет, дата, время, лектор, группа, аудитория, оценки).
7. Группа (номер, факультет, курс, студентов, староста, куратор, кафедра).
8. Поезда (номер, тип, пункт назначения, время, расстояние, цена, мест).
9. Афиша (театр, дата, время, спектакль, режиссер, автор, актеры, цена).
10. Книга (название, код, автор, издательство, год, язык, страницы, формат).
11. Продукция (предприятие, цех, наименование, дата, цена, партия).
12. Расписание ВЦ (номер, дата, начало, окончание, руководитель, мест).
13. Погода (дата, время, температура, облачность, осадки, давление).
14. Служащие (имя, номер, год, должность, дата, оклад, подразделение).
15. Учащиеся (имя, пол, возраст, город, школа, класс, оценки, предметы).
16. Спортсмены (имя, возраст, дата, вид, тренер, место, вид спорта, результат).
17. Поставки (дата, изделие, количество, предприятие, город, транспорт).
18. Дом (номер, улица, квартир, владелец, тип, год, этажей, жильцов).
19. Программы (наименование, фирма, дата, язык, объем, ОС, версия).
20. Объект (название, адрес, заказчик, подрядчик, тип, дата, срок, стоимость).
21. Товар (название, дата, цена, размер, цвет, тип, масса, упаковка).
22. Ведомость (номер, имя, участок, сумма, дата, вид, период, налоги).
23. Оборудование (номер, название, стоимость, аудитория, кафедра, дата).
24. Поликлиника (номер, город, район, участков, врачей, категория).
25. Кафедра (код, название, преподавателей, предметов, специальность, групп).
26. Строительство (объект, адрес, заказчик, рабочие, материалы, работы).
27. Цех (продукция, работники, сырье, поставщики, потребители, площади).
28. Компьютеры (процессор, дисплей, накопители, память, цена, порты).
29. Города (население, площадь, промышленность, инфраструктура, даты).
30. Автомобили (марка, модель, мест, объем двигателя, цена, цвет, габариты).

## Оглавление