

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по выполнению лабораторных работ
по дисциплине «СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ»
для студентов специальности 10.05.01 Компьютерная безопасность
специализация «Информационно-аналитическая и техническая экспертиза
компьютерных систем»

Ставрополь
2026

Содержание

ВВЕДЕНИЕ.....	7
1. Системы основанные на файлах.....	8
Файловые системы, принципы построения.....	8
Недостатки файловых систем.....	8
Задание.....	9
2. Реляционные таблицы.....	10
Реляционная таблица.....	11
Определение домена.....	11
Создание таблиц в среде Microsoft Access.....	12
Задание.....	13
3. Ключи, организация связи между таблицами.....	14
Реляционные ключи.....	14
Связь между таблицами.....	16
Обеспечение целостности данных.....	17
Построение схемы данных средствами Microsoft Access.....	18
Мастер подстановок.....	19
Задание.....	19
4. Модель “сущность-связь” (ER-модель).....	20
Концепция ER-модели.....	20
Задание.....	22
5. Нормализация данных.....	24
Первая нормальная форма (1NF).....	24
Вторая нормальная форма (2NF).....	25
Третья нормальная форма (3NF).....	25
Нормальная форма Бойса-Кодда (BCNF).....	26
Четвертая нормальная форма (4NF).....	26
Пятая нормальная форма (5NF).....	26
Задание.....	26
6. Язык SQL, оператор SELECT.....	27
Выборка значений из таблиц – SELECT.....	27
Порядок сортировки – ORDER BY.....	28
Ограничение набора данных – WHERE.....	28
Предикат существования EXISTS.....	30
Агрегатные функции.....	30
Группировка данных – Group By.....	31
Псевдонимы столбцов.....	31
Псевдонимы таблиц.....	31
Объединение нескольких таблиц.....	32
Задание.....	33
7. Язык SQL, операторы манипулирования данными.....	34
Вставка новой записи – INSERT.....	34
Редактирование данных – UPDATE.....	34
Удаление записей – DELETE.....	35
Задание.....	35
8. Язык SQL, операторы определения данных.....	36
Основные типы данных SQL-92.....	36
Язык определения данных – DDL.....	38
Работа с базой данных.....	38
Работа с таблицами.....	39
Работа с представлениями.....	41

Работа с индексами.....	42
Работа с хранимыми процедурами	42
Работа с триггерами.....	43
Задание.....	43
9. Разработка отчетов БД в среде Access.....	44
Подготовка отчета в среде Access.....	44
Задание.....	45
10. Разработка интерфейса приложения в среде Access.....	46
Создание форм для ввода данных в Microsoft Access.....	46
Рекомендации по проектированию пользовательского интерфейса.....	47
Задание.....	48
11. Доступ к данным средствами ActiveX Data Object (ADO).....	49
Строка соединения ADO.....	50
Соединение с хранилищем данных, компонент TADOConnection.....	51
Установка соединения.....	51
Информирование о БД.....	53
Задание.....	54
12. Общие черты компонентов доступа к данным.....	55
Базовый класс доступа к данным TDataSet.....	56
Открытие и закрытие набора данных.....	56
Перемещение по набору данных.....	56
Создание закладок и переход к закладке	57
Редактирование записей в наборе.....	58
Фильтрация набора данных.....	59
Организация поиска данных.....	60
Взаимодействие с элементами управления данными	60
Задание.....	60
13. Работа с полями набора данных.....	62
Поле таблицы – класс TField.....	62
Классификация полей по функциональному назначению.....	62
Классификация полей по типу обслуживаемых данных	63
Обращение к отдельному объекту-полю.....	63
Доступ к данным поля.....	63
Размер поля.....	64
Значение по умолчанию.....	65
Ограничения на ввод данных.....	65
Маска ввода.....	65
Индексные поля.....	65
Отображение данных	65
Задание.....	66
14. Вычисляемые поля и поля подстановки.....	67
Поля подстановки.....	67
Вычисляемые поля.....	70
Организация отношения главная-подчиненная таблица	70
Задание.....	72
15. Управление индексами, поля BLOB.....	73
Особенности набора данных при работе с вторичными индексами	73
Поля BLOB.....	73
Задание.....	77
16. Разработка интерфейса клиентского приложения.....	78
Источник данных – компонент TDataSource	78
Общие черты компонентов отображения данных.....	79

Сетка базы данных – компонент TDBGrid.....	79
Статический текст – компонент TDBText.....	81
Строка ввода БД – компонент TDBEdit.....	81
Многострочный текстовый редактор БД – TDBMemo.....	81
Изображение БД – компонент TDBImage.....	81
Список БД – TDBListBox.....	81
Комбинированный список БД – TDBComboBox.....	82
Флажок БД – TDBCheckBox.....	82
Радиогруппа БД – TDBRadioGroup.....	82
Компонент – TDBCtrlGrid.....	82
Навигатор – TDBNavigator.....	83
Задание.....	84
17. Создание БД, таблиц и представлений в Microsoft SQL Server.....	85
Создание базы данных.....	85
Удаление базы данных.....	85
Создание таблиц.....	86
Пример создания таблиц средствами Transact SQL.....	87
Создание представлений.....	88
Задание.....	88
18. Введение в Microsoft Transact SQL.....	89
Определение и использование переменных.....	89
Операторы управления Transact-SQL.....	90
Базовые функции Transact-SQL.....	90
Функции преобразования типов данных.....	92
Хранимые процедуры.....	92
Триггеры.....	93
Задание.....	93
19. Разработка приложений по технологии клиент-сервер.....	94
Запрос TADOQuery.....	94
Выполнение запроса SQL.....	94
Параметры запроса.....	94
Хранимая процедура TADOStoredProc.....	96
Соединение с хранимой процедурой.....	96
Выполнение хранимой процедуры.....	96
Транзакции и их изоляция.....	96
Управление транзакциями и компонент TADOConnection.....	97
Задание.....	98
20. Слабо структурированные данные и язык XML.....	99
Построение простейшего документа XML.....	99
Атрибуты.....	100
Определение документа DTD.....	100
Задание.....	102
21. Создание БД в InterBase.....	103
Определение домена в консоли администрирования.....	104
Таблицы и консоль администрирования.....	105
Создание значений ключа с помощью генератора.....	106
Представления и консоль администрирования.....	107
Задание.....	107
22. Процедурный SQL, хранимые процедуры.....	108
Переменные.....	108
Выборка данных с помощью SELECT ... INTO.....	109
Условный оператор IF... THEN... ELSE.....	109

Цикл WHILE...DO.....	110
Цикл выборки данных FOR SELECT...DO.....	110
Оператор SUSPEND.....	112
Оператор EXIT.....	112
Задание.....	112
23. Процедурный SQL, триггеры и события.....	113
Триггеры.....	113
Контекстные переменные.....	113
Ввод значений по умолчанию.....	114
Поддержка корпоративной целостности данных.....	115
События.....	115
Задание.....	116
24. Функции UDF.....	117
Размещение UDF-библиотеки.....	117
Подключение внешней функции к БД.....	117
Подключение UDF в консоли администрирования.....	117
вызов UDF.....	118
Разработка UDF-библиотек в Delphi.....	118
Работа со строками.....	119
Особенности разработки в C++ Builder.....	120
Задание.....	121
25. Подключение к БД из клиентского приложения.....	122
Доступ к базе данных, компонент TIBDatabase.....	122
Соединение с базой данных.....	122
Разрыв соединения.....	125
Управление транзакциями, компонент TIBTranscation.....	126
Параметры транзакции.....	126
Автоматическое управление транзакцией.....	127
Управление транзакцией в ручном режиме.....	128
Задание.....	129
26. Основные компоненты IBX.....	130
Компонент быстрой разработки TIBTable.....	130
Запрос, компонент TIBQuery.....	130
Хранимая процедура, компонент TIBStoredProc.....	132
Универсальный набор данных TIBDataSet.....	133
Формирование запросов.....	133
Задание.....	135
27. Скрипты SQL.....	136
Выполнение скриптов, TIBScript.....	136
Разработка помощника установки БД.....	137
Получение метаданных, TIBExtract.....	138
Разработка генератора скриптов.....	139
Задание.....	140
28. Компоненты для администрирования БД.....	141
Общие черты компонентов.....	141
Свойства сервера, TIBServerProperties.....	143
Конфигурирование БД, TIBConfigService.....	146
Отключение и перезапуск БД.....	147
Задание.....	147
29. Создание БД средствами MySQL.....	148
Создание базы данных.....	148
Изменение параметров БД.....	149

Создание таблиц	149
Столбец-перечисление	150
Столбец-множество	151
Внешние ключи и связи между таблицами	151
Создание индексов	152
Удаление индекса	153
Задание	153
30. Процедурные расширения языка SQL в MySQL	154
Переменные	154
Переменные, определяемые пользователем	154
Системные переменные	154
Переменные в хранимой процедуре	156
Составной оператор BEGIN...END	156
Условные операторы	156
Оператор IF...THEN...ELSE	156
Оператор-селектор CASE	157
Циклы	158
Цикл с предусловием WHILE	158
Цикл с постусловием REPEAT	159
Петлевой цикл LOOP	160
Прерывание цикла — LEAVE	161
Пропуск итерации — ITERATE	161
Задание	161
Приложение 1. Настройка строки соединения ADO	163
Приложение 2. Встроенные функции InterBase	164
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ	165
СПИСОК ИСПОЛЬЗУЕМОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ	166
Пояснительная записка	169
План-график выполнения СРС по дисциплине	169
Методические рекомендации к СРС	169
Форма контроля СРС	170
Задания для СРС	170
Примерные темы рефератов	170
Примерные темы докладов	172
Примерные темы индивидуальных творческих заданий	173
Требования к представлению и оформлению результатов СРС	174
Рекомендуемая литература и интернет-ресурсы	176
1 Общие положения	178
2 Структура и содержание курсовой работы	178
3 Оформление курсовой работы	181
4 Примерный перечень тем курсовых работ	186
ПРИЛОЖЕНИЕ А Образцы библиографической записи	189

ВВЕДЕНИЕ

Дисциплина «Системы управления базами данных» является одной из основных при подготовке специалистов в области компьютерной безопасности. Материал, представленный в учебном пособии по дисциплине «Системы управления базами данных» направлен на формирование набора общепрофессиональных и профессиональных компетенций специалистов, обучающихся по специальности 10.05.01 «Компьютерная безопасность», специализации «Информационно-аналитическая и техническая экспертиза компьютерных систем». В том числе:

ОПК-9 – способен решать задачи профессиональной деятельности с учетом текущего состояния и тенденций развития методов защиты информации в операционных системах, компьютерных сетях и системах управления базами данных, а также методов и средств защиты информации от утечки по техническим каналам, сетей и систем передачи информации;

ОПК-14 – способен проектировать базы данных, администрировать системы управления базами данных в соответствии с требованиями по защите информации;

Главная задача учебного пособия – вооружить будущего специалиста в области компьютерной безопасности всеми необходимыми знаниями для создания профессиональных программных продуктов.

Задачи учебного пособия (лабораторного практикума) совпадают с задачами изучения дисциплины «Системы управления базами данных» и могут быть сформулированы следующим образом:

1. представление об инновационном подходе к вопросам разработки и сопровождения баз данных;
2. эксплуатация и администрирование СУБД;
3. применение языка SQL;
4. основы применения информационных хранилищ и банков данных;
5. использования слабоструктурированных данных и языка XML;
6. понимание построения современных систем управления базами данных и
7. перспективы их развития (объектно-ориентированные БД, документ-ориентированные БД).

Учебное пособие по дисциплине «Системы управления базами данных» представляет собой лабораторный практикум по дисциплине «Системы управления базами данных» и составлено в соответствии с рабочей программой дисциплины для специалистов 10.05.01 «Компьютерная безопасность», специализации «Информационно-аналитическая и техническая экспертиза компьютерных систем».

В методических указаниях приведены алгоритмы, структуры и технологии, широко используемые для разработки современных баз данных. Структурно каждая из работ, в предлагаемом учебном пособии, состоит из краткого изложения теоретического материала, необходимого для ее выполнения, листингов с примерами исходного кода на языках SQL и C++, заданий к лабораторной работе, методики выполнения заданий, контрольных вопросов, списка литературы, рекомендуемой к использованию в данной теме.

1. Системы основанные на файлах

Вид занятия – лабораторная работа

Время занятия – 4 часа

Программное обеспечение – C++ Builder

Одна из самых первых фирм, нацелившая свой научный потенциал на исследование способов хранения и обработки больших массивов данных называлась North American Aviation (NAA)¹. В середине 60-х годов XX века ее инженеры разработали программное обеспечение (ПО) с очень серьезным названием: Generalized Update Access Method (GUAM). Это было не что иное, как прототип представления данных в виде древообразных структур – **иерархическая модель данных** (*hierarchical data model*). Несколько позднее идея, заложенная в GUAM, получила свое развитие в стенах компании IBM. Результат труда IBM реализовался в рамках ПО Information Management System (IMS). Плоды совместного творчества NAA и IBM не только нашли свое отражение в одном из ключевых теоретических направлений развития БД, но и внесли существенный вклад в более осязаемый проект – благодаря IMS проводились расчеты первых полетов американского космического корабля Apollo на Луну.

Однако, не смотря на существенный успех, NAA и IBM несколько не монополизировали рынок будущих систем управления базами данных. Примерно в это же время компания General Electric разработала альтернативную систему Integrated Data Store (IDS). С этим проектом связано одно из ключевых имен исследователей СУБД – имя Чарльза Бачмана. Именно его идеи организации данных в дальнейшем воплотились в принципиально новое направление баз данных – **сетевую модель данных** (*network data model*).

Раз мы заговорили об именах, то нельзя не упомянуть имя ученого, внесшего существенный вклад в становление **реляционной модели данных** (*relational data model*). Это Э.Ф. Кодд, сотрудник одной из научно-исследовательских лабораторий IBM. Благодаря его теоретическим исследованиям в 1976 году в свет вышел проект System-R, послуживший родоначальником всех современных реляционных СУБД.

Файловые системы, принципы построения

Одним из направлений развития программного обеспечения, внесшего существенный вклад в становление современных БД, стали **системы, основанные на файлах** (*file-based system*) или, как их чаще называют **файловые системы**. Знание основных идей, заложенных в файловых системах, и присущих им недостатков позволит нам более глубоко разобраться со способами хранения данных в современных БД и, что очень важно, избежать ошибок свойственных файловым системам.

ФАЙЛОВАЯ СИСТЕМА (ФС) – программа (набор программ) позволяющая добавлять, редактировать, удалять и просматривать данные. Каждая программа определяет свой собственный формат хранения данных и свой собственный способ управления этими данными.

Первые разработки ФС были нацелены на решение элементарных задач – организацию обработки данных, хранившихся в обычных ручных картотеках. С точки зрения программирования решение подобных задач обычно производится в рамках разработки программ, применяющих типизированные файлы.

Недостатки файловых систем

Самый первый недостаток файловых систем проявился очень скоро. Вследствие того, что каждый программист выбирал свой собственный формат хранения данных, создаваемые ими программы и типизированные файлы были просто **несовместимы**. Действительно, разрабатывая структуру, пусть даже предназначенную для описания одних и тех же данных, каждый программист шел по своему пути. Один под поле фамилии отводил 20 символов, другой тридцать... Варьировались не только размерности полей, но и их порядок следования. Сколько разработчиков – столько и мнений...

Второй недостаток также не заставил себя долго ждать. Закладываемая на стадии разработки ПО жесткая физическая структура данных приводила к созданию весьма неповоротливых приложений. Так если в нашем примере для хранения имени студента отводилось 10 символов, то появление необходимости увеличить (или уменьшить) это поле даже на один единственный символ влекло целый каскад проблем. Программист должен был не только переписать все свои программы, но и разработать конвертор для ранее набранных данных в новый формат. Эта проблема называется **зависимостью от данных**.

Вследствие первой пары недостатков файловые системы “обогатились” очередной отрицательной чертой. Несовместимость файлов и зависимость программ от данных привели к тому, что информационные файлы **дублировались** по всему предприятию. Один программист написал программу учета студентов для дека-

¹ Сегодня эта компания называется Rockwell International

ната, другой для отдела кадров, третий для кафедры. Даже если они договорились об едином типе данных, все равно, везде хранится одно и то же, что весьма неэффективно. Другой аспект этого недостатка еще важнее – данные могут стать противоречивыми. Деканат отчислил нерадивого студента, а бухгалтерия продолжает выдавать ему стипендию, ведь в их версии данных этот студент продолжает учиться. Снежный ком имеет особенность к накоплению своей массы. В конечном итоге дублирование данных приводит к их **изолированности**. Во всех отделах и службах предприятия, обрабатывая изначально одинаковые данные, пользователи наполняют их разным (и что самое страшное не всегда корректным) содержанием.

Напоследок расскажем о пятом недостатке файловых систем. Он связан с лавинообразным **разрастанием количества приложений**, обслуживающих файловую систему. В первую очередь это объясняется тем, что любая элементарная задача (например, появление нового запроса к данным) приводит к необходимости кардинальной переработке старого приложения либо к созданию нового.

Таким образом, файловые системы обладают целой плеядой недостатков:

1. Несовместимость данных.
2. Зависимость от данных.
3. Дублирование данных.
4. Изолированность данных.
5. Быстрое разрастание количества приложений.

Это далеко не полный перечень минусов файловых систем, однако их вполне достаточно для того, чтобы сделать вывод о необходимости дальнейшего совершенствования способов хранения данных.

Задание

В среде C++ Builder (или в другой среде проектирования по согласованию с преподавателем) разработайте консольное приложение системы, основанной на файлах. Приложение должно позволять: добавлять, редактировать, удалять и просматривать данные следующего вида:

- В.1. Сотрудники фирмы (фамилия, имя, отчество, должность, зарплата).
- В.2. Список товаров (наименование, стоимость за единицу, количество, поставщик).
- В.3. Телефонный справочник (фамилия, инициалы, номер телефона).
- В.4. Домашняя библиотека (автор, название произведения, жанр).
- В.5. Телепередачи (название телевизионного канала, название передачи, дата и время эфира).
- В.6. Фонотека (исполнитель, название концерта, год выхода).
- В.7. Справочник адресов (фамилия, инициалы, город, улица, дом, квартира).
- В.8. Склад телевизоров (фирма-изготовитель, модель, диагональ экрана, разрешение экрана).
- В.9. Кадры (должность, фамилия и инициалы, номер кабинета, рабочий телефон).
- В.10. Тарифы мобильного оператора (название тарифа, число доступных минут, число доступных СМС, стоимость в месяц).

Советы:

1. На начальном этапе выполнения лабораторной работы хорошей идеей станет разработка структуры **struct** для хранения отдельной записи (см. лаб. работу № 1 “Методы программирования”).
2. Для организации доступа к бинарному файлу допускается использовать низкоуровневые функции для работы с файлами или наиболее подходящие классы (например `TFileStream` в C++ Builder).

Внимание!

Разрабатываемая система, основанная на файлах, должна хранить данные в **бинарном** (двоичном файле)!

2. Реляционные таблицы

Вид занятия – лабораторная работа

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access

Подавляющее большинство недостатков характерных для файловых систем появились вследствие всего из-за двух причин. **Первая** из них в том, что в файловых системах **способы описания данных** (их структуры) **хранились внутри исполняемого кода приложения**. В этом мы убедились на прошлом занятии, определив структуру TStudent в исходном коде проекта. В результате знаниями о том, каким образом можно обратиться к данным типизированного файла, обладал только наш исполняемый файл. Для всех сторонних разработчиков это осталось загадкой. **Вторая** причина заключается в **отсутствии универсального инструмента обеспечивающего доступ к данным**, хранящимся в любом типизированном файле. Таким образом, до содержащейся в файле информации способно добраться только его “родное” приложение.

Выявив причины первых неудач, разработчики программного обеспечения пришли к однозначному выводу: **программа и данные должны стать независимыми** (program-data independence). Но каким образом этого добиться? Достаточно просто – надо вывести описание структур из программы. Данные должны содержать не только информацию, но и ее характеристики. Прежде чем обратиться к информации, приложение знакомится с метаданными (так называется описание данных). выяснив используемые типы данных, их размерности, и если необходимо другие дополнительные параметры, программа без проблем получала доступ к основной информации. Описание данных программисты называют **системным каталогом** (system catalog).

Вынос описаний данных из состава приложения стал ключевым шагом, навстречу будущим базам данных.

БАЗА ДАННЫХ – это совместно используемое пользователями единое хранилище логически связанных данных и их описаний.

База данных не жизнеспособна сама по себе, она нуждается в помощнике – программе обслуживания. Эта программа должна выполнять посредническую роль при организации общения конечного пользователя и находящихся в базе данных. Такая программа называется системой управления базами данных.

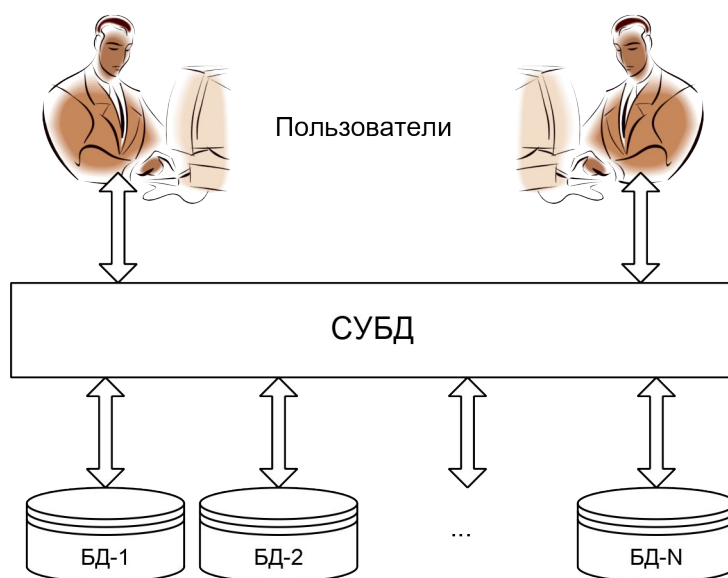


Рис. 2.1. Взаимодействие пользователя и БД

СИСТЕМА УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ – это программное обеспечение, позволяющее создавать БД, хранить и извлекать информацию из БД, а также обеспечивать контроль за доступом к БД.

На плечи СУБД возложено большое количество разнообразных задач. Пока мы упомянем лишь то, что СУБД дает пользователю абстрактное представление данных, скрывая особенности физического хранения данных и управления ими.

В качестве примеров современных реляционных СУБД отметим системы, работающие по технологии клиент-сервер: Oracle, Informix, InterBase, Microsoft SQL Server и однопользовательские системы Microsoft Access, Microsoft FoxPro, Paradox.

Реляционная таблица

Родоначальником реляционной модели данных считается Э.Ф.Кодд (E.F.Codd). Именно этот ученый в 1970 году опубликовал фундаментальную статью "Реляционная модель данных для больших совместно используемых банков данных". Анонсированная в статье идея реляционной модели основывается на теории множеств и логике предикатов. Фундаментальным понятием реляционной модели считается отношение. Отношения предназначены для хранения информации об объектах, представленных в базе данных. Пример отношения представлен на рисунке 2.2.

ОТНОШЕНИЕ – двумерная таблица, состоящая из столбцов и строк.

	Имя столбца				Столбец
	Key	Surname	FName	SpCod	YearNum
	1	Петров	Николай	Комп.безопасность	1
	2	Алфёрова	Елена	ОиТЗИ	2
	3	Савченко	Алексей	Комп.безопасность	4
Строка	3	Пальцев	Роман	Комп.безопасность	4

Рис. 2.2. Пример отношения

В современной литературе при описании отношения зачастую встречаются разные определения. Стоит знать, что в качестве синонима “строка отношения” применяются термины “запись” и “кортеж”. Кроме термина “столбец отношения” можно услышать “атрибут” и “поле”. “Отношение” также имеет свой синоним – это “таблица”.

Определение домена

Любая СУБД позволяет формировать столбец таблицы таким образом, чтобы он стал максимально удобным для своих данных. На стадии проектирования таблицы разработчик обязан уделить этому самое скрупулезное внимание, такой процесс называется **определением домена**.

ДОМЕН – это набор допустимых значений для одного или нескольких столбцов отношения.

Благодаря домену разработчик БД получает возможность централизованно определять смысл и источник значений, которые могут получать поля таблицы. Кроме предотвращения ввода некорректных данных домен поможет избежать семантически некорректных операций. Например, бессмысленного сравнения цены товара и номера телефона.

Характеристики любого столбца реляционной таблицы всегда определяется индивидуальными свойствами данных, которые мы намерены хранить в этом поле. Если столбец предназначен для фамилии или имени человека, то мы выберем текстовый тип данных и ограничим его 15-20 символами. Если речь идет о количественных значениях, то мы воспользуемся числовыми типами данных. В таблице 2.1 представлены основные типы данных, которые могут иметь поля в Microsoft Access.

Таблица 2.1. Основные типы данных Microsoft Access

Тип данных	Назначение	Размер
Текстовый	Хранение текстовых данных.	До 255 символов
Поле МЕМО	Большое количество неформатированных текстовых данных.	До 65 536 символов.
Числовой	Хранение целых и действительных значений.	1, 2, 4 или 8 байт
Денежный	Денежные величины.	8 байт
Дата/время	Значения дат и времени.	8 байт
Счетчик	Автоинкрементное значение.	4 байта
Логический	Булевы значения (истина/ложь).	1 бит

В отличие от всех языков программирования базы данных построены на 3-значной логике. Помимо значений True (истина) и False (ложь) мы можем столкнуться с определителем NULL. Определитель NULL воспринимается как неопределенное (или неизвестное) значение.

Определитель NULL указывает, что значение поля в настоящий момент неизвестно или недопустимо для записи

Определившись с типом данных и их размерностью, программист может наложить на данные дополнительные ограничения:

1. Поставить признак обязательного поля. Если признак не установлен, то надо быть готовым к тому, что пользователь не станет заполнять поле данными. Таким образом, его содержимое станет неопределенным – NULL.
2. Указать значение, передаваемое в поле по умолчанию.
3. Ввести ограничения на допустимые значения (например, диапазон значений, которые разрешено передавать в поле).
4. Назначить маску ввода, определяющую жесткий порядок ввода и представления значений.

Создание таблиц в среде Microsoft Access

Попробуем повторить простую задачу по учету студентов ВУЗа, но на этот раз не с помощью файловых систем, а на основе таблиц. В качестве инструмента воспользуемся услугами среды Access – программного продукта Microsoft предназначенного для построения реляционных баз данных малой степени сложности.

Запустите приложение Access. Воспользовавшись пунктом меню: Файл – Создать, создайте новый файл базы данных и сохраните его под любым именем, например “students.mdb”.

Access предлагает два способа создания таблиц (рис. 2.3):

1. В режиме конструктора.
2. Путем ввода данных.

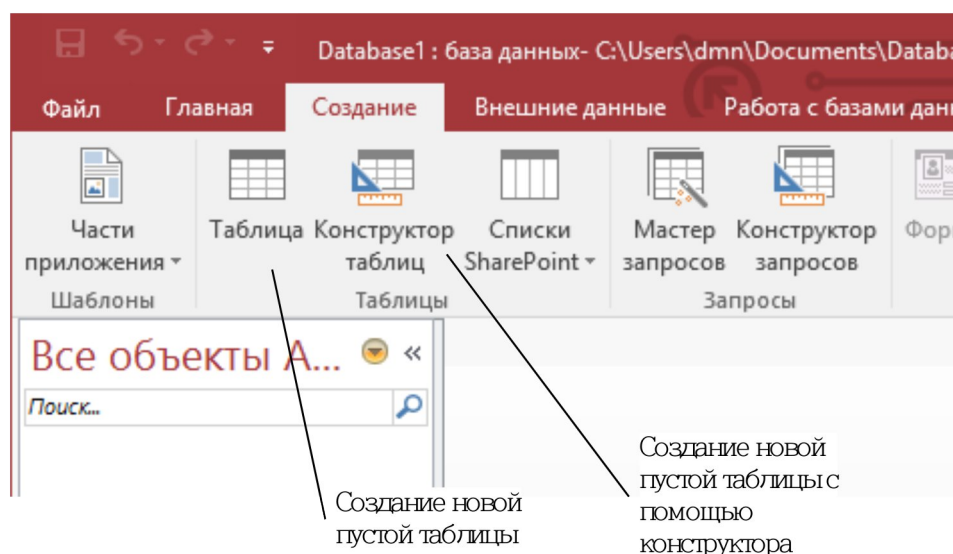


Рис. 2.3. Режимы создания таблиц в среде Access

Мы направимся по наиболее функциональному пути и выберем режим конструктора. Именно этот режим предоставляет пользователю максимальные возможности по формированию будущей таблицы. Двойной щелчок по строке “Создание таблицы в режиме конструктора” вызовет окно конструктора (рис. 2.4).

Совет! При назначении имен столбцам и выборе названия таблицы старайтесь не применять символы национальных алфавитов

Students

Имя поля	Тип данных	Описание (необязательно)
SName	Короткий текст	Фамилия
FName	Короткий текст	Имя
SPCOD	Короткий текст	Код специальности
YEARNUM	Числовой	Курс

Свойства поля

Общие	Подстановка
Размер поля	30
Формат поля	
Маска ввода	
Подпись	
Значение по умолчанию	
Правило проверки	
Сообщение об ошибке	
Обязательное поле	Нет
Пустые строки	Да
Индексированное поле	Нет
Сжатие Юникод	Да
Режим IME	Нет контроля
Режим предложений IME	Нет
Выравнивание текста	Общее

Имя поля может содержать не более 64 знаков (включая пробелы). Для получения справки по именам полей нажмите клавишу F1.

F1 = справка.

Рис. 2.4. Конструктор таблицы

В конструкторе назначаются имена полей (столбцов) будущей таблицы, определяются типы данных и их ограничения. Например, для хранения фамилии мы определяем поле “SName”, тип данных “Текстовый”, размер столбца – 30 символов.

Созданная нами таблица пока не может претендовать на высокое звание реляционной таблицы (R-таблицы). Для этого она должна пройти несколько стадий нормализации, но уже сейчас в ней присутствуют некоторые родовые черты R-таблиц:

1. Таблица поименована.
2. Каждое поле таблицы имеет уникальное имя.
3. Каждое поле содержит только атомарное (неделимое) значение.
4. Значения полей описываются доменами.

Задание

В среде Microsoft Access в режиме конструктора создаете таблицу, предназначенную для хранения следующих данных:

В.1. Деловые знакомства (фамилия, имя, отчество, пол, название фирмы, должность, дата рождения, рабочий телефон).

В.2. Склад магазина (наименование товара, артикул, стоимость, количество, поставщик товара, номер телефона поставщика, дата поставки, место хранения (номер стеллажа)).

В.3. Телефонный справочник (фамилия, инициалы, номер телефона, адрес абонента).

В.4. Домашняя библиотека (автор, название произведения, жанр, издательство, год издания, количество страниц).

В.5. Телепередачи (название телевизионного канала, название передачи, дата и время эфира).

В.6. Домашняя фонотека (исполнитель, название концерта, год выхода, формат данных (CD, DVD, mp3), страна).

3. Ключи, организация связи между таблицами

Вид занятия – лабораторная работа

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access

Создаваемые нами на прошлом занятии таблицы нельзя назвать в полной мере реляционными. Они не удовлетворяют ряду требований, предъявляемых к реляционным отношениям. Сегодня мы уделим внимание двум из них:

1. Обеспечению уникальности строк в R-таблице.
2. Устранению избыточности данных в R-таблице.

Реляционные ключи

Реляционная модель требует, чтобы каждая отдельная строка отношения уникально идентифицировалась по значениям, содержащимся в ее столбцах.

Рассмотрим таблицу, предложенную на рисунке 3.1.

SName	FName	SPCOD	YEARNUM
Петров	Михаил	10.05.01	2015
Алферова	Елена	10.05.03	2016
Савченко	Алексей	10.05.01	2017
Пальцев	Роман	10.05.03	2017
*			0

Рис. 3.1. Данные таблицы Students

В таблице содержится только 4-ре строки. Технически все эти записи уникальны, ведь нет ни одного повторения в содержащейся в строках информации. Но это не добавляет нам уверенности, в том, что такая благоприятная ситуация будет продолжаться и дальше. Можно ли с вероятностью в единицу исключить возможность поступления на один и тот же курс, на одну и ту же специальность двух человек с одинаковыми именами и фамилиями? Допустим, что кроме Петрова Михаила (с отчеством Алексеевич) в его же группе учился Петров Михаил Антонович. Внеся данные о втором Петрове в таблицу, мы сразу получим две абсолютно одинаковые строки. С этого момента никто не сможет отличить записи друг от друга. С точки зрения реляционной модели это недопустимо. Элементарный практический пример. Допустим, что второй Петров не сдал сессию и подлежит отчислению. Вопрос: какую строку из таблицы следует удалить?

Что делать, чтобы добиться уникальности каждой записи в таблице? Добавить в наше отношение поле отчества (LName)? Хорошее решение, но согласитесь, что все-таки, можно допустить и такое совпадение, что нам встретятся полные тески – Ивановы Иваны Ивановичи. Надо идти дальше – ввести в таблицу даты рождения, либо номер паспорта, а еще лучше то и другое. Одним словом, обеспечение уникальности процесс творческий, требующий опыта и интуиции.

Суперключ (superkey) – это столбец (множество столбцов), которые единственным образом идентифицирует запись в таблице

Вполне вероятно и обратная ситуация, что в одной и той же строке отношения содержится несколько полей (или групп полей), каждое из которых в отдельности уникальным образом идентифицирует запись в таблице. Это могут быть номер паспорта, индивидуальный номер налогоплательщика, номер мобильного телефона. В таком случае говорят о существовании нескольких **потенциальных ключей**.

Задача разработчика базы данных заключается в выборе (или создании) наиболее подходящего ключа в отношении и назначении его на роль первичного ключа таблицы.

Первичный ключ – это потенциальный ключ, который выбран для уникальной идентификации записей внутри таблицы

Первичный ключ может включать одно поле таблицы или несколько ее полей. В последнем случае говорят о **составном первичном ключе**.

Многие СУБД позволяют использовать в качестве ключа поле автоинкрементного типа. Подобная возможность имеется и в Access, для этого предназначен счетчик. Для присвоения полю признака первичного ключа щелкните правой кнопкой мышки по имени поля и выберите пункт меню “ключевое поле”.

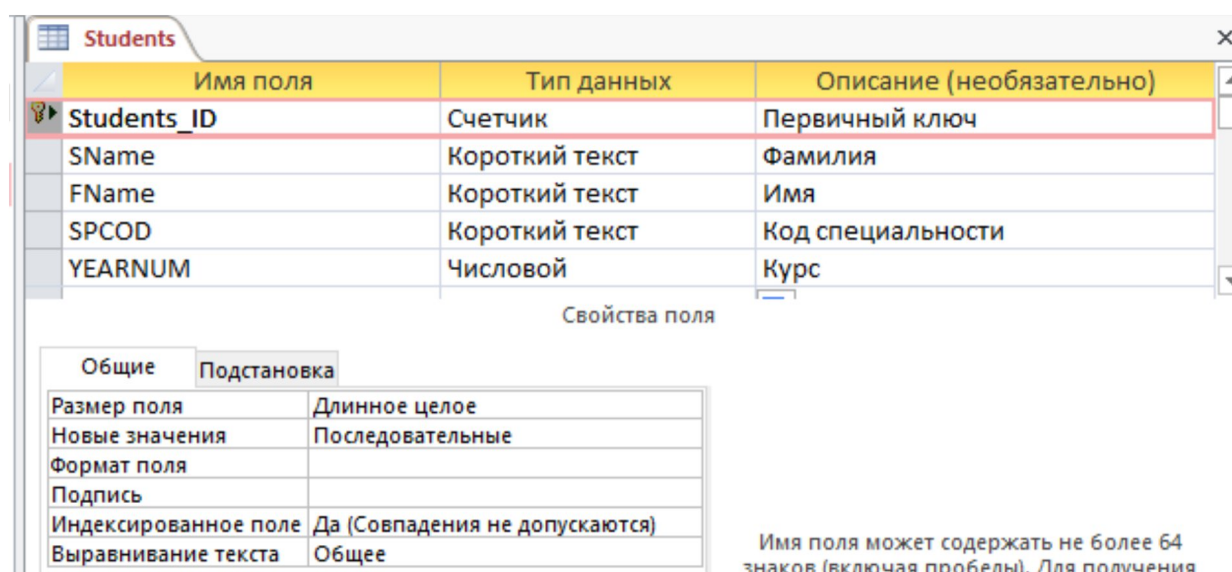


Рис. 3.2. Добавление первичного поля в таблицу

При выборе названия поля, которое станет первичным ключом, используйте название таблицы и аббревиатуру “ID” (от англ. identification). Например: Students_ID.

Внеся необходимые дополнения в таблицу, сохраните ее. Вставьте в таблицу новую запись и посмотрите на поведение поля “Students_ID” – поле автоматически получило новое значение (предыдущее значение + 1). Таким образом, СУБД гарантирует, что каждая новая строка отношения получит очередной **уникальный идентификатор**. Даже если мы удалим последнюю строку в таблице, значение старого первичного ключа больше никогда не будет задействовано.

Students_ID	SName	FName	SPCOD	YEARNUM	Щелки
1	Петров	Михаил	10.05.01	2015	
2	Алферова	Елена	10.05.03	2016	
3	Савченко	Алексей	10.05.01	2017	
4	Пальцев	Роман	10.05.03	2017	
5	Нагорный	Олег	10.05.03	2016	
(№)				0	

Рис. 3.3. Поведение поля-счетчика при добавлении новой записи

Рассказывая о ключах, мы пока не упомянули еще один важный вид ключа – внешний. О нем мы поговорим в следующем подразделе, в котором мы рассмотрим один из способов устранения избыточности в R-таблицах.

Связь между таблицами

Внимательно изучите данные из таблицы Students. Не замечаете ли вы в ней очередного изъяна? Профессиональному разработчику БД достаточно одного взгляда на поле “SPCOD” чтобы сделать вывод о том, что наша таблица весьма неуважительно относится к дисковому пространству компьютера. И действительно, столбец, содержащий информацию о специальности, хранит многократно повторяющиеся значения: строка 10.05.01 (код специальности “Компьютерная безопасность”) встречается дважды, а строка 10.05.03 целых три раза. Наличие избыточных данных в таблице это один из верных признаков, что она не является реляционной.

Один из способов – разбиение таблицы с избыточными данными на две таблицы. В нашем случае за пределы таблицы Students выносятся информация о специализации. Новая таблица будет содержать всего два поля:

1. Поле первичного ключа Spec_ID.
2. Поле названия специализации SpName.

Создайте эту таблицу и назовите ее “Spec” (рисунок 3.4).

Имя поля	Тип данных	Описание (необязательно)
Spec_ID	Счетчик	Первичный ключ
SpName	Короткий текст	Код специальности

Свойства поля

Общие Подстановка

Размер поля	15
Формат поля	
Маска ввода	
Подпись	
Значение по умолчанию	
Правило проверки	
Сообщение об ошибке	
Обязательное поле	Да
Пустые строки	Да
Индексированное поле	Да (Совпадения не дс)
Сжатие Юникод	Да
Режим IME	Нет контроля
Режим предложений IM	Нет
Выравнивание текста	Общее

Имя поля может содержать не более 64 знаков (включая пробелы). Для получения справки по именам полей нажмите клавишу F1.

Рис. 3.4. Таблица специализации

Заполните вновь созданную таблицу названиями специальностей – “10.05.01” и “10.05.03”.

Students		Spec	
Spec_ID	SpName	Щелкните для добавления	
1	10.05.01		
2	10.05.03		
*	(№)		

Рис. 3.5. Содержание таблицы специальностей

Наша таблица “Students” пока не способна воспринимать данные из вновь созданной справочной таблицы “Spec”. Для этого таблица студентов должна быть оснащена специальным полем, благодаря которому будет организовано взаимодействие с внешней таблицей. Такое поле называется **внешним ключом**, в него станут передаваться значения первичных ключей соответствующих специальностей.

Внешний ключ – это столбец (или множество столбцов таблицы), которое соответствует первичному ключу некоторой (может быть, той же самой) таблицы.

Для создания внешнего ключа (рис. 3.6):

1. Откройте таблицу списка студентов в режиме конструктора.
2. Создайте числовое поле Spec_ID (размер поля “Длинное целое” – 4 байта). Это поле и будет внешним ключом.
3. Удалите из таблицы “Students” старое поле “SpCod”.

Имя поля	Тип данных	Описание (необязательно)
Students_ID	Счетчик	Первичный ключ
SName	Короткий текст	Фамилия
FName	Короткий текст	Имя
YEARNUM	Числовой	Курс
Spec_ID	Числовой	Внешний ключ к таблице Spec

Свойства поля

Общие	Подстановка
Размер поля	Длинное целое
Формат поля	
Число десятичных знаков	Авто
Маска ввода	
Подпись	
Значение по умолчанию	0
Правило проверки	
Сообщение об ошибке	
Обязательное поле	Нет
Индексированное поле	Нет
Выравнивание текста	Общее

Имя поля может содержать не более 64 знаков (включая пробелы). Для получения справки по именам полей нажмите клавишу F1.

Рис. 3.6. Создание поля внешнего ключа в таблице “Students”

Подготовив поле внешнего ключа, нам необходимо связать обе таблицы. В Access этого результата можно достичь, воспользовавшись сервисной возможностью среды – схемой данных.

Обеспечение целостности данных

При разработке таблиц и организации связи между таблицами разработчик БД обязан определить правила целостности данных. Различают два основных правила поддержания целостности, называемые: **целостностью сущностей** и **ссылочной целостностью**. Первое и второе правила имеют прямое отношения к порядку назначения первичного и внешних ключей.

Правило целостности сущностей. В таблице ни один столбец, входящий в состав первичного ключа не может содержать неопределенных (NULL) значений.

Заложенный в правиле смысл абсолютно прозрачен. Первичный ключ — это минимальный идентификатор, который используется для уникального определения строк в таблицах. Допуская, что, хотя бы одно поле составного ключа содержит NULL, мы разрушаем саму логику построения составного ключа — получается, что не все поля ключа необходимы для уникальной идентификации записи.

Правило ссылочной целостности. Значение внешнего ключа таблицы должно либо соответствовать значению первичного ключа некоторой строки в базовой таблице, либо задаваться определителем NULL.

В соответствии с правилом ссылочной целостности считается допустимым создание записи с информацией о новом студенте с указанием определителя NULL вместо номера группы, в котором этот студент учится. Такая ситуация может иметь место в том случае, когда студент зачислен в институт, но еще не приписан к какому-то конкретному отделению.

Построение схемы данных средствами Microsoft Access

Вызовите схему данных (пункт меню “Сервис – Схема данных”) и добавьте в нее обе наши таблицы. Удерживая в нажатом состоянии левую кнопку мышки, перетащите поле “Spec_ID” из таблицы “Spec” в таблицу “Students”. В появившемся окне “Изменение связей” включите опцию “Обеспечение целостности данных”. Связь с включенными условиями целостности данных обеспечивает соответствие каждого значения, которое вводится в столбец внешнего ключа, существующему значению в связанном столбце первичного ключа. В результате наших действий между таблицами будет организована связь “один-ко-многим” (рис. 3.7).

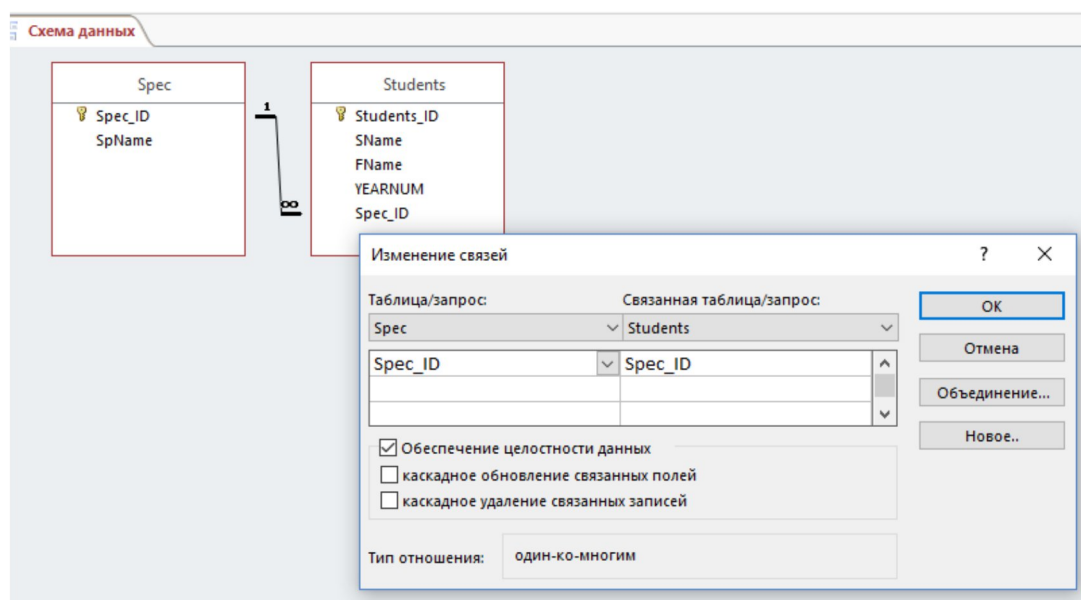


Рис. 3.7. Схема данных БД

В нашем случае связь “один-ко-многим” означает, что одна специальность из таблицы “Spec” может соответствовать многим студентам. Рассмотрим рисунок 3.8. Теперь вместо ввода названия специальности в поле внешнего ключа списка студентов достаточно передать его ключ. Значение 1 соответствует специальности “10.05.01”, 2 – “10.05.03”.

Таблица Students					Таблица Spec	
Students_ID	SName	FName	YearNum	Spec_ID	Spec_ID	SpName
1	Петров	Михаил	1	1	1	10.05.01
2	Алфёрова	Елена	2	2	2	10.05.03
3	Савченко	Алексей	4	1		
4	Пальцев	Роман	4	2		
5	Нагорный	Олег	2	2		

Рис. 3.8. Таблицы с данными

Мастер подстановок

Вынеся повторяющиеся данные во внешнюю таблицу, мы избавились от избыточности, но утратили наглядность представления данных. Раньше пользователю было достаточно одного взгляда на таблицу студентов, чтобы узнать по какой специальности учится тот или иной человек. Теперь же увидев в столбце Spec_ID студента Петрова значение 1, необходимо обратиться к справочнику специальностей, чтобы выяснить, что единица соответствует компьютерной безопасности.

Попробуем сделать список студентов более приветливым. Для этого откройте таблицу студентов в режиме конструктора. выберите поле внешнего ключа Spec_ID. В нижней части окна откройте страничку “Подстановка”. На этой странице расположена всего одна строка “Тип элемента управления” – укажите в ней, что мы намерены работать с полем со списком. Сразу после этого действия на странице отобразятся дополнительные свойства поля (рис. 3.9).

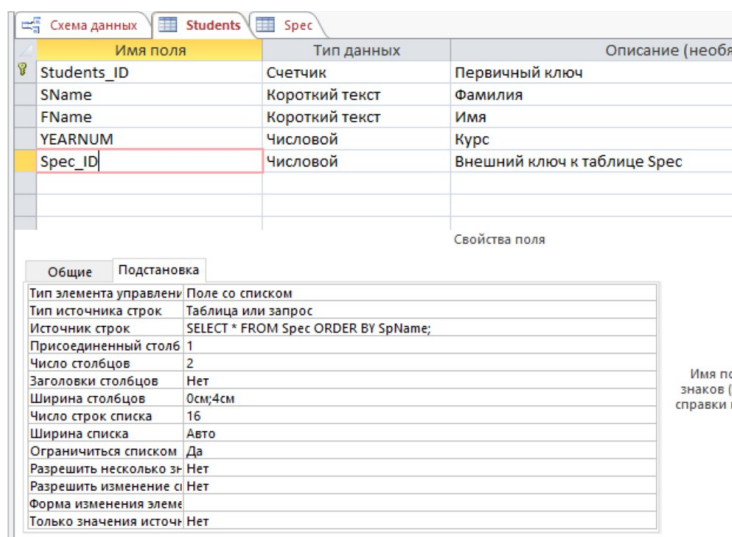


Рис. 3.9. Настройка поля Spec_ID таблицы студентов

В элементе “Источник строк” введите инструкцию на структурированном языке запросов.

```
SELECT * FROM Spec ORDER BY SpName;
```

Эта команда укажет, на то, что в качестве источника данных будет выступать весь перечень специальностей из таблицы, в дополнение ко всему список упорядочится по алфавиту.

Осталось внести два изменения. Укажите, что присоединяются два столбца (свойство “Число столбцов”) и размеры этих столбцов 0 и 4 сантиметра (свойство ширина столбцов). Закройте конструктор, не забыв сохранить изменения. Откройте таблицу Students в режиме редактирования данных. Благодаря мастеру подстановок вместо невыразительных значений ключей поле Spec_ID научилось отображать истинные названия специальностей (рис. 3.10).

Students_ID	SName	FName	YEARNUM	Spec_ID	Щелкните здесь, чтобы увидеть список
1	Петров	Михаил	2015	10.05.01	
2	Алферова	Елена	2016	10.05.03	
3	Савченко	Алексей	2017	10.05.01	
4	Пальцев	Роман	2017	10.05.03	
5	Нагорный	Олег	2016	10.05.03	
*	(№)		0	10.05.01	
				10.05.03	

Рис. 3.10. Поведение поля с подстановкой

Задание

Проанализируйте таблицы, созданные вами на лабораторной работе № 2. Определите ключевые поля, выявите и устраните повторяющиеся данные.

Изучите какие еще варианты полей позволяет создать Access.

4. Модель “сущность-связь” (ER-модель)

Вид занятия – практическое занятие

Время занятия – 2 часа

Программное обеспечение – Microsoft Visio

При проектировании структур сравнительно простых баз данных разработчик не сталкивается с существенными трудностями. Тому подтверждение пример БД по учету студентов ВУЗа. Мы легко разработали 2-х табличную физическую модель, позволяющую хранить требуемые данные. Ситуация кардинально меняется, при постановке задач более высокой степени сложности. Даже профессиональному создателю БД тяжело с первой попытки сформировать корректную физическую модель будущей базы данных. Именно поэтому при разработке серьезных БД процессу физического моделирования предшествует построение более простых моделей БД: концептуальной и логической. На сегодняшнем занятии мы рассмотрим один из методов формирования высокоуровневой концептуальной модели предложенный в 1976 году Питером Ченом. Анонсируемая модель называется моделью “сущность-связь” (Entity-Relationship model, или просто ER-моделью).

Концепция ER-модели

Идея модели “сущность-связь” зиждется на трех понятиях: тип сущности (*entity type*), атрибут и тип связи (*relationship type*).

ТИП СУЩНОСТИ – это физический или абстрактный объект, подлежащий учету в БД. Позднее (на физическом этапе моделирования) тип сущности превратится в реляционную таблицу

Для выявления типов сущностей необходимо проанализировать всю информацию, подлежащую учету в будущей БД. Если объект отвечает на вопрос “Кто?” или “Что?”, то это первый признак, что перед нами тип сущности. Примерами типов сущностей могут стать: “Студент”, “Специальность”, “Автомобиль”, “Товар”. Типы сущности разделяются на **сильные** и **слабые**.

Под **слабым типом сущности** понимается, тип сущности, зависящий от какого-то другого типа сущности, **сильный тип сущности** не зависит от какого-то другого типа сущности

В нашем примере тип сущности “Студент” зависит от типа сущности “Специальность”, таким образом “Студент” считается слабым, а “Специальность” – сильным типом сущности. На диаграммах сильный тип сущности изображается в виде прямоугольника с именем сущности внутри него, а каждый слабый тип сущности – в виде прямоугольника с двойным контуром (рис. 4.1).

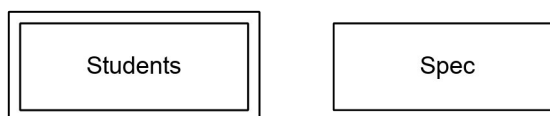


Рис. 4.1. Изображение типов сущностей на диаграммах

Каждый физический или абстрактный объект обладает какими-то характеристиками. У человека есть, имя, пол, возраст, профессия... У автомобиля: марка, цвет, объем двигателя... Для описания этих объектов типы сущности также должны быть вооружены определенными свойствами, называемыми атрибутами.

АТТРИБУТ – свойство типа сущности. На физическом этапе моделирования на основе атрибутов создаются столбцы таблиц.

В первую очередь атрибуты подразделяются на **простые**, **составные** и **производные**. Простой атрибут описывают одно независимое свойство типа сущности, составной – способен работать с несколькими свойствами. Примером простого атрибута может стать пол человека или дата рождения, примером составного – имя, отчество и фамилия. Производный (вычисляемый) атрибут представляет собой значение, полученное за счет обработки данных, содержащихся в других атрибутах типа сущности. Например, возраст человека в годах можно получить за счет вычитания из текущей даты данных хранящихся в атрибуте даты рождения.

При изображении модели атрибуты рисуются в виде эллипсов, присоединенных линией к соответствующей сущности и помеченных именем атрибута. Эллипс, содержащий производный атрибут окружается пунктирным контуром. Если атрибут является составным, то входящие в его состав атрибуты изображаются в виде присоединенных к нему эллипсов. Если атрибут входит в состав первичного ключа, то его название подчеркивается. На рисунке 4.2 предложен один из вариантов ER-модели будущей таблицы студентов.

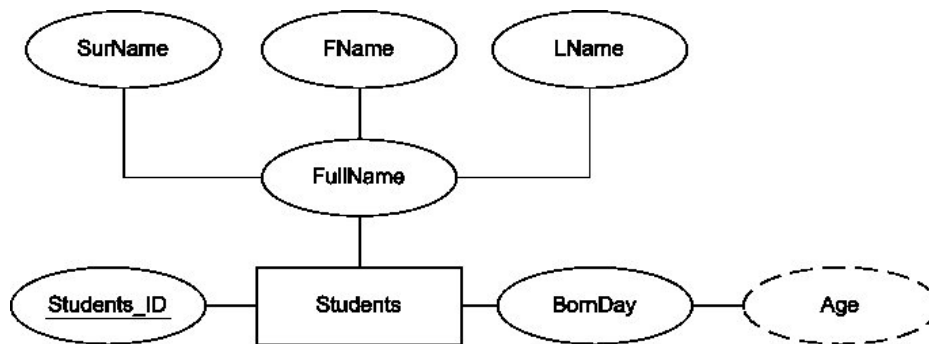


Рис. 4.2. Представление атрибутов в ER-модели

Зачастую, на первом этапе формирования ER-модели, проектировщик БД ошибочно принимает типы сущностей за атрибуты. Так и мы бы могли дополнить тип сущности “Students” атрибутом специальности. Такая ошибка выявляется при исследовании способов взаимодействия типов сущностей между собой.

Для организации взаимодействия между типами сущностей применяются типы связи.

ТИП СВЯЗИ представляет собой ассоциацию между типами сущностей.

Отличительным признаком наличия типа связи между двумя типами сущностей считается возможность вставки глагола между названиями типов сущностей. Например,

“Студент” *учится* по “Специальности”.

“Фирма” *расположена* в “Городе”.

На диаграммах связь обозначается в виде ромба, внутри которого записывается имя связи. Ромбик имеет двойной контур, если связь соединяет слабый тип сущности с сильным типом сущности. Для снижения уровня детализации допускается, чтобы на ER-диаграмме отображались только первичные ключи типов сущностей (рис. 4.3).

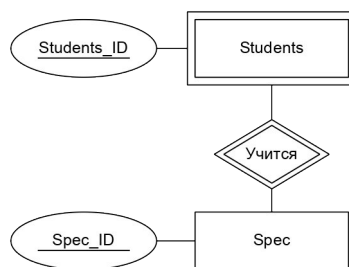


Рис. 4.3. Представление типа связи на ER-диаграмме

Вполне возможна ситуация, когда типом связи охватывается одна и та же таблица. Такое часто бывает при построении иерархических структур, например организационной структуры предприятия. Такой тип связи называется **рекурсивной связью**. В последнем случае на каждом плече связи помечается **ролевое имя** этой связи (рис. 4.4.).

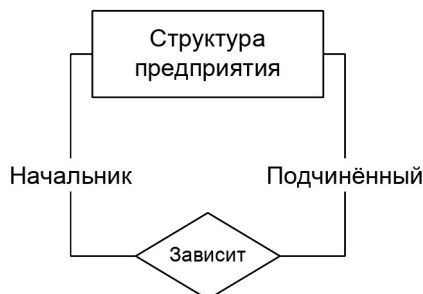


Рис. 4.4. Рекурсивный тип связи

К базовым видам связи относят:

- а) Связь “один к одному”.
- б) Связь “один ко многим”.
- в) Связь “многие ко многим”.

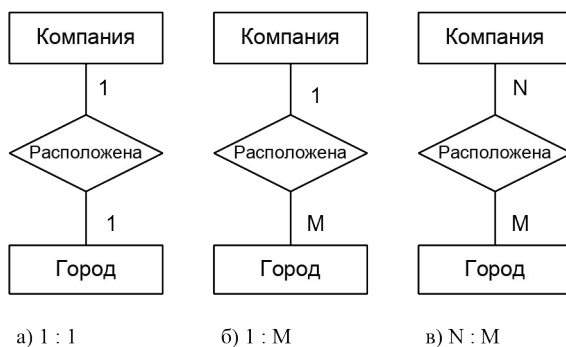


Рис. 4.5. Варианты видов связи на ER-диаграмме

Для выявления какой именно вид связи должен использоваться между двумя типами сущностей строят диаграмму для экземпляров сущностей. О наличии связи “один к одному” (рисунок а) следует говорить в тех случаях, когда каждая из гипотетических компаний ($K_1, K_2, \dots, K_n$) в состоянии обслуживать только в один из городов. И наоборот – каждый из городов в состоянии содержать только одну компанию. На рисунке б) приведен пример связи один – ко многим. Такой вид связи проявляется в тех случаях, когда, например, в одном городе может размещаться несколько компаний, а компания, в свою очередь, в состоянии обслуживать только один город. И, наконец, связь “многие ко многим” (рисунок в) имеет место в случае отсутствия ограничений на количество компаний в городах.

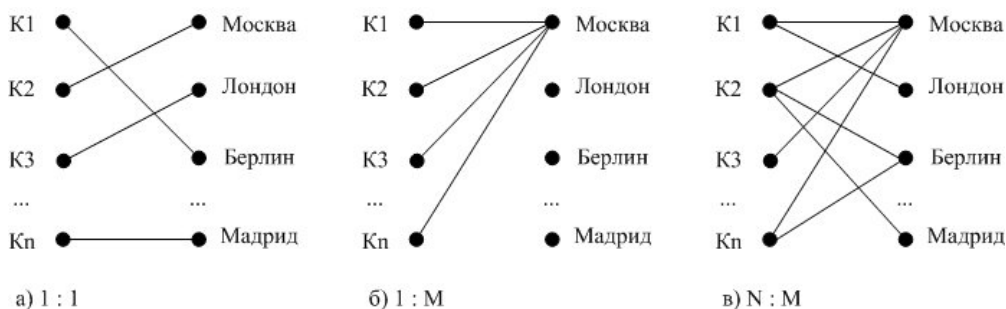


Рис. 4.6. Варианты видов связи

Стоит отметить, что возникновение на диаграмме связи “один к одному” как правило свидетельствует, что один из типов сущностей выбран не верно, и он скорее всего является обычным атрибутом.

Появление на ER-диаграмме связи “многие ко многим” приводит к тому, что при построении физической модели появляется третья таблица. Она выполняет роль диспетчера (коммутатора между основными таблицами). Диспетчер преобразует связь “многие ко многим” в две связи вида “один ко многим”.

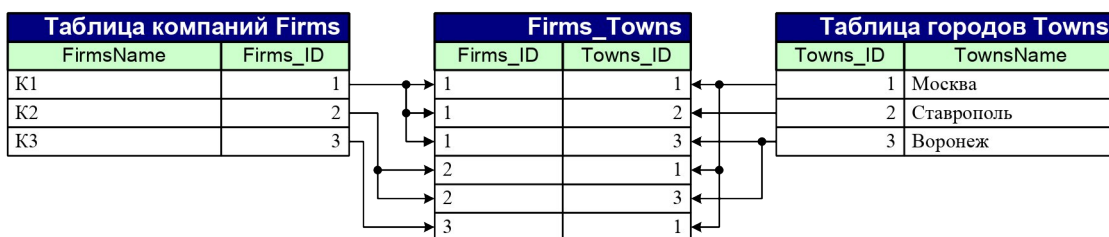


Рис. 4.7. Пример реализации связи многие ко многим с помощью таблицы-коммутатора

Задание

В соответствии с полученными вариантами разработайте ER-модель для будущей базы данных:

1. “База данных поликлиники”. Учету подлежат: больные (ФИО, домашний адрес), лечащий персонал (ФИО, специальность), дата посещения больным врача и поставленный диагноз.
2. “База данных футбольных матчей”. Учету подлежат: спортивный клуб, игроки команд (ФИО), дата матча, игравшие команды, забитые мячи, авторы голов.
3. “База данных “Бюро по туризму””. Учету подлежат: клиенты фирмы (ФИО, домашний адрес), список курортов с ценами и продолжительностью путевок, даты поездок клиентов на курорт.

4. “База данных “Магазин видеофильмов”. Учету подлежат: название фильма, жанр, год выхода, киностудия, режиссер, цена, носитель (DVD/Видеокассета), число экземпляров.
5. “База данных “Домашняя фонотека”. Учету подлежат: исполнитель, страна к которой принадлежит исполнитель, название альбома, год выхода альбома, музыкальный жанр альбома, наименование и продолжительность музыкальных композиций, входящих в альбом, формат хранения (CD, mp3, ...).
6. База данных “Склад автомобильных запчастей”. Учету подлежат: название детали, производитель, цена, в какой марке автомобиля применяется, поставщик детали.
7. База данных “Метеостанция”. Учету подлежат: названия станций, их географические координаты, персонал станций (ФИО, должность), каждая метеостанция ежечасно сохраняет данные о температуре окружающей среды.
8. База данных “Библиотека”. Учету подлежат: жанр книги, авторы книги, название книги, количество экземпляров, место хранения книги, читатель (ФИО и адрес), дата получения читателем, дата возврата книги.
9. База данных “Склад телевизоров”. Учету подлежат: поставщик, производитель телевизора, модель телевизора, дата поставки телевизора, дата списания телевизора со склада, не менее 5 характеристик телевизора (производитель, диагональ, вес, и т.п.), цена телевизора от поставщика, отпускная цена телевизора, количество телевизоров.
10. База данных “Автовокзал.” Учету подлежат: автобусы (марка, дата выпуска и государственный регистрационный номер), водители (ФИО, дата рождения, водительская категория, какой автобус закреплен), расписание движения автобусов (наименование населенного пункта, день и время отправления, день и время прибытия, стоимость билета, автобус выполняющий рейс).
11. База данных “Магазин одежды.” Учету подлежат: товар (название, размер, цена, материал, производитель), дата поступления, дата продажи.
12. База данных “Почтовое отделение” Учету подлежат: поступающая на почту заказная корреспонденция (письма, ценные бандероли и посылки). Дата поступления, дата выдачи, паспортные данные получателя корреспонденции.
13. База данных “Аптечный киоск” Учету подлежат: лекарства (наименование, дозировка, производитель, отпускная цена, место хранения, выдача только по рецепту или нет, показания и противопоказания).
14. База базы данных “Расписание занятий в институте” Учету подлежат: специальность, учебная группа, учебные дисциплины (название и вид занятия), день недели, номера часов проведения занятий, преподаватель, номер аудитории
15. База данных “Гостиница” Учету подлежат: номера гостиницы (номер помещения, число спальных мест, наличие душа, туалета, холодильника, телевизора, сплит-системы, стоимость проживания), занятость номера (дата заселения, паспортные данные жильцов, дата освобождения номера)
16. База данных “Кинотеатр”. Учету подлежат: название кинофильма, жанр кинофильма, возрастные ограничения, дата и время киносеанса, название кинозала, номер посадочного места, цена билета.
17. База данных “Торговый зал”. Учету подлежат: название товара, отпускная цена товара, количество товара в зале, дата выпуска, срок годности. База данных должна предусматривать отпуск товара покупателю со списанием товара из зала.
18. База данных “Учет электроэнергии”. База учитывает количество потребленной электроэнергии в небольшом поселке и ежемесячно подготавливает индивидуальные квитанции об оплате. Учету подлежат: адрес дома, владелец дома, количество потребленной электроэнергии за месяц, стоимость за кВт/ч. Стоимость электроэнергии может изменяться!
19. База данных “Отдел кадров завода”. Учету подлежат: ФИО, рабочая специальность, квалификация (разряд), образование работника, дата трудоустройства, стаж работы, должностной оклад, надбавка (зависит от квалификации и стажа).
20. База данных “Художественный музей” Учету подлежат: автор картины, название картины, краткое описание сюжета, дата написания картины, художественное направление (реализм, абстракция, и т.д.), зал в котором выставлена картина.

Примечания

- ~ Выполненное задание сдается в письменном (распечатанном) виде и заверяется подписью преподавателя.
- ~ При построении ER-модели рекомендуется воспользоваться услугами специализированных редакторов. Например, ER/Studio входящего в поставку Embarcadero RAD Studio или Microsoft Visio.

5. Нормализация данных

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access

Процесс нормализации был впервые предложен Э. Ф. Коддом в 1972 году, он преследует две цели:

1. Исключение избыточности данных.
2. Обеспечение наибольшей гибкости структуры таблиц на случай будущих изменений.

Нормализация осуществляется за счет нескольких последовательных этапов, каждый из которых соответствует некоторой нормальной форме, обладающей известными свойствами. Различают 5 нормальных форм (normal forms), то есть мы можем говорить о 5-ти этапах нормализации. Кроме того, существует промежуточная форма нормализации, получившая название нормальной формы Бойса-Кодда (НФБК), она расположена между 3-й и 4-й нормальными формами.

Первая нормальная форма (1NF)

Первая нормальная форма представляет собой основу реляционной модели данных.

Приведенная к **1NF** таблица не может содержать повторяющихся групп, кроме того, каждое поле таблицы должно быть неделимым.

Что такое **неделимое поле** и как достичь неделимости на практике? Предположим, что мы работаем над базой данных домашней библиотеки (рис. 5.1). Взгляните на ненормализованный вариант таблицы Library. Она состоит из двух полей: Writer – автор книги и Books – название произведения. В этой таблице представлены сразу две ошибки, связанные с неделимостью поля. Наиболее грубая из них – в поле Books. Обратите внимание, что атрибут поля одновременно содержит названия двух и более произведений. Это неверно – для каждого произведения должна быть отведена отдельная строка таблицы. Вторая ошибка связана с полем Writer, где мы храним имя, отчество и фамилию писателя. В таких целях целесообразно сформировать три отдельных поля: имя, отчество и фамилия. Причин тому несколько. Представьте себе, что при вводе информации в колонку Write пользователь вместо «Лев Николаевич Толстой» ввел «Толстой Лев Николаевич». С точки зрения человеческой логики это абсолютно одинаковые значения, но компьютер воспримет их как разные, что сразу отразится на работе программы. Вторая причина – разделив колонку Writer на три (по смыслу уже неделимых) колонки мы значительно упростим задачу по сортировке и выборке данных средствами SQL запросов. Например, теперь мы получим возможность отобрать записи авторов без отчества, и т.п. Заметьте, что приведенная к первой нормальной форме таблица приобрела явную избыточность, но на этом этапе это не страшно.

Ненормализованная таблица "Library"			
Writer		Books	
Лев Николаевич Толстой		Война и Мир Анна Каренина	
Александр Сергеевич Пушкин		Евгений Онегин Сказка о царе Салтане	
Александр Дюма		Три мушкетёра Десять лет спустя Граф Монте-Кристо	



Таблица "Library" приведённая к 1NF			
FName	LName	Name	Books
Лев	Николаевич	Толстой	Война и мир
Лев	Николаевич	Толстой	Анна Каренина
Александр	Сергеевич	Пушкин	Евгений Онегин
Александр	Сергеевич	Пушкин	Сказка о царе Салтане
Александр		Дюма	Три мушкетёра
Александр		Дюма	Десять лет спустя
Александр		Дюма	Граф Монте-Кристо

Рис. 5.1. Создание неделимых полей

Теперь поговорим о повторяющейся группе. **Повторяющаяся группа** представляет собой поле, которое повторяется внутри строки, с целью хранения нескольких значений для атрибута. Допустим, что мы разрабатываем базу данных, предназначенную для хранения данных о температуре воздуха, причем измерения

производятся ежедневно, каждый четный час. На рисунке 5.2 изображена классическая ошибка начинающего программиста. Для хранения даты в таблице «temperature» используется поле TmprDate, а для хранения температуры поле Tmpr. В одном и том же поле заносится несколько значений, разделенных точкой с запятой. Несостоятельность такого подхода при проектировании таблицы не вызывает и сомнения. Представьте себе, как бы выглядела таблица температур, если бы измерения проводились каждые четверть часа, или каждую минуту... Можете поверить, что даже если повторяющихся групп не много, то они все равно затруднят построение запросов и форматирование отчетов, поэтому посмотрим, как от них избавиться. Вернемся к рисунку 5.2. Для того, чтобы преобразовать таблицу температур к 1NF нам потребовалось создать таблицу, состоящую из трех столбцов – даты, времени и температуры. И здесь (как и в случае с приведением к 1NF таблицы авторов) мы получили некоторую избыточность, но зато все логично.

Ненормализованная таблица "temperature"	
TmprDate	Tmpr
01.01.2003	-10.5; -10.5; -11.0; ...
02.01.2003	-12.0; -13.0; ...



"temperature" приведена к 1NF		
TmprDate	TTime	Tmpr
01.01.2003	00:00	-10.5
01.01.2003	02:00	-10.5
01.01.2003	04:00	-11.0
...	...	...
02.01.2003	20:00	-12.0
02.01.2003	22:00	-13.0

Рис. 5.2. Пример устранения повторяющейся группы

Вторая нормальная форма (2NF)

Этап приведения таблицы ко второй нормальной форме может начинаться только после того, как таблица была приведена к 1NF.

Приведенная ко **2NF** таблица представляет собой таблицу, у которой каждое поле, не входящее в состав первичного ключа, характеризуется полной функциональной зависимостью от этого первичного ключа.

Другими словами, у таблицы 2NF все не ключевые поля полностью зависят от первичного ключа или (в случае составного первичного ключа) от каждого поля первичного ключа. Таким образом любое, хранящееся в не ключевом столбце таблицы, значение однозначно определяется значением первичного ключа.

"Library"				
Library_ID	FName	LName	SurName	Books
1	Лев	Николаевич	Толстой	Война и мир
2	Лев	Николаевич	Толстой	Анна Каренина
3	Александр	Сергеевич	Пушкин	Евгений Онегин
4	Александр	Сергеевич	Пушкин	Сказка о царе Салтане
5	Александр		Дюма	Три мушкетёра
6	Александр		Дюма	Десять лет спустя
7	Александр		Дюма	Граф Монте-Кристо

Рис. 5.3. Приведение таблицы к 2NF

Итак, на втором этапе приведения таблицы к реляционному виду необходимо создание первичного ключа. Рассмотрим очередной пример. А для этого предлагаю вернуться к таблице Library. Для преобразования этой таблицы ко второй нормальной форме просто дополняем ее столбцом – Library_ID (рис. 5.3). В качестве первичного ключа разработчики баз данных очень часто используют поля автоинкрементного типа. В этом случае при вставке новой записи, значение ее первичного ключа получает приращение на одну единицу, что гарантирует уникальность этого ключа.

Третья нормальная форма (3NF)

Этап приведения таблицы к третьей нормальной форме может начинаться только после того, как таблица была приведена к 2NF.

Приведенная к **3NF** таблица не имеет избыточных не ключевых полей, связанных с не ключевыми полями других таблиц

Ненадолго возвратимся к рисунку 3.3 (лабораторная работа “Ключи, организация связи между таблицами”). Представленная на рисунке таблица “Students” приведена ко 2NF, но полному соответствию 3NF мешает поле SpCod. В этом столбце содержится данные о названиях специальностей. Это поле не ключевое и содержит повторяющиеся данные. Поэтому оно должно быть вынесено в отдельную таблицу (рис. 3.8). Представленные на этом рисунке таблицы “Students” и “Спец” приведены к 3NF.

Нормальная форма Бойса-Кодда (BCNF)

Технически нормальная форма Бойса-Кодда просто усиливает таблицу, находящуюся в 3NF.

Таблица находится в **BCNF** тогда, когда каждое ее поле (или группа полей) является потенциальным ключом.

У приведенных к 3NF таблиц нарушения требований BCNF крайне редки, они могут произойти только в случаях, когда:

- ~ в таблице имеется более одного составного потенциального ключа;
- ~ имеющиеся потенциальные ключи перекрываются, т.е. ими совместно используется, по крайней мере, один общий атрибут.

Четвертая нормальная форма (4NF)

Четвертая нормальная форма предназначена для борьбы с многозначной зависимостью.

В таблице, приведенной к **4NF** наложен запрет на хранение независимых элементов, когда между этими элементами существует связь типа “многие ко многим”.

Для демонстрации процесса перевода таблицы к 4NF возвратимся, к заключительной части лабораторной работы IV, посвященной ER моделированию. На рисунке 4.7 представлен пример построения связи “многие ко многим” между таблицами компаний и городов.

Пятая нормальная форма (5NF)

Пятая нормальная форма требует обеспечения беспрепятственной возможности перестройки данных в нормализованных таблицах. Приведение таблицы к пятой нормальной форме крайне редкий случай. Это действие имеет смысл, только если таблица содержит **зависимые сочетания**.

Задание

На основе разработанной вами на предыдущем занятии ER-модели базы данных, создайте физическую модель данных. При формировании таблиц добейтесь, чтобы таблицы были нормализованы до четвертой нормальной формы. Реализуйте базу данных в среде Microsoft Access.

6. Язык SQL, оператор SELECT

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – среда Microsoft Access

Язык SQL существенно отличается от языков программирования высокого уровня. Во-первых, SQL не является процедурным языком. На нем нет необходимости шаг за шагом описывать все операции последовательно определяющее решение задания. Задача SQL указать, что нам собственно надо, остальное лежит в сфере ответственности СУБД. Во-вторых, SQL работает в режиме трехзначной логики (3VL), здесь, к привычным для нас значениям TRUE (истина) и FALSE (ложь) добавляется понятие NULL (неопределенность). Более того, в операциях сравнения NULL с чем-то еще возвращаемый результат вполне возможно примет не значения FALSE или TRUE, а UNKNOWN.

В первоначальной минимальной нотации структурированный язык запросов был нацелен на решение трех базовых задач:

1. Создание БД и таблиц с исчерпывающим описанием их структуры.
2. Запрос от БД информации и представление ее в удобном для пользователя виде.
3. Манипуляции с данными – добавление, редактирование и удаление данных.

Несколько позже SQL был обучен:

1. Обработке транзакций.
2. Управлению курсором.
3. Определению прав пользователей.

Исходя из задач, стоящих перед SQL можно классифицировать команды языка по функциональному назначению. В самом общем случае выделяются шесть категорий:

1. **Язык запросов DQL** (Data Query Language) предназначен для извлечения данных из таблиц и в своей основе опирается на инструкцию SELECT.
2. **Язык определения данных DDL** (Data Definition Language) в первую очередь нацелен на решение вопросов создания CREATE и удаления DROP различных объектов – таблиц TABLE, представлений VIEW, индексов INDEX, курсоров CURSOR, определений доменов DOMAIN, и т.д.
3. **Язык манипулирования данными DML** (Data Manipulation Language) осуществляет операции вставки, редактирования и удаления данных и базируется на инструкциях INSERT, UPDATE и DELETE.
4. **Язык управления доступом к данным DCL** (Data Control Language) устанавливает ограничения на права пользователей при работе с объектами базы данных. Характерными представителями инструкций этой категории выступают GRANT и REVOKE.
5. **Язык обработки транзакций TPL** (Transaction Processing Language) включает инструкции BEGIN TRANSACTION, COMMIT и ROLLBACK. Язык позволяет объединять несколько команд управления данными DML в группу, называемую транзакцией. Если хотя бы одна из команд, входящих в транзакцию не будет выполнена, то осуществляется откат и всех других.
6. **Язык управления курсором CCL** (Cursor Control Language) выбирает для обработки одну строку результирующего множества запроса.

Оператор языка SQL включает в себя зарезервированные слова (INSERT, UPDATE, SELECT, и т.д.) и слова определяемые пользователем (названия таблиц, столбцов, триггеров, и т.д.). SQL-операторы не чувствительны к регистру символов, поэтому, при их описании, могут использоваться как строчные, так и прописные буквы. Однако это правило не соблюдается при вводе данных. Например, при поиске всех записей, содержащих значение “ПЕТРОВ” в результирующий набор, будут возвращены строки, содержащие значение “ПЕТРОВ”, а не “Петров”, “петров”, и т.д.

Выборка значений из таблиц – SELECT

Оператор SELECT предназначен для описания запроса на получение информации из базы данных. Запрос, осуществляя обработку всех строк и столбцов таблицы (или нескольких таблиц) отсеивает все не удовлетворяющие критериям запроса данные и обеспечивает группировку или сортировку отобранных данных в заданном порядке. В некотором роде запрос представляет собой что-то вроде фильтра, пропускающего только необходимую на данный момент информацию.

SELECT [DISTINCT|ALL]

```
{ {функция_агрегирования...| выражение для вычисления значения [AS имя_столбца] } ., ... }
| спецификатор.* }
| *
FROM
{имя_таблицы [AS псевдоним], ...}
| {имя_представления [AS псевдоним] }
[WHERE предикат]
GROUP BY { [имя_таблицы | псевдоним] } ., ... [HAVING предикат]
[ORDER BY {столбец_результат [ASC | DESC], ...} ]
```

Простейший запрос должен включать в себя операторы SELECT и FROM, имя таблицы и спецификатор “*”.

```
SELECT * FROM WRITER;
```

Спецификатор “*” говорит о том, что выбираются все столбцы из таблицы Writer. Это крайне нежелательный способ организации запроса, так как выборка всех столбцов значительно загружает сетевой трафик и замедляет выполнение запроса. Поэтому принято явно указывать минимально-необходимый набор столбцов.

```
SELECT Surname, FName, LName FROM WRITER
```

Порядок сортировки – ORDER BY

Порядок сортировки возвращаемой информации определяется при помощи ключевых слов ORDER BY. Нижеприведенный пример осуществит сортировку таблицы Writer по фамилии.

```
SELECT Surname, FName, LName FROM WRITER ORDER BY Surname;
```

В порядке сортировки допускается указать сразу несколько столбцов, разделяя их запятыми. По умолчанию записи сортировки в порядке возрастания. Если надо упорядочить данные по убыванию, то после имени столбца укажите оператор DESC.

```
SELECT * FROM Writer ORDER BY Surname DESC, FName DESC;
```

Вне зависимости от используемых в запросе ключевых слов, фраза ORDER BY всегда должны быть последним элементом в предложении SELECT.

Ограничение набора данных – WHERE

Для того, чтобы ограничить набор выводимых данных необходимо воспользоваться предложением WHERE. Различают пять типов ограничений:

1. **Сравнение.** Осуществляется сравнение результатов вычисления одного выражения с другим.
2. **Попадание в диапазон.** Проверяется – попадает ли результат вычисления выражения в определенный диапазон значений.
3. **Соответствие шаблону.** Проверяется – соответствует ли некоторое строковое значение определенному шаблону.
4. **Неопределенное значение NULL.** Содержит ли поле неопределенное значение NULL.
5. **Принадлежность множеству значений.** Входит ли результат в множество значений.

Сравнения осуществляется с помощью операторов: =, <, >, <=, >=. Результат сравнения может принимать значения: True, False и Unknown.

Сравнение

Инструкция

```
SELECT * FROM Surname WHERE bday>'01/01/1900';
```

вернет всех авторов, дата рождения которых больше 1-го января 1900 года.

Другой вариант более сложного запроса возвратит всех авторов с фамилией «Толстой» и именем «Лев».

```
SELECT Surname, FName, LName, bday FROM Writer
WHERE Surname='Толстой' AND FName='Лев';
```

Попадание в диапазон – предикат BETWEEN

Предикат BETWEEN осуществляет контроль за тем, чтобы Значение1 входило в диапазон от Значения2 до Значения3.

Значение1 **BETWEEN** Значение2 **AND** Значение3

или

Значение1 **NOT BETWEEN** Значение2 **AND** Значение3

Например, запрос

```
SELECT * FROM Surname WHERE BDay BETWEEN '01/01/1800' AND '01/01/1900';
```

вернет всех авторов, родившихся в 19 веке.

Соответствие шаблону – предикат подобия LIKE

Предикат LIKE определяет условия поиска с указанием шаблонов.

LIKE::= шаблон подстроки

[**NOT**] **LIKE** шаблон подстроки

[**ESCAPE** служебный символ подстановки]

Поиск указанной подстроки в строке. Например, для выбора всех авторов, в фамилии которых есть символ «о» мы подготовим такой запрос:

```
SELECT Surname, FName FROM Writer WHERE Surname LIKE '%о%';
```

Запрос вернет всех авторов, в фамилии которых есть такой символ. В шаблоне подстроки помимо набора интересующих нас символов допускается использовать два трафаретных символа:

1. Символ подчеркивания «_», он используется вместо любого одиночного символа в выражении.
2. Символ процента «%», который заменяет последовательность любых символов.

Если столбец входит в состав индекса, то для ускорения процесса поиска записей желательно, чтобы шаблон подстроки не начинался с трафаретных символов.

Например, для отбора всех авторов, у которых во второй позиции фамилии есть символ «о» и фамилия заканчивается на «ой» та выражение должно выглядеть так: '_о%ой'. В случае, когда искомая строка должна включать служебный символ, используемый в качестве символа подстановки, то перед указанием служебного символа ставится ключевое слово **ESCAPE**.

```
SELECT * FROM Writer WHERE Surname LIKE 'П#%' ESCAPE '#';
```

Проверка неопределенного значения – предикат NULL

Предикат NULL разработан для определения наличия значения.

NULL ::=конструктор значений строки **IS** [**NOT**] **NULL**

Следующий пример вернет все записи, у которых не заполнен столбец BDAY.

```
SELECT Surname,FName FROM Writer WHERE BDAY IS NULL;
```

Если вы наоборот хотите выбрать не пустые строки, то запрос должен выглядеть так

```
SELECT Surname,FName FROM Writer WHERE BDAY IS NOT NULL;
```

Принадлежность множеству – предикат подзапроса IN

Предикат подзапроса IN производит проверку – входит ли результат вычисления в заданное множество. Множество может описываться как поэлементно, так и быть результатом другого, вложенного подзапроса.

IN::=значения строки [**NOT**] **IN** табличный подзапрос|элементы множества

Например,

```
SELECT * FROM Writer WHERE Surname IN ('ПУШКИН', 'ДОСТОЕВСКИЙ');
```

Подзапрос называют вложенный в основной запрос, внутри оператора WHERE оператор еще один оператор SELECT. Например, перед нами стоит задача – выяснить автора, которому в таблице Book не сопоставлено ни одной записи:

```
SELECT * FROM Writer WHERE Wrt_Key NOT IN (SELECT Wrt_key FROM Book);
```

Обратная задача решается при удалении ключевого слова NOT:

```
SELECT * FROM Writer WHERE Wrt_Key IN (SELECT Wrt_key FROM Book WHERE Book_key<100);
```

Предикат IN определяет, будут ли значения строки обнаружены в наборе значений, который либо определен, либо получен по результатам табличного подзапроса.

Предикат существования EXISTS

Задача предиката проверить факт существования записей, удовлетворяющих определенному критерию. Предикат может использоваться только совместно с подзапросами.

EXISTS ::= EXISTS подзапрос

Результатом выполнения предиката будет булево значение True или False. Логическая истина возвращается в том случае, если в возвращаемом подзапросом наборе данных присутствует хотя бы одна запись. Ключевое слово NOT EXISTS использует обратные правила обработки.

Основное достоинство предиката существования – высокая скорость выполнения подзапроса, и если вы хотите распознать запрос, основанный на существовании каких-то данных, то не стоит писать следующее:

```
IF (SELECT COUNT(*) FROM Book WHERE условие)>0 ...
```

Такой код, прежде чем оценить условие IF будет вынужден прочитать **все** записи в таблице, и чем больше записей в таблице Book, тем дольше он будет выполняться. Напротив, использование EXISTS практически моментально вернет TRUE, как только подзапрос столкнется с первой же строкой, удовлетворяющей условию:

```
IF EXISTS (SELECT * FROM Book WHERE условие) ...
```

Предикат существования EXISTS никогда не принимает значения UNKNOWN.

Агрегатные функции

Агрегатные (обобщающие) функции в качестве исходных параметров принимают значения, указанные в запросе (после слова SELECT) и вычисляют результат. Как правило, эти функции группируются с помощью предложения GROUP BY.

Таблица 6.1. Агрегатные функции

Функция	Описание
COUNT	Возвращает количество строк. Тип возвращаемого значения – INTEGER. SELECT COUNT(*) FROM BOOK;
AVG	вычисляет среднее арифметическое для указанных элементов. Используется только для полей цифровых типов данных. Тип возвращаемого значения FLOAT. SELECT AVG(имя_столбца) FROM имя_таблицы;
SUM	вычисляет сумму значений. Применяется только для цифровых типов данных. SELECT SUM(имя_столбца) FROM имя_таблицы; Будьте внимательны. Если в результате вычисления суммы итоговое число превысит максимальное значение используемого типа данных, то возникнет ошибка.
MAX	Возвращает наибольшее из всех значений. SELECT MAX(имя_столбца) FROM имя_таблицы;
MIN	Возвращает наименьшее из всех значений. SELECT MIN(имя_столбца) FROM имя_таблицы;

Все агрегатные функции работают с единственным полем таблицы и возвращают единственное значение. Функции COUNT, MIN и MAX применимы, как к числовым, так и к нечисловым полям. Функции SUM и AVG умеют работать только с числовыми полями.

Группировка данных – Group By

Предложение предназначено для осуществления группировки выходных строк. Группировка осуществляется по равенству значений каких-либо столбцов.

```
SELECT Writer.Surname, COUNT(Book.Book)
FROM Book
RIGHT JOIN Writer ON Writer.Wrt_Key=Book.Wrt_Key
GROUP BY Surname
```

Приведенный выше запрос объединяет две таблицы (авторов и книг), группирует данные по фамилии автора и возвращает количество произведений каждого из них.

Все имена полей, приведенные в списке группирующего запроса SELECT должны присутствовать и во фразе GROUP BY. Единственное исключение делается для полей обрабатываемых агрегатной функцией.

В результате выполнения группирующего запроса для каждой отдельной группы создается одна-единственная группирующая строка.

Дополнительная фильтрация группы строк – Having

Ключевое слово HAVING используется после группировки (предложения GROUP BY) и предназначено для дополнительной фильтрации уже сформированных групп строк. Поведение элемента HAVING подобно работе ключевого слова WHERE, но он работает не со всеми строками таблиц, а со строками результирующего набора. Фильтрация может осуществляться с помощью агрегирующих функций (COUNT, SUM, AVG, MAX и MIN). Для примера, несколько модифицируем текст предыдущего запроса и научим его выводить информацию об авторах, для которых зарегистрировано более 2 наименований книг.

```
SELECT Writer.Surname, COUNT(Book.Book)
FROM Book
RIGHT JOIN Writer ON Writer.Wrt_Key=Book.Wrt_Key
GROUP BY Surname
HAVING COUNT(Book.Book)>2;
```

Псевдонимы столбцов

Практически все диалекты SQL поддерживают псевдонимы столбцов. Как правило, переименование столбца требуется в случаях использования агрегатных функций или в случаях, когда в объединяемых таблицах есть поля с идентичными именами. Синтаксис задания псевдонимов столбцов различается от диалекта к диалекту. Например:

1) Microsoft SQL Server:

```
SELECT Writer.Surname, COUNT(Book.Book) AS BookCount ...
```

2) СУБД SYBASE:

```
SELECT Writer.Surname, BookCount=COUNT(Book.Book) ...
```

3) Классический ANSI SQL:

```
SELECT Writer.Surname, COUNT(Book.Book) BookCount ...
```

Во всех примерах мы задаем псевдоним BookCount для результата работы функции COUNT.

Псевдонимы таблиц

Вместо того, чтобы в запросе SQL указывать полное имя таблицы при каждой ссылке на нее в команде SELECT можно определить для нее сокращенное название и далее пользоваться этим псевдонимом.

```
SELECT W.Surname, B.Book
FROM Book B, Writer W
WHERE W.Wrt_Key=B.Wrt_Key;
```

Псевдоним указывается в конструкции FROM сразу за полным именем таблицы. В приведенном выше примере для таблицы Book назначен псевдоним B, а для Writer – W.

Объединение нескольких таблиц

вывод данных из одной таблицы представляет собой частный случай. Значительно чаще возникает задача объединения данных из различных таблиц. Для этого используют ряд синтаксических структур, позволяющих соединять таблицы. Простейшим способом объединения двух и более таблиц является использование предиката WHERE с определением полей, применяемых для соединения таблиц. Кроме того, для выполнения объединения после ключевого слова FROM необходимо через запятую перечислить имена объединяемых таблиц. Например, мы хотим соединить таблицы писателей (Writer) и таблицу книг (Book). Данные таблицы объединяются полями Wrt_Key обеих таблиц.

```
SELECT Surname, FName, LName, Book FROM Writer, Book
WHERE Book.Wrt_Key=Writer.Wrt_Key;
```

Сразу разберемся в терминологии. При объединении таблиц зачастую оперируют терминами внутренняя и внешняя таблица. Таблица, описанная слева от знака равенства, называется внешней, справа – внутренней. Кроме того, эти таблицы еще называют левой и правой. Тип представленного ранее объединения называют внутренним. Это один из наиболее распространенных способов объединения в реляционных базах данных.

Большее распространение для организации объединения двух и более таблиц получил синтаксис ANSI с использованием оператора JOIN. При этом различают пять типов соединений: внутренне, левое, правое, полное и перекрестное.

Таблица 6.2. Основные типы соединений

Тип соединения	Синтаксис
CROSS Перекрестное	таблица1 CROSS JOIN таблица2 Это прямое декартово произведение, обеспечивающее все возможные объединения строк из каждой таблиц
NATURAL Естественное	таблица1 [NATURAL] [тип соединения] JOIN таблица2 Соединение внешнего ключа со связанным с ним родительским ключом.
UNION Объединение	таблица1 UNION JOIN таблица2 Соединение, в которое входят те строки каждой из таблиц, для которых совпадения не были установлены.
ON Объединение посредством предиката	таблица1 [тип соединения] JOIN таблица2 ON предикат где тип соединения {INNER LEFT RIGHT FULL OUTER}
{INNER LEFT RIGHT FULL OUTER} USING Объединение посредством имен столбцов	таблица1 [тип соединения] JOIN таблица2 USING (имя столбца ., ...) где тип соединения {INNER LEFT RIGHT FULL OUTER}

Аналог объединения WHERE на синтаксисе ANSI обязательно содержит оператор JOIN и выглядит примерно так:

```
SELECT Writer.Surname, Writer.FName, Writer.LName, Book.Book
FROM Writer
INNER JOIN Book ON Writer.Wrt_Key=Book.Wrt_Key;
```

Такое объединение называют **внутренним**.

Коренное отличие применения оператора JOIN от оператора WHERE в операциях объединения таблиц заключается в том, что после слова FROM нет необходимости указывать перечень всех таблиц.

Для того, чтобы осуществить **левое** объединение таблиц (другими словами заставить вывести все записи из левой таблицы вне зависимости от того есть ли у них подчиненные записи в правой таблице) конструкция незначительно изменится:

```
SELECT Writer.Surname, Writer.FName, Writer.LName, Book.Book
```

```
FROM Writer
LEFT JOIN Book ON Writer.Wrt_Key=Book.Wrt_Key;
```

Для создания **правого** объединения замените в предыдущем примере ключевое слово LEFT на RIGHT.

```
SELECT Writer.Surname, Writer.FName, Writer.LName, Book.Book
FROM Writer
RIGHT JOIN Book ON Writer.Wrt_Key=Book.Wrt_Key;
```

Теперь запрос возвратит все записи из правой таблицы (Book) и соответствующие ей записи левой таблицы. Так как в правой таблице не указаны книги Достоевского, то в результирующей таблице опять вы не увидите записей с его фамилией.

Полное объединение требует использования ключевого слова FULL.

```
SELECT Writer.Surname, Writer.FName, Writer.LName, Book.Book
FROM Writer
FULL JOIN Book ON Writer.Wrt_Key=Book.Wrt_Key;
```

Такое объединение позволит вернуть все записи как из левой, так и из правой таблиц. Если бы, например, в таблице книг оказалась еще одна запись о книге без автора (другими словами ключ автора Wrt_Key был установлен в Null).

Редко используемый тип **перекрестного** объединения CROSS заставит запрос вернуть все возможные комбинации строк из объединяемых таблиц.

```
SELECT Writer.Surname, Writer.FName, Writer.LName, Book.Book
FROM Writer
CROSS JOIN Book;
```

Ценность, сформированной таким образом, выходной таблицы весьма условна, но, тем не менее, если такая возможность существует, то о ней надо помнить.

Далее приведен пример запроса SQL формирующий единую результирующую таблицу из четырех.

```
SELECT Writer.Surname, Writer.FName, Writer.LName, Writer.BDay,
       Publisher.Publisher, Book.Book
FROM Base
LEFT JOIN Publisher ON Publisher.Pbl_Key=Base.Pbl_Key
LEFT JOIN Book ON Book.Bk_Key=Base.Bk_Key
RIGHT JOIN Writer ON Writer.Wrt_Key=Book.Wrt_Key;
```

Порядок соединения нескольких таблиц имеет критическое значение. Если таблицы соединяются в правильном порядке, то общее число обрабатываемых строк будет меньше. Описывая запрос, следует выполнять сначала максимально ограничивающий поиск, чтобы отфильтровать как можно большее число строк на ранних фазах выполнения запроса с соединениями.

Задание

С помощью оператора SELECT разработайте группу запросов к своей базе данных. В результате работы у вас должны быть созданы запросы позволяющие:

1. сортировать данные во всех отдельных таблицах БД;
2. отбирать данные из отдельных таблиц;
3. объединяющие все пары таблиц, состоящие в отношении “один ко многим”;
4. объединяющие все таблицы, состоящие в отношении “многие ко многим”;
5. суммарный запрос, отображающий все данные из всех таблиц БД.

Защита лабораторной работы осуществляется путем выполнения, полученного у преподавателя индивидуального задания по реализации инструкции SELECT.

7. Язык SQL, операторы манипулирования данными

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access

На занятии мы изучим одно из подмножеств SQL – язык манипулирования данными DML (Data Manipulation Language). DML осуществляет операции вставки, редактирования и удаления данных и базируется на инструкциях INSERT, UPDATE и DELETE.

Вставка новой записи – INSERT

Назначение оператора – вставка в таблицу одной и более строк.

```
INSERT INTO имя_таблицы  
[ (имя столбца., ...) ]  
VALUES  
выражение запроса | конструктор значений таблицы | {DEFAULT VALUES}
```

Имя пополняемой таблицы указывается сразу после ключевых слов INSERT INTO. С помощью данного оператора также допускается вставка новой записи и в представление, при условии, что представление состоит из одной таблицы. Сразу за именем таблицы следует перечень столбцов, в которые мы собираемся внести новую информацию. Если список столбцов окажется неполным (в нем не будет явным образом указан весь их перечень), то в оставшиеся столбцы вносится значение по умолчанию. Если, в свою очередь значение по умолчанию не было определено заранее (при создании таблицы) и на столбец наложено ограничение NOT NULL, то работа оператора будет прервана. Еще одним поводом для приостановки операции вставки новой записи будет попытка вставить новое значение в столбец, который самостоятельно генерирует новое значение (например, в столбцы автоинкрементного типа).

```
INSERT INTO Writer  
(Surname, FName, LName, BDay)  
VALUES  
( 'Пушкин', 'Александр', 'Сергеевич', '26/05/1799' );
```

В команде на добавление новой строки перечисление столбцов не является обязательным, если вы осуществляете вставку данных во все столбцы. Но в этом случае последовательность передаваемых значений должна четко соответствовать физическому порядку столбцов таблицы.

Редактирование данных – UPDATE

Оператор осуществляет редактирование существующей записи.

```
UPDATE имя_таблицы  
SET {имя_столбца={выражение | значение | NULL | DEFAULT}}.,...  
[WHERE предикат]
```

Минимальный код позволяющий исправить **все** записи в таблице выглядит следующим образом:

```
UPDATE Writer SET LName='Кукушкин';
```

Неоспоримое удобство оператора UPDATE заключается в возможности практически мгновенного изменения данных во всей таблице. Например, требуется повысить зарплату всем сотрудникам предприятия в 2 раза.

```
UPDATE Sales SET PplMoney= PplMoney*2;
```

Еще раз отметим, что приведенные примеры заставят СУБД внести изменения во все записи таблицы, поэтому старайтесь ограничивать набор редактируемых записей с помощью оператора WHERE.

```
UPDATE Writer SET BDay= '26/05/1799'  
WHERE Surname= 'Пушкин' AND FName= 'Александр';
```

Обратите внимание, что в данном случае синтаксис WHERE на сто процентов соответствует синтаксису WHERE в операторе SELECT. Вместе с тем есть и второй тип применения оператора WHERE – WHERE CURRENT OF. Такая инструкция используется при работе с курсором. В таком случае будет отредактирована запись, на которой позиционирован курсор.

Во многих СУБД “редактирование” данных осуществляется следующим образом: строка со старой информацией помечается как удаленная, а в таблицу вставляется новая строка, с обновленными данными. А для того, чтобы не нарушить целостность данных в эту запись переносится старое значение первичного ключа.

Удаление записей – DELETE

Оператор DELETE предназначен для удаления строк из таблиц данных.

DELETE FROM имя_таблицы [{**WHERE** предикат}]

Минимальный синтаксис включает ключевые слова DELETE FROM и имя таблицы.

DELETE FROM Book

Такая команда **полностью очистит таблицу**. Для того, чтобы осуществить адресное удаление строк, например, только конкретного автора необходимо после ключевого слова WHERE описать условие удаления.

DELETE FROM Book **WHERE** Surname= 'Толстой' **AND** FName= 'Лев';

Задание

Применительно к созданной вами БД разработайте запросы позволяющие:

1. очищать таблицы;
2. заполнять таблицы данными;
3. редактировать определенную строку в таблице;
4. удалять определенную строку в таблице.

Защита лабораторной работы осуществляется путем выполнения, полученного у преподавателя индивидуального задания по реализации инструкций INSERT, UPDATE и DELETE.

8. Язык SQL, операторы определения данных

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access

Язык определения данных DDL (Data Definition Language) в нацелен на решение вопросов создания CREATE и удаления DROP различных объектов базы данных – таблиц TABLE, представлений VIEW, индексов INDEX, курсоров CURSOR, определений доменов DOMAIN, и т.д.

Основные типы данных SQL-92

Каждое поле в реляционной таблице базы данных SQL характеризуется определенным типом данных. Основные типы данных практически всех СУБД эквивалентны друг другу. Указание типа данных в SQL понадобится в первую очередь при создании таблиц. С достаточной степенью условности можно говорить о существовании семи основных типов данных SQL (по крайней мере, такое разделение предусматривает стандарт):

1. CHARACTER STRING (текст);
2. NATIONAL CHARACTER (символы национальной кодировки);
3. BIT STRING (двоичные данные);
4. APPROXIMATE NUMERIC (число, имеющее приближенное значение);
5. EXACT NUMERIC (точное число);
6. DATETIME (дата и время);
7. INTERVAL (период времени).

Первые два типа данных представляют собой обычный текст, с той лишь разницей, что NATIONAL CHARACTER способен работать с национальным набором символов.

Таблица 8.1. Текстовые данные

Спецификация	Описание
CHARACTER [n] или CHAR [n]	Тип данных предназначен для создания текстовой строки фиксированной длины. Количество символов в строке определяется в квадратных скобках после указания типа данных – <i>n</i> . Если в поле типа CHAR помещается текстовое значение меньшего размера, то оставшиеся позиции символов забиваются пробелами.
CHARACTER VARYING [n] или VARCHAR [n]	Текстовая строка переменной длины. Максимальный размер строки определяется в квадратных скобках. Преимущество такого типа данных над типом CHAR заключается в том, что здесь пустые позиции не заполняются пробелами и соответственно таблица требует меньшего размера памяти.
NATIONAL CHARACTER [n] или NCHAR [n]	Строка специального (национального) символьного набора фиксированной длины.
NATIONAL CHARACTER VARYING [n] или NCHAR VARYING [n]	Строка национального символьного набора переменной длины. Национальные типы данных описывают символы в системе UNICODE, в этом случае для хранения одного символа требуется два байта.

Тип данных **двоичная (битовая) последовательность** предназначен для хранения последовательностей двоичной информации. Такой тип данных часто называют Binary Large Objects (BLOB). Тип данных достаточно универсален и позволяет хранить как простейшие логические данные, так и огромные изображения, звуковые файлы и т.д.

Таблица 8.2. Двоичные последовательности

Спецификация	Описание
BIT [n]	Битовая последовательность фиксированной длины. Аргумент <i>n</i> устанавливает длину последовательности в битах. Если аргумент отсутствует (установлен в 1), то тип данных используется для создания полей логического типа (Да/Нет).

BIT VARYING [n]	Битовая последовательность переменной длины. Максимальное значение битовой последовательности указывается в аргументе <i>n</i> .
-----------------	--

На базе битовых последовательностей в различных СУБД реализованы свои собственные типы данных. Как уже упоминались ранее, эти типы данных умеют хранить большие объемы форматированного и неформатированного текста, изображения, OLE объекты, файлы мультимедиа и др.

Тип данных **округленные числа** предназначен для определения чисел с плавающей точкой, осуществляющий хранение числа в научном формате (мантисса плюс порядок) и имеет аргумент точность(*n*), но в отличие от типа `numeric` не имеет масштаба. По этой причине `APPROXIMATE NUMERIC` склонен к ошибке округления и предназначен для хранения значений не требующих высокой точности.

Таблица 8.3. Округленные числа

Спецификация	Описание
REAL	Точность и предел значений определяется автоматически СУБД. Как правило, занимает в памяти 6 байт и может хранить число в интервале от -3,4E-38 до +3,4E+38 с точностью до 7 цифр после запятой
FLOAT [(n)]	Параметром <i>n</i> определяется (но не обязательно) минимальное значение точности. Как правило, занимает в памяти 8 байт (если значение <i>n</i> не указано). Тип данных способен хранить число в интервале от 1,7E-308 по +1,7E-308 с точностью до 15 знаков
DOUBLE PRECISION	Точность определяется автоматически СУБД, но должна превышать точность REAL

Тип данных **точные числа** предназначен для хранения чисел, точно соответствующих их цифровой записи. При задании такого типа данных необходимо указать два аргумента: «точность» (*n*) и «масштаб» (*m*). Точность задает общее число значащих цифр, используемых при отображении числа. Масштаб определяет число значащих цифр справа от десятичной точки. Безусловно должно соблюдаться условие: точность больше масштаба ($n > m$). Аргумент «масштаб» не является обязательным и если его не указывать, то считается равным 0.

Таблица 8.4. Точные числа

Спецификация	Описание
NUMERIC [(n[,m])]	Точное число, описываемое аргументами <i>n</i> и <i>m</i> .
DECIMAL [(n[,m])] или DEC [(n[,m])]	В отличие от <code>NUMERIC</code> реально хранит число с большей точностью, чем определено в аргументе <i>m</i> . Другими словами говорят, что <code>NUMERIC</code> задает реальное значение точности, а <code>DECIMAL</code> – минимальное значение точности.
INTEGER или INT	Тип данных, предназначен для хранения целых чисел. Это частный случай типа данных <code>NUMERIC</code> , у которого масштаб установлен в 0, а точность определена возможностями СУБД. Как правило, занимает в памяти 4 байта.
SMALLINT	Тип данных предназначен для хранения малых целых чисел. Также, как и <code>INTEGER</code> это частный случай типа данных <code>NUMERIC</code> . Как правило, занимает в памяти 2 байта.

Количество байт, необходимых для хранения `NUMERIC` и `DECIMAL` в памяти компьютера, состоит в прямой зависимости от размерности параметра *n*. Например, если вы хотите хранить число из 15-16 значащих цифр потребуется 8 байт, 34-36 цифр – 16 байт.

Практически во всех СУБД для работы с денежными величинами на базе `NUMERIC` реализован свой собственный специализированный тип данных. Такие типы данных хранят действительные числа с точностью до 4-го знака после запятой.

По сути, тип данных **дата и время** (`DateTime`) представляет собой структуру с отдельными полями для хранения даты и времени.

Таблица 8.5. Дата и время

Спецификация	Описание
DATE	Тип данных включает три поля: YEAR (год) – от 0001 до 9999 MONTH (месяц) – от 01 до 12

TIME [(n)]	DAY (день) – от 01 до 31 Формат записи: “уууу–мм–дд”. Полное число позиций (вместе с разделителями) 10. Тип данных включает три поля: HOUR (год), MINUTE (минута), SECOND (секунда). Если не указан аргумент точности (n), то полное число позиций (вместе с разделителями) равно 8, а формат записи: “hh:mm:ss”. Если вы укажете аргумент точности, то получите возможность работать с долями секунд.
TIMESTAMP [(n)]	Комбинация типов данных DATE и TIME.

Типы данных TIME и TIMESTAMP обладают еще одним аргументом – WITH TIME ZONE. Аргумент предназначен для определения зонального времени.

Тип данных **интервал** предназначен для задания интервала времени между двумя значениями времени. Основное назначение Interval – осуществление приращения (уменьшения) переменной типа DateTime. Структура типа данных Interval состоит из таких же полей, что и DateTime и, кроме того, обладает знаком, для того чтобы указать направление смещения времени.

Язык определения данных – DDL

Язык определения данных решает задачи создания и удаления объектов базы данных. По стандартам SQL-92 к таким объектам относятся:

- ~ базы данных (схемы);
- ~ таблицы;
- ~ индексы;
- ~ представления;
- ~ курсоры.

Каждый объект в базе однозначно описывается его именем и имеет владельца. Подавляющее число операторов DDL начинается с ключевых слов CREATE (создать) или DROP (удалить).

Работа с базой данных

Существующий на настоящее время стандарт не обязывает разработчиков СУБД создавать базы данных каким-то определенным образом. Синтаксис создания баз данных варьируется от поставщика к поставщику. Так в синтаксисе сервера InterBase операция создания новой базы данных выглядит примерно следующим образом:

```
CREATE DATABASE 'C:\Data\IB\NewBase'
LENGTH=100 USER имя_пользователя PASSWORD пароль;
```

Microsoft SQL Server для создания новой базы данных «по минимуму» потребует следующих строк кода:

```
CREATE DATABASE NewBase
ON (NAME = NewBase,FILENAME = 'c:\mssql7\data\NewBase.mdf');
```

В СУБД ORACLE база данных формируется во время инсталляции системы, а пользовательские объекты размещаются в этой БД. В состав СУБД INGRES включена утилита CREATEDB с помощью которой создается новая база данных.

Простейший способ удаления базы данных сводится к вызову операторов уничтожения базы и ее имени.

```
DROP DATABASE имя_базы_данных [, ...n]
```

Операцию CREATE SCHEMA можно считать альтернативой CREATE DATABASE. Схема представляет собой комбинацию связанных друг с другом объектов базы данных – таблиц, представлений, доменов, и т.п. В минимальной нотации оператор определения схемы выглядит следующим образом:

```
CREATE SCHEMA [имя_схемы | AUTHORIZATION имя_создателя]
```

Синтаксис удаления схемы выглядит следующим образом:

```
DROP SCHEMA имя_схемы [RESTRICT | CASCADE]
```

Здесь ключевое слово **RESTRICT** (вызывается по умолчанию) указывает на то, что перед удалением схема должна быть пустой, иначе удаление будет отменено. Для уничтожения схемы содержащей объекты нам потребуется другой ключ – **CASCADE**.

Работа с таблицами

После создания базы данных можно переходить к созданию таблиц. Ниже представлен общий формат оператора **CREATE TABLE**:

```
CREATE [ {GLOBAL | LOCAL} TEMPORARY] TABLE имя_таблицы
({определение поля | [ограничение таблицы]}., ...
[ON COMMIT {DELETE | PRESERVE} ROWS]) :
определение поля ::=
имя_поля {имя домена | тип данных [размер]}
[ограничение поля...]
[DEFAULT значение по умолчанию]
[COLLATE имя сравнения]
```

Минимальная конструкция, нацеленная на создание простейшей таблицы должна включать операторы **CREATE TABLE**, имя создаваемой таблицы и перечень полей с их размерами и типами. В следующем примере создается таблица **Peoples**, включающая в пять полей. Первое поле целочисленное, второе, третье и четвертое поля символьные и последнее поле предназначено для хранения даты.

```
CREATE TABLE Peoples
(Peoples_ID Int,
Surname Char(30),
FName Char(30),
LName Char(30),
BDay Date);
```

В таблице может быть одно или несколько определений поля. Определение поля обязательно должно включать имя поля, тип данных и при необходимости размер поля. В различных диалектах SQL могут различаться названия типов полей и формат хранения данных (например, даты и времени). По умолчанию столбцы таблицы допускают помещение в них значения **NULL**.

Таблица 8.6. Ограничения поля и таблицы

Ключ	Описание
NOT NULL	Запрет на вставку в поле значения NULL .
UNIQUE	Значение поля должно быть уникальным.
PRIMARY KEY	Значение поля должно быть уникальным, оно не может содержать NULL , в таблице это ограничение может использоваться только один раз.
CHECK	Ограничение-проверка на допустимое значение поля. В скобках за оператором CHECK указывается предикат, проверяющий допустимость значения.
FOREIGN KEY и REFERENCES	Оба ограничения весьма похожи, единственное различие между ними в том, что FOREIGN KEY – ограничение внешнего ключа для таблицы, а REFERENCES – ограничение внешнего ключа для поля. В случае установки ограничения для таблицы сразу за FOREIGN KEY в скобках необходимо указать перечень столбцов, относящихся к ключу. В остальном синтаксис одинаков. Ограничения внешнего ключа требуют, чтобы все значения, присутствующие во внешнем ключе, соответствовали значениям родительского ключа, т.е. внешний ключ должен иметь ссылочную целостность.

Приведенный ниже пример создает на сервере базы данных таблицу **DInfo**. В качестве первичного ключа назначается поле **DKey**, символьное поле **DName** должно быть не пустым. Поле **IzmKey** допускает только ввод значений, существующих в поле **IzmKey** другой таблицы **Izmer**.

```
CREATE TABLE DInfo
(DKey INT PRIMARY KEY,
DName CHAR(40) NOT NULL,
IzmKey INT REFERENCES Izmer(IzmKey));
```

Установка значения по умолчанию производится при помощи оператора **DEFAULT**. Таким образом, обеспечивается автоматический ввод в столбец некоторого значения, если его явным образом не укажет пользователь в момент вставки новых данных.

```
CREATE TABLE tmp
(IntValue INT DEFAULT 100,
 FloatValue FLOAT DEFAULT 0.5,
 CharValue CHAR(10) DEFAULT 'A');
```

Временные таблицы существуют только на время текущего сеанса SQL и по окончании его автоматически удаляются. При создании временной таблицы требуется указать параметр `TEMPORARY`. В зависимости от того создаем мы глобальные временные таблицы или локальные временные таблицы можно (но не обязательно) использовать ключевые слова `GLOBAL` или `LOCAL` соответственно.

```
CREATE GLOBAL TEMPORARY TABLE TmpTable
(IntValue INT, FloatValue FLOAT);
```

Оператор `COLLATE` предназначен для явного указания порядка сортировки в таблице. Дело в том, что наборы символов, используемые в языке SQL, обычно отсортированы по коду каждого символа, т.е. набор строчных символов следуют после прописных символов (как, например, в наборе символов ASCII). Если Вас такая ситуация не устраивает, то вы должны самостоятельно создать объект сортировки (`CREATE COLLATION`) в базе данных и сослаться на него в таблице.

Оператор `DROP TABLE` предназначен для удаления таблиц.

```
DROP TABLE имя_таблицы CASCADE | RESTRICT
```

Если определен параметр `RESTRICT`, то на удаляемую таблицу не должно быть ни каких ссылок в ограничениях или просмотрах (представлениях). Таким образом СУБД обеспечивает целостность данных и таблица не будет удалена если она связана с другими объектами базы данных. Параметр `CASCADE` осуществляет каскадное удаление всех ссылающихся на таблицу объектов (таблиц, ограничений, просмотров).

Минимальный набор команд не требует использования параметра `CASCADE | RESTRICT`.

```
DROP TABLE tmp;
```

Предложение `ALTER TABLE` вносит изменения в определение уже существующей таблицы.

```
ALTER TABLE имя_таблицы
{ADD [COLUMN] определение столбца}
| {ALTER [COLUMN] имя изменяющегося столбца}
| {DROP [COLUMN] имя_столбца RESTRICT | CASCADE}
| {ADD определение ограничения для таблицы}
| {DROP CONSTRAINT имя ограничения RESTRICT | CASCADE}
```

Таблица 8.7. Изменение структуры таблицы

Оператор	Описание
ADD [COLUMN]	Добавляет в таблицу новый столбец. Новый столбец определяется так же, как в операторе <code>CREATE TABLE</code> .
ALTER [COLUMN]	Используется для создания или отмены значения по умолчанию для столбца.
DROP [COLUMN]	Удаляет из таблицы столбец. При использовании параметра <code>RESTRICT</code> перед удалением столбца СУБД проверит наличие ссылок на него из других таблиц, представлений и т.п. и если такие ссылки есть, то столбец не будет удален. Наоборот, при использовании параметра <code>CASCADE</code> вместе со столбцом будут удалены все объекты, ссылающиеся на него.
ADD	Предложение <code>ADD</code> будет добавлять к таблице новое ограничение.
DROP CONSTRAINT	Удаляет уже существующие ограничения. Если в предложении задан параметр <code>RESTRICT</code> , то в этот момент столбец не должен использоваться как родительский ключ для внешнего ключа другой таблицы. Если передается параметр <code>CASCADE</code> , то внешние ключи, имеющие ссылки или ограничения <code>FOREIGN KEY</code> будут уничтожены.

Для того, чтобы добавить к таблице книг еще один столбец, способный хранить целое число воспользуйтесь следующим примером:

```
ALTER TABLE Book ADD COLUMN BookPages INT;
```

Для изменения типа уже существующего столбца:

```
ALTER TABLE Book ALTER COLUMN BookPages FLOAT;
```

И, наконец, для удаления столбца:

```
ALTER TABLE Book DROP COLUMN BookPages;
```

Работа с представлениями

Представление – это хранимое определение набора инструкций SELECT, задающего таблицы и способ их объединения, столбцы и накладывающего ограничения на выводимые строки которые нужно выбрать при вызове данного представления.

Если объяснить проще, то представление — это удачный способ отображения информации, хранимой в СУБД. Причем настолько удачный, что пользователям может казаться, что они работают с реальными таблицами. Представление используется для организации доступа к одной таблице, части таблицы или нескольким таблицам. К представлению, как и к обычной таблице, можно обратиться, используя ключевое слово SELECT.

```
CREATE VIEW имя_таблицы [(список столбцов)]
AS (SELECT оператор
[WITH [CASCADED | LOCAL] CHECK OPTION])
```

Ряд примеров создания представлений на основе одной таблицы предложен ниже. Простейшее однотабличное представление, возвращающее все поля из таблицы Writer.

```
CREATE VIEW View_Writer_1
(AS
SELECT * FROM Writer WHERE Wrt_Key>100);
```

Создание представления на основе объединения нескольких таблиц. выводятся только три поля.

```
CREATE VIEW View_Writer_2
(AS
SELECT Writer.Surname, Writer.FName, Book.Book
FROM Writer
INNER JOIN BOOK ON Book.Bk_Key= Writer.Bk_Key);
```

выражение CREATE VIEW создает представление таблицы или нескольких таблиц иногда называемых виртуальной таблицей. Определение представления содержит обычный запрос на базе оператора SELECT, но с некоторыми ограничениями: например, нельзя использовать оператор ORDER BY, упорядочивающий набор строк. При объединении двух и более таблиц надо быть внимательным при совпадении имен полей таблиц. В таких случаях синтаксис SELECT * ... (выбор всех полей) является недопустимым и совпадающее имя надо переименовать при помощи оператора AS. Также явный список имен полей после оператора SELECT необходимо описать в случае, если поля содержат вычисляемые значения или существуют объединенные поля с разными именами в соответствующих им таблицам.

Как правило, представления доступны только в режиме «только для чтения», однако если представление создано на основе одной единственной таблицы и не включает в себя операторы UNION, EXCEPT, INTERSECT, GROUP BY, HAVING, DISTINCT, то представление допускает операции вставки INSERT и редактирования UPDATE.

Для удаления представления используют инструкцию

```
DROP VIEW имя_представления CASCADE | RESTRICT
```

Минимальный синтаксис требует оператора DROP VIEW и имени представления.

```
DROP VIEW Temp_View
```

Если вы используете ключевое слово RESTRICT, то на удаляемое представление не должно быть никаких ссылок, иначе операция удаления будет отменена. Параметр CASCADE напротив, осуществит каскадное удаление всех объектов связанных с данным представлением.

Работа с индексами

Индексы предназначены для обеспечения максимально быстрого поиска и сортировки данных в таблице. Физически индекс представляет собой уникальный набор значений и список указателей на страницы данных где эти значения находятся.

```
CREATE [UNIQUE] [CLUSTERED] INDEX имя_индекса ON имя_таблицы (имя_столбца [, ... n])
```

В случае создания индекса с опцией **UNIQUE** СУБД прекращает допускать ввод повторяющихся значений в столбец (столбцы), включенные в определение индекса.

Кластерный индекс **CLUSTERED** – специальный индекс, заставляющий СУБД хранить данные в таблицах именно в том порядке, который задает этот индекс. В таблице может быть определен только один кластерный индекс.

```
CREATE UNIQUE INDEX siName ON WRITER (SurName, FName, LName);
```

Использование кластерного индекса способствует значительному сокращению времени доступа к данным, поэтому основной индекс таблицы всегда стоит определять **CLUSTERED**.

В различных диалектах СУБД синтаксис создания индекса несколько разнится. Например, в InterBase допускается создавать индексы, осуществляющие сортировку данных в обратной последовательности при помощи ключа **DESCENDING**.

```
CREATE DESCENDING INDEX siName_ZA ON WRITER (SurName, FName, LName);
```

Для удаления индекса используют конструкцию

```
DROP INDEX имя_индекса
```

Синтаксис удаления индекса включает в себя ключевые слова **DROP INDEX** и непосредственно имя таблицы и индекса, разделенные точкой.

Работа с хранимыми процедурами

В стандартном наборе инструкций SQL-92 команды, позволяющей создавать или удалять хранимую процедуру, мы не обнаружим. Вместе с тем практически все SQL серверы поддерживают технологию хранимых процедур, поэтому в стандарте SQL-92 предусмотрены инструкции **CREATE PROCEDURE** и **DROP PROCEDURE**.

Изучите синтаксическую конструкцию, позволяющую создать хранимую процедуру

```
CREATE PROCEDURE имя_процедуры  
[параметр1 тип_параметра, ..., параметрN тип_параметра]  
AS  
ТЕЛО ПРОЦЕДУРЫ
```

Например, создавая процедуру для добавления строки в таблицу БД Microsoft SQL Server (диалект Transact-SQL) мы бы написали следующий текст запроса:

```
CREATE PROCEDURE stp_InsertRecord  
@Param1 INT, @Param2 VARCHAR(30), @Param3 BIT, @PrimaryKey INT OUTPUT  
AS  
INSERT INTO MyTable (Field1, Field2, Field3)  
VALUES (@Param1, @Param2, @Param3)  
SET @PrimaryKey =@@IDENTITY;
```

Еще один пример процедуры, в результате выполнения возвращающей набор данных, но в этот раз на диалекте сервера InterBase:

```
CREATE PROCEDURE stp_SelectRecord  
(ParamDate DATE)  
AS  
BEGIN  
FOR  
SELECT * FROM DayTable WHERE FieldDate <: ParamDate  
DO SUSPEND;  
END;
```

Хранимая процедура удаляется командой `DROP PROCEDURE`.

DROP PROCEDURE имя_процедуры

Работа с триггерами

Триггер – это программа, описанная на языке SQL и выполняющаяся в момент возникновения события, обычно связанного с вставкой, удалением или изменением данных. Как правило, триггер представляет собой последний рубеж обороны по поддержке бизнес-правил базы данных.

```
CREATE TRIGGER имя_триггера ON имя_таблицы  
FOR ([DELETE] | [INSERT] | [UPDATE])  
AS инструкция_на_языке_SQL
```

В соглашении о SQL-92 нет четкого описания синтаксиса создания триггера. Поэтому приводим достаточно условную синтаксическую конструкцию. Ключевые слова `DELETE`, `INSERT`, `UPDATE` указывают после какого события (удаления, вставки, редактирования) должен выполняться триггер.

В InterBase допускается использование ключевого слова `BEFORE` (например: `BEFORE DELETE`), говорящего, что событие происходит перед операцией (в данном случае удаления).

```
CREATE TRIGGER Delete_Book FOR Writer  
ACTIVE BEFORE DELETE  
AS  
DELETE FROM Book WHERE Book.Wrt_Key=Writer.Wrt_Key;
```

Предложенный пример триггера (синтаксис InterBase) удаляет все книги из таблицы `Book`, написанные конкретным автором перед тем, как и сам автор будет удален из таблицы `Writer`.

Для удаления триггера воспользуйтесь инструкцией

DROP TRIGGER имя_триггера

Как и все операции удаления, операция удаления триггера начинается ключевым словом `DROP`, далее следует пояснение – какой объект из базы данных удаляется – `TRIGGER`, и в заключении через точку передается имя таблицы и имя триггера.

Задание

Используя изученные на занятии операторы SQL, подготовьте последовательность инструкций SQL позволяющих создать приведенные к четвертой нормальной форме таблицы в соответствии с ER-моделью подготовленной вами на занятии 4. Модель “Сущность-связь” (ER-модель). Предложите инструкции SQL, позволяющие реструктурировать имеющиеся таблицы (добавлять или удалять поля, изменять характеристики полей).

9. Разработка отчетов БД в среде Access

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access

Отчеты (reports) предоставляют наиболее гибкий и одновременно наглядный способ отображения информации, содержащейся в базе данных. Отчеты позволяют выбрать из базы данных требуемую пользователем информацию и оформить ее в виде документов, которые можно просмотреть и напечатать.

Средствами отчета невозможно внести изменения в содержащиеся в БД данные.

Подготовка отчета в среде Access

В среде Access отчет может создаваться как на основе таблиц, так и на базе заранее подготовленных запросов. Допустим, что нам требуется подготовить отчет по списочному составу студентов нашего ВУЗа. Разработаем простой объединяющий запрос:

```
SELECT Students.*, Spec.*  
FROM Spec  
INNER JOIN Students ON Spec.Spec_ID = Students.Spec_ID  
ORDER BY SName, FName;
```

Сохраните вновь созданный запрос под именем “StudentsList”, в списке объектов выделите только что созданный элемент и щелкните по кнопке “Создание — Отчет” на панели управления базой данных. В результате перейдя в окно предварительного просмотра вы получите простейший вариант бумажного отчета (рис. 9.1).

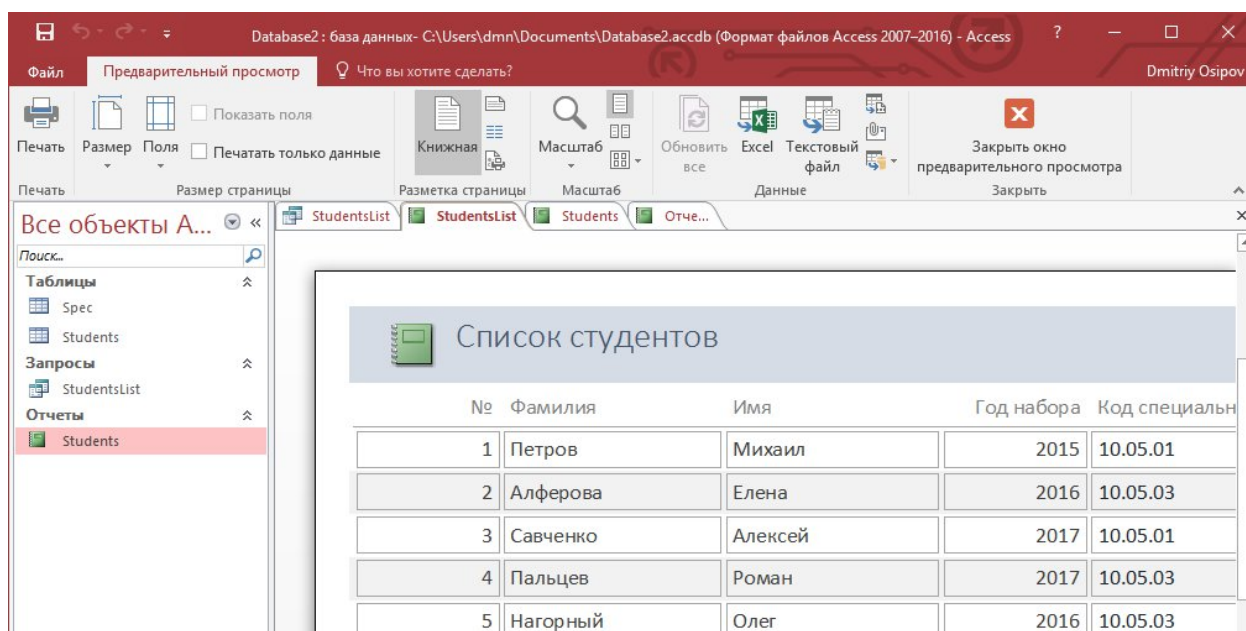


Рис. 9.1. Отчет созданный Access в режиме по умолчанию

Другим вариантом создания отчетов в Access является использование специального помощника — мастера отчетов. Для доступа к нему воспользуйтесь кнопкой “Создание — Мастер отчетов” (рис. 9.2)

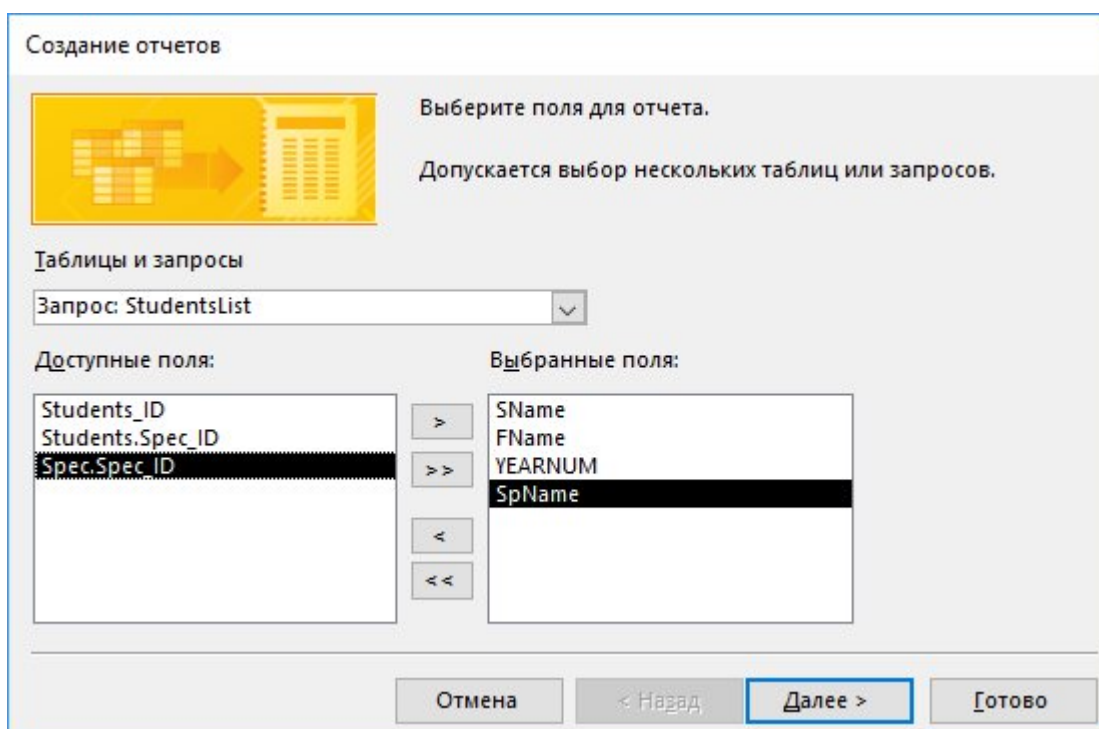


Рис. 9.2. Первое окно диалога мастера отчетов

Для создания профессионального отчета вам потребуется значительно больше усилий. Для этого потребуется поработать вручную в **Конструкторе отчетов**. Отчет сразу “рассыпается” на пару десятков самостоятельных элементов управления. Здесь и обычные компоненты-метки (предназначенные для вывода вспомогательных надписей) и компоненты-поля (отвечающие за получение информации из результирующего набора данных).

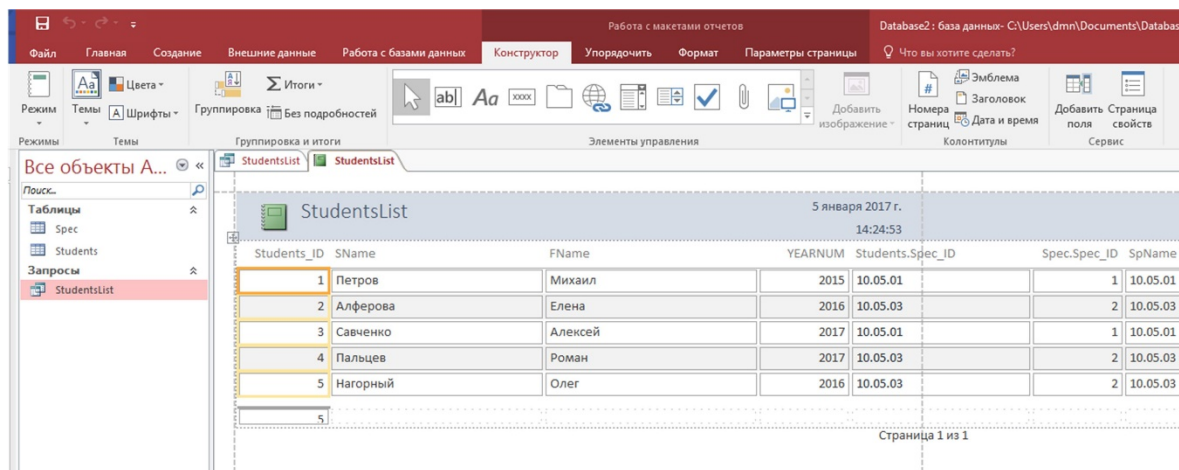


Рис. 9.3. Отчет в режиме конструктора

Для настройки характеристик отдельного элемента выведите на экран его свойства (пункт “Свойства” контекстного меню). В рамках редактора свойств вы сможете управлять всеми параметрами элемента управления.

Задание

В соответствии с полученными заданиями подготовьте 3-4 отчета. Основной отчет должен объединять в себе информацию из всех таблиц БД. Один из отчетов должен быть создан с помощью мастера отчетов Access.

Важно! Подготовленные вами отчеты должны быть готовы к печати на листах бумаги формата А4.

10. Разработка интерфейса приложения в среде Access

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access

В задачи разработчика баз данных входит не только создание ключевых объектов БД (например, таблиц и запросов), но и разработка интуитивно понятного интерфейса, позволяющего обычному пользователю (не имеющего ни малейшего представления об особенностях физической организации данных) просматривать, добавлять, редактировать и удалять данные.

Создание форм для ввода данных в Microsoft Access

Среди всех сред программирования, специализирующихся на работе с БД Microsoft Access предоставляет едва ли ни самый простой и удобный способ создания интерфейса доступа к базам данных. Здесь предусмотрено несколько способов создания форм. К простейшим из них относится применение средств автоматического создания форм на основе таблицы или запроса (рис. 10.1).

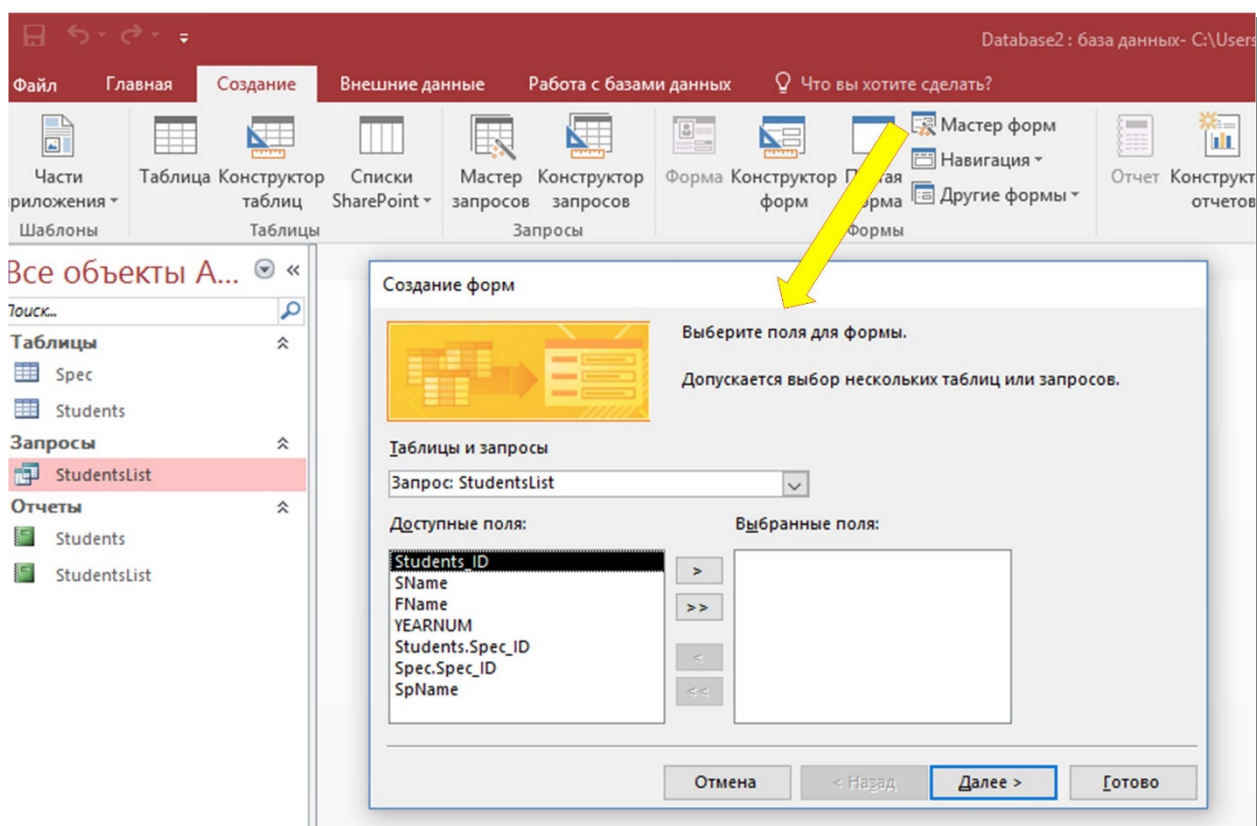


Рис. 10.1. Мастер создания новой формы

Вернемся к нашему проекту по учету студентов ВУЗа (лабораторные работы 2 и 3) и создадим форму для таблицы студентов “Students”.

1. Выберите таблицу “Students” в списке объектов БД.
2. Перейдите на страницу “Создание” среды проектирования.
3. Раскройте список **Другие формы** и выберите в нем элемент **Разделенная форма**.

Автоматически созданная форма далеко не безукоризненна, с точки зрения конечного пользователя у нее как минимум четыре недостатка (рис. 10.3). **Во-первых**, подписи полей не отражают их назначения. **Во-вторых**, на форме присутствует поле первичного ключа, причем создается впечатление, что оно доступно для редактирования, а в этом нет необходимости. **В-третьих**, размеры строк ввода чрезмерны велики. **В-четвертых**, название формы не отражает ее назначения.

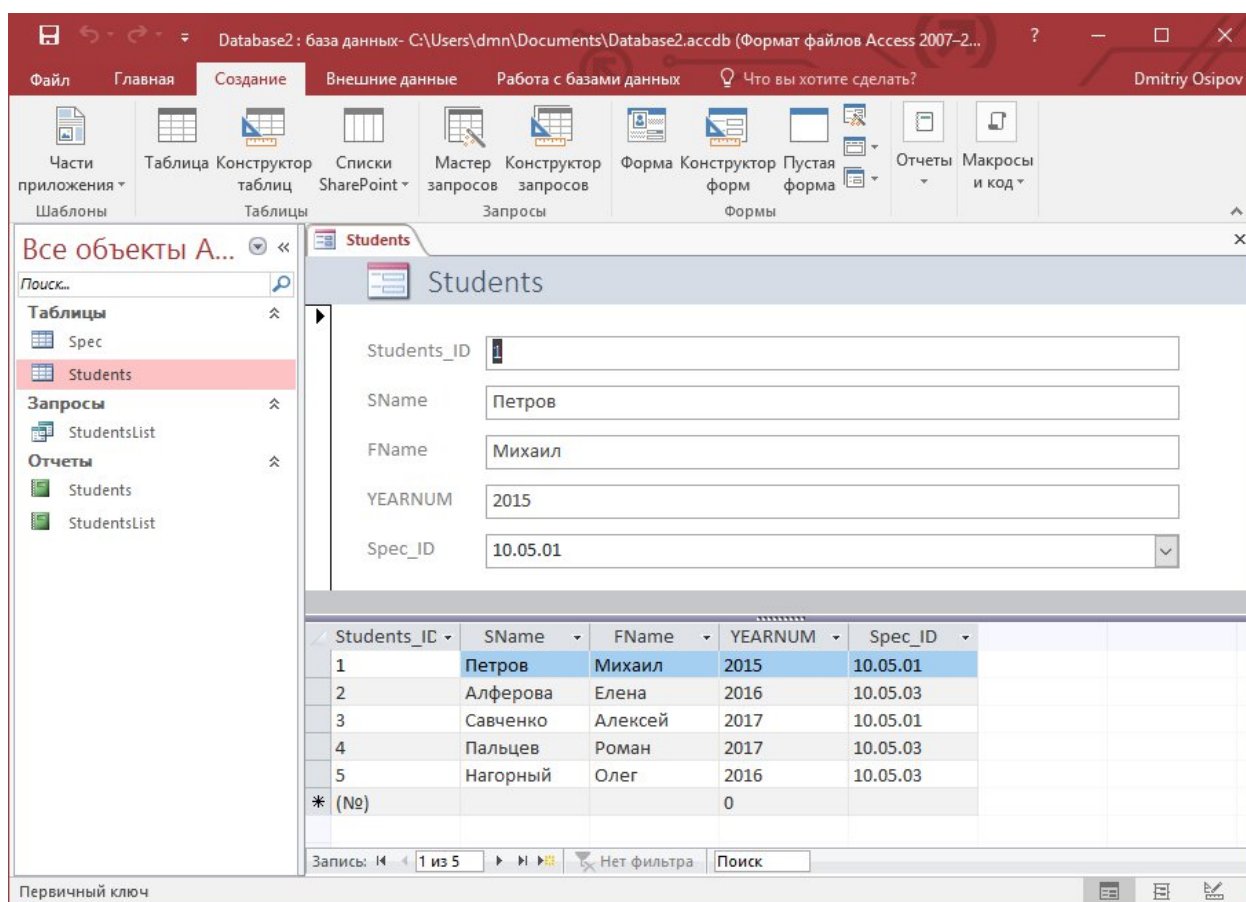


Рис. 10.2. Автоматически созданная форма Students

Для внесения исправлений и доработок в форму необходимо перейти в режим конструктора. Для этого перейдите на вкладку **Главная**, нажмите на кнопку “Режим” на панели инструментов, и в открывшемся списке выберите элемент “Конструктор”. В режиме конструктора вы сможете управлять всеми элементами управления, расположенными на форме (рис. 10.4). Для этого достаточно обратиться к контекстному меню компонента и выбрать пункт “Свойства”.

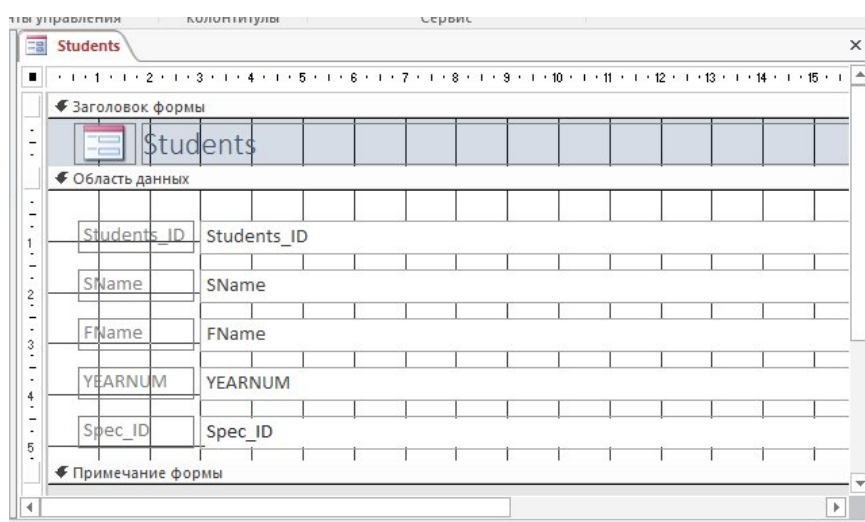


Рис. 10.3. Форма в режиме конструктора

Рекомендации по проектированию пользовательского интерфейса

При разработке пользовательского интерфейса следует руководствоваться рядом рекомендаций. Ниже перечислены основные из них:

- ~ название формы должно отражать ее назначение;
- ~ последовательность полей и их группировка должны быть логически обоснованы;

- ~ названия полей легко узнаваемы;
- ~ форма должна иметь привлекательный вид;
- ~ используется только общепринятая терминология и сокращения;
- ~ визуальное выделение пространства и границ полей ввода данных;
- ~ удобные средства перемещения курсора;
- ~ наличие средств исправления отдельных ошибочных символов и целых полей;
- ~ наличие сообщений об ошибках при вводе недопустимых значений;
- ~ выделение обязательных для ввода данных полей;
- ~ средства вывода пояснительных сообщений с описанием полей;
- ~ средства вывода сообщения об окончании заполнения формы.

Задание

Применяя различные подходы к проектированию рабочих форм создайте не менее 5 форм для ввода, редактирования и удаления данных для своего проекта БД. Разберитесь с назначением основных элементов управления, которые разрешено располагать на форме. При проектировании интерфейса форм опирайтесь на рекомендации по разработке пользовательского интерфейса БД.

6 семестр

11. Доступ к данным средствами ActiveX Data Object (ADO)

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access, C++ Builder (Delphi)

Технология объектов данных ActiveX (Microsoft® ActiveX® Data Objects, ADO) появилась на свет в стенах корпорации Microsoft и считается логическим развитием системы доступа к данным OLE DB. У ADO много достоинств, основное из них в том, что технология обеспечивает единый механизм доступа к разнотипным хранилищам данных. Приложение ADO обращается не к конкретной СУБД, а к объекту COM. Многокомпонентная модель объектов (Component Object Model, COM) описывает способ взаимодействия программ любого типа, программа-сервер предоставляет в распоряжение окружающих собственные службы, программа-клиент пользуется услугами доступных служб.

Для доступа к данным в рамках ADO (рис. 11.1) реализована колл-лекция интерфейсов COM-объектов предназначенных для работы, как с реляционными, так и с не реляционными наборами данных, включая майнфреймы, иерархические базы данных, текстовые, графические, географические данные, XML, e-mail и многое другое. Доступ к данным предоставляют драйверы, называемые *провайдером*.

Все элементы управления VCL, нацеленные на работу с объектами данных ADO, собраны на странице dbGo палитры компонентов Delphi XE. Основной программный код всех интересующих нас сегодня классов сосредоточен в модуле ADODB, а описания их интерфейсов в ADOInt. Семёрку компонентов ADO можно условно разделить на три категории (рис. 11.2):

- ~ Компоненты обеспечения централизованного доступа к данным – TADOConnection и TRDSCConnection. Эти компоненты обычно используются при разработке сложных приложений, базирующихся на архитектуре клиент-сервер. Экземпляр класса TADOConnection отвечает за установку соединения и взаимодействие проекта с БД, все остальные компоненты dbGo пользуются его посредническими услугами.
- ~ Компоненты обслуживания данных в стиле Microsoft ADO – командный компонент TADOCommand и набор данных TADODataSet.
- ~ Компоненты, работающие с данными: компонент-таблица TADOTable, запрос TADOQuery и хранимая процедура TADOStoredProc.

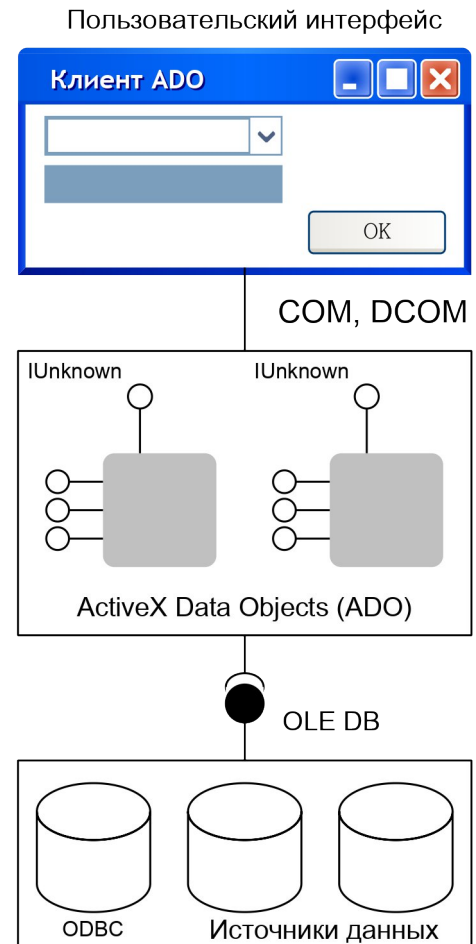


Рис. 11.1. Архитектура универсального доступа к данным ADO

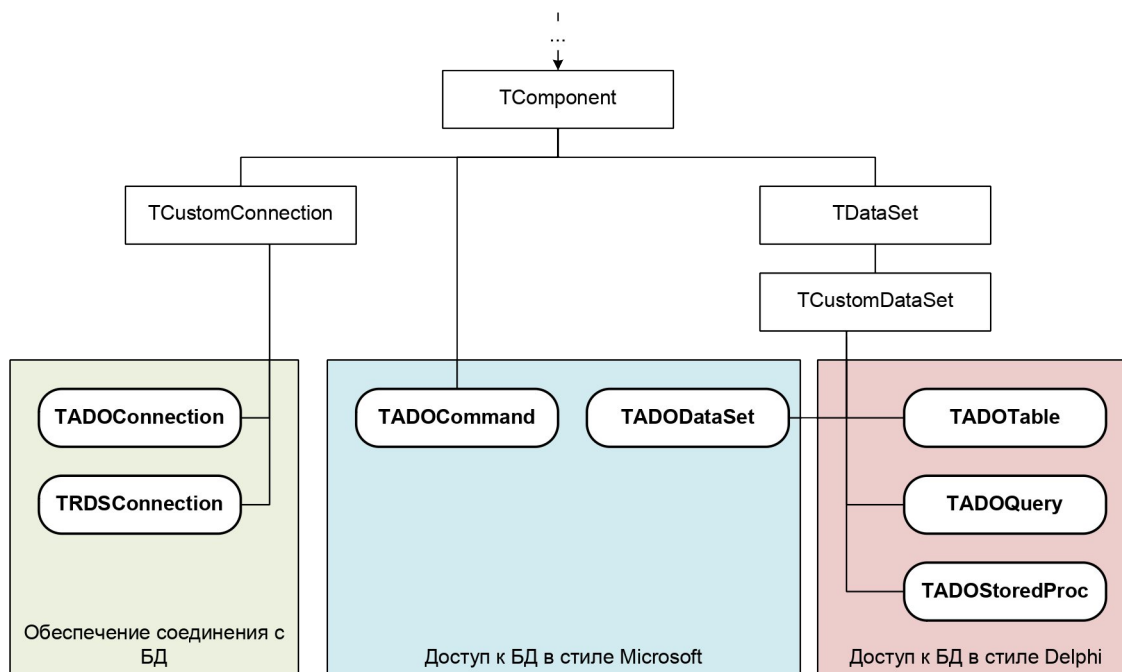


Рис. 11.2. Иерархия компонентов ADO в библиотеке VCL

Строка соединения ADO

Строка соединения ADO представляет собой обычную текстовую строку, содержащую описание параметров соединения с интересующей нас БД. Соединительная строка представляет собой последовательность аргументов и значений, разделяемых точкой с запятой. В качестве символа присвоения применяется знак равенства.

Параметр1=значение1;Параметр2=значение2;

Например, так выглядит строка доступа к базе данных DEMODB развернутой на Microsoft SQL Server с именем MSSQL_HOST:

Provider=SQLOLEDB.1;Persist Security Info=False;User ID=SA;Initial Catalog=DEMOB;Data Source=MSSQL_HOST

Перечень наиболее часто встречаемых в соединительной строке аргументов представлен в таблице 11.1.

Таблица 11.1. Аргументы соединительной строки

Параметр	Описание
Provider	Имя поставщика интерфейса OLE DB. Если этот аргумент не определен, то по умолчанию задействуется провайдер Microsoft OLE DB для ODBC – “MSDASQL”
Persist Security Info	Принимает значение true или false. В состоянии true уведомляет, что в строке соединения имеется конфиденциальная информация о порядке соединения (например, имя пользователя и пароль доступа), состояние false говорит о том, что такая информации отсутствует.
File name	Название файла, содержащего информацию о соединении
User ID	Регистрационное имя пользователя по умолчанию
Password	Пароль пользователя
Data Source	Имя источника данных, например SQL-сервера
Initial Catalog	Каталог инициализации – имя базы данных
Remote Provider	Имя провайдера применяемого на клиентской стороне соединения. Этот параметр необходим только при организации работы со службами удаленных данных RDS
Remote Server	Имя удаленного сервера. Этот параметр необходим только при работе с RDS
URL	Абсолютный URL адрес

Строку соединения допускается хранить в специальном файле связи с данными (UDL-файле). Для создания такого файла достаточно создать пустой текстовый файл и заменить расширение txt на udl. Запуск такого файла автоматически вызовет редактор настройки соединительной строки.

Соединение с хранилищем данных, компонент *TADOConnection*

Хотя основной задачей компонента считается установка соединения с базой данных ADO, но, ко всему прочему, *TADOConnection* умеет:

- ~ регистрировать пользователя в СУБД;
- ~ управлять транзакциями;
- ~ отправлять в адрес СУБД различные команды (инструкции SQL);
- ~ взаимодействовать с подчиненными компонентами – наборами данных ADO;
- ~ получать от БД информацию об ее объектах и особенностях соединения.

Установка соединения

Для описания параметров подключения к БД текст соединительной строки направляется в свойство

property *ConnectionString*: *WideString*;

Если в строке отсутствует информация о поставщике услуг, то его необходимо определить в свойстве:

property *Provider*: *WideString*;

По умолчанию роль провайдера исполняет *MSDASQL*.

Процесс соединения с указанной в соединительной строке базой данных инициируется посредством обращения к методу *Open()*. В простейшей ситуации процедура даже не требует параметров.

procedure *Open*; **overload**;

В перегружаемой версии процедуры *Open()* предусмотрена возможность передачи в адрес сервера имени и пароля пользователя. Но это только для случая, когда эта информация не включена в соединительную строку.

procedure *Open*(**const** *UserID*: *WideString*;
 const *Password*: *WideString*); **overload**;

Если вы полагаете, что при подключении к базе данных пользователь обязан пройти регистрацию и своими собственными руками ввести имя и пароль, то целесообразно установить в *True* свойство:

property *LoginPrompt*: *Boolean*;

В результате перед соединением с базой вызывается стандартное диалоговое окно регистрации.

Для проверки факта подключения компонента *TADOConnection* к БД контролируют состояние свойства:

property *Connected*: *Boolean*;

С процессом установки соединения связана строгая последовательность событий (табл. 11.2) позволяющая программисту практически на всех этапах помогать или мешать приложению получать доступ к БД.

Таблица 11.2. Последовательность событий при соединении *TADOConnection* с набором данных

Событие	Описание
1. property <i>BeforeConnect</i> : <i>TNotifyEvent</i> ;	вызывается перед началом соединения
2. property <i>OnWillConnect</i> : <i>TWillConnectEvent</i> ;	Генерируется после отправки запроса на соединение
3. property <i>OnLogin</i> : <i>TLoginEvent</i> ;	Регистрация пользователя
4. property <i>OnConnectComplete</i> : <i>TConnectErrorEvent</i> ;	Процесс соединения завершается
5. property <i>OnInfoMessage</i> : <i>TInfoMessageEvent</i> ;	вызывается после успешного соединения, в тот момент, когда провайдер возвращает дополнительную информацию о соединении
6. property <i>AfterConnect</i> : <i>TNotifyEvent</i> ;	вызывается по окончании всего процесса соединения

Пример соединения без регистрации пользователя

Благодаря тому, что параметры соединения ADO содержатся в обычной текстовой строке, программист получает отличную возможность управлять процессом подключения к любому хранилищу данных просто редактируя параметры соединительной строки. Предлагаю вашему вниманию пример, позволяющий пользователю выбирать расположение исходного файла с данными в момент старта приложения настольной БД.

Допустим, что в нашем распоряжении имеется база данных в формате Microsoft Access. В составе форм проекта Delphi пока имеется:

- ~ главная форма проекта Form1 (модуль Unit1), данная форма пока не содержит элементов управления — вы их добавите позднее;
- ~ форма ввода пути к файлу БД Form2 (модуль Unit2);
- ~ модуль данных DataModule1:TDataModule (модуль Unit3) на котором размещен компонент ADOConnection1 :TADOConnection.

Вполне очевидно, что для выбора файла БД в момент старта приложения следует выводить окно, уточняющее путь к файлу. Для воспользуемся формой Form2 (рис. 11.3) и размещаем на ее поверхности следующий перечень компонентов:

- ~ строка ввода Edit1:TEdit для редактирования пути к файлу.
- ~ диалог открытия файла OpenFileDialog1. Для того, чтобы компонент реагировал только на файлы Access следует отредактировать отвечающее за фильтрацию файлов свойство Filter:= Моя БД Access 2003|demo_db.mdb|Моя БД Access 2007-2016|demo_db.accdb|Все файлы Microsoft Access|*.mdb; *.accdb. Обратите внимание, на то, что в данном примере предполагается подключение к файлу с именем demo_db, в вашем случае следует передать другое имя.
- ~ справа от строки ввода расположите кнопку Button1:TButton. Щелчок по этой кнопке будет про-буждать от сна диалог открытия файла OpenFileDialog1.
- ~ Два последних компонента — кнопки класса TBitBtn. Первую кнопку называем btnOK и ее свойство Kind установим в bkOK. Свойству Kind второй кнопки присвоим bkCancel.

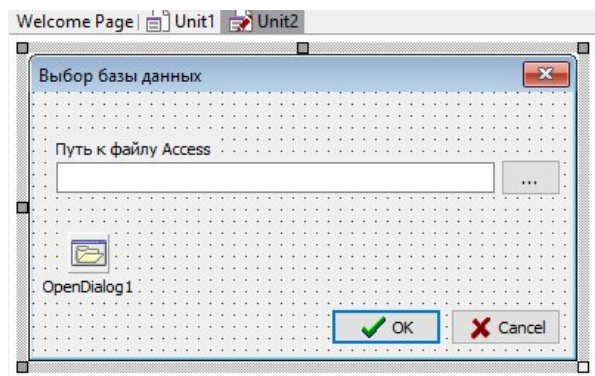


Рис. 11.3. Форма ввода пути к БД

Убедитесь, что новый проект сохранен и перейдите к Form2. Опишем событие щелчка по кнопке Button1 — он отвечает за выбор пути к файлу БД (листинг 11.1).

Листинг 11.1. Щелчок по кнопке — определение пути к файлу БД

```
procedure TForm2.Button1Click(Sender: TObject);
begin
if OpenFileDialog1.Execute then
    LabeledEdit1.Text:=OpenDialog1.FileName;
end;
```

В рамках события OnClick() мы выводим на экран диалог открытия файла (в котором пользователь выберет имя файла с БД) и передаем имя и путь к файлу в строку edFilePath.

Переходим к главной форме проекта Form1 и обращаемся к ее событию OnShow() — оно генерируется в момент вывода формы на экран (листинг 11.2).

Листинг 11.2. Вывод главной формы проекта на экран

```
uses Unit2, Unit3;
```

```

procedure TForm1.FormShow(Sender: TObject);
begin
if NOT DataModule1.ADOConnection1.Connected then
  if Form2.ShowModal=mrOK then
  begin
    {формируем соединительную строку}
    DataModule1.ADOConnection1.ConnectionString:=
      'Provider=Microsoft.ACE.OLEDB.12.0;Data Source='+
      Form2.LabeledEdit1.Text+';Persist Security Info=False';
    DataModule1.ADOConnection1.LoginPrompt:=False;
    DataModule1.ADOConnection1.Connected:=True;
  end;
end;

```

В результате мы получим следующий результат — в момент вывода на экран главной формы проекта над ним будет возникать модальное окно, в котором пользователь должен указать путь к целевому файлу.

Информирование о БД

В состав методов компонента TADOConnection входит процедура способная предоставить нам системную информацию о базе данных

```

procedure OpenSchema(const Schema: TSchemaInfo;
  const Restrictions: OleVariant;
  const SchemaID: OleVariant; DataSet: TADODataSet);

```

Первый параметр метода Schema: TSchemaInfo определяет какую именно информацию мы рассчитываем получить. Например, передав в параметр константу siIndexes — мы приобретем информацию об индексах БД, siTables — таблицах, siViews — представлениях. Второй параметр Restrictions представляет собой массив ограничений на запрашиваемую информацию. Третий по счету параметр SchemaID применяется только в ситуации, когда глобальный уникальный идентификатор GUID провайдера данных не определен в стандартной спецификации OLE DB. Как правило, такая необходимость возникает в случаях, когда аргумент Schema установлен в siProviderSpecific. И, наконец, последний параметр DataSet является ссылкой на элемент управления класса TADODataSet, в его множество записей Recordset будут направлены результаты выполнения процедуры. Полученные данные программист может обрабатывать по своему усмотрению, например, просто отобразить их в сетке TDBGrid.

Апробируем приобретенные знания на практике. Разместите на поверхности модуля данных DataModule1 проекта компоненты:

```

~      соединение ADO — ADOConnection1;
~      набор данных ADO — ADODataset1;
~      источник данных — DataSource1.

```

Перейдите к инспектору объектов. С помощью свойства Connection элемента управления ADODataset1 свяжите компонент с ADOConnection1. Затем выберите источник данных DataSource1 и, воспользовавшись свойством DataSet, соедините его с ADODataset1.

Создайте новую пустую форму Form4 (Unit4) и расположите на ее поверхности единственный компонент — сетку — DBGrid1. Подключите в строку uses новой формы ссылку на модуль данных (в нашем примере это Unit3), затем присоедините сетку DBGrid1 к источнику данных DataSource1.

Переходим к программированию (листинг 11.2).

Листинг 11.3. Получение сведений о всех таблицах БД

```

uses Unit3, Data.Win.ADODB;
procedure TForm4.FormShow(Sender: TObject);
begin
if DataModule1.ADOConnection1.Connected then
begin
  DataModule1.ADODataset1.Close;
  DataModule1.ADOConnection1.OpenSchema(siTables, EmptyParam, EmptyParam, DataMod-
  ule1.ADODataset1);
end;
end;

```

Если все сделано безошибочно, то в результате запуска проекта мы получим данные обо всех отношениях БД (в случае Access будут отображены не только созданные вами таблицы, но и скрытые таблицы и запросы).

На основе процедуры `OpenSchema()` построен ряд справочных методов, способных снабдить нас данными о таблицах, полях таблиц и хранимых процедурах базы данных.

```
procedure GetTableNames(List: TStrings; SystemTables: Boolean = False);
```

```
procedure GetFieldNames(const TableName: String; List: TStrings);
```

```
procedure GetProcedureNames(List: TStrings);
```

Результаты выполнения процедур передается в список строк `List: TStrings`.

Задание

1. В среде проектирования Delphi (C++ Builder) на основе компонента `TADOConnection` создайте приложение, соединяющееся с разработанной вами БД MS Access.
2. Предусмотрите в своем проекте возможность регистрации пользователя в БД.
3. Предусмотрите в своем проекте возможность сбора служебной информации о БД (имена доступных таблиц и их полей).

12. Общие черты компонентов доступа к данным

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access, C++ Builder (Delphi)

Более десятка визуальных элементов управления среды программирования Delphi предназначены для решения вопросов отображения содержимого отдельных полей, или коллекции строк из набора данных, многие из них обучены редактировать данные, а кое-кто даже способен управлять процессом ввода/удаления данных из таблиц. Благодаря этим компонентам, мы, не написав ни одной строки кода, сможем строить несложные приложения, работающие с базой данных. Для проверки этого утверждения дополните разработанный на лабораторной работе 11 проект новой формой и разместите на главной форме следующий перечень элементов управления:

1. Таблица `ADOTable1`, этот компонент вы обнаружите на страничке `dbGo` палитры компонентов.
2. Источник данных `DataSource1` (страничка `Data Access`).
3. Навигатор данных `DBNavigator1` (страничка `Data Controls`).
4. Сетка для отображения данных `DBGrid1` (страничка `Data Controls`).

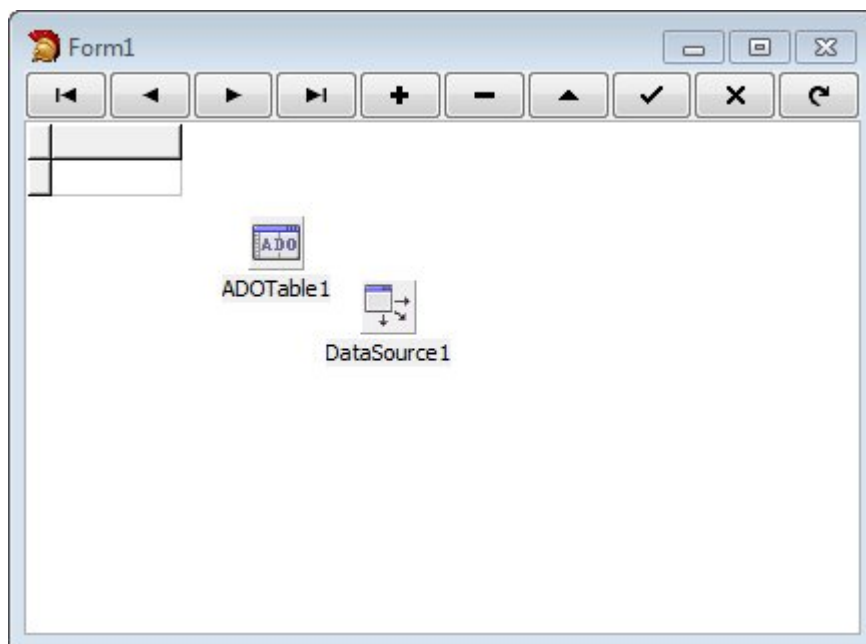


Рис. 12.1. Внешний вид первого проекта БД

Разместив все перечисленные компоненты на форме проекта (рис. 12.1), выберите таблицу – компонент `ADOTable1`. Нам предстоит проделать наиболее важную операцию – подключить компонент таблицы к физической таблице, которая в разработанном вами файле БД. Для этого выберите свойство `Connection` компонента и подключите его к компоненту-соединителю `ADOConnection1`.

Закончив настройку строки соединения, мы получаем возможность подключить компонент `ADOTable1` к файлу с таблицей. Для этого в инспекторе объектов находим свойство `TableName` и вводим имя любой таблицы из вашей БД, например `TableName:='SUPPLIERS'`. Настройка подключения завершена, нам осталось только активировать его, для этого устанавливаем свойство компонента `Active` в состояние `true`.

Настроим оставшиеся три компонента.

1. В свойство `DataSet` компонента `DataSource1` передаем имя компонента `ADOTable1`.
2. В свойство `DataSource` компонента `DBGrid1` передаем имя источника данных `DataSource1`.
3. В свойство `DataSource` компонента `DBNavigator1` передаем имя источника данных `DataSource1`.

Работа над проектом завершена.

Базовый класс доступа к данным *TDataSet*

В распоряжение разрабатывающего приложения БД программиста средой Delphi предоставляется более десятка компонентов, позволяющих получить доступ к данным. Однако эта лабораторная работа посвящена изучению ни какого-то отдельного элемента управления, мы рассмотрим предтечу всех компонентов доступа к данным – абстрактный класс *TDataSet*. Класс нацелен на решение следующих основных задач:

1. Открытие и закрытие набора данных.
2. Перемещение по набору данных.
3. Редактирование записей в наборе.
4. Организация доступа к полям таблиц.
5. Фильтрация набора данных.
6. Обеспечение поиска данных.

Из-за того, что *TDataSet* возглавляет иерархию классов, предназначенных для доступа к данным, и служит в своем роде первопроходцем в этом направлении, значительная часть объявленных в нем методов не наполнена содержанием – они являются абстрактными.

Открытие и закрытие набора данных

Простейшей проверкой на то, в каком состоянии находится набор данных, может стать контроль свойства

property Active: Boolean;

Если оно вернет значение `true`, то это признак того, что набор данных активен и, как минимум доступен для просмотра. Совместно со свойством работают методы

procedure Open; // открытие набора

procedure Close; // закрытие набора

Перевод в `true` свойства `Active` последовательно вызовет события

property BeforeOpen: TDataSetNotifyEvent; //перед открытием

property AfterOpen: TDataSetNotifyEvent; //после открытия

Вызов метода `Close()` (присвоение значения `false` свойству `Active`) также вызовет два события

property BeforeClose: TDataSetNotifyEvent; //перед закрытием

property AfterClose: TDataSetNotifyEvent; //после закрытия

Перемещение по набору данных

Мы уже знаем, что открытый непустой набор данных обеспечивает доступ к своим записям. Текущая запись набора обычно называется активной записью, именно на ней установлен курсор набора. Перемещение курсора приводит к смене активной (текущей) записи. Естественно в классе *TDataSet* реализован целый ряд методов, обеспечивающих перемещение курсора по набору. К наиболее часто используемым процедурам такого рода стоит отнести

procedure First; //переход на первую запись

procedure Prior; //возврат к предыдущей записи

procedure Next; //переход на следующую запись

procedure Last; //переход на последнюю запись

Для прыжка сразу на несколько записей понадобится помощь метода

function MoveBy(Distance: Integer): Integer;

В параметр `Distance` передается дистанция, на которую мы хотим переместиться по набору, от текущей записи. Если значение отрицательное, то перемещение осуществляется к началу набора, положительное – к концу. За отправную точку принимается текущая запись набора. Функция возвращает реальное число записей, на которое удалось переместить курсор. Вызов методов перемещения имеет смысл только в том случае, если набор не пуст. Узнать об этом поможет метод

function IsEmpty : Boolean;

Если в наборе нет ни одной строки, то функция возвратит `true`. Если в наборе есть записи, то общее количество доступных строк нам поведаст свойство

property RecordCount : Integer;

Порядковый номер текущей записи храниться в свойстве

property RecNo : Integer;

Это свойство доступно только в компоненте TADOTable.

Два специальных свойства сигнализируют о местонахождении курсора. Если курсор находится на самой первой записи в наборе, то в True установится свойство

property Bof : Boolean; //сокр. от англ. "begin of file"

Если курсор окажется на самой последней записи, то в состояние True установится свойство

property Eof : Boolean; //сокр. от англ. "end of file"

С помощью перечисленных выше свойств часто реализуют циклы, перебирающие все записи внутри набора

Листинг 12.1. Просмотр всех строк в таблице

```
with ADOTable1 do
begin
DisableControls; //отключение визуальных элементов управления от набора данных
    First;
    while EOF<>true do
    begin
        Memo1.Lines.Add(FieldByName('Field1').AsString);
        Next;
    end;
EnableControls; //подключение элементов управления
end;
```

С перемещением курсора по набору данных связано два события

property BeforeScroll: TDataSetNotifyEvent; //перед перемещением

property AfterScroll: TDataSetNotifyEvent; //после перемещения

Создание закладок и переход к закладке

В Delphi реализован простой способ запоминания текущего положения курсора (создание закладки) с возможностью последующего перемещения к данной записи. Для того, чтобы запомнить позицию текущей строки набора данных мы вызываем метод

function GetBookmark: TBookmark;

Функция возвращает данные в переменную-закладку. Для перехода к закладке следует воспользоваться услугами процедуры

procedure GotoBookmark(Bookmark: TBookmark);

Вместо функций GetBookmak() и GotoBookmark() допускается использовать свойство

property Bookmark: TBookmark;

Когда необходимость в услугах закладки отпадает, распределенный ресурс освободит метод

procedure FreeBookmark(Bookmark: TBookmark);

Листинг 12.2. Работа с закладками

```
private
    BM:TBookmark; //переменная для запоминания закладки
    //...
procedure TForm1.btnGetBookmarkClick(Sender: TObject);
begin
    BM:=ADOTable1.GetBookmark; //помечаем запись закладкой
end;

procedure TForm1.btnGoToBookmarkClick(Sender: TObject);
begin
    if ADOTable1.BookmarkValid(BM)=true then //проверка закладки
    begin
        ADOTable1.GotoBookmark(BM); //возвращаемся к закладке
```

```

ADOTable1.FreeBookmark(BM); //удаляем закладку
btnGoToBookmark.Enabled:=False;
end;
end;

```

Редактирование записей в наборе

Модификация данных в наборе данных возможна только в ситуации, когда свойство

```
property CanModify: Boolean; //только для чтения
```

возвращает false — это признак того, что набор данных возражает против редактирования своих строк.

Получив потенциальное согласие набора данных на внесение изменений, можно добавить в конец набора новую, пустую запись. Для этого используйте метод

```
procedure Append;
```

Если требуется, чтобы новая запись оказалась в месте положения курсора, то применяйте

```
procedure Insert;
```

Вызов методов Append и Insert подготовит набор данных к приему новой записи. Если, например, средством визуализации избрана сетка (TDBGrid), то после вызова функции на экране компьютера пользователь увидит новую пустую строку, готовую к получению данных. Ему останется ввести информацию с клавиатуры. Если вы не доверяете пользователю, то одновременно с операцией добавления (или вставки) новой строки можно заполнить запись данными. Для этого используют методы:

```

procedure AppendRecord(const Values: array of const);
procedure InsertRecord(const Values: array of const);

```

Здесь в качестве параметров передается массив значений, непосредственно вносимых в таблицу. Будьте внимательны — последовательность значений, передаваемых в массиве должна совпадать с физической последовательностью полей в таблице.

```
Table1.InsertRecord(['Петров', 'Петр', 'Петрович', NULL]);
```

С процессом вставки новой записи связаны три последовательно возникающие события.

Таблица 12.1. События связанные со вставкой новой строки

Событие	Описание
property BeforeInsert: TDataSetNotifyEvent;	Вызывается перед вставкой записи
property OnNewRecord: TDataSetNotifyEvent;	Вызывается в момент вставки новой записи
property AfterInsert: TDataSetNotifyEvent;	Вызывается по окончании вставки записи

Для перевода текущей записи в режим редактирования предназначена процедура

```
procedure Edit;
```

С операциями редактирования взаимодействуют события

```

property BeforeEdit: TDataSetNotifyEvent; //перед редактированием
property AfterEdit: TDataSetNotifyEvent; // после редактирования

```

Для того, чтобы сохранить внесенные изменения вызывают метод

```
procedure Post;
```

Листинг 12.3. Вставка и сохранение новой записи

```

with ADOTable1 do
begin
Append;
FieldByName('City').Value:= 'Ставрополь';
Post;
end;

```

Соответствующие процессу сохранения обработчики событий

```
property BeforePost: TDataSetNotifyEvent; //перед сохранением
property AfterPost: TDataSetNotifyEvent; //после сохранения
```

Метод `Post()` следует вызывать только в случае, когда в результате редактирования в наборе данных возникли изменения. Узнать об этом поможет свойство

```
property Modified: Boolean; //только для чтения
```

Свойство возвратит `True`, если набор нуждается в сохранении. И наоборот значение `False` свидетельствует о том, что набор данных не изменялся.

Для удаления текущей записи используйте метод

```
procedure Delete;
```

Процедуру обслуживают события

```
property BeforeDelete: TDataSetNotifyEvent; //вызывается перед удалением
property AfterDelete: TDataSetNotifyEvent; //после удаления
```

Для отмены последних изменений в тексте используйте метод

```
procedure Cancel;
```

Учтите, что метод `Cancel` сохраняет работоспособность только до вызова метода `Post()`. Соответствующие обработчики событий:

```
property BeforeCancel: TDataSetNotifyEvent; //вызывается перед отменой
property AfterCancel: TDataSetNotifyEvent; //после отмены
```

Фильтрация набора данных

Фильтрация данных позволяет осуществлять отбор части записей по определенному критерию (критериям) описанным в свойстве:

```
property Filter: string;
```

Правила описания фильтра достаточно просты и похожи на синтаксис, используемый в языке SQL в предложении `WHERE`. Далее приведем несколько примеров по синтаксису строки фильтра.

```
ADOTable1.Filter:= 'Town='+ QuotedStr(МОСКВА);
ADOTable1.Filter:= 'Town<>'+ QuotedStr(МОСКВА);
ADOTable1.Filter:= 'Price>1000 and Price<=2000';
```

Для включения (отключения) фильтра необходимо установить в `true (false)` свойство:

```
property Filtered : Boolean;
```

Некоторые особенности в организацию фильтрации данных вносит свойство:

```
property FilterOptions: TFilterOptions;
```

Значение `foCaseInsensitive` определяет чувствительность фильтра к регистру символов. Значение `foNoPartialCompare` разрешает отфильтровывать частичные совпадения.

```
ADOTable1.Filter:= 'Town>M*';
```

С процессом фильтрации связан специализированный обработчик события:

```
type TFilterRecordEvent = procedure (DataSet: TDataSet; var Accept: Boolean) of object;
property OnFilterRecord: TFilterRecordEvent;
```

В рамках указанного события программист получает возможность осуществить более тонкую настройку процесса фильтрации. Параметр `DataSet` возвращает ссылку на фильтруемый набор данных. С помощью переменной `Accept` управляется процесс отбора (фильтрации) записей: `true` – включать запись в результирующий набор, `false` – не включать.

Листинг 12.4. Событие фильтрации строк набора данных

```
procedure TForm1.ADOTable1FilterRecord(DataSet: TDataSet; var Accept: Boolean);
begin
```

```
{настраиваем фильтр на отбор записей на основе значения в поле Price, в результирующий
набор будут включены записи >500 и меньше 1000}
ACCEPT:=(DataSet['Price']>'500') and (DataSet['Price']<'1000');
end;
```

Обработчик события будет работать только в случае, если в true установлено свойство Filtered.

Организация поиска данных

На уровне класса TDataSet реализован самый универсальный метод поиска Locate. Универсальность метода заключается в том, что он способен осуществлять поиск даже в неиндексированных полях.

```
function Locate(const KeyFields: String; const KeyValues: Variant;
Options: TLocateOptions): Boolean;
```

Здесь: KeyFields – имя поля (или полей, разделенных точкой с запятой) в которых производится поиск. KeyValues – шаблон (или, когда поиск идет в нескольких полях – массив шаблонов) по которым будет осуществлен поиск. Параметр Options : TLocateOptions передается в квадратных скобках, внутри скобок через запятую значения:

```
~ loCaseInsensitive – поиск нечувствителен к регистру символов.
~ loPartialKey – допускается поиск по части ключа.
```

Также допускается оставлять скобки пустыми.

В листинге приведен пример вызова функции Locate() для поиска некоего гражданина по фамилии “Петров” рожденного в 1980 году. Метод осуществит поиск сразу в двух полях таблицы, не обращая внимания на регистр символов.

Листинг 12.5. Поиск строки в наборе данных

```
with ADOTable1 do
Locate('Name; BDay', VarArrayOf(['Петров', '1980']), [loCaseInsensitive]);
```

В случае обнаружения требуемых значений метод позиционирует курсор на этой записи и возвращает значение True.

Взаимодействие с элементами управления данными

Как правило, потомки класса TDataSet (элементы управления TADOTable, TADOQuery, и т.д.) еще в период разработки приложения связываются с элементами управления, отвечающими за отображение и редактирование данных. Посредником в этих операциях выступает компонент TDataSource (источник данных). Для подключения к набору данных источника данных (компонент класса TDataSource) предназначено одноименное свойство

```
property DataSource: TDataSource;
```

Задание

Улучшите начатый на лабораторной работе 11 проект БД. Используя компоненты-таблицы TADOTable, TDataSource и TDBGrid научите проект БД следующим операциям:

1. Перемещаться по строкам таблицы.
2. Создавать закладки и возвращаться к ним.
3. Добавлять, редактировать и удалять строки из таблицы.
4. Фильтровать строки в таблице (например, строить фильтр по дате).
5. Производить поиск строки в таблице, по значению поля.

Внимание! При выполнении задания нельзя использовать компонент TDBNavigator.

13. Работа с полями набора данных

Вид занятия – самостоятельная работа

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access, C++ Builder (Delphi)

При формировании библиотеки VCL программисты Embarcadero реализовали целый арсенал компонентов, предназначенных для организации доступа к данным. К моменту появления Delphi перечень этих элементов управления настолько расширился, что даже простое их перечисление способно внушить благоговение любому программисту. Вместе с этим, не смотря на впечатляющий перечень классов, компоненты достаточно просты в изучении, так как построены по схожим принципам. Как уже отмечалось на предыдущем занятии, все компоненты доступа к данным обладают общим предком – классом `TDataSet`. Материал, сегодняшнего занятия, также является общим для всех компонентов доступа к данным. Он посвящен особому классу `TField` – полю набора данных.

Поле таблицы – класс `TField`

Класс `TField` (от англ. *field* поле) инкапсулирует физический объект – поле набора данных. Сам по себе класс `TField` является абстрактным и посему в первоначальном виде не используется. А вот богатое потомство класса интегрируется во все построенные на основе класса `TDataSet` компоненты. При открытии набора данных (например, при помощи компонентов `TADOTable` или `TADOQuery`) автоматически рождается ровно столько объектов `TField`, сколько столбцов физически доступно в этом наборе. Объекты-поля созданные таким образом (без явного вмешательства программиста) называют **динамическими**. Основная особенность подобных объектов в том, что они уничтожаются сразу после отключения от набора данных.

Однако, в некоторых ситуациях, и на долю программиста выпадает обязанность немного поработать. Особенно если идет речь о реализации:

- ~ полей синхронного просмотра (*lookup*);
- ~ вычисляемых полей (*calculated*);
- ~ агрегатных полей (*aggregate*).

Для этого в среде Delphi реализован специальный редактор “Fields Editor”, вызываемый по двойному щелчку кнопкой мышки по компоненту-набору данных. Все объекты-поля, созданные руками программиста, именуют **статическими**. В отличие от динамического поля, жизненный цикл его статического собрата начинается практически в момент старта приложения, продолжается до завершения его работы и ни коим образом не зависит от того, открыт или закрыт набор данных.

Один и тот же набор данных не в состоянии одновременно обладать, как динамическими, так и статическими полями. И если в наборе определено хотя бы одно статическое поле, то компонент, утратит возможность самостоятельно создавать его динамических коллег.

Теперь немного об основных обязанностях класса `TField`, потомки класса умеют:

1. Читать данные из текущей записи конкретного столбца таблицы.
2. Редактировать значения поля, текущей записи с синхронной проверкой допустимости вводимых данных.
3. Указывать компонентам отображения данных, каким образом следует представлять содержимое поля на экране.
4. Обрабатывать вычисляемые поля.
5. Искать значения поля синхронного просмотра в другом наборе данных

Классификация полей по функциональному назначению

Для того, чтобы выяснить назначение поля сразу стоит обратиться к его свойству:

property `FieldKind`: `TFieldKind`;

В подавляющем числе случаев обращение к этому свойству несет справочный характер, и редактировать хранящееся в нем значение не только излишне, но и даже опасно. Список возможных вариантов ответов не велик и представлен в таблице 13.1.

Таблица 13.1. Значения `TFieldKind`

<i>Значение</i>	<i>Описание</i>
fkData	Объект представляет собой поле данных.
fkCalculated	Это вычисляемое поле, содержимое которого определяется в обработчике события OnCalcFields() содержащего его компонента.
fkLookup	Поле синхронного просмотра.
fkInternalCalc	Внутренне вычисляемое поле.
fkAggregate	Агрегатное поле.

Самый распространенный тип поля – fkData. На физическом уровне это чаще всего поле обычной таблицы, доступ к которому мы получили с помощью компонентов TTable, TQuery или их коллег.

Классификация полей по типу обслуживаемых данных

Ключевой характеристикой поля служит тип данных, который может в нем содержаться. Для выяснения этого прочитайте значение свойства

```
property DataType: TFieldType;
```

Например, поле предназначенное для работы с 32-х разрядными целыми числами имеет тип данных ftInteger, вещественное поле – ftFloat, строковое – ftString.

Обращение к отдельному объекту-полю

Для того чтобы выяснить, какому набору данных принадлежит поле стоит обратиться к свойству

```
property DataSet: TDataSet;
```

Свойство вернет ссылку на родительский компонент.

Каждое поле владеет информацией об имени связанного с ним столбца таблицы

```
property FieldName : string;
```

В наборе данных (потомке TDataSet) поля хранятся в свойстве Fields. В этом списке, каждое поле описывается уникальным индексом.

```
property Index: Integer;
```

Изменяя индекс, мы получаем возможность корректировать последовательность полей в этом наборе. Порядковый номер поля в наборе данных редактировать нельзя. Это свойство доступно только для чтения

```
property FieldNo: Integer; //только для чтения;
```

Оно отображает физическую последовательность столбцов в наборе.

Доступ к данным поля

Для того, чтобы проверить содержит ли поле значение обращайтесь к свойству:

```
property IsNull: Boolean;
```

Если значение поля неопределенное, то свойство вернет true. Если значение поля определено, то вы имеете полное право выяснить, что там находится. Для чтения и записи данных в поле предназначается свойство

```
property Value: Variant;
```

```
if ADOTable1.Fields[0].IsNull=False then  
Result:= Table1.Fields[0].Value;
```

Для полной очистки содержимого поля стоит применять процедуру

```
procedure Clear;
```

После вызова метода значение поля будет соответствовать NULL.

Поле можно перевести в режим “только для чтения” применив свойство

```
property ReadOnly: Boolean;
```

Это свойство позволяет программисту программным образом запрещать или разрешать редактировать данные. Но перед этим стоит проверить доступно ли для правки поле на физическом уровне, ведь оно, например, может оказаться вычисляемым, т.е. не редактируемым в принципе. Для этого требуется проанализировать состояние свойства:

```
property CanModify: Boolean; //только для чтения
```

Свойство информирует нас о том, разрешено (`true`) или запрещено (`false`) модифицировать данные поля.

При вводе новой или редактировании существующей записи важную роль играет свойство:

```
property Required: Boolean;
```

Значение `true` свидетельствует о том, что это так называемое “**обязательное**” поле. Другими словами, оно должно быть заполнено, иначе (если при сохранении записи поле окажется пустым) – мы получим сообщение об ошибке.

Приведение типов

Один и тот же объект-поле способен представлять содержащееся в нем значение в виде некоторых стандартных типов данных. Для этого в классе `TField` инкапсулировано несколько свойств (табл. 13.2).

Таблица 13.2. Приведение типов данных в поле `TField`

Свойство	Описание
property AsBCD: TBcd;	Содержит значение поля в формате двоично-кодированного десятичного числа. Значения такого типа иногда применяются вместо денежных (<code>currency</code>) типов данных
property AsBoolean: Boolean;	Содержит значение в логическом формате
property AsCurrency: Currency;	Денежный формат
property AsDateTime: TDateTime;	Формат дата/время
property AsFloat: Double;	Вещественное число
property AsInteger: Integer;	Целое число
property AsSQLTimeStamp: TSQLTimeStamp;	Значение поля представлено в формате <code>TSQLTimeStamp</code> . Это один из способов хранения даты/времени, в виде записи, применяемый в серверах SQL
property AsString: String;	Строка символов
property AsVariant: Variant;	Универсальный формат

Следующая строка кода демонстрирует способ передачи данных из поля в компонент-метку.

```
Label1.Caption:=Query1.FieldByName('CityName ').AsString;
```

Или обратный пример, показывающий каким образом можно забрать значение из компонента-календаря:

```
Table1.Fields[1].AsDateTime:= DateTimePicker1.Date;
```

Размер поля

При обращении к значению поля иногда требуется знать физический размер, занимаемый этим значением. Для таких целей предназначено свойство:

```
property DataSize: Integer;
```

Свойство возвратит размер в байтах. Иногда часто это свойство применяют для того, чтобы зарезервировать область памяти для хранения содержимого поля.

Еще одно свойство:

```
property Size: Integer;
```

для поля строкового типа указывает максимальное число символов, допустимых для ввода в это поле. А для остальных типов полей – размерность в единицах, зависящих от типа данных поля.

Значение по умолчанию

Значение по умолчанию, вносимое в поле при вводе новой записи определяется в свойстве.

property DefaultExpression: String;

Как правило, в этом свойстве указываются наиболее вероятные данные, что освобождает пользователя от повинности многократного ввода одного и того же значения.

```
FieldCurrentTime.DefaultExpression='12:00:00';
```

Для наибольшей универсализации свойство сделано строковым. Это позволяет задавать значения по умолчанию для большинства типов полей.

Ограничения на ввод данных

Объекты поля способны осуществлять строгий контроль за корректностью вводимых данных. Набор свойств и методов поля позволяет, как импортировать доменные ограничения, заложенные в ядре СУБД, так и определять ограничения на уровне клиентского приложения. Благодаря этому строится гибкая и многоуровневая система защиты от ввода некорректных данных.

Ограничения на диапазон допустимых значений определяются в свойстве:

property CustomConstraint: String;

Это ни что иное, как обычная строка, в которой описывается выражение на языке SQL. Допустим, что физическое имя поля “field1”, тогда примерно так будет выглядеть выражение, определяющее допустимые пределы вводимого значения:

```
CustomConstraint='field1>=10 and field1<100';
```

При попытке пользователя ввести недопустимое значение набор данных, генерирует исключительную ситуацию. Для того, чтобы сообщение об ошибке было понятно для пользователя целесообразно описать, вероятную причину ошибки в свойстве:

property ConstraintErrorMessage: string;

Если ограничения определены на уровне СУБД, то вы можете лишь ознакомиться с ними, посмотрев содержимое свойства:

property ImportedConstraint: string;

Попытка внести изменения в импортируемые ограничения обречена на провал.

Еще одно справочное свойство предназначено для проверки – есть ли какие-нибудь ограничения на вводимые значения

property HasConstraints: Boolean; //только для чтения

Маска ввода

Маска ввода, предназначена для определения формата вводимых пользователем значений и для повышения наглядности представляемых данных.

property EditMask: TEditMask;

Индексные поля

На основе поля или группы полей в таблице может быть построен индекс. Если поле входит в состав установленного в настоящий момент индекса набора данных свойство:

property IsIndexField: Boolean;

возвращает значение true.

Отображение данных

Класс TField и его потомки не вооружены методами, позволяющими визуализировать свое содержимое. Однако компоненты линейки **Data Controls** палитры компонентов, отвечающие за отображение информации вполне способны узнать у класса TField каким образом он хотел бы представлять свои данные на экране.

Данные, предназначенные для вывода на экран, хранятся в свойстве

```
property DisplayText: String; //только для чтения
```

Все компоненты (DBText, DBEdit, DBGrid, и т.д.) при отображении данных обратятся именно к этому свойству поля.

Если поле находится в режиме редактирования, то все изменения (до момента их сохранения) аккумулируются в свойстве.

```
property Text: String;
```

В период редактирования содержимое свойств DisplayText и Text различается.

При назначении имен заголовков колонок сетка TDBGrid обязательно проверит содержимое свойства объекта Field. В этом свойстве по умолчанию хранится физическое имя поля, сетка DBGrid “подхватит” это имя и с его помощью определит заголовок колонки. Альтернативный способ назначения подписей полей обеспечивает свойство

```
property DisplayLabel : String;
```

В этом свойстве по умолчанию находится имя физическое имя поля, однако программист имеет право вместо невразумительного “ColumnCity” написать “Город”. И тогда имена колонок сразу получают осмысленные названия.

Еще одно свойство, связанное с отображением данных, называется:

```
property DisplayName: String; //только для чтения
```

Содержимое этого свойства используется при выводе сообщения об ошибке, возникшей при редактировании данных в поле.

Задание

1. Разработайте функцию, на вход которой поступает имя любой таблицы вашей БД. В результате выполнения функция возвратит все строки из заданной таблицы в многострочный редактор TМemo.
 2. Разработайте функцию, на вход которой поступает имя любой таблицы вашей БД. В результате выполнения функции вы должны получить сведения о составе полей этой таблицы, включая: имя поля, тип поля, размер данных поля.
-

14. Вычисляемые поля и поля подстановки

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access, C++ Builder (Delphi)

Опытные программисты знают, что без особой необходимости в таблицах не следует хранить информацию, дублирующую уже существующие данные. Также следует отказываться от накопления информации, которая может быть получена путем проведения простейших операций над существующими данными. Поясним это утверждение на примере. Допустим, что перед нами поставлена задача – разработать базу данных содержащую сведения о зарплате сотрудников предприятия и размере подоходного налога. В такой ситуации заводить отдельное поле для внесения в него суммы налога избыточное действие, ведь зная сумму жалования значение налога можно получать при помощи простейшей математики.

Поля подстановки

Поля подстановки (lookup fields) активно применяются программистами для построения связи между главной и справочной таблицей. Изучите схему данных, представленную на рисунке 14.1. В нашем распоряжении две таблицы. В справочную таблицу DEPARTMENT занесены названия отделов и служб предприятия. Во второй таблице SALARY вы найдете текстовое поле FIO, предназначенное для хранения фамилии и инициалов и одноименное с именем таблицы поле SALARY специализирующееся на запоминании должностного денежного оклада работника предприятия.

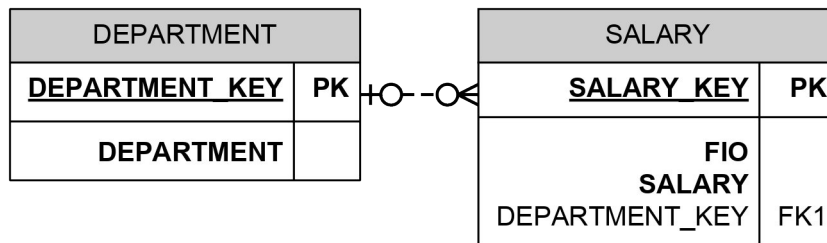


Рис. 14.1. Модель данных для примера работы с полем подстановки

Связь между таблицами осуществляется по ключевому полю DEPARTMENT_KEY. При вводе новой записи в таблицу SALARY в поле внешнего ключа DEPARTMENT_KEY передается значение, соответствующее первичному ключу в справочной таблице DEPARTMENT. Однако такое представление информации абсолютно неприемлемо для пользователя, вместо непонятных цифр в таблице ему хотелось бы видеть реальное название отдела. К счастью это не сложно, механизм, осуществляющий поиск данных в справочной таблице, с последующим показом найденных данных в исходной таблице осуществляется при посредничестве поля подстановки.

Создадим базу данных в Microsoft Access. Для этого проведем следующие операции:

1. Создаете новый проект базы данных в среде Access и сохраните ее в отдельном каталоге, например под именем demo.mdb.
2. Создайте таблицу DEPARTMENT включающую поле первичного ключа DEPARTMENT_KEY автоинкрементного типа; текстовое поле длиной 30 символов DEPARTMENT для хранения названия отдела предприятия.
3. Создайте таблицу SALARY (рис. 14.2) состоящую из четырех полей: поле первичного ключа SALARY_KEY автоинкрементного типа; текстовое поле длиной 30 символов FIO для хранения фамилии сотрудника; поле денежного типа SALARY с данными о ежемесячном жаловании и внешнего ключа DEPARTMENT_KEY целочисленного типа.
4. У поля SALARY установите значение по умолчанию равное 0.

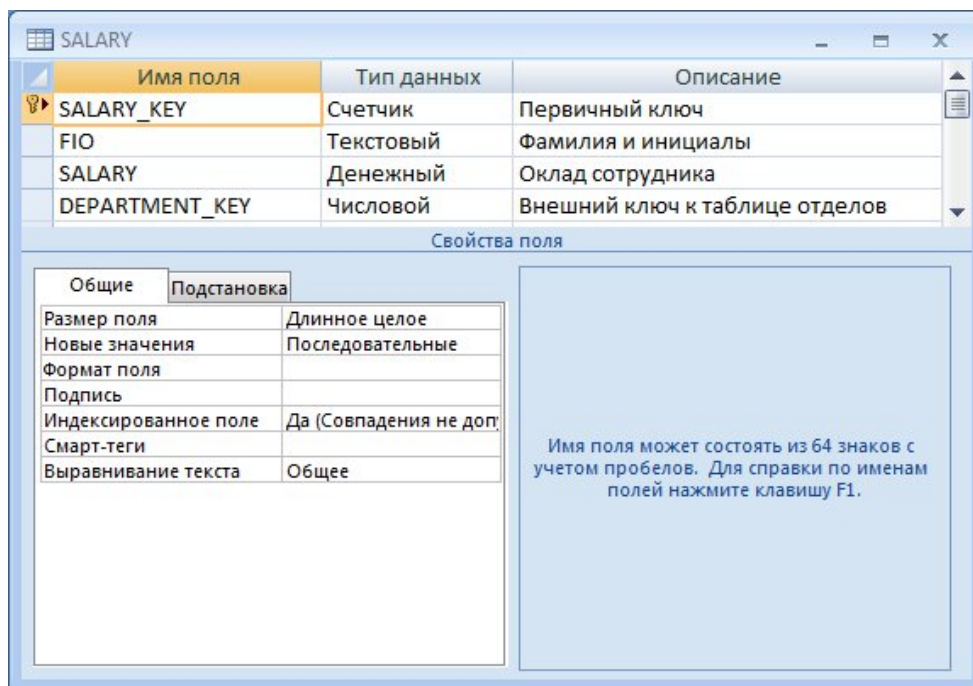


Рис. 14.2. Таблица SALARY в режиме конструктора

Введите в таблицу DEPARTMENT несколько записей и закрывайте программу Access, в ближайшее время она нам не понадобится. Теперь приступим к проектированию приложения Delphi. Начните новый проект. Переименуйте главную форму проекта Name:=frmMain и расположите на ее поверхности следующие компоненты (рис. 14.3):

- ~ Две таблицы (компоненты класса TADOTable). Первый компонент предназначен для взаимодействия с таблицей DEPARTMENT, поэтому переименуем его в tblDEPARTMENT. Второй компонент станет обслуживать таблицу SALARY, назовем его tblSALARY.
- ~ Источник данных DataSource1. Компонент предназначен для взаимодействия с таблицей SALARY, переименуйте его в dsSALARY. С помощью свойства DataSet подключите источник данных к компоненту tblSALARY.

Сетку для отображения данных DBGrid1 и компонент для управления данными DBNavigator1. С помощью свойства DataSource присоедините оба компонента к dsSALARY.

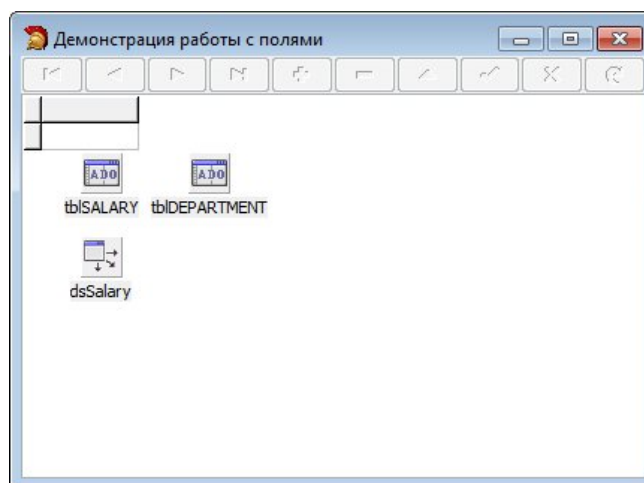


Рис. 14.3. Вид главного окна приложения

Сохраните проект под именем fielddemo.dpr в том же каталоге, где находится созданная ранее база данных demo.mdb. выберите компонент tblDEPARTMENT и, воспользовавшись свойством ConnectionString, подключите его к нашей базе данных. Скопируйте эту же строку в свойство ConnectionString второго компонента – таблицы tblSALARY, ведь ему нужен доступ к этой же самой БД.

Закончив работать с соединительной строкой, подключите компоненты к соответствующим таблицам:

- ~ компонент tblDEPARTMENT к таблице отделов и служб TableName:= 'department.db';

компонент `tblSALARY` к таблице сотрудников `TableName='salary.db'`.

Проверьте корректность подключения, активировав свойства `Active` обеих таблиц.

Дважды щелкните левой кнопкой мышки по компоненту `tblSALARY` для вызова редактора полей этой таблицы. Во всплывающем меню редактора полей выберите пункт “Add all fields” (Добавить все поля). После этого в редакторе появится все поля таблицы. В результате форма заполнится списком полей, входящими в состав нашей таблицы `SALARY`. Осуществленную нами операцию называют определением статическим полей. В результате для каждого из полей таблицы `SALARY` среда Delphi поставила в соответствие новый объект – потомок класса `TField`. Класс поля определяется типом хранящихся в нем данных, например, для текстового поля создается объект класса `TStringField`, для денежного – `TFMTBCDField`; для поля счетчика – `TAutoIncField`, и т.д. Каждый объект-поле получает имя путем сложения имен компонента-таблицы и физического имени поля. В нашем случае поле, хранящее данные о фамилии сотрудника будет названо `Table1+FIO=tblSALARYFIO`.

Редактор “New Field” не предназначен для физического добавления поля в структуру таблицы. Он лишь способен создавать экземпляры поля (потомки класса `TField`) на программном уровне. В качестве основных клиентов редактора выступают вычисляемые (*calculated*) поля и поля подстановки (*lookup*).

Вновь вызовите контекстное меню редактора полей и найдите пункт меню “New field...” (Новое поле). Это действие вызовет окно настройки (рис. 14.4) в котором следует описать характеристики поля подстановки:

- ~ в строке “Name” (имя поля) укажем имя создаваемого поля – `luDEPARTMENT`;
- ~ в списке “Type” (тип данных) определим тип данных этого поля – `String`;
- ~ в группе переключателей “Field type” (тип поля) отметим переключатель `Lookup`.

Заполняем четыре комбинированных списка в группе “Lookup definition”. Ввод данных производим слева направо, сверху вниз. Приступим. выпадающий список “Key fields” определяет поле таблицы сотрудников, являющееся внешним ключом, то есть предназначенным для взаимодействия с таблицей отделов и служб. Внешним ключом в таблице сотрудников является поле `DEPARTMENT_KEY`, поэтому останавливаем свой выбор на нем. В выпадающем списке “Dataset” указываем имя справочного набора данных. В нашем случае это таблица `tblDEPARTMENT`. Во втором ряду, в списке “Lookup Keys” требуется ввести имя первичного ключа таблицы-справочника отделов и служб – поле `DEPARTMENT_KEY`. И, наконец, в последнем списке “Result Field” необходимо выбрать имя поля, которое мы хотим видеть в результирующем наборе данных – это название отдела – `DEPARTMENT`. Закончив описание нового поля, нажимаем кнопку `OK`.

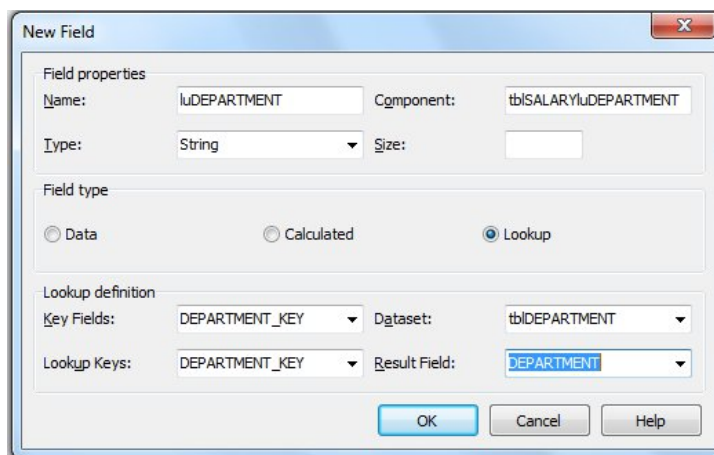


Рис. 14.4. Создание подстановочного поля

С этого момента наш проект приобретает работоспособность, но, к сожалению, нельзя сказать, что он лишен недостатков. Один из них – избыточность информации отображаемой в сетке `DBGrid1`. Этот компонент выводит на экран столбцы которые совсем необязательно видеть пользователю, это служебные поля первичного (`SALARY_KEY`) и вторичного (`DEPARTMENT_KEY`) ключей таблицы `tblSALARY`. Рекомендую вам найти свойства `visible` названных полей и присвоить им значения `false`, что сделает поля невидимыми. Кроме того имеет смысл настроить свойство `DisplayLabel` всех оставшихся видимыми полей так, чтобы сетка `DBGrid1` выводила осмысленные названия подсказывающие пользователю назначение того или иного столбца (рис. 14.5).

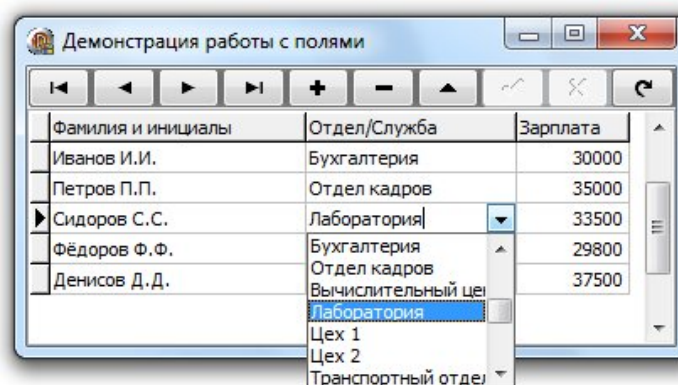


Рис. 14.5. Поле подстановки в действии

Вычисляемые поля

Вновь возвращаемся к примеру. Еще раз обратимся к всплывающему меню редактора полей `tblSALARY` и выберем пункт “New Field”. На экран будет вызван редактор, позволяющий создавать новый экземпляр поля.

В нашем случае мы конструируем вычисляемое поле. Для этого в строке `Name` указываем имя нового поля – `CALC_TAX`, в строке `Type` – определяем тип поля `Currency`. Убедитесь, чтобы в группе переключателей “Field type” была выделена кнопка “Calculated” (рисунок 14.6) и нажмите на кнопку OK. После того, как окно редактора закроется, загляните в инспектор объектов. В нем появился новый объект – поле `tblSALARYCALC_TAX`, а его свойство `FieldKind` установилось в состояние `fkCalculated`.

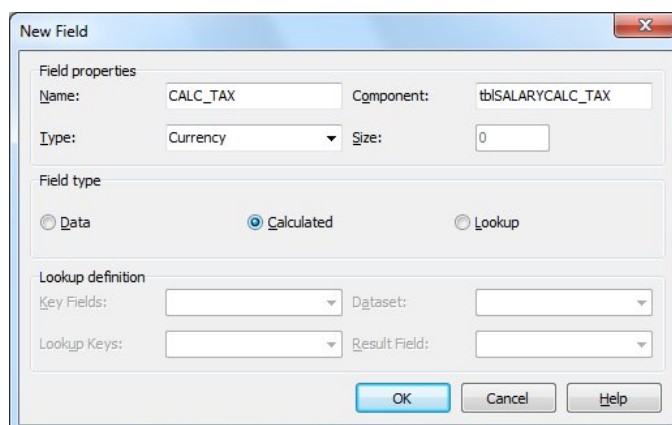


Рис. 14.6. Создание вычисляемого поля

Наступил черед поломать голову над листингом программы. У компонента `tblSALARY` находим обработчик события `OnCalcFields()` и внутри него напишем всего одну строку.

Листинг 14.1. Пример расчета значения для вычисляемого поля

```
procedure TfrmMain.tblSALARYCalcFields(DataSet: TDataSet);
begin
  tblSALARYCALC_TAX.Value:=tblSALARYSALARY.AsCurrency*13/100;
end;
```

Проверьте состояние свойства `AutoCalcFields` таблицы, оно должно быть установлено в `True`. Опять “включаем” таблицу (`Active:=True`) и запускаем проект на выполнение. Теперь отображающая данные сетка `DBGrid1` дополнилась новой колонкой, содержащей рассчитанное значение подоходного налога.

Организация отношения главная-подчиненная таблица

Один из полезных приемов, которым владеют компоненты таблицы, и в том числе `TADOTable` связан с построением отношения главная-подчиненная (master-detail) таблица. Подобное отношение имеется и в нашей демонстрационной БД книжного магазина (рис. 14.7). Таблице поставщиков `SUPPLIERS` подчиняется таблица контрактов `CONTRACTS`. Отношение подчиненности реализуется через хранящее ключ поставщика ключевое поле `SUPPLIER_KEY`.

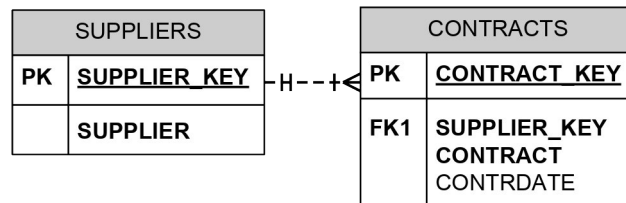


Рис. 14.7. Отношение “главный-подчиненный” между таблицами

Функциональные возможности компонента TADOTable позволяют формировать отношение главный-подчиненный при желании даже без единой строки кода. Для претворения такого плана в жизнь достаточно помнить о существовании свойств: MasterSource и MasterFields. Все перечисленные свойства имеют прямое отношение к подчиненной таблице и настраиваются в строгой последовательности. В первую очередь в свойстве:

property MasterSource: TDataSource;

указывается на подключенный к главной таблице источник данных (компонент TDataSource). Затем программист определяет перечень полей, участвующих в организации взаимодействия между таблицами. Для этого в свойстве:

property MasterFields: String;

указывают имена столбцов, входящих в первичный ключ главной таблицы.

Для практического претворения в жизнь нашего замысла можете взять за основу один из разработанных в прошлой главе проектов. Благо он уже обучен подключаться к хранилищу данных. Добавьте к предыдущей разработке новую форму и следующий перечень компонентов (рис. 14.8):

- ~ две таблицы TADOTable. Переименуйте первую таблицу в tblSUPPLIERS, этот компонент станет обслуживать таблицу поставщиков. Вторую таблицу назовем tblCONTRACTS, мы позднее подключим этот компонент к таблице контрактов.
- ~ два источника данных TDataSource. Первый источник переименуем в dsSUPPLIERS и в инспекторе объектов подключаем к таблице поставщиков (свойство DataSet:= tblSUPPLIERS). Вторым источником называем dsCONTRACTS и присваиваем свойству DataSet:= tblCONTRACTS.
- ~ две сетки TDBGrid. Сетки позволят нам увидеть содержимое таблиц. Первый компонент называем dbgSUPPLIERS и располагаем в левой части формы. Подключаем сетку к источнику данных dsSUPPLIERS (свойство DataSource:= dsSUPPLIERS). Оставшаяся сетка получит имя dbgCONTRACTS, свойство DataSource:=dsCONTRACTS.

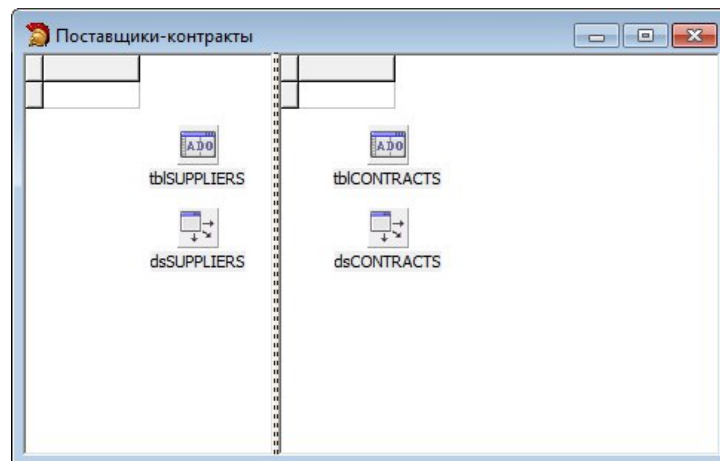


Рис. 14.8. Форма, демонстрирующая отношение главный-подчиненный

Переименуйте форму в frmSuppliersContracts и найдите событие OnShow() вызываемое в момент вывода формы на экран и внесите в него следующие строки кода.

Листинг 14.2. Настройка параметров компонентов

```
procedure TfrmSuppliersContracts.FormShow(Sender: TObject);
begin
with tblSUPPLIERS do
begin
```

```
    Connection:=frmDM.ADOConnection1;  
    TableName:='SUPPLIERS';  
    Open;  
end;  
  
with tblCONTRACTS do  
begin  
    Connection:=frmDM.ADOConnection1;  
    TableName:='CONTRACTS';  
    MasterSource:=dsSUPPLIERS;  
    MasterFields:='SUPPLIER_KEY';  
    Open;  
end;  
end;
```

Программа готова. Запустите приложение на выполнение внимательно проследите за его работой. выбор в сетке dbgSUPPLIERS того или иного поставщика приводит к автоматическому отображению в сетке dbgCONTRACTS сведений о заключенных с ним контрактов.

Задание

Дополните ваш проект БД примерами:

1. полей подстановки.
2. вычисляемых полей.
3. отношением главная таблица — подчиненная таблица.

При разработке приложения рекомендуется воспользоваться следующими компонентами: TADOTable, TDataSource, TDBNavigator и TStringGrid.

15. Управление индексами, поля BLOB

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Access, C++ Builder (Delphi)

Индекс представляет собой часть БД, в нем содержится информация об организации данных в таблицах БД. Индекс аналогичен предметному указателю, обычно приводимому в конце книги. Благодаря нему, при поиске нужной информации исключается необходимость полного просмотра книги.

Индекс отвечает за ускорение процесса сортировки и поиска записей в наборе данных.

Различают первичные и вторичные индексы. **Первичный индекс** это ничто иное, как первичный ключ таблицы – он гарантированно содержит уникальные значения для всех записей в таблице. Если поле (поля) индексирования не является первичным ключом, то оно называется **вторичным индексом**.

Особенности набора данных при работе с вторичными индексами

Одна и та же таблица вправе обзавестись некоторым перечнем вторичных индексов. Но в один и тот же момент времени таблица имеет право пользоваться услугами одного единственного индекса.

Для выяснения списка имен доступных индексов вызывается процедура:

```
procedure GetIndexNames(List: TStrings);
```

Перечень имеющихся в наличии индексов процедура возвратит в набор строк List.

Подключение индексов по его имени или по названиям входящих в индекс полей осуществляется с помощью свойств:

```
property IndexName: WideString; //имя индекса  
property IndexFieldNames: string; //поля индекса
```

Ниже предложен листинг, демонстрирующий способ выбора текущего индекса пользователем во время работы программы.

Листинг 15.1. Подключение индекса

```
procedure TForm1.ADOTable1AfterOpen(DataSet: TDataSet);  
begin  
  ADOTable1.GetIndexNames(ComboBox1.Items); //сбор индексов  
end;  
procedure TForm1.ComboBox1Change(Sender: TObject);  
begin  
  if ComboBox1.ItemIndex <> -1 then  
    ADOTable1.IndexName := ComboBox1.Items.Strings[ComboBox1.ItemIndex];  
end;
```

Сразу после соединения с таблицей (в момент возникновения события AfterOpen) производится сбор имен индексов в комбинированный список ComboBox1. Теперь пользователь приобрел право определить нужный ему режим сортировки данных, для этого он просто выбирает требуемый индекс из списка ComboBox1.

Поля BLOB

Класс TBlobField реализован для обеспечения работы с двоичными большими объектами (*binary large object*, BLOB) переменной длины. Поля этого типа и его производные представляют собой универсальное хранилище большого объема разнотипной информации. Поля BLOB широко представлены практически во всех СУБД. Например: в таблицах Paradox к таковым относятся большие текстовые поля (Memo), поля форматированного текста (FormattedMemo), графическое поле (Graphic).

Если рассуждать о специализации BLOB полей, то стоит обратить внимание на свойство:

```
property BlobType: TBlobType;
```

где

```
type TBlobType = ftBlob..ftOraClob;
```

Тип данных TBlobType это всего-навсего подмножество TFieldType. Значения, которые может принимать это тип данных, вы можете просмотреть в справочной системе. Хотя свойство BlobType доступно как

для чтения, так и для записи, но если возникла необходимость, поменять тип поля целесообразнее вызвать метод:

Почти все методы BLOB поля нацелены на решение вопросов чтения или записи данных.

Для загрузки данных в поле предназначены два метода:

```
procedure LoadFromFile(const FileName: string);
procedure LoadFromStream(Stream: TStream);
```

В первом случае требуется передать имя графического файла FileName. Реализация второго метода несколько сложнее, в нем работа осуществляется с областью памяти – потоком Stream.

Для выгрузки данных из поля реализованы два симметричных метода с аналогичными параметрами:

```
procedure SaveToFile(const FileName: string);
procedure SaveToStream(Stream: TStream);
```

Для копирования данных из другого BLOB-поля определена процедура.

```
procedure Assign(Source: TPersistent);
```

Здесь в качестве параметра передается источник данных – Source. В роли источника данных могут выступать: другие BLOB поля, набор строк TStrings, графические объекты, например TBitmap.

Факт изменения данных внутри поля можно проверить, просмотрев свойство:

```
property Modified: Boolean;
```

Значение True является признаком, что поле изменялось.

Разработаем небольшой пример, демонстрирующий работу с BLOB полем. Задача заключается в следующем: нам требуется научить приложение сохранять в таблице растровую картинку и имя файла, соответствующего этому изображению. Начнем с проектирования структуры таблицы. Для решения поставленной задачи нам достаточно определить три поля: первичный ключ PICTURE_KEY, поле PICTUREBLOB для хранения графики и текстовое поле для хранения имени файла PICTURENAME. Воспользуемся услугами Microsoft Access и создадим в ней файл БД с именем blobdemo.accdb. База данных будет содержать единственную таблицу PICTURETABLE (рис 15.1) с описанными выше характеристиками.

PICTURETABLE		
Имя поля	Тип данных	
PICTURE_KEY	Счетчик	Ключевое поле
PICTURENAME	Текстовый	Название файла с рисунком
PICTUREBLOB	Поле объекта OLE	Объект BLOB

Рис. 15.1. Структура таблицы для работы с полем BLOB

Создайте новый проект и разместите на главной форме следующий перечень компонентов: таблица – ADOTable1, диалог доступа к файлам – FileOpenDialog1, объект Image1 – для отображения рисунков, метку для отображения названия файла – StaticText1, и семь кнопок класса TSpeedButton. Первые четыре кнопки (внешний вид формы на рисунке 15.2) предназначены для перемещения по записям внутри таблицы. Назовем их следующим образом: btnFirst – для перемещения на самую первую запись, btnPrior – предыдущая запись, btnNext – следующая запись, btnLast – последняя запись. Кроме имен кнопок потребуется изменить значения их свойств tag. Для btnFirst – 0, btnPrior – 1, btnNext – 2, btnLast – 3. И осталось три кнопки, отвечающие непосредственно за редактирование данных в таблице: btnAppend – добавляет новую запись в конец набора, btnEdit – редактирует текущую запись и btnDelete – отвечает за удаление.

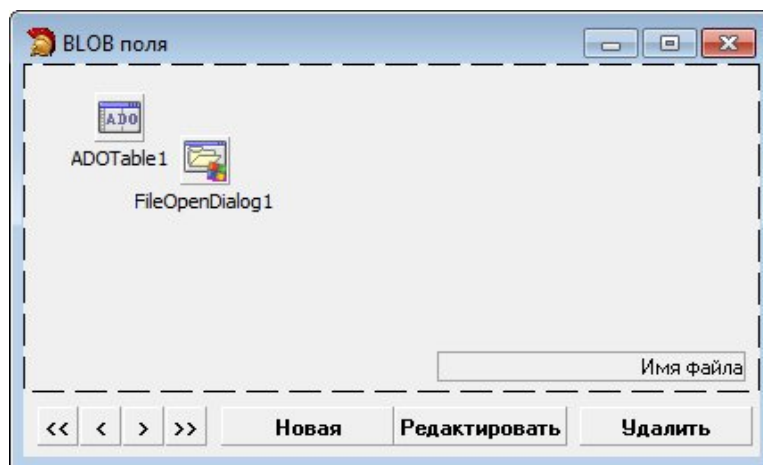


Рис. 15.2. Внешний вид формы для работы с BLOB-полем

Сохраните проект в том же каталоге где и расположен файл blobdemo.accdB. Главную форму проекта по традиции назовем frmMain.dfm, соответствующий ей модуль – main.pas, а весь проект – blobdemo.dpr.

Настройте строку соединения для компонента ADOTable1.

В секции частных объявлений опишем заголовки двух процедур SetData() и GetData(). Процедура SetData() предназначена для организации загрузки файла рисунка в BLOB поле. Для этого вызывается диалог открытия изображения. Если пользователь выберет файл и при этом таблица находится в режиме вставки (dsInsert) или редактирования (dsEdit) оба поля записи будут заполнены соответствующими данными.

Листинг 15.2. Отправка данных в поле BLOB

```

procedure TfrmMain.SetData;
begin
  if (FileOpenDialog1.Execute) and
    ((ADOTable1.State=dsInsert) or (ADOTable1.State=dsEdit)) then
    begin
      (ADOTable1.FieldByName('PICTUREBLOB') as
        TBlobField).LoadFromFile(FileOpenDialog1.FileName);
      ADOTable1.FieldByName('PICTURENAME').AsString:=
        ExtractFileName(FileOpenDialog1.FileName);
    end;
end;

```

Процедура GetData() предназначена для извлечения данных из текстового и BLOB поля. Название файла загружается в Caption компонента StaticText1. Если поле Image не пусто, то мы создаем поток M: TMemoryStream, загружаем в него данные из поля, и передаем их в компонент Image1. Обращая внимание, на то что, перед передачей данных из потока его следует установить на нулевую позицию. В заключении освобождаем ресурсы, занимаемые потоком, вызвав метод Free().

Листинг 15.3. Получение данных из поля BLOB

```

procedure TfrmMain.GetData;
var M : TMemoryStream;
begin
  with ADOTable1 do
    begin
      StaticText1.Caption:=FieldByName('PICTURENAME').AsString;
      if (FieldByName('PICTUREBLOB') as TBlobField).IsNull=False then
        begin
          M:=TMemoryStream.Create;
          (FieldByName('PICTUREBLOB') as TBlobField).SaveToStream(M);
          M.Position:=0;
          Image1.Picture.Bitmap.LoadFromStream(M);
          M.Free;
        end else Image1.Picture.Bitmap.FreeImage;
    end;
end;

```

После того, как были созданы ключевые процедуры чтения и записи данных, переходим к описанию основных событий. Первое из них – событие `OnCreate()` главной формы проекта. В нем мы осуществим подключение компонента `ADOTable1` к базе данных, определим параметры диалога открытия изображений.

Листинг 15.4. Создание формы

```

procedure TfrmMain.FormCreate(Sender: TObject);
var s:string;
    FT:TFileTypeItem;
    const db='\\blobdemo.accdb';
begin
    {подключаемся к базе данных}
    s:=ExtractFilePath(Application.ExeName)+db;
    if FileAge(s)>0 then
        begin
            ADOTable1.ConnectionString:='Provider=Microsoft.ACE.OLEDB.12.0;
                                         Data Source='+s+';Persist Security Info=False';
            ADOTable1.TableName:='PICTURETABLE';
            ADOTable1.Open;
            GetData;
            FT:=FileOpenDialog1.FileTypes.Add;
            FT.FileMask:='*.bmp';
            FT.DisplayName:='Растровый файл';
        end
        else
            begin
                Application.MessageBox(pChar('Не найден файл'+#13+S), 'Ошибка', MB_OK);
                Application.Terminate;
            end;
        end;
end;

```

выберите кнопку `btnFirst`, отвечающую за перевод курсора на самую первую запись в таблице и опишите ее обработчик события `OnClick()`. Особенность события в том, что оно будет применяться для всех четырех кнопок перемещения. Если помните, кнопки перемещения по записям таблицы различаются свойством `tag`, `btnFirst` – 0, `btnPrior` – 1, `btnNext` – 2, `btnLast` – 3. Анализируя свойство `tag` в селекторе `Case` мы вызываем соответствующий метод таблицы.

Листинг 15.5. Щелчок по кнопкам перемещения по строкам таблицы

```

procedure TfrmMain.btnFirstClick(Sender: TObject);
begin
    with ADOTable1 do
        begin
            {перемещаем курсор в соответствии со значением свойства tag кнопок}
            case (sender as TComponent).Tag of
                0: First;    //на первую запись
                1: Prior;    //на предыдущую запись
                2: Next;     //на следующую запись
                3: Last;     //на последнюю запись набора данных
            end;
            GetData; //получаем данные из таблицы
        {с помощью методов BOF и EOF проверяем местоположение текущей записи}
        btnFirst.Enabled:=NOT(BOF); {кнопка отключается, если курсор на первой записи}
        btnPrior.Enabled:=NOT(BOF);
        btnNext.Enabled:=NOT(EOF); {кнопка отключается, если курсор на последней записи}
        btnLast.Enabled:=NOT(EOF);
    end;
end;

```

Нам осталось описать действия добавления, редактирования и удаления записи.

Листинг 15.6. Щелчки по кнопкам редактирования строк таблицы

```

procedure TfrmMain.btnAddClick(Sender: TObject); //добавление
begin

```

```

ADOTable1.Append; {добавление новой записи в конец набора}
SetData; {вызов выбора изображения}
GetData; {передаем данные в компонент Image1 и StaticText1}
end;

procedure TfrmMain.btnEditClick(Sender: TObject); //редактирование
begin
    ADOTable1.Edit; {перевод таблицы в режим редактирования}
    SetData;
    GetData;
end;

procedure TfrmMain.btnDeleteClick(Sender: TObject); //удаление
begin
    if MessageBox(frmMain.Handle,
        pWideChar('Удалить запись?'),
        pWideChar('Удаление'),
        MB_YESNO + MB_ICONQUESTION)=idYes then
    begin
        ADOTable1.Delete; {удаление записи}
        Image1.Picture.Bitmap.FreeImage; {освобождаем ресурс картинки}
        StaticText1.Caption:=''; {очищаем заголовок метки}
        GetData;
    end;
end;

```

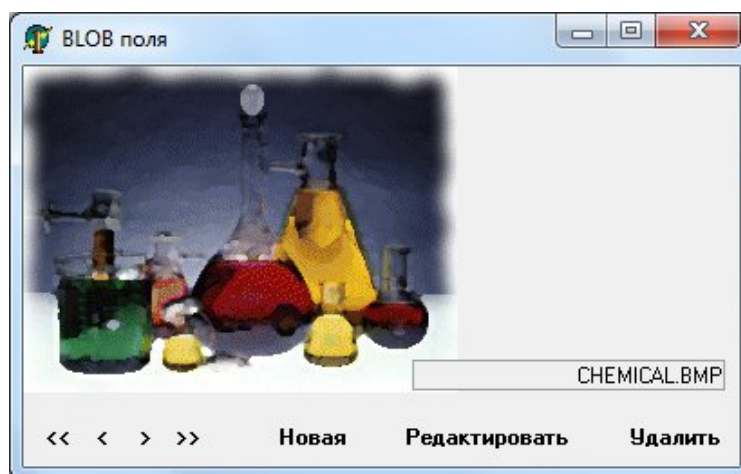


Рис. 15.3. Внешний вид приложения

Задание

Внесите в свой проект БД следующие доработки:

1. Возможность сортировать данные по индексному полю.
 2. Добавьте к одной из таблиц поле предназначенное для хранения фотографии, и разработайте форму для просмотра, ввода, редактирования и удаления данных из этого поля.
-

16. Разработка интерфейса клиентского приложения

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – среда Microsoft Access, C++ Builder (Delphi)

Источник данных – компонент TDataSource

Невизуальный элемент управления TDataSource минималист, у него всего 4 опубликованных свойства, пара методов и плюс три обработчика событий. Но, не смотря такую скромность, источник данных самым выдающимся образом решает следующие задачи:

Во-первых, источник данных незаменим при организации взаимодействия между компонентом доступа к набору данных (элементы TADOTable, TSQLQuery, TIBStoredProc, и т.п.) и элементами управления данными (TDBNavigator, TDBEdit, TDBGrid). Источник берет на себя ответственность за передачу информации в компоненты отображения данных и возвращает назад осуществленные пользователем изменения (рис. 16.1).

Во-вторых, компонент TDataSource способен организовать связь между главным и подчиненным наборами данных. Этот вопрос мы уже рассматривали в лабораторных работах 13 и 14.

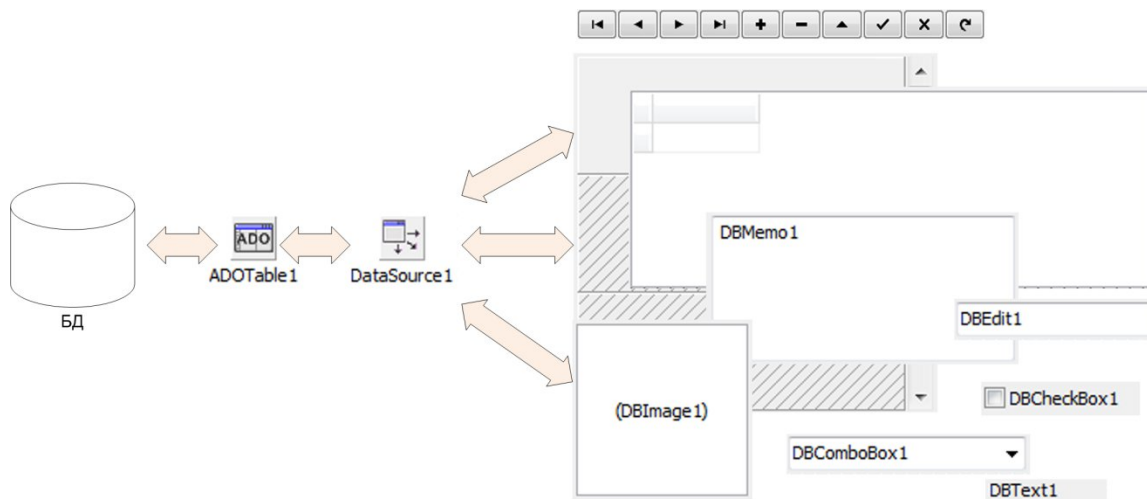


Рис. 16.1. Место компонента TDataSource в приложении БД

Ключевым свойством компонента по праву считается

property DataSet: TDataSet;

Благодаря этому свойству и поддерживается связь с набором данных. Свойство доступно как в период разработки приложения, так и во время выполнения. Во время разработки инспектор объектов предоставит программисту список из всех доступных в приложении потомков TDataSet (таблиц, запросов, хранимых процедур), нам лишь останется выбрать нужный из них.

Для проверки, связан ли в текущий момент источник данных с набором DataSet, предназначена функция

function IsLinkedTo(DataSet: TDataSet): Boolean;

Контроль состояния набора данных, к которому в данный момент присоединен DataSource, осуществляет с помощью свойства:

property State: TDataSetState; //только для чтения

Оставшиеся два свойства имеют прямое отношение к элементам отображения данных. Первое из них приспособлено для их централизованного включения/отключения:

property Enabled: Boolean;

Второе свойство (в состоянии true) обеспечивает автоматический перевод в режим редактирования связанного с DataSource компонента – набора данных

property AutoEdit: Boolean;

Такой переход осуществляется при внесении пользователем изменений в элементах отображения данных. Еще один способ перехода в редактирование сводится к вызову метода:

```
procedure Edit;
```

При любом изменении в данных (например: при переходе от одной записи к другой или в момент модификации записи) вызывается обработчик события:

```
property OnDataChange: TDataChangeEvent;  
type TDataChangeEvent = procedure (Sender: TObject; Field: TField) of object;
```

здесь параметр `Field` указывает, в каком из полей произошли изменения. Но если одновременно модифицировалось два (и более) поля параметр окажется пустым – `nil`.

При каждом изменении состояния связанного набора данных происходит событие:

```
property OnStateChange: TNotifyEvent;
```

Перед передачей в базу сделанных пользователем модификаций вызывается:

```
property OnUpdateData: TNotifyEvent;
```

Данный обработчик события может использоваться для проверки правильности данных перед их сохранением.

Общие черты компонентов отображения данных

Практически все компоненты отображения данных построены на базе стандартных элементов управления и являются их логическим развитием. Соединение компонента с набором данных реализуется при прямом посредничестве элемента управления `TDataSource`. Поэтому, первой и наиболее важной объединяющей чертой всех элементов управления, предназначенных для работы с данными, является не что иное, как источник данных. Для подключения к источнику потребуется передать имя необходимого компонента `TDataSource` в свойство

```
property DataSource: TDataSource;
```

Компоненты отображения данных, специализирующиеся на предоставлении доступа к отдельному полю набора данных (`TDBText`, `TDBEdit`, `TDBMemo`, и т.д.) для подключения к столбцу потребуют передать его имя в свойство

```
property DataField: String;
```

Если соединение осуществилось без проблем, то в свойстве

```
property Field: TField; //только для чтения
```

мы обнаружим ссылку на соответствующий обслуживаемому столбцу таблицы экземпляр объекта-поля `TField`.

Все компоненты, способные не просто отображать, но еще и редактировать данные, переводятся в режим только для чтения свойством

```
property ReadOnly: Boolean;
```

Сетка базы данных – компонент *TDBGrid*

Компонент `TDBGrid` один из наиболее часто используемых элементов управления для отображения и редактирования данных. Его основное преимущество над всеми своими коллегами заключается в том, что сетка способна вывести на экран сразу несколько строк и столбцов набора данных, все остальные компоненты (за исключением `TDBCtrlGrid`) ориентированы на работу только с текущей записью и одним единственным полем.

Каждая подключенная к активному набору данных сетка знает, содержимое каких именно полей предстоит ей обслуживать. Этой информацией сетка способна поделиться и с нами. Свойство

```
property Fields[Index: Integer]: TField; //только для чтения
```

предоставляет доступ ко всем объектам `TField` набора данных, как к элементам массива. Общее количество элементов в массиве хранится в свойстве

```
property FieldCount: Integer; //только для чтения
```

Объект-поле, соответствующий выделенной в сетке ячейке

property SelectedField: TField;

Индекс выбранной колонки находится в свойстве

property SelectedIndex: Integer;

Свойства SelectedField и SelectedIndex не только информационные. С их помощью вы получаете возможность выбирать требуемую колонку в коде программы.

Некоторые особенности поведения сетки определяются свойством

property Options: TDBGridOptions;
TDBGridOptions = **set of** TDBGridOption;

Таблица 16.1. Описание опций сетки

Значение	Описание
dgEditing	Сетка допускает редактирование своего содержимого, но при условии, что в состав опций не входит значение dgRowSelect.
dgAlwaysShowEditor	Ячейки сетки постоянно готовы к редактированию своих данных, пользователю даже нет необходимости нажимать клавиши Enter или F2.
dgTitles	Самая верхняя строка сетки отводится для показа заголовков колонок.
dgIndicator	Напротив текущей записи появляется указатель.
dgColumnResize	Разрешается перестановка местами и изменение ширины колонок.
dgColLines	Колонки сетки разделяются линией.
dgRowLines	Строки сетки разделяются линией.
dgTabs	Нажатие клавиш Tab и Shift+Tab позволяет перемещаться между ячейками сетки.
dgRowSelect	Пользователь теряет возможность выбрать отдельную ячейку сетки. Вместо этого в сетке выделяется вся строка целиком.
dgAlwaysShowSelection	Сетка продолжает выделять цветом выбранные пользователем ячейки, даже если она сама потеряла фокус ввода.
dgConfirmDelete	Перед удалением строки из сетки вызывается диалоговое окно предлагающее подтвердить эту операцию.
dgCancelOnExit	Автоматически отменяет несохраненные изменения в данных при потере сеткой фокуса ввода.
dgMultiSelect	Допускается одновременный выбор более одной строки сетки.

В таблице 16.2 представлены шесть событий компонента TDBGrid обеспечивающих взаимодействие пользователя и элемента управления.

Таблица 16.2. События сетки TDBGrid

Событие	Описание
property OnColEnter: TNotifyEvent;	Событие генерируется в момент получения фокуса ввода одной из колонок сетки.
property OnColExit: TNotifyEvent;	Одна из колонок сетки теряет фокус ввода.
type TDBGridClickEvent = procedure (Column: TColumn) of object ;	Событие вызывается при щелчке кнопкой мышки в ячейке сетки. Здесь Column – колонка, которой принадлежит данная ячейка.
property OnCellClick: TDBGridClickEvent;	
type TDBGridClickEvent = procedure (Column: TColumn) of object ;	Щелчок мышкой по заголовку колонки.
property OnTitleClick: TDBGridClickEvent;	
type TMovedEvent = procedure (Sender: TObject; FromIndex, ToIndex: Longint) of object ;	Событие вызывается после перемещения колонки из позиции FromIndex в позицию ToIndex.
property OnColumnMoved: TMovedEvent;	
property OnEditButtonClick: TNotifyEvent;	Щелчок по кнопке внутри ячейки сетки.

Статический текст – компонент *TDBText*

Элемент управления *TDBText* предназначен только для отображения текущего значения поля и не имеет никаких навыков по редактированию связанных с ним данных. Для подключения компонента к набору данных достаточно выполнить стандартные операции определения источника данных (свойство *DataSource*) и указать какое именно поле нас интересует (свойство *DataField*).

Для обеспечения автоподстройки размеров компонента при выводе текста с различным количеством символов установите в *true* свойство:

property *AutoSize*: Boolean;

Если текстовые данные настолько необъятны, что ни каким образом не желают помещаться в одну строку, следует перевести в *true* свойство:

property *WordWrap*: Boolean;//по умолчанию false

В этом случае элемент управления вспомнит про свои навыки вывода информации в несколько строк.

Строка ввода БД – компонент *TDBEdit*

Компонент класса *TDBEdit* представляет собой строку ввода способную не только отображать но и редактировать текущее значение поля.

Многострочный текстовый редактор БД – *TDBMemo*

Элемент управления *TDBMemo* спроектирован для работы с текстовыми полями больших размеров или с BLOB полями, специализирующихся на хранении текстовых данных.

Изображение БД – компонент *TDBImage*

Элемент *TDBImage* специализируется на обработке графических данных, содержащихся в BLOB полях. Ключевое свойство компонента:

property *Picture*: TPicture;

Оно инкапсулирует весь функционал для работы с растровой графикой и метафайлами. Для ускорения работы с BLOB полями в элементе управления объявлено свойство:

property *AutoDisplay*: Boolean;

Задача свойства – включение/отключение автозагрузки данных из поля. Если свойство установлено в *false*, то для инициализации процесса загрузки изображения в компонент вызывают метод

procedure *LoadPicture*;

Список БД – *TDBListBox*

Список *TDBListBox* отображает текущее значение поля (при условии совпадения его с одним из элементов списка) и позволяет пользователю передавать в поле таблицы одно из заранее подготовленных текстовых значений. Перечень допустимых значений храниться в унаследованном свойстве:

property *Items* : TStrings;

Заполнение списка строк может осуществляться как в период разработки проекта во встроенном редакторе, так и во время работы приложения.

Комбинированный список БД – *TDBComboBox*

Комбинированный список выбора *TDBComboBox* призван решать задачи по выбору значения из списка. Текущее значение связанного с компонентом поля отображается в строке редактирования элемента управления. При желании пользователь может передать в базу данных одно из значений, хранящихся в списке компонента. Список значений содержится в свойстве *Items*, индекс текущего значения – *ItemIndex*.

Флажок БД – *TDBCheckBox*

Элемент позволяет пользователю согласиться или отказываться от условия. Хотя в первую очередь флажок нацелен на взаимодействие с полями данных типа Boolean, но его также можно научить передавать в базу текстовые значения. Эта возможность осуществляется с помощью двух свойств:

property *ValueChecked*: String; //значение на включение флажка

```
property ValueUnchecked: String;    //значение на отключение флажка
```

по умолчанию содержащими значения `True` и `False` соответственно. Теперь, если в присоединенном к компоненту текстовом поле базы данных обнаруживается значение `True`, то этот факт найдет свое отражение – в виде “галочки” на поверхности элемента. Значение `False` наоборот – прогонит всех галок. Компонент способен отслеживать одновременно несколько значений, но в этом случае значения должны быть разделены точкой с запятой

```
DBCheckBox1.ValueChecked := 'Да;Вкл';
DBCheckBox1.ValueUnchecked := 'Нет;выкл';
```

Для управления компонентом из программы потребуются свойства:

```
property Checked: Boolean;
```

и

```
type TCheckBoxState = (cbUnchecked, cbChecked, cbGrayed);
property State: TCheckBoxState;
```

Радиогруппа БД – TDBRadioGroup

Компонент реализует группу кнопок выбора, состояние которых определяется значением текущего поля присоединенного набора данных. Одновременно может быть отмечена только одна из кнопок группы. При выборе пользователем кнопки, связанное с ним значение передается в базу данных.

Список заголовков кнопок определяется содержимым свойства `Items`:

```
property Items: TStrings;
```

Перечень возможных значений в свою очередь заполняется в свойстве:

```
property Values: TStrings;
```

Оба свойства доступны во время разработки и во время выполнения программы. Отдельно подчеркну, что количество строк в обоих списках должно быть одинаковым.

Текущее значение связанного поля мы можем выяснить из свойства:

```
property Value: String;
```

Индекс выбранной кнопки находится в свойстве:

```
property ItemIndex: Integer;
```

Компонент – TDBCtrlGrid

Компонент `TDBCtrlGrid` представляет собой набор динамически создаваемых панелей. После открытия, связанного с ним набора данных компонент самостоятельно создает несколько панелей. Каждая панель ассоциируется с отдельной записью из набора данных. Главная черта `TDBCtrlGrid` в том, что этот элемент управления может являться владельцем других компонентов, предназначенных для работы с БД (рис. 16.2).

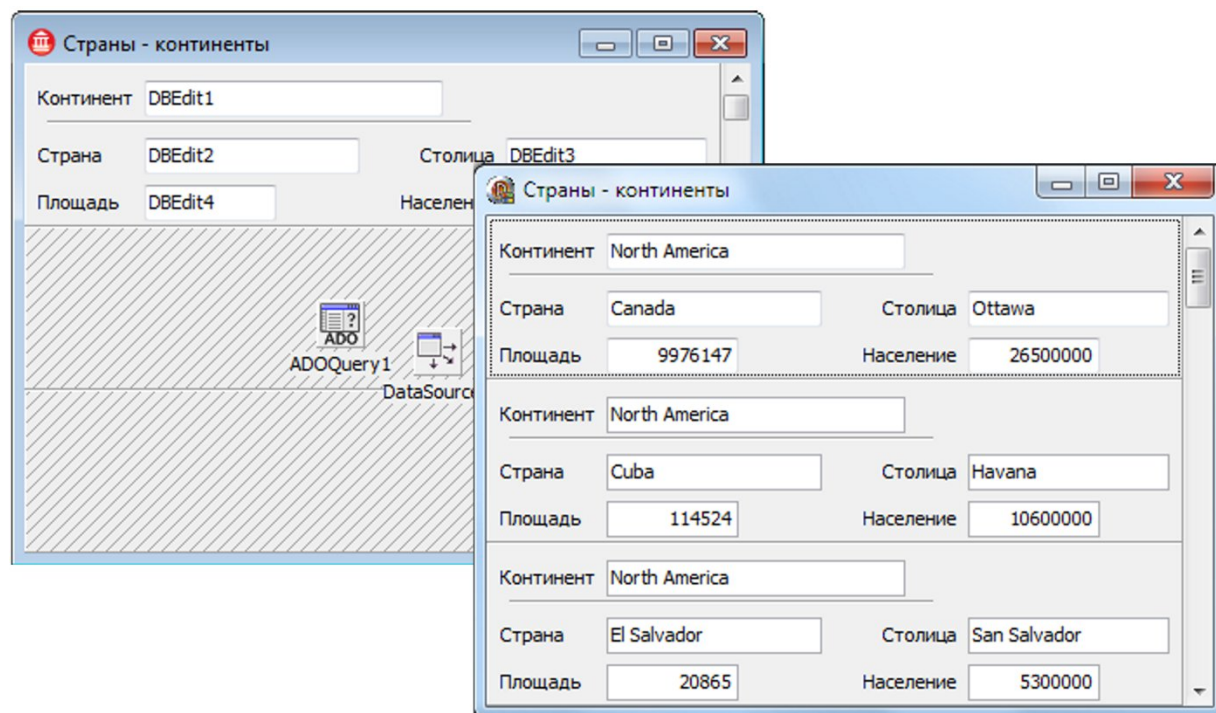


Рис. 16.2. Работа с компонентом *TDBCtrlGrid*

Порядок следования записей в компоненте *TDBCtrlGrid* определен свойством

property Orientation: *TDBCtrlGridOrientation*;
type *TDBCtrlGridOrientation* = (goVertical, goHorizontal);

По умолчанию свойство принимает значение *goVertical*, что соответствует последовательности сверху-вниз.

По умолчанию панели компонента размещаются в одном столбце. Для изменения этой традиции требуется настроить свойство

property ColCount: Integer;

Количество рядов видимых в клиентской области элемента управления определяется свойством:

property RowCount: Integer;

За геометрические размеры панелей отвечают свойства:

property PanelHeight: Integer;

property PanelWidth: Integer;

Индекс панели текущей записи хранится в свойстве:

property PanelIndex: Integer;

Это же свойство можно применять при необходимости переместиться к панели с определенным номером.

Навигатор – *TDBNavigator*

Навигатор по набору данных визуально представляет собой группу кнопок (рисунок 16.3). Компонент инкапсулирует в себе золотой минимум методов достаточных для создания элементарного приложения работающего с таблицей базы данных. Сразу после подключения к источнику данных при помощи свойства *DataSource* элемент управления сможет обеспечить перемещение по записям в наборе, вставку, редактирование и удаление записи – для этого достаточно щелкнуть по соответствующей кнопке. По умолчанию компонент содержит 10 кнопок, однако по желанию программиста вполне можно отказаться от показа ненужных. Для этого надо настроить свойство

property VisibleButtons: *TButtonSet*;

type *TButtonSet* = **set** of *TNavigateBtn*;

type *TNavigateBtn* = (nbFirst, nbPrior, nbNext, nbLast, nbInsert, nbDelete, nbEdit, nbPost, nbCancel, nbRefresh);

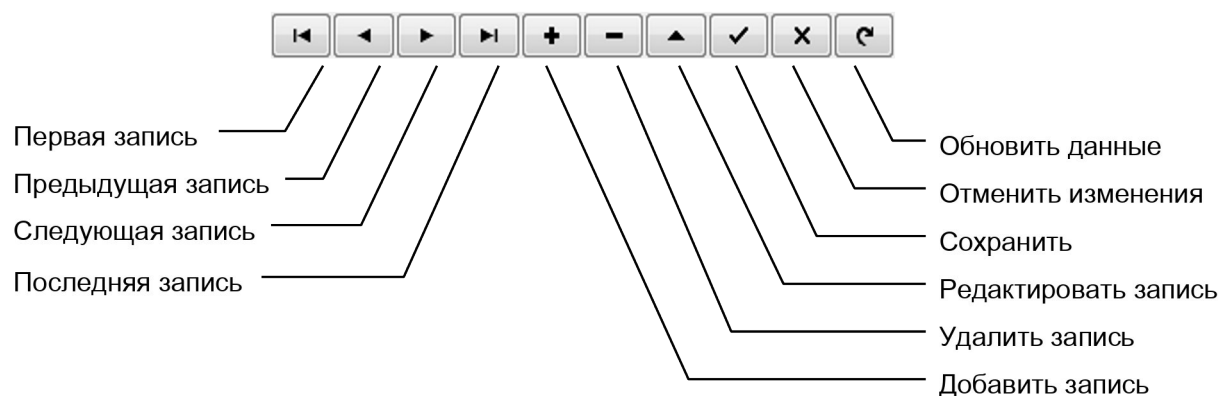


Рис. 16.3. Элемент управления *TDBNavigator*

Как и практически все визуальные элементы управления VCL навигатор обладает элементарным обработчиком события `OnClick()`. С той только разницей, что он способен различать по какой из кнопок был произведен щелчок.

property `OnClick`: `ENavClick`;

type `ENavClick` = **procedure** (Sender: `TObject`; Button: `TNavigateBtn`) of object;

для этого в событии объявлен дополнительный параметр – `Button`.

Задание

Используя полученные на занятии знания, разработайте интерфейс для вашего приложения БД. В приложении должны быть обязательно задействованы компоненты `TDataSource`, `TDBGrid`, `TDBCtrlGrid`, `TDBEdit`, `TDBListBox` (или `TDBComboBox`), `TDBCCheckBox`, `TDBRadioGroup` и другие компоненты на ваш выбор.

17. Создание БД, таблиц и представлений в Microsoft SQL Server

Вид занятия – практическое занятие

Время занятия – 2 часа

Программное обеспечение – Microsoft SQL Server

К настоящему времени насчитывается не более полутора десятков современных систем управления базами данных. Среди них наибольшей популярностью пользуются серверы баз данных Oracle, Informix, Interbase и Microsoft SQL Server.

Создание базы данных

В терминологии Microsoft SQL Server (MS SQL) база данных это логический объект, в которые MS SQL помещает таблицы, индексы и другие объекты. Физически база данных располагается в одном (или нескольких файлах).

Для создания базы данных можно воспользоваться программой SQL Server Enterprise Manager:

1. Запустите SQL Server Enterprise Manager. выберите сервер, с которым вы станете работать и выделите папку “Databases”.
2. выберите команду “Действие – Создать”.
3. В окне “Database properties” определите параметры новой БД (как минимум название БД, допустим “Demo”).
4. Щелкните по кнопке ОК.

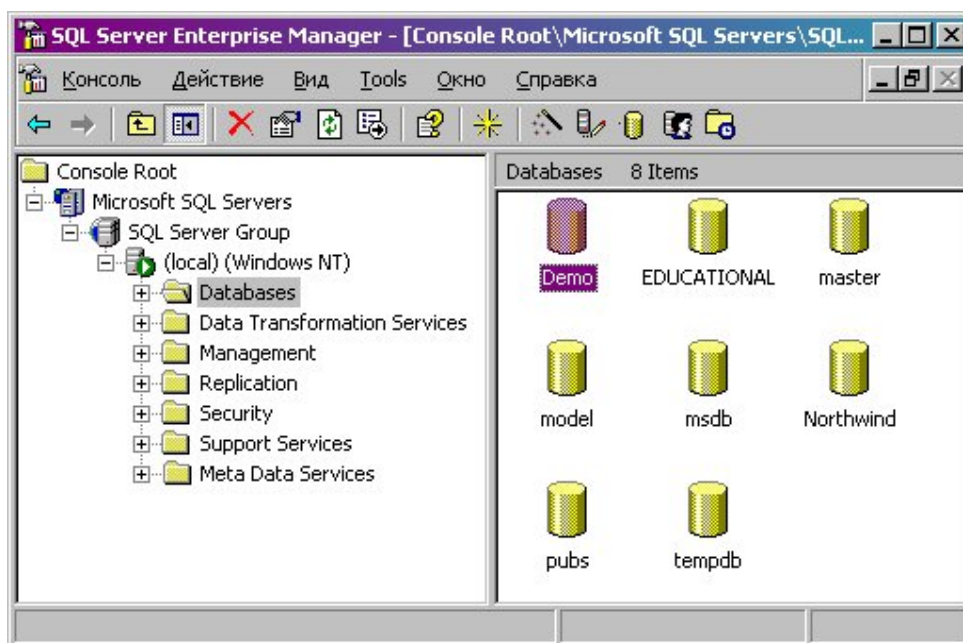


Рис. 17.1. Окно SQL Server Enterprise Manager с созданной БД “Demo”

В результате MS SQL создаст, пустую БД (рис. 17.1). Такого же результата можно добиться, воспользовавшись командой SQL “Create database”, рассмотренной на 8-м занятии.

Для внесения изменений в параметры вновь созданной БД достаточно выделить пиктограмму БД и в контекстном меню воспользоваться командой Action – Properties (Действие – Свойства).

Удаление базы данных

При удалении БД вместе с ней удаляются все принадлежащие ее объекты, и освобождается пространство, занимаемое БД на диске. Для удаления БД выберите команду “Действие – Удалить”.

Альтернативным способом “борьбы” с ненужной БД является использование команды SQL Drop Database.

Создание таблиц

Для создания таблиц в MS SQL используется программа SQL Server Enterprise Manager или команда Create Database. Рассмотрим процесс создания таблиц на примере схемы данных, представленной на рисунке 17.2.



Рис. 17.2. Схема данных БД “Demo”

Для создания новой таблицы в SQL Server Enterprise Manager выберите нужный сервер, а затем откройте папку Databases и БД “Demo” (которой будет принадлежать таблица). Затем выполните команду: “Действие – Создать – Таблица”. В результате на экран компьютера будет выведено окно редактора таблицы (рис. 17.3).

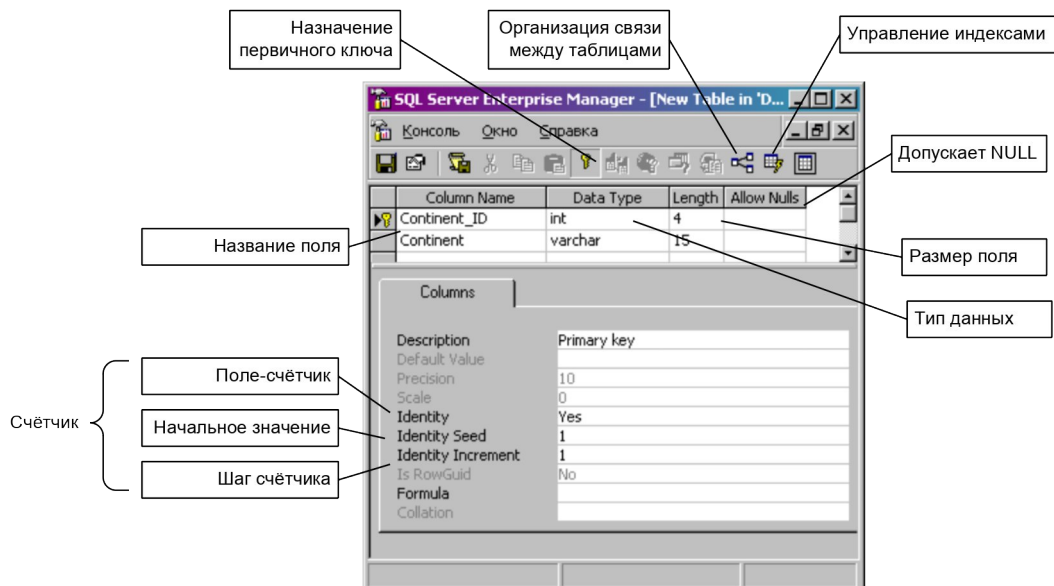


Рис. 17.3. Окно создания таблицы

Завершив описание будущей таблицы, сохраните изменения. Для этого нажмите на кнопку с изображением дискеты. Вам останется ответить на вопрос об имени создаваемой таблицы.

При создании таблицы Country необходимо позаботиться об организации связи один ко многим между таблицей стран и континентов. Для этого в состав полей таблицы Country введено поле Continent_ID, которое станет содержать внешний ключ (Foreign Key). Для определения связей между главной и подчиненной таблицами на панели инструментов требуется нажать кнопку “Manage Relationships...”. На рисунке 17.4. показано окно редактора свойств таблицы с активной страницей “Relationships”.

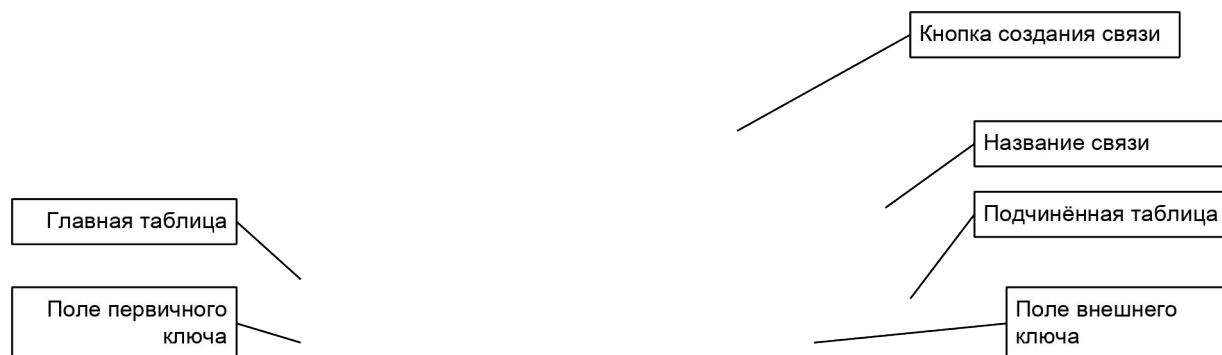


Рис. 17.4. Страница определения связи между таблицами

Ограничения Foreign Key и Primary Key гарантируют, что соответствующие строки связанных таблиц всегда останутся согласованными. Помимо ограничения внешнего ключа MS SQL сервер позволяет накладывать на поля таблицы дополнительные ограничения:

1. Ограничение поддержки уникальности (Unique). Применяется к любому столбцу таблицы с целью предотвращения ввода в него повторяющихся значений.
2. Ограничение проверки данных (Check). Налагает дополнительные условия на значения, которые можно ввести в один или несколько столбцов таблицы. Одним из примеров ограничения может служить диапазон вводимых значений.
3. Значение по умолчанию (Default). Применяется для того, чтобы автоматически вводить в столбец некоторое значение.

Пример создания таблиц средствами Transact SQL

Все сделанные нами операции по созданию таблиц можно представить в виде нескольких строк команд на языке Transact SQL (диалект SQL, используемый в MS SQL).

Листинг 17.1. Код TSQL для создания таблиц для СУБД MS SQL

```

/*Создание таблицы континентов*/
CREATE TABLE Continent
    (Continent_ID INTEGER IDENTITY (1, 1) PRIMARY KEY,
    Continent VARCHAR(15) NOT NULL UNIQUE)
/*Создание таблицы стран*/
CREATE TABLE Country
    (Country_ID INTEGER IDENTITY (1, 1) PRIMARY KEY,
    Continent_ID INTEGER,
    Country VARCHAR(20) NOT NULL UNIQUE,
    City VARCHAR(20) NOT NULL UNIQUE,
    Population INT,
    Area INT,
    CHECK (Population>0), CHECK (Area>0),
    FOREIGN KEY (Continent_ID) REFERENCES Continent (Continent_ID));
  
```

Создание представлений

Одним из способов показа информации в MS SQL является представление.

Представление (view) это хранимое определение оператора SELECT.

Чтобы создать новое представление необходимо сделать следующие шаги:

1. Запустите SQL Server Enterprise Manager.
2. Находясь в текущей базе данных, выберите команду “Создать – View...”.
3. В окне “New View” выберите таблицы и поля, включаемые в представление (рис. 17.5).
4. Закройте окно.
5. Назначьте имя нового представления, например “vn_Continent_Country”

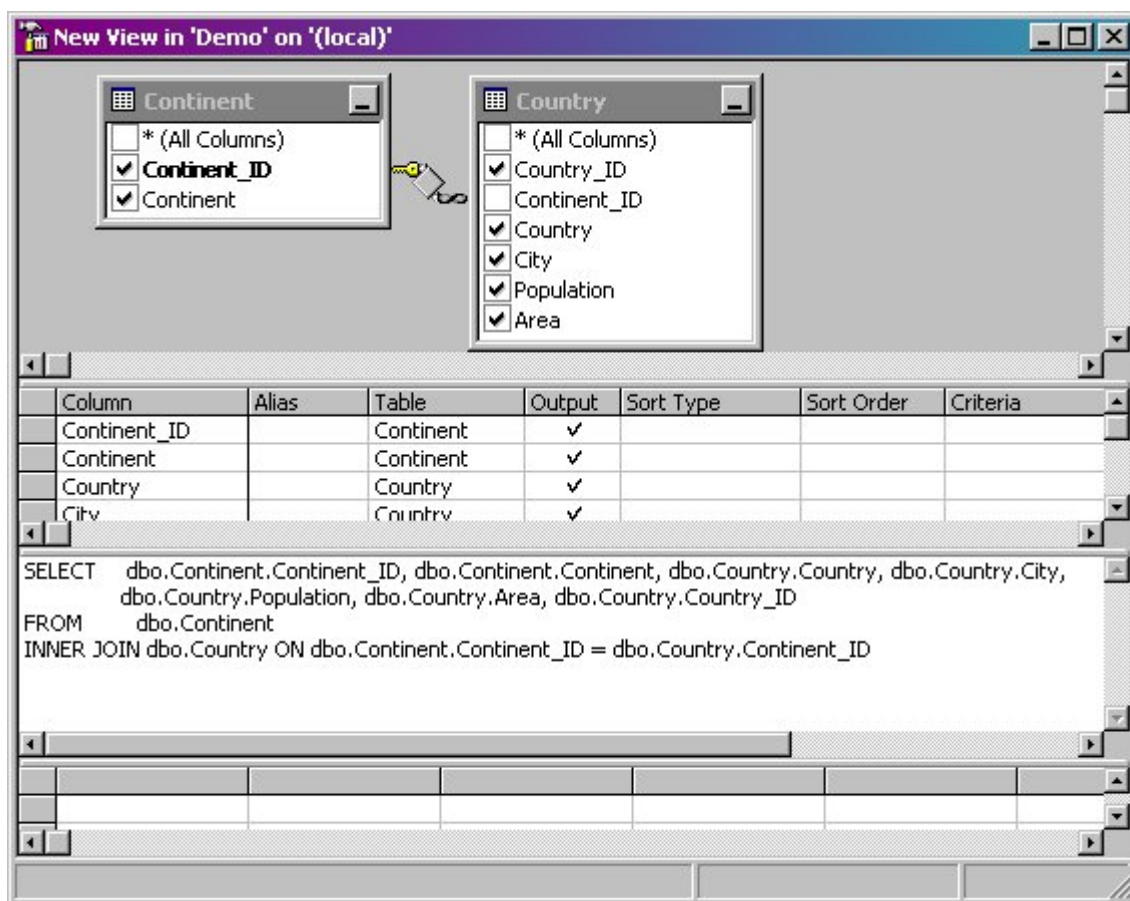


Рис. 17.5. Создание представления с помощью мастера

Этого же результата можно добиться, выполнив следующую команду на языке SQL

Листинг 17.2. Инструкция TSQL для создания представления для СУБД MS SQL

```
CREATE VIEW vn_Continent_Country
AS
SELECT Continent.Continent_ID, Continent.Continent, Country.Country,
Country.City, Country.Population, Country.Area, Country.Country_ID
FROM Continent
INNER JOIN Country ON Continent.Continent_ID = Country.Continent_ID
```

Задание

В соответствии со схемой данных разработанной на 4-й лабораторной работе создайте свой проект БД средствами Microsoft SQL Server.

18. Введение в Microsoft Transact SQL

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – Microsoft SQL Server

Каждая из современных реляционных СУБД внесла свои дополнения в стандартизированный еще в прошлом веке язык структурированных запросов SQL-92. Это занятие посвящено некоторым особенностями одного из самых известных диалектов SQL – языка Transact-SQL.

В рамках Transact-SQL практически полностью поддержан язык SQL-92, кроме того, Transact-SQL обладает несколькими сотнями новых функций и хранимых процедур.

Определение и использование переменных

Переменные в Transact-SQL могут быть локальными или глобальными. Локальные переменные определяются с помощью оператора `DECLARE`. Перед именем переменной всегда ставится символ “@”, а перед именем глобальной два таких символа.

DECLARE @имя_переменной тип_данных

Например

DECLARE @S CHAR(20), @I INTEGER

Для вывода переменной на экран можно использовать команду `PRINT`.

Для преобразования типа данных применяются функции `CONVERT` или `CAST`.

Для присвоения переменной значения применяется оператор `SET` или `SELECT`.

В MS SQL объявлен ряд глобальных переменных, которые вы сможете применять в своей работе. Некоторые из них представлены в таблице 18.1.

Таблица 18.1. Основные глобальные переменные в TSQL

Название	Описание
@@CONNECTIONS	Общее количество соединений с сервером или попыток входа в систему
@@ERROR	Номер последней системной ошибки; в случае отсутствия ошибок это значение равно 0
@@FETCH_STATUS	Состояние последнего выполненного оператора <code>FETCH</code>
@@IDENTITY	Значение последней вставленной в столбец <code>IDENTITY</code> величины
@SIDLE	Суммарное время простоя центрального процессора сервера.
@@IO_BUSY	Суммарное время, затраченное сервером на выполнение операций ввода/вывода.
@@LANGUAGE	Название используемого в данный момент языка
@@LOCK_TIMEOUT	Величина тайм-аута для операций блокировки (в миллисекундах)
@@MAX_CONNECTIONS	Максимальное число одновременных подключений.
@@MAX_PRECISION	Максимальный уровень точности для числовых и десятичных типов данных
@@NESTLEVEL	Номер текущего уровня вложения при вызове подпрограмм
@@OPTIONS	Возвращает установки текущих опций, заданных с помощью оператора <code>SET</code>
@@PROCID	Идентификатор текущей хранимой процедуры
@@REMSERVER	Имя удаленного сервера
@@ROWCOUNT	Количество строк, на которые повлиял последний запрос
@@TRANCOUNT	Общее число активных транзакции для текущего пользователя

Операторы управления Transact-SQL

В составе Transact-SQL имеется несколько операторов, которые применяются для изменения порядка выполнения команд внутри запроса SQL.

Ключевые слова IF...ELSE используются для реализации аналога условного оператора, знакомого Вам из традиционных языков программирования:

```
IF выражение
    Операторы
[ELSE]
    [IF выражение]
        Операторы
```

Ключевые слова BEGIN и END используются для обозначения набора операторов SQL, выполняемых как единое целое

```
BEGIN
    операторы
END
```

Ключевое слово WHILE используется для определения условия, на основании которого выполняется один или несколько операторов SQL, пока это условие истинно (равно TRUE):

```
WHILE <булево выражение>
    <операторы SQL>
```

Внутри конструкции WHILE могут применяться операторы BREAK и CONTINUE, которые соответственно предназначены для прерывания работы конструкции или для продолжения ее выполнения.

Листинг 18.1. Организация цикла WHILE на языке TSQL

```
DECLARE @X INT, @Y TINYINT
SET @X=1 SET @Y=1
WHILE @X<5
BEGIN
    PRINT 'X<5'
    SET @X=@X+1
    SET @Y=@Y+1
    IF @Y=2
    BEGIN
        PRINT 'Y=2'
        BREAK
    END
END
PRINT 'ЦИКЛ ЗАВЕРШЕН'
```

Базовые функции Transact-SQL

В таблицах 18.1 - 18.4 представлены наиболее часто применяемые функции transact-SQL.

Таблица 18.2. Основные математические функции в TSQL

Функция	Описание	
ACOS	Арккосинус числа	Значение угла выражено в радианах
ASIN	Арксинус числа	
ATAN	Арктангенс числа	
ATAN2	Арктангенс числа	
COS	Тригонометрический косинус угла	
COT	Тригонометрический котангенс угла	
SIN	Тригонометрический синус угла	
TAN	Тригонометрический тангенс угла	

DEGREES	Преобразование радиан в градусы
RADIANS	Преобразование градусов в радианы
CEILING	Наименьшее целое значение, которое больше или равно значению выражения
FLOOR	Наибольшее целое значение, которое меньше или равно значению выражения
EXP	Экспонента значения выражения
LOG	Натуральный логарифм значения выражения
LOG 10	Десятичный логарифм значения выражения
PI ()	Значение числа ПИ
POWER	Значение выражения в степени у
ABS	Абсолютное значение выражения
RAND	Случайное действительное число, находящееся в промежутке от нуля до единицы либо до значения, указанного в необязательном аргументе integer expression
ROUND	Округление значения с точностью до integer
SIGN	Возвращает значение 1, 0 или -1, в зависимости от знака выражения. Имеет тот же тип, что и исходное выражение
SQRT	Квадратный корень из значения выражения

Таблица 18.3. Основные функции для работы со строками в TSQL

Функция	Описание
CHAR	Преобразует ASCII код в символ
LOWER	Преобразует прописные буквы в строчные
UPPER	Преобразует строчные буквы в прописные
LTRIM	Удаляет из строки начальные пробелы
RTRIM	Удаляет из строки конечные пробелы
CHARINDEX	Осуществляет поиск подстроки в строке
STUFF	Вставляет подстроку в строку
SUBSTRING	Извлекает из строки подстроку
(+)	Сложение строк

Таблица 18.4. Основные функции для работы с датой в TSQL

Функция	Описание
DATENAME	Возвращает часть даты в виде символьной строки
GETFATE	Возвращает текущее значение даты/времени
DATEADD	Добавляет к дате значение
DATEDIFF	Возвращает разницу между двумя датами

Таблица 18.5. Основные системные функции в TSQL

Функция	Описание
HOST_NAME ()	Имя компьютера-сервера
HOST_ID ()	Идентификационный номер компьютера-сервера
SUSER_ID	Идентификатор пользователя, назначаемый при входе в систему
SUSER_NAME	Имя пользователя, используемое для входа в систему
USER_ID	Идентификационный номер пользователя, назначаемый при подключении к базе данных

USER_NAME	Имя пользователя, используемое для подключения к базе данных
DB_NAME	Имя базы данных
DB_ID	Идентификационный номер базы данных
OBJECT_ID	Идентификационный номер объекта базы данных
OBJECT_NAME	Имя объекта базы данных
INDEX_COL	Имя индексного столбца
COL_LENGTH	Длина столбца
COL_NAME	Имя столбца
DATALENGTH	Реальная длина выражения для некоторого типа данных
COALESCE	Возвращает первое ненулевое выражение (не равное NULL)
ISNULL	Заменяет нулевое значение выражения (NULL) заданным значением
NULLIF	Возвращает элемент NULL, если оба выражения идентичны

Функции преобразования типов данных

Для приведения типов данных предназначены функции `CONVERT` и `CAST`. Синтаксис этих функций следующий

CONVERT (тип данных [(длина)] , выражение [, стиль])

CAST (выражение **AS** тип данных)

Пример работы с функциями предложен в листинге 18.2.

Листинг 18.2. Преобразование типов данных

```
DECLARE @X decimal (5, 2)
SET @X = 193.57
SELECT CAST(CAST(@X AS varbinary(20)) AS decimal(10,5))
SELECT CONVERT(decimal(10,5), CONVERT(varbinary(20), @X))
```

Хранимые процедуры

Хранимые процедуры представляют собой подпрограммы, выполняемые на сервере. Эти подпрограммы могут запускаться из клиентского приложения, из других процедур и из триггеров.

В хранимую процедуру можно передавать значения, а они, в свою очередь, способны возвращать результаты своей работы.

К основным преимуществам хранимых процедур стоит отнести следующее:

- ~ Производительность.
- ~ Преимущества при разработке систем клиент-сервер.
- ~ Безопасность.
- ~ Поддержание дополнительных бизнес-правил на ввод данных.

Для создания хранимой процедуры применяется команда `CREATE PROCEDURE`. Вашему вниманию предложен код хранимой процедуры позволяющей добавлять в таблицу континентов новый материк.

Листинг 18.3. Пример хранимой процедуры TSQL

```
CREATE PROCEDURE Continent_Insert (@Continent CHAR(15), @Continent_ID INT OUTPUT)
AS
IF NOT EXISTS (Select * FROM Continent WHERE Continent=@Continent)
BEGIN
INSERT INTO Continent (Continent) VALUES (@Continent)
SET @CONTINENT_ID=@@IDENTITY
END ELSE @CONTINENT_ID=-1
```

Процедура с именем “Continent_Insert” обладает двумя параметрами: @Continent (в него передается название материка) и выходной параметр @Continent_ID (он возвращает значение первичного ключа вновь вставленной записи). Перед добавлением новой строки процедура проверяет нет ли в таблице континентов материка с названием @Continent, и только после этого добавляет новую запись.

Триггеры

Сервер БД способен осуществлять контроль за логикой работы приложения и прохождении данных по таблицам. Один из мощнейших инструментов такого рода называется триггером.

Триггер – специальный тип процедуры, которая выполняется при вставке, модификации или удалении данных в некоторой таблице. Триггеры запускаются в самую последнюю очередь, и если работа триггера завершается аварийно, то информация в БД не обновляется.

Синтаксическая конструкция описания триггера выглядит следующим образом:

```
CREATE TRIGGER имя_триггера
ON имя_таблицы
FOR [INSERT, UPDATE, DELETE]
AS Операторы
```

По сути, триггер – это событие, возникающее после вставки (*insert*), редактирования (*update*) или удаления записей (*delete*) из таблицы.

В листинге 18.4 предложен пример триггера обеспечивающего каскадное удаление данных из подчиненной таблицы, по факту удаления записи из главной таблицы.

Листинг 18.4. Пример триггера TSQL

```
CREATE TRIGER Continent_TR_DELETE
ON Continent
FOR DELETE
AS
DECLARE @Continent_ID INT
Select @ Continent_ID = Continent_ID From INSERTED
DELETE FROM Country WHERE Continent_ID=@Continent_ID
```

Задание

На основе БД разработанной на 17 лабораторной работе. Разработайте хранимые процедуры предназначенные для:

- ~ вставки, редактирования и удаления данных из таблиц;
- ~ просмотра данных в отдельных таблицах БД;
- ~ просмотра сводных данных (объединение всех таблиц БД);
- ~ поиска данных;
- ~ фильтрации данных.

19. Разработка приложений по технологии клиент-сервер

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – Microsoft SQL Server, Delphi, C++ Builder

Занятие посвящено изучению ряда компонентов Delphi предназначенных для просмотра, вставки, редактирования и удаления данных с применением команд на языке структурированных запросов SQL. Рассматриваемые на занятии компоненты обычно применяются для построения клиентских приложений БД.

Запрос TADOQuery

Компонент TADOQuery предназначен для взаимодействия с базой данных на основе структурированного языка запросов SQL. Для этих целей он инкапсулирует команду на языке SQL и отправляет ее в адрес сервера БД по команде клиентского приложения.

Ключевым свойством компонента является

property SQL: TStrings;

Это набор строк, в котором программист описывает команды на языке SQL. Для заполнения набора строк можно использовать все функциональные возможности инкапсулированные в классе TStrings. А как минимум надо помнить о существовании пары методов: Clear() – очистка содержимого набора строк и Add() – добавить новую строку в конец набора.

Листинг 19.1. Пример работы с компонентом TADOQuery

```
with ADOQuery1 do
begin
SQL.Clear;
SQL.ADD('Select * from writer');
Open;
end;
```

Весь текст запроса можно проконтролировать, обратившись к свойству:

property Text: String; //только для чтения

Перед началом работы с компонентом TADOQuery надо обязательно подключить компонент к БД. Для этого следует воспользоваться свойством ConnectionString или Connection

Выполнение запроса SQL

После того, как программист опишет текст SQL он получает право отправить команду на выполнение. Для этого компонент TADOQuery вооружен двумя методами, отправляющими команду на выполнение запроса SQL:

```
procedure Open;
procedure ExecSQL;
```

Метод Open() унаследован от универсального набора данных TDataSet. Он применяется только тогда, когда при помощи компонента TADOQuery программист планирует получить некоторый массив строк. Метод ExecSQL() просто приказывает БД отработать поступившую команду SQL.

Если текст запроса SQL начинается с команды SELECT (другими словами мы планируем получить от БД информацию) — используйте метод Open(). Если же мы редактируем данные (команды INSERT, UPDATE, DELETE), создаем объекты БД (команды CREATE ...), уничтожаем объекты (DROP ...), то в этом случае наш верный союзник метод ExecSQL().

Параметры запроса

Для нас уже не секрет, что по сравнению с TADOTable компонент TADOQuery предлагает программисту весьма гибкий способ получения набора данных. Дело за малым – на языке SQL описываем текст запроса и наш новый знакомый позволит объединять данные из нескольких таблиц, отбирать только требуемые поля, наложить ограничения на получаемые данные, сортировать их в указанной последовательности. Взглянем на строки, в которых предложен способ получения перечня стран с населением более 100 000 человек.

Листинг 19.2. Выборка данных с помощью TADOQuery

```
ADOQuery1.SQL.Clear;
ADOQuery1.SQL.Add('SELECT * FROM Country WHERE Population>=100000');
```

Беда в одном, при необходимости поменять условия отбора данных приходится очищать предыдущие SQL-инструкции и писать все заново, а это весьма неудобно. Хотя, проявив немного изобретательности, мы вместо статичного запроса сможем создать более эффективную конструкцию (листинг 19.1)

Листинг 19.3. Динамическое изменение текста запроса

```
procedure TForm1.Edit1Change(Sender: TObject);
var X : integer;
begin
X:= StrToIntDef(Edit1.Text,0);
  with ADOQuery1 do
  begin
    If Active Then Close;
    SQL.Clear;
    SQL.Add('SELECT * FROM Country');
    SQL.Add('WHERE Population>='+IntToStr(X));
    Open;
  end;
end;
```

Пользователь вводит число жителей в строку ввода Edit1. Мы динамически обновляем условие отбора данных в момент события OnChange().

Параметры окажут неоценимую помощь в запросах, в которых ограничения на отбор данных могут меняться, или в командах модифицирующих данные в таблицах. Продолжим эксперименты с таблицей Country, хранящей данные о названии страны (поле Name) имя столицы (поле Capital) и количество жителей в стране (поле Population). Пример универсальной команды SQL, позволяющей использовать ее неоднократно для вставки данных в таблицу предложен ниже:

Листинг 19.4. Подготовка к TADOQuery вставке строки

```
with ADOQuery1 do
begin
SQL.Clear;
SQL.Add('INSERT INTO Country (Name, Capital, Population)');
SQL.Add('VALUES (:Name, :Capital, :Population)');
end;
```

Данная процедура вызывается только один раз, например, при старте приложения. Обратите внимание на строку

```
SQL.Add('VALUES (:Name, :Capital, :Population)');
```

Символ двоеточия признак того, что следующее за ним слово является именем параметра. Теперь, для вставки в таблицу новой записи достаточно передать в текст запроса соответствующие значения. Для хранения параметров в запрос инкапсулировано специальное свойство

```
property Parameters: TParameters;
```

Для обращения к параметру надо указать его индекс от 0 до ParamCount-1.

О количестве параметров нас информирует свойство:

```
property ParamCount: Word; //только для чтение
```

Листинг 19.5. Процедура вставки новой строки

```
Procedure InsertTown(Name, Capital :String; Population :Cardinal);
begin
ADOQuery1.Parameters [0].AsString := Name;
ADOQuery1.Parameters [1].AsString := Capital;
ADOQuery1.Parameters[2].AsInteger := Population;
ADOQuery1.ExecSQL;
end;
```

При передаче значений параметров главное не перепутать их индекс. Если же вы не знаете очередности параметров, то вместо порядкового номера параметра целесообразнее применять метод

```
function ParamByName (const Value: String): TParam;
```

Функция возвращает параметр с именем Value

```
ADOQuery1.Parameters.ParamByName('Population').AsInteger:= Population;
```

Хранимая процедура TADOStoredProc

В отличие от своих коллег (таблицы TADOTable и запроса TADOQuery), компонент TADOStoredProc не предназначен для прямого обращения к набору данных (таблице или объединению таблиц). TADOStoredProc предназначен для вызова описанной на SQL сервере хранимой процедуры (stored procedure). В виду этой особенности TADOStoredProc не найдет применения в приложениях, работающих с локальными данными (например такими как Access), так как драйверы этих баз данных не поддерживают такой вид объекта. Но, в клиентских приложениях, обслуживаемых SQL-сервером, компонент чувствует себя превосходно.

Соединение с хранимой процедурой

Для подключения TStoredProc к размещенной на сервере процедуре программист должен описать два свойства. В первую очередь в с помощью свойства Connection следует подсоединиться к компоненту TADOConnection. Затем, в свойстве:

```
property StoredProcName: String;
```

указывается имя хранимой процедуры, которую станет вызывать компонент. Кроме того, если хранимая процедура обладает параметрами, то необходимо проверить список параметров в свойстве:

```
property Parameters: TParameters;
```

Как правило, список параметров заполняется автоматически после подключения компонента к расположенной на сервере процедуре.

Выполнение хранимой процедуры

Для выполнения хранимой процедуры TADOStoredProc может воспользоваться двумя методами:

```
procedure Open;
```

```
procedure ExecProc;
```

Процедура Open () используется только в тех случаях, когда хранимая процедура возвращает набор данных (процедура построена на основе команды SELECT). Во всех остальных случаях нам нужен метод ExecProc.

Листинг 19.6. Обращение к хранимой процедуре

```
ADOStoredProc1.Parameters.Params[0].AsString := Edit1.Text; //передача значения
ADOStoredProc1.ExecProc;                                     //выполнение процедуры
Result:= ADOStoredProc1.Params[1].Value;                    //получение результатов
```

При работе с SQL-серверами Sybase и Microsoft SQL пригодится метод:

```
procedure GetResults;
```

Этот метод возвращает выходные параметры хранимой процедуры в клиентское приложение.

Транзакции и их изоляция

При разработке БД нацеленных на обслуживание более чем одного пользователя одновременно необходимо досконально разбираться в процессах выполнения транзакций и блокировок.

Стоит выделить 4-ре основных принципа транзакций:

1. **Целостность.** Транзакция может быть завершенной или нет. Транзакция не может быть выполнена частично.
2. **Согласованность.** После выполнения (или после отката) транзакции система должна находиться в некотором известном состоянии.
3. **Изолированность.** Транзакции должны быть автономными и не должны воздействовать на другие транзакции, а также зависеть от них.

4. **Устойчивость.** После завершения транзакции ее цель считается достигнутой, и отменить ее уже нельзя.

Существуют ограничения, которые нельзя выполнить с помощью транзакций. Это операции которые нельзя отменить без существенного влияния на компоненты системы. В частности внутри транзакции нельзя использовать команды на создание (CREATE), удаление (DROP) и реконфигурацию (ALTER) объектов БД.

При построении многопользовательских БД разработчик программы должен предусмотреть поведение программного обеспечения в случае одновременного доступа к одним и тем же данным нескольких пользователей. Ведь нельзя исключить ситуацию, когда пара пользователей попытается отредактировать одну и ту же строку в таблице. Что окажется в ней после их работы? Способ защиты информации многопользовательских БД называется **уровнем изоляции транзакций**. В MS SQL существует 4-ре уровня изоляции:

1. Уровень **Read Uncommitted**. Транзакции разрешено читать данные, в настоящий момент обрабатываемые другими транзакциями. Говоря о таком процессе, часто применяют термин **грязное чтение**.
2. Уровень **Read Committed**. Транзакция может читать только завершенные изменения других транзакций.
3. Уровень **Repeatable Read** (повторяющееся чтение). Уровень изоляции гарантирует, что на просматриваемые пользователем данные не повлияют транзакции, выполняемые другими транзакциями. Недостаток уровня в том, что Ваша транзакция захватывает данные в монопольное пользование и не разрешает обращение к ним из других транзакциям, а это полный запрет на параллельную обработку данных.
4. Уровень **Serializable**. Рекомендуемый уровень изоляции. Он поддерживает все ограничения уровня Repeatable Read, но в тоже время предотвращает чтение фиктивных данных.

Управление транзакциями и компонент TADOConnection

Для запуска новой транзакции предназначен метод

```
function BeginTrans: Integer;
```

Метод возвращает целочисленное значение, соответствующее уровню вложения транзакции. Если запуск новой транзакции осуществился без заминок, то это приведет к установке в true свойства

```
property InTransaction: Boolean;
```

Значение true свидетельствует о том, что TADOConnection инициировал транзакцию и эта транзакция находится в стадии выполнения. С окончанием транзакции свойство вернется в состояние false. Сразу после успешного запуска новой транзакции происходит событие

```
property OnBeginTransComplete: TBeginTransCompleteEvent;  
TBeginTransCompleteEvent = procedure (Connection: TADOConnection;  
TransactionLevel: Integer; const Error: Error;  
var EventStatus: TEventStatus) of object;
```

Параметр EventStatus информирует нас, насколько успешно прошел процесс запуска транзакции. Если операция выполнялась корректно, то в параметре окажется значение esOK.

Для сохранения результатов выполнения транзакции программист вызывает процедуру

```
procedure CommitTrans;
```

Сразу после сохранения транзакции возникает событие

```
property OnCommitTransComplete: TConnectErrorEvent;  
TConnectErrorEvent = procedure (Connection: TADOConnection; Error: Error;  
var EventStatus: TEventStatus) of object;
```

Если в процесс выполнения транзакции вкралась ошибка, то для ее отмены воспользуйтесь методом

```
procedure RollbackTrans;
```

Откат транзакции сопровождается событием

```
property OnRollbackTransComplete: TConnectErrorEvent;
```

Если разрабатываемая база данных подлежит эксплуатации в многопользовательском режиме, то особое внимание стоит уделить вопросам обеспечения параллельной обработки данных. Накал борьбы клиентских приложений за записи таблиц можно снизить, установив соответствующий уровень изоляции транзакций

```
property IsolationLevel: TIsolationLevel;
```

Уровень разграничения транзакций определяет возможность одновременного совместного доступа пользователей к одним и тем же данным. В таблице 19.1 приведены допустимые значения уровней изоляции транзакций в ADO. Блокировки упорядочены по степени возрастания пессимистических настроений. Другими словами каждая последующая строка все более и более затрудняет параллельную работу нескольких пользователей с одними и теми же данными, но зато снижает вероятность появления конфликтов.

Таблица 19.1. Возможные уровни изоляции транзакции ADO – *TIsoolationLevel*

Уровень изоляции	Описание
<code>ilUnspecified</code>	Сервер может остаться в любом режиме изоляции транзакций, так как приложение не предъявляет к СУБД каких либо особых требований
<code>ilChaos</code>	Изменения, сделанные транзакцией с более высоким уровнем изоляции, не могут быть переписаны текущей транзакцией.
<code>ilReadUncommitted</code> или <code>ilBrowse</code>	Грязное чтение. Допустимо чтение данных незавершенных транзакций.
<code>ilCursorStability</code> или <code>ilReadCommitted</code>	Исключение грязного чтения. Данные доступны только после завершения транзакции.
<code>ilRepeatableRead</code>	Повторяющееся чтение. Изменения, произведенные другими транзакциями к предварительно прочитанным данным не видны, пока не будет осуществлено повторное чтение данных.
<code>ilSerializable</code> или <code>ilIsolated</code>	Полная изоляция. Максимальная степень изоляции транзакций.

Задание

На основе изученных компонентов разработайте процедуры и функции, позволяющие получать, добавлять, редактировать и удалять данные из таблиц вашей БД. Операции, связанные с модификацией данных должны быть заключены в транзакции.

20. Слабо структурированные данные и язык XML

Вид занятия – лабораторное занятие

Время занятия – 2 часа

Программное обеспечение – среда Microsoft Visual Studio, C++ Builder (Delphi)

Расширяемый язык разметки (eXtensible Markup Language, XML) был разработан специально для создания структурированных документов – он как никто другой позволяет представлять и манипулировать элементами данных. Любой документ XML может рассматриваться как символьная строка, состоящая из символов разметки и данных. Благодаря тому, что двоичные данные в XML отсутствуют, документ на основе расширяемого языка разметки может быть открыт и отредактирован в любом текстовом редакторе, от элементарного блокнота до текстового процессора класса Microsoft Word.

Построение простейшего документа XML

Правила создания рядового документа XML столь просты, что для написания демонстрационного проекта в формате XML уйдет не более минуты.

Листинг 20.1. Документ XML

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<!--Это комментарий-->
<example>
    <line1>Простой пример работы</line1>
    <line2>с языком XML</line2>
</example>
```

Каждому открывающему тэгу должен сопоставляться закрывающий тэг (за исключением первой строки документа). Признаком того, что тэг является закрывающим служит наклонная черта “/”.

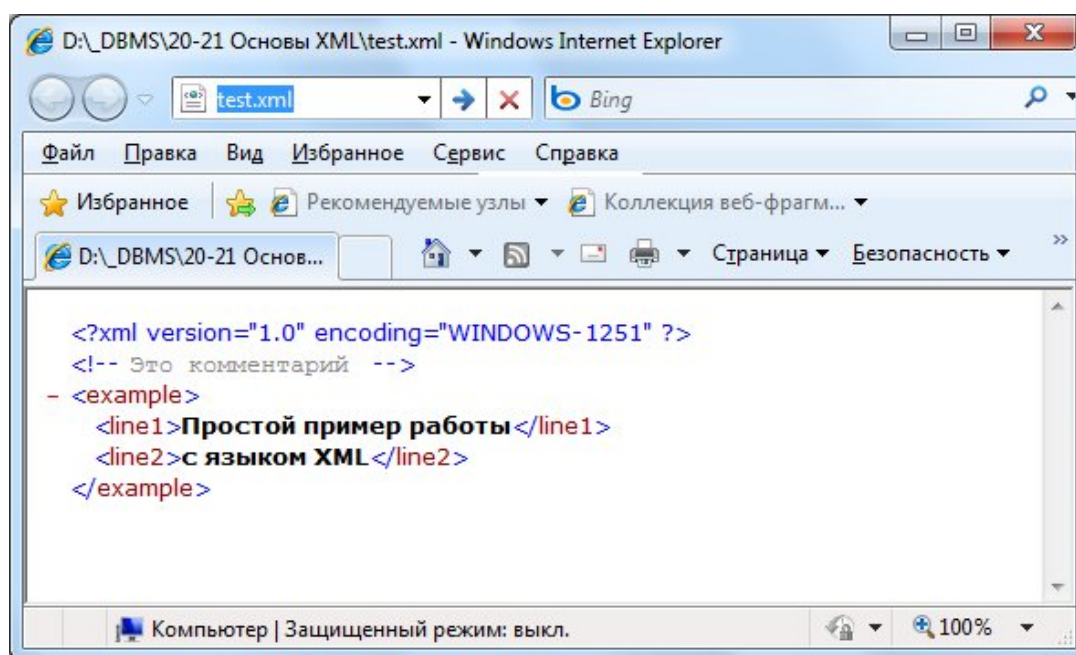


Рис. 20.1. Внешний вид XML-документа в браузере Windows Internet Explorer

Совокупность открывающего, закрывающего тэга и данных внутри них

```
<line3>данные</line3>
```

называется **элементом** документа.

Базовые правила определения элементов XML:

1. Каждому открывающему тэгу должен соответствовать закрывающий тэг.
2. Документ XML позволяет использовать вложенные друг в друга тэги, при условии, что тэги не должны перекрываться.

3. В документе XML должен быть всего один корневой элемент.
4. Имена тэгов должны начинаться с буквы. После первого символа разрешается применять цифровые и буквенные символы. Имя не должно содержать символы пробела и двоеточия ":". В именах запрещено применять аббревиатуры "XML", "xml", "Xml", и т.п.
5. Имена чувствительны к регистру.

Атрибуты

Предположим, что мы через Интернет получаем прайс-лист книжного магазина. В нем содержатся данные систематизированные по жанру произведения, а внутри элементов жанра <GENRE> .. </GENRE> хранятся элементы с описанием книг <BOOK>..</BOOK>.

Листинг 20.2. Документ XML с атрибутами

```
<?xml version="1.0" encoding="WINDOWS-1251"?>
<pricelist>
<GENRE>Роман
  <BOOK WRITER="Толстой Л.Н." PUBLISHER="Мир" PRICE="100.00">
    Война и мир</BOOK>
  <BOOK WRITER="Толстой Л.Н." PUBLISHER="Детская литература" PRICE="57.00">
    Анна Каренина</BOOK>
  <BOOK WRITER="Достоевский Ф.М." PUBLISHER="Кавказ" PRICE="210.00">
    Братья Карамазовы</BOOK>
</GENRE>
<GENRE>Поэма
  <BOOK WRITER="Гете И.В." PUBLISHER="Худ. литература" PRICE="45.5">
    Фауст</BOOK>
</GENRE>
</pricelist>
```

Самое главное в новом примере то, что открывающий тэг описания книг содержит **атрибуты**. Атрибут состоит из названия и значения. Порядок именования атрибутов ничем не отличается от правил задания имен для элементов документа. Значения атрибутов заключаются в двойные или одинарные кавычки, в нашем примере в них содержатся дополнительная информация, описывающая ту или иную книгу.

Определение документа DTD

Нельзя исключить вероятность того, что заложенная разработчиком смысловая нагрузка в элементы документа окажется не столь очевидной как для пользователя, так и для анализатора XML. Примеров неоднозначностей в толковании разметки документа достаточно много. Например, создавая элемент <PHONENUM> мы рассчитываем, что в нем будет храниться 11-значный номер телефона, но пользователь предположит, что достаточно ограничиться сокращенным 6-значным номером абонента. Для решения подобных проблем в базах данных разработан весьма эффективный аппарат поддержки целостности и непротиворечивости данных, а как обстоят дела в XML?

В ситуации неоднозначности толкования разметки стоит воспользоваться еще одним козырем XML – способностью подключения к документу дополнительного описания называемого **определением типа документа** (*Document Type Definition, DTD*). DTD конкретизирует, что именно должно содержаться в элементах и атрибутах документа.

Описание структуры документа XML с помощью DTD начинается сразу после обязательного заголовка документа и включает слово DOCTYPE. Признаком завершения секции описания выступает закрывающая квадратная скобка и закрывающий тэг >]. Таким образом простейшее описание будет выглядеть следующим образом:

```
<!DOCTYPE имя_элемента [ ]>
```

Внутри секции содержится некоторый набор тэгов <! и > в которые заключаются имена элементов структуры или списки атрибутов. Восклицательный знак указывает на то, что это **элемент объявления**. Сразу за восклицательным знаком (без пробела) следует слово DOCTYPE и имя элемента структуры.

В простейшем случае описание DTD заносится непосредственно в документ XML, сразу после заголовка с версией. В представленном ниже примере мы создаем блок определения типа документа, информирующего нас о том, что:

1. Каждый информационный блок документа pricelist состоит из двух элементов (GENRE и BOOK).

- 2. Оба элемента представляют собой разобранные символьные данные (Parsed Character Data, PCDATA) или, проще говоря, элементы хранят обычный текст.
- 3. Элемент BOOK содержит список атрибутов (тэг <!ATTLIST>). Из них два атрибута обязательны для заполнения (#REQUIRED) и один необязательный (#IMPLIED).

Листинг 20.3. Документ XML с описанием DTD

```
<?xml version="1.0" encoding="WINDOWS-1251" standalone="yes"?>
<!DOCTYPE pricelist [
    <!ELEMENT GENRE (#PCDATA)>
    <!ELEMENT BOOK (#PCDATA)>
    <!ATTLIST BOOK
        WRITER CDATA #REQUIRED
        PUBLISHER CDATA #IMPLIED
        PRICE CDATA #REQUIRED>]>
<pricelist>
    <GENRE>Роман
    <BOOK
        WRITER="Толстой Л.Н."
        PUBLISHER="Мир"
        PRICE="100.00">
        Война и мир</BOOK>
    </GENRE>
</pricelist>
```

В процессе чтения блоков данных XML, анализатор проверяет их на соответствие требованиям DTD. Если выявляются отсутствующие (или наоборот лишние) элементы и атрибуты, или элемент данных находится не в ожидаемом месте, или содержимое элементов и атрибутов не соответствует объявлению, то анализатор делает вывод о том, что документ некорректен.

В нашем примере элемент BOOK содержит объявления атрибутов. Список атрибутов начинается с тэга <!ATTLIST, далее следует имя элемента, которому принадлежит список, затем описание атрибутов и закрывающая угловая скобка. Ключевое достоинство атрибута в том, что тип хранимых в нем данных не ограничивается простым CDATA (табл. 20.1).

Таблица 20.1. Типы атрибутов DTD

Тип	Описание
CDATA	В качестве значения атрибута могут использоваться символьные данные
ID	Признак, что значение атрибута должно быть уникальным. Весьма полезное качество при работе с базами данных, так как позволяет однозначно идентифицировать элемент, которому принадлежит уникальный атрибут
IDREF	В атрибуте хранится ссылка на идентификатор (ID) элемента
IDREFS	В атрибуте находится разделенная пробелами последовательность значений IDREF
ENTITY	Ссылка на внешний объект (например, файл мультимедиа). При проверке корректности документа подобный объект пропускается анализатором
ENTITIES	Разделенные пробелами ссылки на внешние объекты
NMTOKEN	Строка символьных данных, которая будет рассматриваться анализатором как корректное значение
NMTOKENS	Строка, содержащая несколько разделенных пробелами NMTOKEN
Перечислимый атрибут	Атрибут, содержащий список допустимых значений. Например: <!ATTLIST video format (MPEG1 MPEG2 MPEG4) "MPEG2"> В скобках мы перечислили все допустимые для атрибута значения видеформата. По умолчанию предлагается применять формат видеозаписи MPEG2

Описание DTD позволяет формировать достаточно сложные модели XML-данных. Допустим нам необходимо построить иерархическую структуру состава системного блока персонального компьютера (листинг).

Листинг 20.4. Описание DTD для системного блока

```
<!DOCTYPE COMPUTERS [
<!ELEMENT COMPUTER (CPU, HDD+, VIDEO, RAM, OTHER*)>
<!ELEMENT CPU (#PCDATA)>
```

```
<!ELEMENT HDD (#PCDATA)>
<!ELEMENT VIDEO (#PCDATA)>
<!ELEMENT RAM (#PCDATA)>
<!ELEMENT OTHER (#PCDATA)>
]>
```

указывают на то, что элемент “COMPUTER” должен содержать информацию о процессоре “CPU”, жестком диске “HDD”, видеокарте “VIDEO”, памяти “RAM” и другом оборудовании “OTHER”. С этой целью названия вложенных элементов перечисляются в круглых скобках после слова “COMPUTER”. Подобное перечисление требует, чтобы при вводе XML данных строго соблюдался порядок следования элементов, на первом месте процессор, на втором – винчестер, и т.д. Если обозначенный порядок, по каким-то причинам нарушится, то анализатор документа немедленно известит нас об ошибке в документе.

Обратите внимание еще на одну особенность описания DTD для примера с компьютером. Во второй строке листинга после названий элементов “HDD” и “OTHER” появились ранее не встречавшиеся нам символы “+” и “*”. Благодаря им мы приобретаем дополнительные возможности (табл. 20.2) по введению ограничений на количество элементов данных и установке признака обязательности и необязательности элемента.

Таблица 20.2. Ограничение количества элементов

Определитель	Описание
?	Признак необязательного элемента. Он может вообще отсутствовать в XML данных, либо появляться один раз.
+	Элемент может появиться один или несколько раз
*	Элемент может не появляться или появляться несколько раз
Пустой определитель	Значение по умолчанию. Элемент обязателен и может появляться только один раз

С помощью символов “+” и “*” мы сумели создать определение DTD для XML-документа с переменным числом элементов. Теперь, при описании состава компьютера нам позволено подключать один или несколько жестких дисков и неограниченное число дополнительного оборудования “OTHER”, при этом документ XML остается корректным.

Листинг 20.5. Пример XML файла с данными системного блока

```
<?xml version="1.0" encoding="WINDOWS-1251" standalone="no"?>
<!DOCTYPE COMPUTERS SYSTEM "computers.dtd">
<COMPUTERS>
  <COMPUTER>
    <CPU>INTEL CoreDuo2400</CPU>
    <HDD>320GB</HDD>
    <VIDEO>RODEON8800</VIDEO>
    <RAM>2GB</RAM>
  </COMPUTER>
  <COMPUTER>
    <CPU>AMD 3000</CPU>
    <HDD>500GB</HDD>
    <HDD>500GB</HDD>
    <VIDEO>NVidia</VIDEO>
    <RAM>4GB</RAM>
    <OTHER>ТВ-тюнер Aver</OTHER>
    <OTHER>Принтер HP LJ P1005</OTHER>
  </COMPUTER>
</COMPUTERS>
```

выше предложен фрагмент документа XML ссылающийся на файл с описанием computers.dtd.

Задание

Разработайте DTD-описание и описание XML Schemas документа для экспорта всех данных (за исключением ключевых полей) из вашей БД. В соответствии с полученным описанием разработайте и заполните XML-документ данными (соответствующим 10-15 строкам таблиц БД).

7 семестр

21. Создание БД в InterBase

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – СУБД InterBase (FireBird)

Основным инструментом администратора СУБД InterBase выступает консоль **IBConsole**, доступ к которой вы сможете получить через меню. По умолчанию консоль администрирования IBConsole.exe окажется в каталоге

C:\Program Files (x86)\Embarcadero\Studio\..n\InterBaseXE7\bin

Воспользуйтесь консолью и подключитесь к экземпляру сервера. На запрос регистрационных данных введите имя *SYSDBA* с паролем *masterkey*. Для создания новой БД обратитесь к пункту контекстного меню **Create Database** щелкнув правой кнопкой по узлу Database (рис. 21.1).

По умолчанию для получения административного доступа к СУБД используется имя “SYSDBA” с паролем “masterkey”. В период проектирования разработчик БД может не менять этот логин и использовать его в своей работе, однако при развертывании СУБД у конечного пользователя следует обязательно сменить общеизвестный пароль!

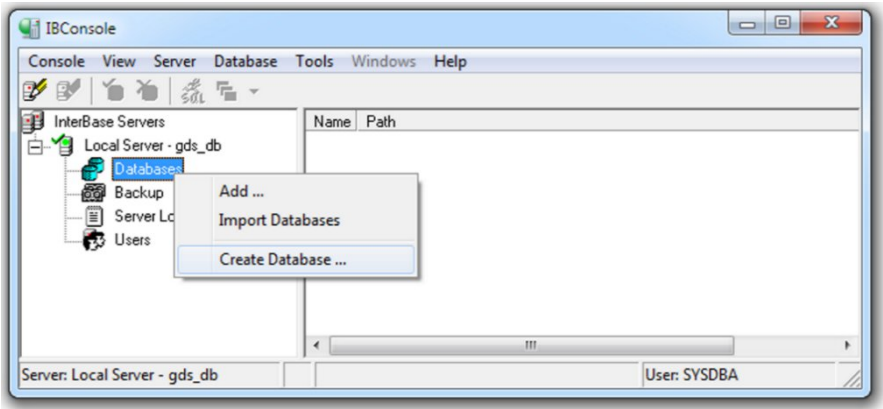


Рис. 21.1. Создание новой БД средствами IBConsole

В простейшем случае для создания нового экземпляра БД в достаточно лишь указать имя и путь к файлу с будущей БД (рис. 21.2). Если требуется более тонкая настройка платформы БД, то изучите группу опций **Options**.

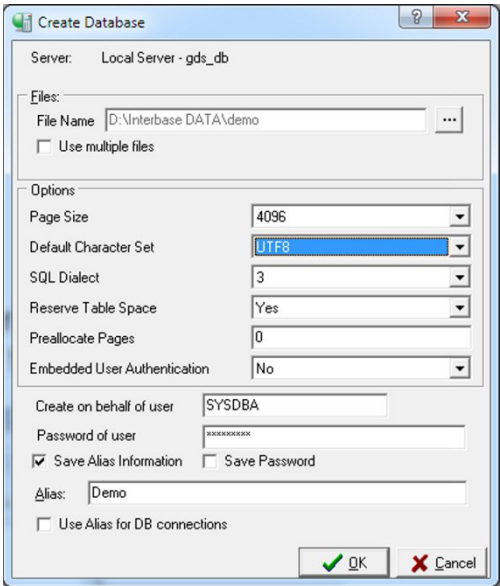


Рис. 21.2. Настройка параметров создаваемой БД

Определение домена в консоли администрирования

Если вы хотите пойти по наиболее простому пути создания домена, то сразу обращайтесь к консоли администрирования IBConsole. Для этого в консоли следует развернуть дерево целевой БД и выбрать в нем узел **Domains**. Щелчок правой кнопкой мышки вызовет контекстное меню, в котором нам останется выбрать пункт **Create** (рис. 21.3).

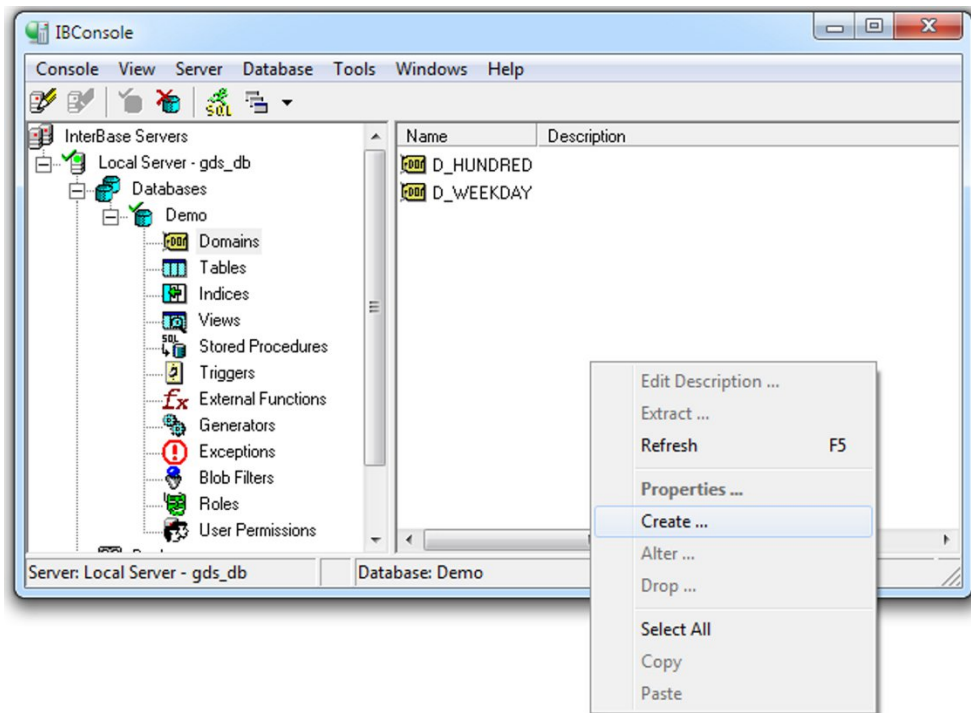


Рис. 21.3. Создание домена средствами IBConsole

В ответ на это действие перед программистом раскроется окно редактора домена (рис. 21.4) в котором, кроме собственно уникального имени домена **Domain Name**, следует:

- ~ определить опорный тип данных **Data Type**;
- ~ назначить значение по умолчанию **Default**;
- ~ подтвердить запрет на вставку неопределенного значения **Not Null**;
- ~ описать выражение-ограничение на перечень допустимых значений **Check**.

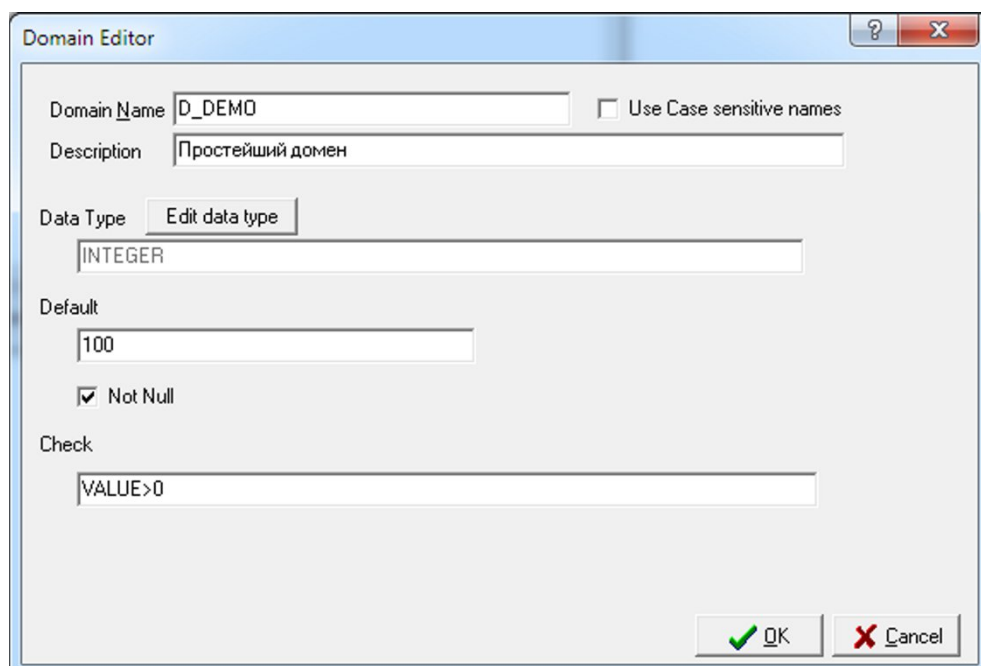


Рис. 21.4. Создание домена средствами IBConsole

Таблицы и консоль администрирования

Сторонникам работы с таблицами с помощью консоли администрирования сервера безусловно понравится удобный и одновременно простой интерфейс утилиты. Для создания новой таблицы достаточно выбрать узел **Tables** в дереве объектов проектируемой БД и затем щелкнуть правой кнопкой мышки в области списка таблиц и вызвать пункт меню **Create** (рис. 21.5).

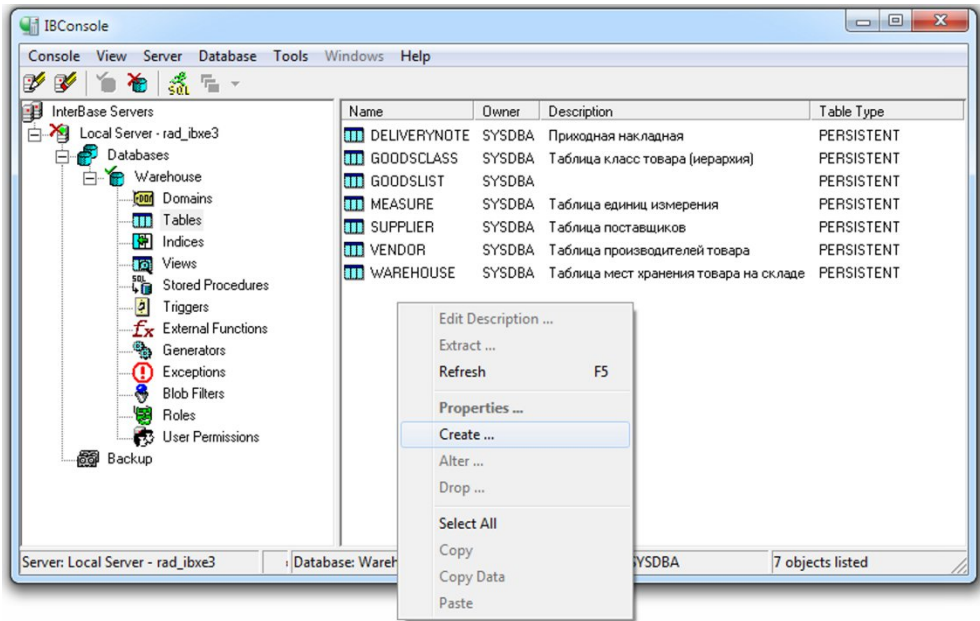


Рис. 21.5. Команда на создание новой таблицы в IBConsole

В ответ на это действие перед разработчиком появится новое окно редактора таблицы (**Table Editor**) в котором следует определить (рис. 21.6):

- ~ имя создаваемой таблицы (в строке ввода **Table Name**);
- ~ перечень полей таблицы (кнопка **Add Field**);
- ~ первичный ключ (вкладка **Primary Key**);
- ~ поля с уникальными значениями (вкладка **Unique Keys**);
- ~ внешние ключи (вкладка **Foreign Keys**) для связи с другими таблицами;
- ~ дополнительные ограничения накладываемые на поля таблицы (вкладка **Check Constraints**).

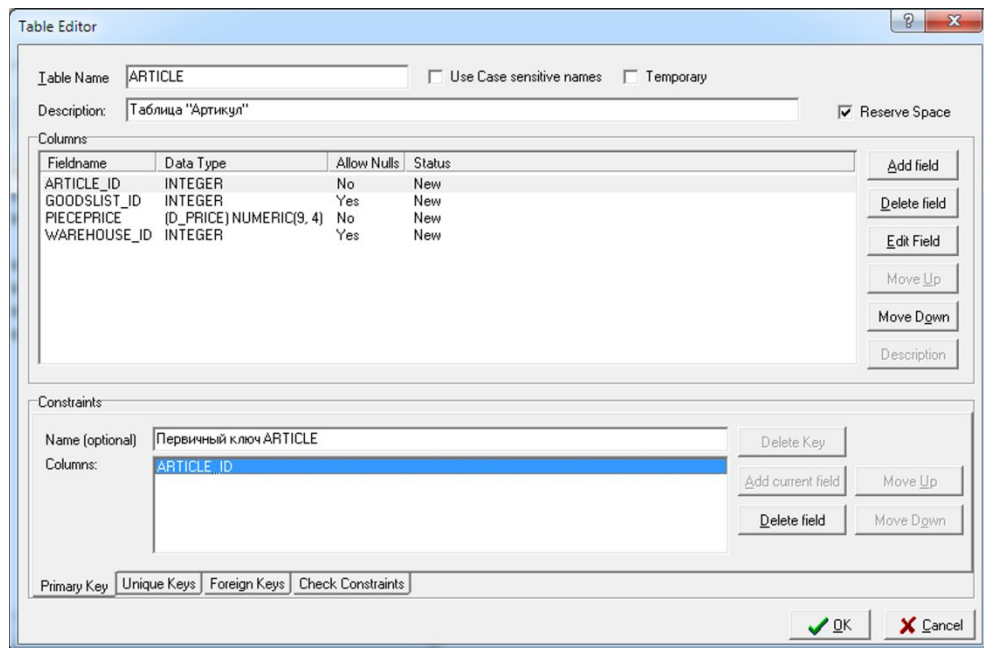


Рис. 21.6. Создание новой таблицы в консоли IBConsole

Помимо создания таблиц, консоль администрирования позволяет просматривать свойства, редактировать и удалять таблицы.

Создание значений ключа с помощью генератора

Для автоматического ввода значений первичных ключей разработчики СУБД InterBase придумали оригинальное решение. Оно заключается в создании отдельного объекта–генератора, представляющего собой счетчик автоинкрементного типа. Для того, чтобы вдохнуть жизнь в генератор следует воспользоваться инструкцией из листинга 21.1.

Листинг 21.1. Создание и инициализация генератора с помощью SQL

```
CREATE GENERATOR G_WAREHOUSE_ID;
SET GENERATOR G_WAREHOUSE_ID TO 0;
```

В этом примере в первой строчке в базе данных создается генератор с именем G_WAREHOUSE_ID, а затем этому генератору присваивается стартовое значение 0.

Для обращения к генератору предусмотрена функция:

```
GEN_ID(имя_генератора, шаг_приращения);
```

Названная функция обладает двумя параметрами, в которые передаются имя генератора и шаг приращения значения. Для того, чтобы подготовленное генератором число попало в столбец первичного ключа таблицы функция GEN_ID вызывается в триггере (листинг 21.2) или хранимой процедуре.

Листинг 21.2. Генерация значения ключа в триггере

```
CREATE TRIGGER T_WAREHOUSE_BI_GEN
FOR WAREHOUSE
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
IF (NEW.WAREHOUSE_ID IS NULL) THEN
    NEW.WAREHOUSE_ID=GEN_ID(G_WAREHOUSE_ID, 1);
END;
```

Несмотря на некоторые сложности работы с генераторами у них есть неоспоримое достоинство – они предоставляют программисту дополнительную степень свободы при управлении значениями первичных ключей.

Если при отладке БД вы осуществляли ввод новых записей в таблицу не применяя генератор (например, отключив триггер и формируя значение первичного ключа вручную), то затем не забудьте перестроить текущее значение генератора, установив в него значение не ниже текущего значения первичного ключа таблицы. Для этого вам потребуются команда SET GENERATOR.

Представления и консоль администрирования

Для быстрого создания представления достаточно обратиться к консоли администрирования **IBConsole**. Найдя узел **Views** в иерархии объектов базы данных щелкните правой кнопкой мышки в области списка представления и выберите элемент меню **Create**. В ответ на эти действия перед вами появится редактор представления **View Editor** (рис. 21.7).

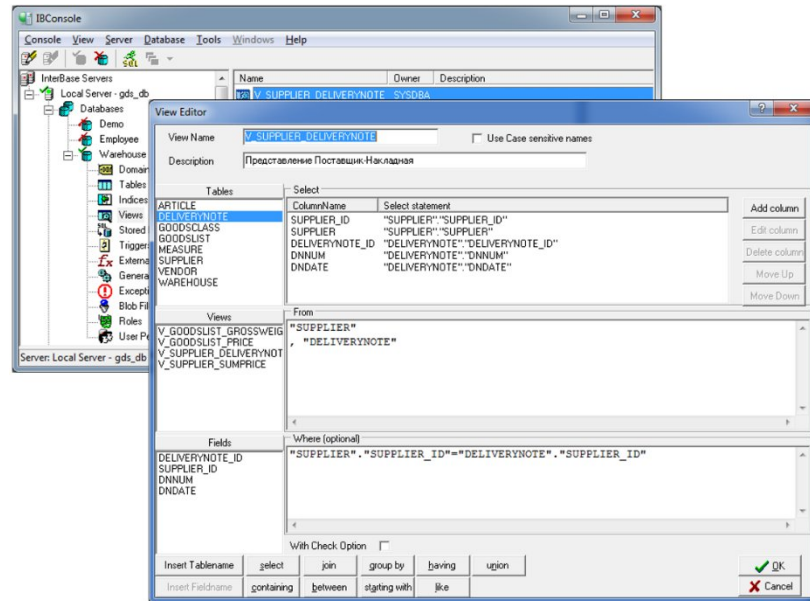


Рис. 21.7. Создание представления с помощью консоли администрирования IBConsole

Для описания нового представления следует определиться со следующими атрибутами:

- ~ имя представления (строка ввода **View Name**);
- ~ состав столбцов входящих в представление (кнопка **Add column**). При описании отдельного столбца редактор попросит указать его псевдоним, он станет отображаться в колонке **ColumnName** списка **Select**;
- ~ в многострочное поле ввода **From** следует поместить перечень таблиц и уже существующих представлений на основе которых будет создано новое представление;
- ~ если представление формируется на базе двух или более исходных отношений, то в многострочное поле ввода **Where (optional)** записывается условие объединения.

Закончив описание нажимаем на кнопку **OK** — InterBase проверит корректность описания представления и создаст новый объект.

Задание

В соответствии с утвержденной ER-моделью базы данных сделайте следующее:

1. Создайте БД средствами консоли администрирования InterBase (или Firebird).
2. Создайте не менее 3 доменных ограничений, расширяющих возможности различных типов данных.
3. Создайте все таблицы БД с применением доменных ограничений и генераторов для первичных ключей.
4. Убедитесь, что все таблицы обладают механизмом генерации первичного ключа.
5. Создайте не менее 3 представлений, одно из них должно объединить все таблицы БД.

Воспользовались пунктом меню **View Metadata** получите метаданные разработанной БД.

22. Процедурный SQL, хранимые процедуры

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – среда InterBase (FireBird)

Хранимая процедура представляет собой подпрограмму на языке SQL, сохраняемую в системном каталоге СУБД. Благодаря тому, что процедура выполняется на сервере, она не требует ресурсов от клиентских станций, что в значительной степени повышает производительность.

В код хранимой процедуры выносятся как наиболее часто вызываемые, так и максимально “тяжеловесные” и трудоемкие команды. В качестве таких распространенных операций стоит упомянуть процедуры вставки, модификации, удаления и выборки данных из таблиц. Благодаря тому, что процедура может содержать блоки команд на языке SQL, переменные, условные операторы и циклы, она позволяет строить весьма сложные логические конструкции, позволяющие решать нетривиальные задачи.

Обобщенная синтаксическая конструкция создания хранимой процедуры выглядит следующим образом

```
CREATE PROCEDURE имя_процедуры [(входной_параметр1 тип, ...)]
                                [RETURNS (выходной_параметр1 тип, ...)]
```

AS

[блок объявления переменных]

BEGIN

тело_процедуры

END

Процедуры можно создавать как с помощью консоли администрирования (рис 22.1), так и прямым вызовом инструкции **CREATE PROCEDURE**.

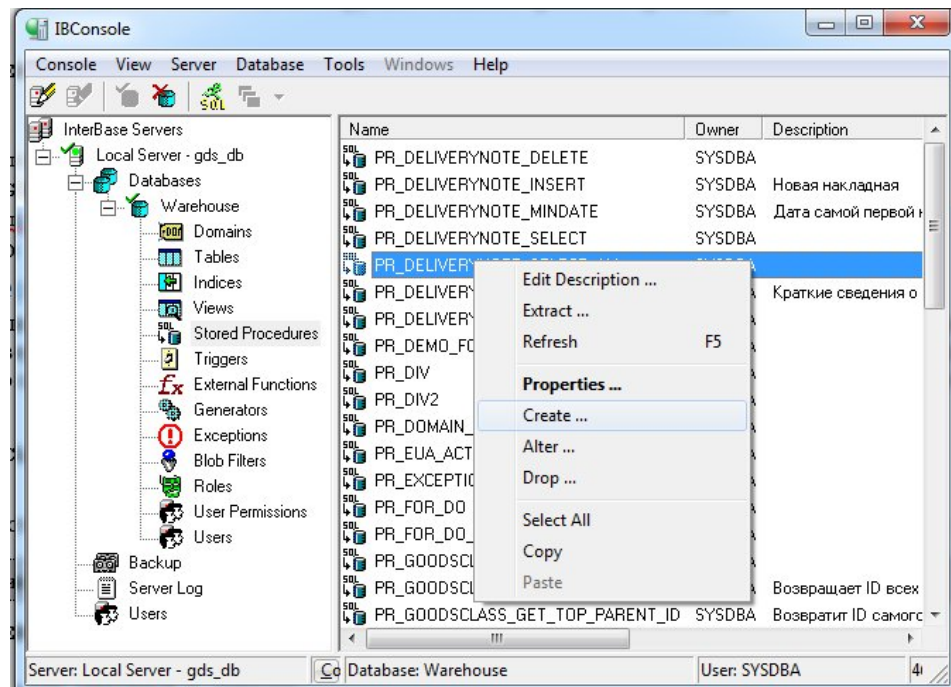


Рис. 22.1. вызов редактора хранимой процедуры из окна IBConsole

Переменные

Если вы вспомните свои первые шаги при изучении любого высокоуровневого языка программирования, то наверняка отправной точкой стала работа с переменными. Мы не станем изменять этой традиции, тем более, что процедуры и триггеры обладают правом объявлять локальные переменные. Синтаксис объявления таков

```
DECLARE VARIABLE имя_переменной тип_данных;
```

Листинг 22.1 демонстрирует порядок работы с переменными в рамках хранимой процедуры

Листинг 22.1. Объявление переменных в хранимой процедуре

```
CREATE PROCEDURE PR_VARDEMO
AS
DECLARE VARIABLE A INTEGER;
DECLARE VARIABLE B INTEGER;
DECLARE VARIABLE C FLOAT;
BEGIN
```

```

A=5;
B=2;
C=A/B;
/*...*/
END

```

В InterBase существует следующее соглашение по работе с переменными. При обращении к переменным внутри процедур и триггеров достаточно просто указать идентификатор переменной, однако если переменная используется в конструкции SQL (например, совместно с оператором SELECT), то имени переменной должен предшествовать *маркер двоеточия* “:”.

Листинг 22.1 раскрывает еще одну важную синтаксическую конструкцию — *составной оператор* BEGIN...END. Все операторы, входящие в составной оператор рассматриваются компилятором как единый блок кода. Внутри BEGIN...END могут находиться не только операторы присваивания, но и условные операторы; структуры циклов; операторы вызова исключительных ситуаций; инструкции SQL; и т. п.

Выборка данных с помощью SELECT ... INTO

Обычно запросы на выборку основанные на инструкции SELECT предназначены для возвращения результирующего набора, включающего несколько строк с данными. Однако это не догма, случаются ситуации, когда создатель запроса планирует получить всего одну строку, например содержащую результат выполнения одной из функций агрегирования. В таких случаях (когда в результате выполнения запроса всегда будет возвращена лишь одна строка с данными) можно воспользоваться услугами оператора INTO, позволяющего явным образом передать значения из столбцов таблицы в заранее подготовленные переменные. Синтаксис подобного запроса выглядит следующим образом

```

SELECT <столбец1> [, < столбец2> ...]
FROM имя_таблицы WHERE <условия отбора>
INTO :переменная1 [, :переменная2 ...];

```

Сохраненные в переменных значения могут быть задействованы в условном операторе, в циклах или подвергнуты какой-то иной обработке. Листинг 22.2 демонстрирует порядок применения инструкции SELECT...INTO в хранимой процедуре

Листинг 22.2. Получение названия класса товара по его первичному ключу

```

CREATE PROCEDURE PR_GOODSCLASS_SELECTNODE
(AGOODSCLASS_ID INTEGER)
RETURNS
(AGOODSCLASS VARCHAR(255))
AS
begin
    SELECT GOODSCLASS FROM GOODSCLASS
    WHERE GOODSCLASS_ID=:AGOODSCLASS_ID
    INTO :AGOODSCLASS;
end

```

Условный оператор IF...THEN...ELSE

Организуя вычислительный процесс, программист должен обладать возможностью направлять ход вычислений в то или иное русло в зависимости от полученных в ходе выполнения программы промежуточных результатов или введенных пользователем исходных данных. Для этих целей в языке SQL имеется условный оператор IF...THEN.

Условный оператор, (или как его иногда называют *оператор ветвления*) задается с помощью инструкции

```

IF (условие) THEN <оператор> [ELSE <оператор>];

```

Поведение условного оператора определяется описываемым сразу за ключевым словом IF условием. В качестве условия может использоваться любой правильный предикат. Если условие истинно, то выполнению подлежит оператор (или блок операторов) следующий после THEN, иначе (если условие ложно) — обрабатываются альтернативные операторы, принадлежащие необязательной секции ELSE.

При необходимости условие может быть составным, в этом случае элементы условия объединяются с помощью логических операторов AND, OR и NOT. Предположим, что в после какого-то действия с БД мы обращаемся к паре переменных ASUPPLIER и CITYCODE. Дальнейшие действия зависят от состояния этих переменных (листинг 22.3).

Листинг 22.3. Пример условного оператора

```

IF ((CITYCODE >=100) AND (CITYCODE <=200))
    OR
    (ASUPPLIER IS NULL) THEN
BEGIN
    /*...*/
END

```

Как видите, составной оператор BEGIN...END подлежит выполнению только при условии, что значение переменной CITYCODE принадлежит диапазону чисел от 100 до 200 или если в переменной ASUPPLIER находится неопределенность NULL.

Для проверки значения на неопределенность следует использовать выражение IS NULL или IS NOT NULL. Все другие варианты неприемлемы!

Цикл WHILE...DO

Циклы предназначены для многократного выполнения заданной последовательности операторов тех пор, пока соблюдается условие выполнения цикла. В InterBase предусмотрена следующая синтаксическая конструкция цикла

```

WHILE (условие) DO
BEGIN
<операторы цикла>
END

```

Это так называемый цикл с предусловием, выполнение которого начинается с проверки истинности условия. Условие должно быть определено внутри круглых скобок, при необходимости условие может быть сложным и содержать несколько предикатов, объединенных логическими операторами.

Пример цикла WHILE...DO предложен в листинге 22.4. В данном случае мы разработали процедуру PR_LOOPDEMO осуществляющую последовательное приращение значения локальной переменной.

Листинг 22.4. Цикл WHILE...DO

```

CREATE PROCEDURE PR_LOOPDEMO
RETURNS (Y INTEGER)
AS
DECLARE VARIABLE X INTEGER;
BEGIN
    X=0;
    WHILE (X<10) DO
    BEGIN
        X=X+1;
    END
    Y=X;
END

```

Запустив процедуру на выполнение с помощью команды

```
EXECUTE PROCEDURE PR_LOOPDEMO;
```

мы получим результат: Y=10.

Цикл выборки данных FOR SELECT...DO

Некий симбиоз цикла и операции выборки данных представляет синтаксическая конструкция

```

FOR <предложение_SELECT>
DO <операторы_цикла>

```

Предложенное выражение позволяет построчно перебирать все строки отношения, возвращенного запросом SELECT. В запросе SELECT должно быть задействовано ключевое слово INTO, благодаря ему мы на каждой итерации цикла сможем передавать в переменную результаты запроса, а затем, полученный результат станет обрабатываться в рамках тела цикла.

Простой, но вместе с тем показательный пример цикла выборки предложен в листинге 22.5.

Листинг 22.5. Хранимая процедура выборки всех строк из таблицы поставщиков

```
CREATE PROCEDURE PR_SUPPLIER_SELECT_ALL
RETURNS
( ASUPPLIER VARCHAR(100),
  ID INTEGER,
  NUM INTEGER)
AS
BEGIN
  NUM=0;
  FOR
    SELECT SUPPLIER, SUPPLIER_ID FROM SUPPLIER
      ORDER BY SUPPLIER INTO :ASUPPLIER, :ID
  DO
    BEGIN
      NUM=:NUM+1;
      SUSPEND;
    END
  END
END
```

В данном листинге описывается хранимая процедура выборки всех записей из таблицы поставщиков с упорядочиванием их по алфавиту. На каждой итерации цикла выборки мы осуществляем приращение значения выходного параметра NUM на единицу. Что получится в результате? Для ответа на этот вопрос напомним обычный запрос SELECT в котором после ключевого слова FROM следует имя только что разработанной хранимой процедуры выборки (листинг 22.6).

Листинг 22.6. Обращение к хранимой процедуре из запроса SELECT

```
SELECT NUM, ID, ASUPPLIER FROM PR_SUPPLIER_SELECT_ALL
```

Результат выполнения запроса представлен на экранном снимке (рис. 2.1) — в нашем распоряжении появился виртуальный столбец с порядковым номером записи.

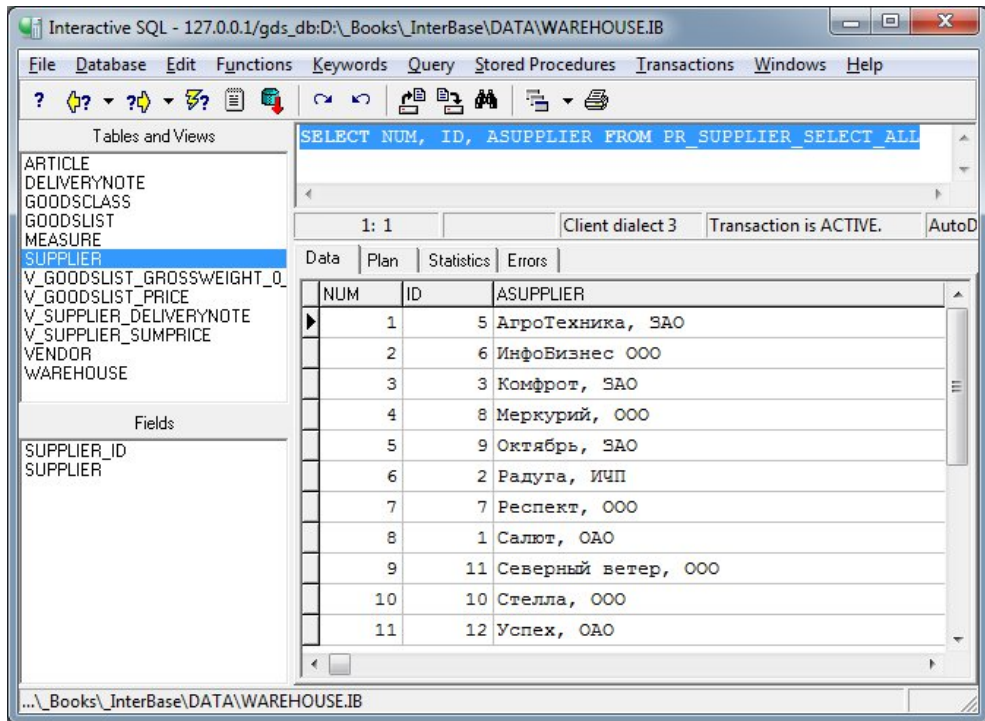


Рис. 22.2. Запрос к хранимой процедуре выборки FOR SELECT ... DO

Оператор SUSPEND

Оператор SUSPEND предназначен для небольшой приостановки выполнения программного кода и обычно применяется совместно с запросами выборки SELECT, в особенности если осуществляется построчная обработка строк отношения (курсоры и FOR SELECT ... DO).

Приостановка выполнения кода нужна с единственной целью — дать возможность системе заполнить выходной буфер только-что полученными значениями строки. Сразу после заполнения буфера выполнение кода продолжится автоматически.

Оператор `SUSPEND` следует применять исключительно в хранимых процедурах выборки, в остальных случаях он не нужен, т.к. вызов `SUSPEND` приводит к переходу к завершающему `END` процедуры.

Оператор *EXIT*

Как и в большинстве процедурных языков программирования для повышения процедурной гибкости в состав InterBase SQL введен оператор прерывания выполнения программного кода `EXIT`. В случае вызова `EXIT` все следующие за ним строки кода пропускаются и управление передается последнему оператору `END` процедуры.

Задание

Разработайте следующий перечень хранимых процедур:

1. Процедуры вставки, редактирования и удаления записей в одну из таблиц БД.
2. Процедура выборки всех строк из 2-х объединенных таблиц на основе выражения `FOR SELECT...DO`.
3. Произвольная процедура демонстрирующая работу условного оператора.
4. Произвольная процедура демонстрирующая работу с циклом.

В отчет о выполненной лабораторной работе необходимо включить исходный код разработанных процедур.

23. Процедурный SQL, триггеры и события

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – среда InterBase (FireBird)

Триггеры

Под триггером понимается особая разновидность хранимой процедуры, вызов которой осуществляется автоматически, вне зависимости от пожеланий пользователя БД. Целевое назначение триггеров в первую очередь связано с обеспечением бизнес-правил на уровне таблицы. Код триггера выступает последней линией обороны по поддержке целостности и непротиворечивости данных.

Создание нового триггера в InterBase осуществляется командой `CREATE TRIGGER`, сразу после которой следует уникальное для проектируемой БД имя триггера и название обслуживаемой таблицы:

```
CREATE TRIGGER имя_триггера FOR имя_таблицы
[ACTIVE | INACTIVE]
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE}
[POSITION номер]
AS
[локальные переменные]
BEGIN
/*код триггера*/
END
```

По умолчанию, сразу после создания триггер приступает к выполнению своих обязанностей. Такое поведение соответствует установке в инструкции SQL необязательного ключа `ACTIVE`. Если же по каким-либо причинам вам понадобится временно “спящий” триггер не реагирующий на изменения записей в таблице, то воспользуйтесь ключом `INACTIVE`. Подобный триггер не сложно оживить позднее с помощью инструкции `ALTER TRIGGER`.

Триггеры InterBase способны отвечать на 3 пары событий, связанных с модификацией данных в обслуживаемой таблице:

- ~ события `BEFORE INSERT` и `AFTER INSERT` возникают соответственно перед тем как новая строка попадет в таблицу и после того как эта строка будет сохранена;
- ~ события `BEFORE UPDATE` и `AFTER UPDATE` генерируется перед началом и после завершения редактирования данных;
- ~ событие `BEFORE DELETE` предшествует, а `AFTER DELETE` и завершает операцию удаления данных из таблицы.

С одной и той же таблицей может быть сопоставлено практически неограниченное число триггеров (от 0 до 32 767), как вы понимаете все они не могут выполняться одновременно — они могут быть вызваны в неопределенном порядке. Если за определенное событие отвечает единственный триггер, то о последовательности его запуска можно не думать, однако если таких триггеров несколько, то стоит определить очередность их срабатывания. Для этой цели в состав инструкции `CREATE TRIGGER` обязательно включите ключ `POSITION` и укажите порядковый номер триггера в очереди.

Контекстные переменные

Внутри тела триггера мы имеем право обращаться к двум разновидностям **контекстных переменных**: `NEW` и `OLD`. Для доступа к ним потребуется небольшое уточнение — имя столбца таблицы, с которым ассоциируется контекстная переменная, например, `NEW.SUPPLIER` или `OLD.VENDOR`.

Контекстная переменная `NEW` применяется в триггерах, вызываемых в момент вставки новой строки или модификации данных в таблице. В этой переменной окажется новое значение, претендующее на попадание в таблицу, но пока не сохраненное в ней. Контролируя передаваемые в таблицу значения, программист сможет построить хорошо эшелонированную линию обороны. Например, в столбцах `WEIGHT` и `GROSSWEIGHT` таблицы `GOODSLIST` должны содержаться величины веса нетто и брутто хранящегося на складе товара. Здравая логика подсказывает, что величина чистого веса нетто не может превышать значение веса товара в упаковке. Исходя из этого утверждения разработаем триггер (листинг 23.1) стоящий на страже только что рассмотренного бизнес-правила.

Листинг 23.1. Контроль правильности значений нетто и брутто

```

CREATE TRIGGER T_GOODSLIST_BI2 FOR GOODSLIST
ACTIVE BEFORE INSERT POSITION 1
AS
BEGIN
    IF ((NEW.WEIGHT IS NOT NULL)
        AND (NEW.GROSSWEIGHT IS NOT NULL)
        AND (NEW.WEIGHT>NEW.GROSSWEIGHT)) THEN
        /* Вес нетто не может превышать вес брутто! */
        EXCEPTION E_WEIGHT_EXCEPTION;
END

```

Столкнувшись с нарушением условия $\text{нетто} \leq \text{брутто}$ триггер вызывает подготовленное заранее исключение `E_WEIGHT_EXCEPTION`, которое в свою очередь прекратит операцию вставки данных и известит об этом пользователя.

В триггере разрешено применять почти все процедурные расширения языка SQL, в том числе условный оператор `IF...THEN...ELSE` и оператор цикла `WHILE...DO`.

В триггерах `BEFORE` значение контекстной переменной `NEW` может быть изменено (в триггерах `AFTER` это занятие не имеет смысла). При желании триггер контроля значений нетто и брутто можно переделать и вместо вызова исключения просто поправить содержание одного из столбцов, например, приравняв нетто и брутто

```
NEW.WEIGHT=NEW.GROSSWEIGHT
```

Обращение к переменной `OLD` имеет смысл в триггерах модификации и удаления данных — в ней хранится старое значение. Предложенный в листинге 23.2 код предназначен для осуществления каскадного удаления записей из таблицы классификатора товаров `GOODSCLASS`. Напомню, что таблица `GOODSCLASS` построена по принципу иерархической структуры в которой дочерние записи могут подчиняться родительским за счет связи ключевых полей `GOODSCLASS_ID` — `PARENT_ID`.

Листинг 23.2. Триггер каскадного удаления записей из таблицы GOODSCLASS

```

CREATE TRIGGER T_GOODSCLASS_BD FOR GOODSCLASS
ACTIVE BEFORE DELETE POSITION 0
AS
BEGIN
    IF (EXISTS (SELECT * FROM GOODSCLASS WHERE
                PARENT_ID=OLD.GOODSCLASS_ID)) THEN
        DELETE FROM GOODSCLASS WHERE PARENT_ID=OLD.GOODSCLASS_ID;
END

```

В коде триггера, вызываемого перед удалением строки из таблицы, осуществляется проверка наличия у подлежащей удалению строки идентифицируемой значением первичного ключа `OLD.GOODSCLASS_ID` подчиненных записей. Если такие обнаруживаются, то к ним применяется инструкция `DELETE`. Таким образом дочерние строки исключаются из таблицы до того, как это произойдет с родительской строкой.

Ввод значений по умолчанию

Еще один аспект применения триггеров связан с обеспечением ввода значений по умолчанию в столбцы, не допускающие неопределенность `NULL`. Напомню, что значение по умолчанию определяется в момент создания таблицы (или доменного ограничения) с помощью ключевого слова `DEFAULT`. СУБД гарантирует попадание заданного значения в столбец только при выполнении инструкции вставки `INSERT`. Если же речь идет о модификации данных, то ничто не запрещает пользователю передать в соответствующий столбец неопределенность `NULL`. Исправить ситуацию поможет триггер предложенный в листинге 23.3.

Листинг 23.3. Поддержка значения по умолчанию для столбца

```

CREATE TRIGGER T_DELIVERYNOTE_BU FOR DELIVERYNOTE
ACTIVE BEFORE UPDATE POSITION 0
AS
BEGIN
    IF (NEW.DNDATE IS NULL) THEN NEW.DNDATE=CURRENT_DATE;
END

```

Как видите при работе со значениями по умолчанию триггер гораздо более эффективен, чем все остальные инструменты, имеющиеся в распоряжении разработчика БД.

Если к таблице подключена цепочка триггеров BEFORE, то физически новое значение NEW попадет в столбец только после выполнения всей последовательности триггеров.

Поддержка корпоративной целостности данных

Зачастую в таблицах могут находиться данные, модификация которых недопустима или должна осуществляться в соответствии с особыми правилами. Эти правила определяются некоторыми внутренними, используемыми на данном предприятии установками. В свою очередь разработчики БД транслируют эти установки в правила поддержки корпоративной целостности данных. Например, в нашей БД при поступлении товаров на склад, товар автоматически закрепляется за складом по умолчанию, и только затем, по мере необходимости сотрудники склада осуществляют передачу товара на другие склады и торговые учреждения. Вполне естественно, что параметры склада по умолчанию (его название, атрибуты и самое главное — значение первичного ключа) должны быть постоянны и не допускать несанкционированных исправлений, иначе система может рухнуть.

В демонстрационной БД описание складов хранится в таблице WAREHOUSE. Понятно, что строка с информацией о складе по умолчанию заносится в эту таблицу еще на этапе введения БД в эксплуатацию и получает первичный ключ WAREHOUSE_ID=1. Как мы станем защищать эту запись от “нападок” не очень хорошо подготовленных пользователей которые зачастую ведут себя еще хуже, чем самые отъявленные вредители? Естественно с помощью триггера, так в листинге 23.4 предложен способ защиты записи от попыток удаления.

Листинг 23.4. Запрет на удаление определенной записи

```
CREATE TRIGGER T_WAREHOUSE_BD FOR WAREHOUSE
ACTIVE BEFORE DELETE POSITION 0
AS
BEGIN /*нельзя удалить склад по умолчанию*/
  IF (OLD.WAREHOUSE_ID=1) THEN EXCEPTION E_DEFAULTWAREHOUSE_EXCEPTION;
END
```

В рассматриваемом примере при попытке удалить запись с первичным ключом равным 1 немедленно генерируется исключение, и операция прерывается. Примерно такой же код необходимо разместить в триггере BEFORE UPDATE и тем самым запретить внесение изменений в эту очень важную запись.

События

В хранимых процедурах и триггерах может использоваться особый механизм, благодаря которому InterBase получает возможность извещать свои клиентские приложения о том или ином событии, произошедшем внутри БД. В простейшем случае в качестве примеров подобных событий может стать какая-либо важная операция модификации строк в таблице. В качестве более глобальных поводов о необходимости немедленного информирования клиентов можно привести событие поступления в торговую сеть новой партии товаров; сообщения о том, что количество запасов какого-то материала достигло критического уровня; сообщение о том, что сервер будет перезагружен и т.д.

Для того, чтобы механизм событий заработал в полную силу необходимо описать порядок взаимодействия между вызывающим событие сервером и ожидающим событие клиентом. С сервером все предельно просто — ему достаточно известить клиентов о каком-то свершившемся факте с помощью оператора

```
POST_EVENT <имя_события>
```

Имя события чувствительно к регистру и ограничивается 64 символами.

Встретив в SQL-коде оператор POST_EVENT сервер передает бразды управления менеджеру событий, который (в случае корректного завершения транзакции) самостоятельно отправит известие о событии клиентам.

Так как в большинстве случаев пользовательское приложение в первую очередь интересуется процедурами вставки, модификации и удаления данных, то для генерации событий лучше всего подходят триггеры AFTER. Например, триггер из листинга 23.5 каждый раз после успешного добавления очередной записи в таблицу поставщиков товаров SUPPLIER будет радовать клиентские приложения событием “SUPPLIER_INSERT”.

Листинг 23.5. Триггер вызывающий событие “вставка новой записи”

```
CREATE TRIGGER T_SUPPLIER_AI_EVENT FOR SUPPLIER
AFTER INSERT AS
BEGIN
  POST_EVENT 'SUPPLIER_INSERT';
END;
```

Заметим, что триггер просто осуществит широковещательную отправку сообщения всем прослушивающим сервер клиентам. Само по себе рассылаемое сообщение никакой дополнительной информационной нагрузки не несет, посему дальнейшее развитие ситуации целиком и полностью возлагается на обрабатывающее событие клиентское приложение.

Событие генерируется только после успешного завершения транзакции, если же осуществляется откат транзакции вызов оператора `POST_EVENT` пропускается.

Если клиент пишется на языке Delphi (или C++ Builder), то для обеспечения реакции клиента на событие достаточно воспользоваться услугами штатного компонента `TIBEvents`.

Задание

1. Предложите примеры 3-4 триггеров, обеспечивающих поддержку корпоративной целостности данных.
2. Для произвольной таблицы создайте три события, уведомляющие о:
 - a. вставке новой записи;
 - b. редактировании записи;
 - c. удалении записи.

24. Функции UDF

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – среда InterBase (FireBird), C++ (Delphi)

Сервер InterBase не может похвастаться большим набором встроенных функций, охватывающих весь спектр вероятных задач способных возникнуть перед разработчиком БД. Скажем больше — встроенных функций меньше десятка (приложение 1). Однако этот факт ни в коем случае не говорит о слабости InterBase. Почему? Потому, что создатели InterBase сделали так, чтобы сервер стал способен подключать к БД внешние, или как их еще называют — определенные пользователем функции (user-defined function, UDF).

Внешние функции могут быть разработаны на таких языках программирования, как Delphi, C++ Builder компании Embarcadero и Visual C++ корпорации Microsoft. Список языков программирования можно и расширить. По большому счету единственным условием попадания того или иного кандидата в перечень языков способных написать UDF выступает единственный критерий — умение компоновать и компилировать динамически подключаемые библиотеки (Dynamic Link Library, DLL). Именно DLL и выступает контейнером для пользовательских функций в случае если СУБД развернута на платформе Windows.

Размещение UDF-библиотеки

Для того, чтобы сервер InterBase смог воспользоваться UDF написанными сторонними разработчиками динамические библиотеки должны быть скопированы в определенные папки. В простейшем случае, вне зависимости от операционной системы вы можете перенести файлы с библиотеками в папку:

`<напка_InterBase>\UDF`

или в папку:

`<напка_InterBase>\intl`

Подключение внешней функции к БД

Размещение файла библиотеки в целевой папке необходимое, но недостаточное условие для работы с UDF. Для того чтобы наша база данных смогла воспользоваться услугами UDF-библиотеки стоит разобраться с процессом подключения внешних функций к нашему проекту. С этой целью рассмотрим синтаксическую конструкцию `DECLARE EXTERNAL FUNCTION`, отвечающую за импорт внешней функции в БД.

```
DECLARE EXTERNAL FUNCTION имя_функции
[тип_данных ;| CSTRING (int) | DESCRIPTOR
[, тип_данных | CSTRING (int) ...] DESCRIPTOR]
RETURNS { тип_данных [BY VALUE] | CSTRING (int) | PARAMETER n} [FREE_IT]
ENTRY_POINT 'имя_функции_в_модуле'
MODULE_NAME 'имя_модуля';
```

В листинге 24.1 продемонстрирован порядок подключения к БД функции `LTRIM()` из поставляемого в составе InterBase модуля внешних функций `ib_udf`.

Листинг 24.1. Подключение внешней функции `IB_UDF_ltrim()`

```
DECLARE EXTERNAL FUNCTION LTRIM
CSTRING(80)
RETURNS CSTRING(80) FREE_IT
ENTRY_POINT 'IB_UDF_ltrim' MODULE_NAME 'ib_udf';
```

Задача только что подключенной функции — подавление всех пробелов, находящихся перед текстом. Это крайне полезная услуга позволит нам бороться с ненужными пробелами, которые могут быть случайно вставлены пользователем в столбец той или иной таблицы.

Подключение UDF в консоли администрирования

Консоль управления сервером InterBase позволяет осуществить процесс подключения к БД пользовательской функции не только за счет непосредственного использования SQL, но и с помощью удобного визуального интерфейса. Для этого необходимо открыть БД и обратиться к узлу **External Functions** утилиты **IB-Console**.

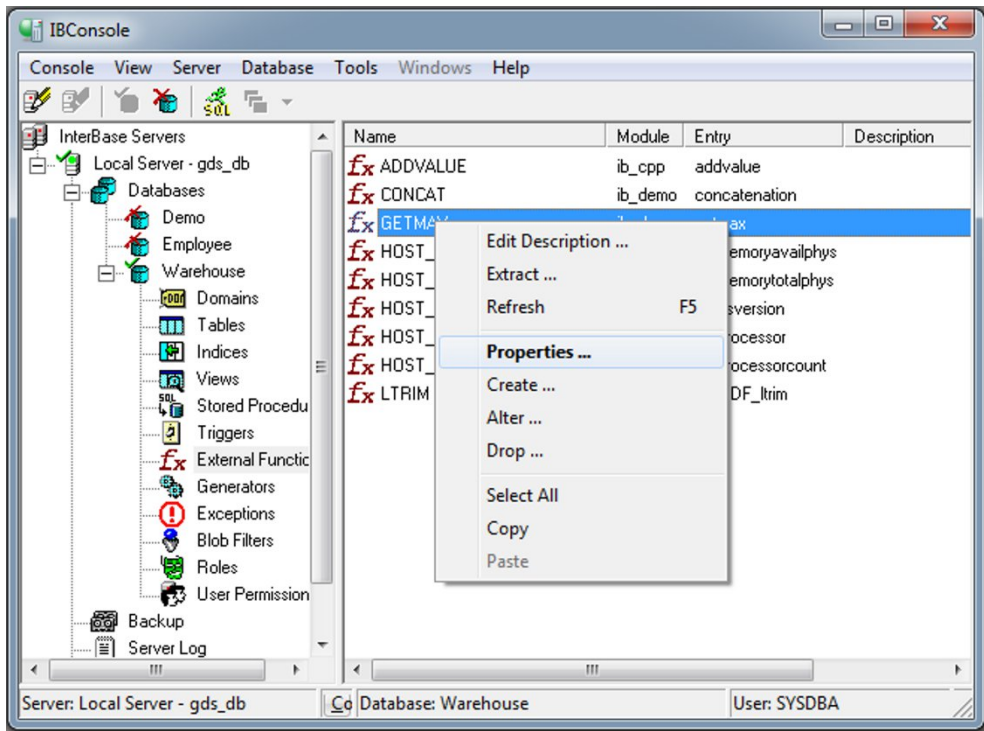


Рис. 24.1. Управление UDF с помощью консоли администрирования IBConsole

Выбор узла внешних функций заставит утилиту построить список всех задействованных в текущей БД пользовательских функций. Управление этими функциями, в том числе подключение, модификация и отключение осуществляется при посредничестве контекстного меню (рис. 24.1).

вызов UDF

Как проверить работоспособность подключенной UDF-функции? Для тестирования можно воспользоваться любой из инструкций DML (SELECT, INSERT, UPDATE и DELETE). Простейший способ проверки — вызов функции в SQL-запросе основанном на ключевом слове SELECT (листинг 24.2).

Листинг 24.2. вызов внешней функции LTRIM()

```
SELECT LTRIM('      ABC') FROM rdb$database;
```

Запустив запрос на выполнение вместо “ ABC” в ответ вы получите строку с удаленными пробелами перед основным текстом — “ABC”.

Объявленные в БД пользовательские функции могут быть вызваны как из запросов динамического SQL, так и из хранимых процедур и триггеров.

Разработка UDF-библиотек в Delphi

Для разработки библиотек содержащих подключаемые пользовательские функций можно воспользоваться услугами языка программирования Delphi. Для этого в главном меню среды разработки выбираем пункт **File | New | Other** и в окне **New Items** находим элемент **Dynamic-link Library**. В ответ на наши действия Delphi создаст шаблон кода, начинающийся с ключевого слова **library** — это и есть визитная карточка проекта библиотеки DLL. Сразу сохраните проект. При выборе имени для проекта помните, что после того как вы нажмете кнопку **F9** это имя определит название для библиотеки. В моем первом примере DLL станет называться **ib_demo**.

Для 64-битной версии InterBase следует разрабатывать исключительно 64-битные библиотеки!

Напишем функцию **getmax()**, сравнивающую два целочисленных значения и возвращающих наибольшее из них (листинг 24.3).

Листинг 24.3. Библиотека на языке Delphi с функцией getmax()

```
library ib_demo;

uses
  System.SysUtils,
```

```

System.Classes;

{$R *.res}

function getmax(var x:integer; var y:integer):integer; cdecl; export;
begin
    if x>y then result:=x
    else result:=y;
end;

exports getmax; //секция экспорта

begin
end.

```

Для того чтобы сервер InterBase (как, впрочем, и любое стороннее приложение) смогло воспользоваться описанными в библиотеке функциями, эти функции должны быть помечены, как экспортируемые. Синтаксис объявления экспортируемых библиотекой функций имеет две особенности:

- ~ между экспортирующей функцию библиотекой и выступающим в роли импортера приложением должно быть заключено *соглашение о вызовах*, оно необходимо для обеспечения корректной передачи параметров и возврата результата выполнения функции. Определяя соглашение о вызовах для функции, предназначенной для вызова из InterBase задействуйте стандартное соглашение `stdcall` или соглашение в стиле языка Си `cdecl`;
- ~ объявление экспортируемой функции со спецификатором `export` и использование секции `exports` для указания имен экспортируемых процедур и функций.

Откомпилировав библиотеку из листинга 24.3 перенесите `dll` в каталог `<напка_InterBase>\UDF`

и перейдите в консоль управления InterBase. Теперь нам предстоит подключить UDF к базе данных (листинг 24.4).

Листинг 24.4. Подключение функции GETMAX() к БД

```

DECLARE EXTERNAL FUNCTION GETMAX
INTEGER, INTEGER
RETURNS INTEGER BY VALUE
ENTRY_POINT 'getmax' MODULE_NAME 'ib_demo';

```

Работа со строками

При необходимости работы в UDF с текстовыми параметрами учитывайте тот факт, что фирменные строки Delphi (тип данных `String`) несовместимы с текстовыми типами данных в InterBase. Посему, при определении типов данных для передачи аргументов и возврата результата выполнения функции задействуем исключительно указатели `pAnsiChar` (или `pChar`). Представленный в листинге 24.5 пример информирует о типе процессора, используемого в компьютере на котором развернута СУБД.

Листинг 24.5. Возврат UDF на языке Delphi текстового значения

```

function getprocessor:pAnsiChar; CDECL; export;
begin
    case TOSVersion.Architecture of
        arIntelX86 : Result:='Intel x86';
        arIntelX64 : Result:='Intel x64';
        arARM32    : Result:='Архитектура Acorn RISC Machine';
        else      : Result:='Unknown';
    end;
end;

```

При подключении UDF возвращающей текстовые данные воспользуемся типом данных `CSTRING(n)` с указанием максимального числа символов (листинг 24.6).

Листинг 24.6. Подключение функции getprocessor() к БД

```

DECLARE EXTERNAL FUNCTION HOST_PROCESSOR
RETURNS CSTRING(10) CHARACTER SET NONE
ENTRY_POINT 'getprocessor' MODULE_NAME 'ib_host';

```

Рассмотрим еще один небольшой пример, на этот раз демонстрирующий порядок передачи двух текстовых аргументов на вход функции (листинг 24.7).

Листинг 24.7. Передача текстовых аргументов в функцию Delphi

```
function concatenation(A:pAnsiChar; B:pAnsiChar):pAnsiChar; cdecl; export;
var C:Ansistring;
begin
    C:=StrPas(A)+StrPas(B);
    result:=pAnsiChar(C);
end;
```

Мы написали функцию, позволяющую осуществлять конкатенацию двух строк, для ее подключения к БД воспользуйтесь листингом 24.8.

Листинг 24.8. Подключение функции concatenation() к БД

```
DECLARE EXTERNAL FUNCTION CONCAT
CSTRING(127), CSTRING(127)
RETURNS CSTRING(255)
ENTRY_POINT 'concatenation' MODULE_NAME 'ib_demo';
```

Особенности разработки в C++ Builder

Пользовательские функции для InterBase в равной степени удобно проектировать как на Delphi, так и на C++ Builder. Так что если вы предпочитаете язык C++, то все в ваших руках. Листинг 24.9 содержит код элементарной пользовательской функции addvalue(), предназначенной для суммирования двух вещественных значений.

Листинг 24.9. Функция суммирования addvalue() на языке C++

```
#pragma hdrstop
#pragma argsused

extern "C" int _libmain(unsigned long reason)
{
    return 1;
}

extern "C" double __declspec(dllexport) addvalue(double *x, double *y);
//-----
double addvalue(double *x, double *y)
{
    return *x+*y;
}
```

При работе с UDF в C++ (впрочем, как и в случае с Delphi) на вход функции аргументы должны передаваться исключительно по ссылке — это требование InterBase. Поэтому определяя параметры функции из листинга 24.9 мы воспользовались оператором *. Для возврата суммы мы напротив, воспользовались способом передачи данных по значению.

Листинг 24.10 содержит код подключения функции суммирования к БД.

Листинг 24.10. Подключение функции addvalue() к БД

```
DECLARE EXTERNAL FUNCTION ADDVALUE
DOUBLE PRECISION, DOUBLE PRECISION
RETURNS DOUBLE PRECISION BY VALUE
ENTRY_POINT 'addvalue' MODULE_NAME 'ib_cpp';
```

Еще один пример написанной на C++ пользовательской функции демонстрирует порядок возврата текстовых данных (листинг 24.11).

Листинг 24.11. Функция преобразования номера в название месяца на языке C++

```
extern "C" char* __declspec(dllexport) monthname(int *monthnum);
//-----
char* monthname(int *monthnum)
{
```

```

static char* month[12]=
    {"Январь", "Февраль", "Март", "Апрель", "Май", "Июнь", "Июль",
     "Август", "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"};
if (*monthnum>=0 && *monthnum<12){
    return month[*monthnum];
} else return "";
}

```

На этот раз мы разработали функцию которая получив на вход порядковый номер месяца года возвратит нам его текстовое название. Обратите внимание, что на этот раз возврат данных мы производим по ссылке, задействовав тип данных char*.

Импортируя в InterBase написанную на C++ функцию возвращающую текстовую строку хорошим тоном является использование оператора `FREE_IT` уведомляющую вызвавший динамическую библиотеку процесс о том, что теперь можно очистить выделенную для строки память (листинг 24.12)

Листинг 24.12. Подключение функции monthname() к БД

```

DECLARE EXTERNAL FUNCTION MONTHNAME
INTEGER
RETURNS CSTRING(10) CHARACTER SET NONE FREE_IT
ENTRY_POINT 'monthname' MODULE_NAME 'ib_cpp';

```

Завершающий штрих — проверка работоспособности функции. Для этого вызовем из интерактивного SQL последнюю в этой главе строку кода (листинг 24.13).

Листинг 24.13. вызов внешней функции MONTHNAME()

```

SELECT MONTHNAME(4) FROM rdb$database;

```

В ответ мы получим название месяца с индексом 4.

Задание

- Предложите пример работы с любой из встроенных функций (приложение 1).
- Разработайте (на Delphi или C++ Builder) и подключите к БД UDF–библиотеку, содержащую три функции из представленного ниже перечня:
 - произведение двух целых чисел;
 - произведение двух вещественных чисел;
 - возврат целочисленного значения текущего года;
 - генерация случайного вещественного числа в диапазоне от 0 до 1;
 - генерация случайного целого числа в диапазоне от 0 до 100;
 - расчет выражения $(x+y)^2$;
 - получение значения $\sin(x)$ где x — значение угла в градусах;
 - расчет выражения b^2-4ac ;
 - поиск наибольшего значения из 3 целых чисел;
 - поиск наименьшего значения из 4 вещественных чисел.

25. Подключение к БД из клиентского приложения

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – среда InterBase (FireBird), C++ (Delphi)

Современные версии Embarcadero RAD Studio предоставляют разработчику программного обеспечения по крайней мере 4 альтернативных способа по организации доступа клиентского приложения к БД InterBase:

1. классический подход, основанный на линейке компонентов InterBase (IBX), именно ему мы и уделим максимум своего внимания;
2. доступ к данным на основе компонентов dbExpress (DBX);
3. технология FireDAC;
4. кроме того, в составе InterBase предусмотрены библиотеки для поклонников платформы .NET.

Структура опирающегося на компоненты IBX клиентского приложения представлена на рисунке 25.1. Весь механизм взаимодействия между СУБД и клиентом содержится в специальном клиентском программном обеспечении, ядро которого составляет библиотека gds32.dll.

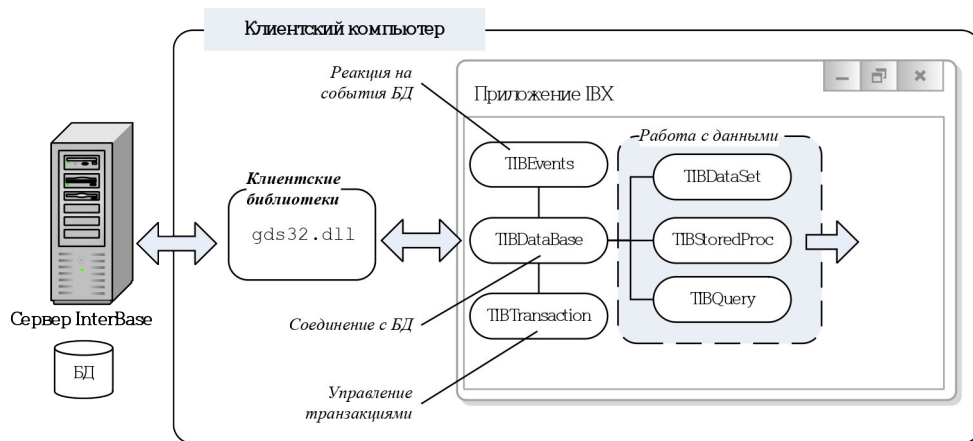


Рис. 25.1. Архитектура клиентского приложения на основе компонентов IBX

Первая версия СУБД InterBase была создана талантливым программистом Джимом Старки (Jim Starkey) для компании Groton Database Systems. Именно поэтому в базовой библиотеке gds32.dll применяется аббревиатура gds, кроме того gds зачастую используется в качестве расширения для файла с данными.

Для работы клиентского приложения InterBase необходимо наличие на компьютере библиотеки gds32.dll. При разработке программного обеспечения следует учитывать, что в составе клона InterBase (СУБД FireBird) также имеется одноименная библиотека gds32.dll, которая не совместима с файлом библиотеки InterBase.

Доступ к базе данных, компонент TIBDatabase

Компонент TIBDatabase предназначен для организации доступа клиентского приложения к базе данных, обслуживаемой сервером InterBase. Наличие компонента TIBDatabase в клиентском приложении обязательное условие, так как его услугами пользуются практически все остальные компоненты доступа к данным (рис. 25.1), для соединения с базой данных у них объявлено свойство Database.

Соединение с базой данных

Современные версии Delphi позволяют создавать как многопользовательские клиент-серверные проекты InterBase, так и простые однопользовательские системы. За режим работы приложения отвечает свойство `property ServerType : string; //по умолчанию IBServer`

Книга посвящена клиент-серверным БД, поэтому в наших проектах свойство должно быть установлено в состояние по умолчанию — IBServer.

Для организации соединения с базой данных необходимо указать имя БД. Для этого предназначено свойство

property DatabaseName: string

Имя определяется названием файла базы данных, а формат его описания зависит от того, под какой операционной системой выполняется СУБД InterBase и сетевого протокола, с помощью которого осуществляется доступ к серверу. В простейшем случае, когда база данных размещается на том же компьютере, на котором установлено наше приложение, мы ограничиваемся обычным путем к файлу

```
IBDatabase.DatabaseName:= 'c:\ibdata\warehouse.ib';
```

Но если InterBase выполняется на удаленном сервере, то порядок определения имени слегка усложняется. Для соединения с СУБД мы можем выбирать любой из наиболее распространенных сетевых протоколов: TCP/IP или NetBEUI. По возможности предпочтение стоит отдавать современному протоколу TCP/IP.

При подключении к сетевой БД с помощью TCP/IP, путь к файлу должен начинаться с указания сетевого имени компьютера

```
IBDatabase.DatabaseName:= 'IBHost:C:\IBData\warehouse.ib';
```

Если по какой-то причине стек протоколов TCP/IP недоступен, то организуем работу с помощью протокола NetBEUI, но в этом случае несколько изменится порядок описания имени базы данных.

```
IBDatabase.DatabaseName:= '\\IBHost\C:\IBData\warehouse.ib';
```

Параметры соединения передаются при посредничестве свойства

property Params: TStrings;

Во время проектирования (чтобы избежать повторения процедуры регистрации при каждом перезапуске приложения) в это свойство можно внести данные об имени и пароле пользователя, при этом, не забыв отключить автоматический вызов диалога регистрации, установив в false свойство

property LoginPrompt : Boolean;

Для обращения к отдельным параметрам соединения можно обратиться к свойству

property DBParamByDPB[const Idx: Integer]: string;

В свойство следует передать имя соответствующей константы, например user_name. Все константы объявлены в модуле IBHeader.

Для очень забывчивых разработчиков может пригодиться метод

procedure CheckDatabaseName;

Он проверяет факт заполнения свойства DatabaseName, и если свойство пусто — вызывает исключительную ситуацию.

Завершив все подготовительные операции, вызываем метод

procedure Open;

либо переводим в true свойство:

property Connected : Boolean;

Ко всему прочему свойство Connected может использоваться для проверки факта соединения с БД (листинг 25.1).

Листинг 25.1. Пример подключения к локальной БД

```
procedure TfrmDataModule.DataModuleCreate(Sender: TObject);
begin
  with IBDatabase1 do
  begin
    DatabaseName:= 'C:\IBData\warehouse.ib';
    LoginPrompt:=False;
    Params.Clear;
    Params.Add('USER "SYSDBA"'); //имя пользователя
    Params.Add('PASSWORD "masterkey"'); //пароль
    Connected:=True;
  end;
```

end;

Такой код применяется только во время разработки проекта, а окончательная версия нашего программного продукта просто обязана потребовать от пользователя ввода регистрационных данных.

Для доступа к серверу InterBase необходимо указать имя и пароль пользователя. По умолчанию права системного администратора предоставит следующая учетная запись: пользователь “SYSDBA”, пароль “masterkey”.

Обычно программисты предпочитают управлять ходом соединения клиентского приложения с БД вручную в коде программы. Но зачастую контакт с базой данных необходим и во время разработки, например для инициализации свойств и параметров компонентов хранимых процедур или для генерации списка статических полей у компонентов наборов данных. После этого перед стартом программы важно не забыть вернуть свойство `Connected` в состояние `false`, в противном случае Delphi сгенерирует исключительную ситуацию. Если подключение/отключение надо произвести пару раз за день, то ничего страшного, но если проект перекомпилируется многократно... К счастью в компоненте `TIBDatabase` острота проблемы существенно снижена благодаря свойству

property `AllowStreamedConnected` : Boolean; //по умолчанию true

Свойство предназначено исключительно для упрощения процесса отладки проекта. Переведя свойство `AllowStreamedConnected` в состояние `false`, мы освобождаемся от необходимости постоянно включать/отключать `Connected` во время разработки. Компонент самостоятельно, без какого-то постороннего вмешательства, отсоединится от базы данных в момент старта проекта (ожидая явной команды на соединение в коде программы), и вновь подключится к БД после его остановки.

Еще одно полезное качество класса `TIBDatabase` — безопасное тестирование соединения. В процессе подготовки соединения перед вызовом метода `Open()` хорошей идеей может стать вызов проверочного метода

procedure `TestConnected`: Boolean;

Процедура проверит корректность параметров соединения и при успехе вернет значение `true`, при ошибке мы получим `false`, но (что очень важно) при этом не будет сгенерирована исключительная ситуация.

Для настройки параметров соединения во время разработки проекта программист может обратиться к контекстному меню компонента и воспользоваться пунктом “Database editor” (рис.25.2).

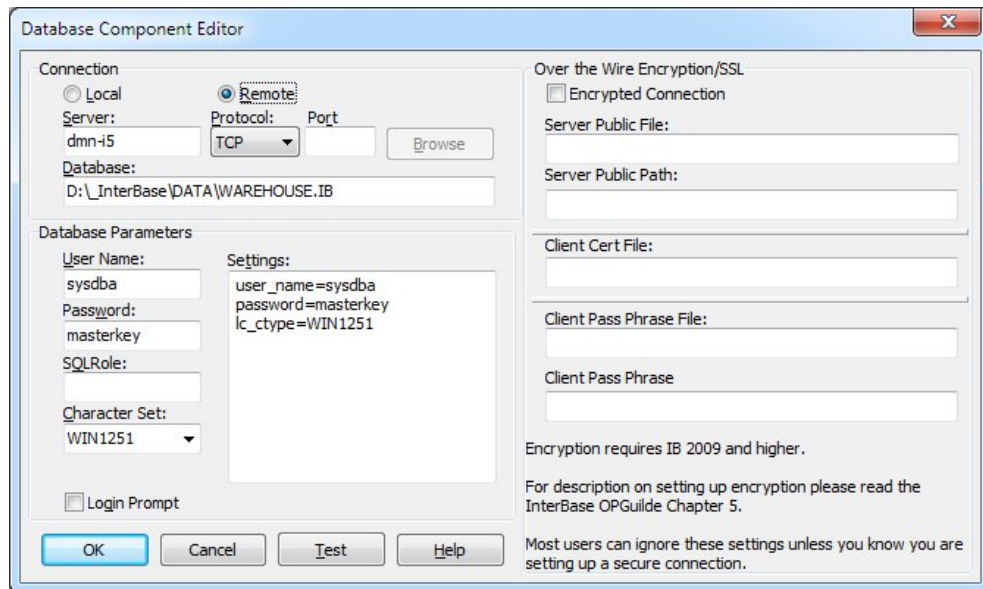


Рис. 25.2. Редактор настройки соединения

С процессом соединения связано два события

property `BeforeConnect`: TNotifyEvent; //перед началом соединения

property `AfterConnect`: TNotifyEvent; //после успешного соединения

Они могут пригодиться для инициализации других компонентов проекта.

Регистрация пользователя

Для вызова штатного диалога регистрации пользователя установите в `true` свойство `LoginPrompt`. В этом случае, обращение к методу `Open()` инициирует событие:

```
property OnLogin: TIBDatabaseLoginEvent;
TIBDatabaseLoginEvent = procedure (Database: TIBDatabase;
                                     LoginParams: TStrings) of object;
```

Регистрационные данные пользователя направляются параметр `LoginParams`.

Допустим, что у нас имеется отдельная форма `TfrmUserLogin` со строками ввода `edUser:TEdit` и `edPassword:TEdit` предназначенными для ввода имени и пароля пользователя. Тогда процесс регистрации можно организовать так, как предложено в листинге 25.2.

Листинг 25.2. Регистрация пользователя на сервере InterBase

```
procedure TfrmDataModule.IBDatabase1Login(Database: TIBDatabase;
                                           LoginParams: TStrings);

var frmUserLogin: TfrmUserLogin;
begin
  frmUserLogin:=TfrmUserLogin.Create(nil); {создаем форму}
  if frmUserLogin.ShowModal=mrOK then
    with LoginParams do
      begin
        Clear;
        Add('USER'+frmUserLogin.edUser.Text); //имя пользователя
        Add('password'+frmUserLogin.edPassword.Text); //пароль
      end;
      frmUserLogin.Release;
end;
```

Разрыв соединения

Кроме анализа состояния свойства `Connect` для проверки наличия соединения с БД программист может использовать методы:

```
procedure CheckActive; //проверка активности соединения
procedure CheckInactive; //проверка пассивности
```

Особенность процедур `Check...` в том, что при невыполнении условия проверки они генерируют исключительную ситуацию (листинг 25.3).

Листинг 25.3. Шаблон кода завершения работы с БД

```
uses IB;
...
procedure ...
begin
  try
    IBDatabase1.CheckInactive;
    {какие-то операции, осуществление которых возможно только
     при отсутствии соединения}
  except
    on EIBClientError do //проверяем класс ИС
      begin
        MessageBox(Handle,
                    PChar('Данная операция невозможна
                           при активном соединении!'),
                    PChar('Ошибка'),
                    MB_ICONERROR+MB_OK);
      end;
  end;
end;
end;
```

Для отключения от базы данных обращаемся к методу `Close()` либо переводим в `false` свойство `Connected`.

Для отключения всех активных наборов данных с сохранением соединения с БД вызываем метод **procedure** CloseDataSets;

После обращения к методу все обслуживаемые нашим компонентом потомки TDataSet перейдут в пассивный режим (Active=false), но TIBDatabase по-прежнему сохранит контакт с базой.

Процесс разрыва соединения сопровождается парой событий

property BeforeDisconnect: TNotifyEvent;

property AfterDisconnect: TNotifyEvent;

События возникают соответственно перед началом разрыва соединения, и по факту разрыва.

При необходимости можно ограничить время простоя соединения. Для этого предназначено свойство:

property IdleTimer: Integer; //по умолчанию 0 — неограниченно

Время исчисляется в миллисекундах. По умолчанию свойство установлено в ноль, это означает, что ограничения отсутствует. По истечению времени простоя генерируется событие:

property OnIdleTimer: TNotifyEvent;

Это событие может быть использовано для разрыва соединения с БД.

Управление транзакциями, компонент TIBTransaction

Компонент TIBTransaction предоставляет услуги по управлению транзакциями из клиентского приложения с использованием одного или нескольких соединений с БД. Это обязательный ингредиент любого клиентского приложения InterBase, так как в услугах транзакций нуждаются все наборы данных (потомки класса TIBCustomDataSet) и компонент TIBSQL — выполняемые ими SQL-инструкции всегда должны выполняться в контексте транзакции.

Клиентское приложение должно как минимум содержать один компонент-транзакцию. Но единственного компонента достаточно только для простейших проектов. В реальности компонентов TIBTransaction должно быть ровно столько, сколько требует логика работы приложения. Конечно же, не стоит присоединять личную транзакцию к каждой таблице, хранимой процедуре и запросу InterBase, это окажется явным перебором. Имеет смысл выделить индивидуальные компоненты TIBTransaction для обслуживания наиболее сложных SQL команд и запросов.

Работа с компонентом обычно начинается с подключения к базе данных. Ссылка на базу данных по умолчанию находится в свойстве

property DefaultDatabase: TIBDatabase;

Параметры транзакции

По умолчанию транзакция устанавливается в наиболее востребованное для большинства клиентских приложений InterBase состояние:

- ~ уровень изоляции — SNAPSHOT;
- ~ доступны операции записи и чтения;
- ~ при обращении к заблокированному ресурсу транзакция переходит в режим ожидания освобождения блокировки.

Вполне естественно, что предустановленные значения не могут подойти ко всем случаям жизни, поэтому компонент TIBTransaction предоставляет нам возможность перенастроить поведение транзакции. Доступ из кода приложения к содержащему параметры буферу, обеспечивает свойство

property Params: TStrings;

или

property TPB: PChar; //только для чтения

В параметрах транзакции содержится информация об уровне изоляции транзакции, порядке использования точек сохранения, разрешение на редактирование данных. Для определения параметров транзакции проще всего воспользоваться услугами редактора транзакции, который вызывается из контекстного меню компонента TIBTransaction (рис. 25.3).

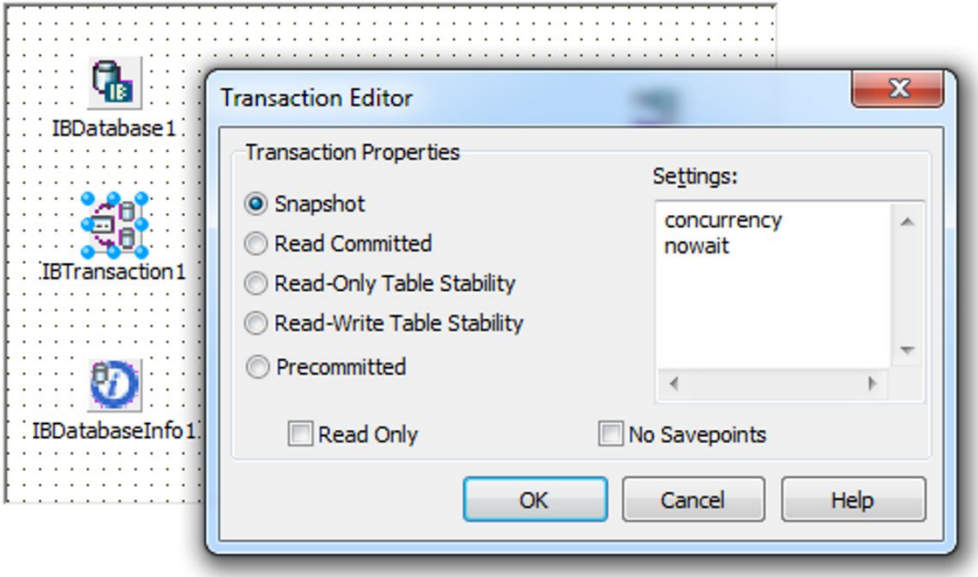


Рис. 25.3. Окно редактора параметров транзакции

Как уже упоминалось на лекции уровни изоляции транзакций InterBase несколько отличаются от требований стандарта SQL. вы можете выбрать наиболее подходящий уровень просто щелкнув мышкой по соответствующей кнопке выбора:

- ~ самую низкую степень изолированности обеспечит уровень Read Committed (неповторяемое чтение).
- ~ уровень Precommitted соответствует уровню Read Committed в режиме только для чтения
- ~ уровень Snapshot (моментальный снимок) соответствует средней степени изоляции.
- ~ наибольшую степень изоляции обеспечивают уровни воспроизводимого чтения Read-Only Table Stability и Read-Write Table Stability, в режиме Read-Only операторы внутри транзакции не могут модифицировать данные, в режиме Read-Write — модификации допускаются.

Автоматическое управление транзакцией

Компонент TIBTransaction достаточно умен для того, чтобы самостоятельно начинать и завершать выполнение транзакции. Степень доверия программиста к компоненту определяется свойствами

property AllowAutoStart : Boolean; //по умолчанию true
property AutoStopAction : TAutoStopAction; //по умолчанию saNone

Свойство AllowAutoStart установленное в состояние true указывает компоненту на то, что при необходимости он должен самостоятельно запустить транзакцию. Свойство AutoStopAction определит действие по умолчанию в момент завершения транзакции. Варианты допустимых значений TAutoStopAction представлены в таблице 25.1.

Таблица 25.1. Возможные значения TAutoStopAction

Значение	Описание
saNone	Компонент не может самостоятельно управлять завершением транзакций
saRollback	Компонент имеет право на самостоятельность при откате транзакции
saCommit	Компоненту разрешено фиксировать транзакцию
saRollbackRetaining	Возврат к точке сохранения. Поддерживается, начиная с InterBase 6.0 Транзакция не завершается пока есть незакрытые наборы данных. Все изменения в связанных с транзакцией данных отменяются. Другими словами, производится откат транзакции с сохранением ее контекста
saCommitRetaining	Транзакция не завершается до тех пор, пока имеются незакрытые наборы данных. Все изменения в охваченных транзакцией записях фиксируются. Контекст транзакции сохраняется

Если транзакция обслуживает компонент TIBSQL, то свойство AutoStopAction должно быть установлено в состояние saNone.

Если транзакция обслуживает наборы данных (компоненты `TIBTable`, `TIBQuery` или `TIBDataSet`) то заложенное в свойство `AutoStopAction` обработает только после того, как будет закрыт связанный с транзакцией набор данных. Другими словами, все изменения, сделанные набором данных в рамках транзакции, не зафиксируются до тех пор пока набор данных остается активным.

Для принудительного вызова действия, указанного в свойстве `AutoStopAction` следует вызвать метод

procedure `CheckAutoStop;`

В автоматический режим управления транзакцией лучше всего ставить тот компонент `TIBTransaction` на который компонент-соединение с БД `TIBDatabase` ссылается с помощью свойства `DefaultTransaction`.

Управление транзакцией в ручном режиме

В тех ситуациях, когда автоматическая работа транзакции нежелательна программисту следует брать бразды правления компонентом `TIBTransaction` в свои руки. В первую очередь для этого в инспекторе объектов следует отключить свойства `AllowAutoStart` и `AutoStopAction` и приступить к программированию.

Первое правило с проверки которого должны начинаться стоки кода заключается в проверке активности компонента-транзакции. Если в текущий момент времени выполняется стартовавшая ранее транзакция, то попытка вновь воспользоваться услугами компонента может отменить предыдущую транзакцию. Об активности судят по свойству

property `Active: Boolean;`

Если свойство возвратит `true`, то компонент лучше пока не трогать — он в работе. Заметим, что свойство доступно и для записи, оно позволяет стартовать транзакцию даже во время визуального проектирования, поэтому не стоит злоупотреблять доброжелательностью компонента (это может сослужить плохую службу), тем более что перевод `Active` в состояние `false` аварийно завершает текущую транзакцию вызывая метод `Rollback`. Поэтому будем полагать, что свойство `Active` следует использовать только в режиме для чтения.

Альтернативным способом проверки состояния транзакции может стать обращение к свойству

property `InTransaction: Boolean; //только для чтения`

Значение `true` свидетельствует, что транзакция еще не завершена.

Убедившись, что транзакция не активна, для явного запуска новой транзакции обращаемся к процедуре

procedure `StartTransaction;`

При нормальном завершении транзакции вызываем метод

procedure `Commit;`

фиксирующий изменения в БД, в противном случае, для отката транзакции идем на поклон к методу

procedure `Rollback;`

Предложенный в листинге 25.4 фрагмент кода раскрывает порядок управления транзакцией, в состав которой входит две хранимых процедуры.

Листинг 25.4. Пример работы с компонентом `TIBTransaction`

```
if IBTransaction1.Active=False then
begin
  IBTransaction1.StartTransaction; //старт транзакции
  try
    with IBStoredProc1 do // операция с 1-ой процедурой
      begin
        //заполняем параметры процедуры
        ExecProc;
      end;

    with IBStoredProc2 do //операция со 2-ой процедурой
      begin
        //заполняем параметры процедуры
        ExecProc;
      end;
    IBTransaction1.Commit; //фиксируем транзакцию
  except
    //действия при возникновении ошибки
```

```

    IBTransaction1.Rollback; //откат транзакции
end;
end;

```

Методы Commit() и Rollback() завершают выполнение транзакции и освобождают ее контекст. После вызова любого из перечисленных методов возвратится к транзакции невозможно

Если мы работаем с InterBase не ниже 6 версии, то для отката всех изменений в данных, осуществленных в рамках текущей транзакции, разрешено воспользоваться процедурой

```

procedure RollbackRetaining;

```

При этом сохраняется контекст транзакции, другими словами транзакция не завершается и может быть продолжена. Для фиксации изменений осуществленных транзакцией с сохранением ее контекста используйте метод

```

procedure CommitRetaining;

```

Еще раз заострю внимание читателя на том, что метод не завершает транзакцию, а просто сохраняет все изменения в текущей версии. Транзакция останется открытой до тех пор, пока не будет вызван метод Commit() или Rollback().

Задание

С использованием языка Delphi (C++ Builder) разработайте клиентское приложение VCL способное соединиться с БД InterBase (Firebird). Приложение должно позволять пользователю:

1. выбирать/вводить имя сервера;
2. указывать путь и имя с файлом БД;
3. вводить имя и пароль пользователя;
4. подготовьте к работе компонент-транзакцию, он понадобится для выполнения следующих работ.

26. Основные компоненты IBX

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – среда InterBase (FireBird), C++ (Delphi)

Компонент быстрой разработки TIBTable

Компонент–таблица TIBTable предоставляет возможность создания простейших клиент-серверных приложений за кратчайшее время с минимальными затратами на программирование. Для простейшего приложения понадобится помощь шести компонентов (рис. 26.1):

1. Компонент IBDataBase1 отвечающий за соединение с БД.
2. Транзакция IBTrandaction1 обеспечит доступ к контексту транзакции.
3. Компонент IBTable1 предоставит доступ к таблице.
4. Источник данных DataSource1 (страничка Data Access).
5. Навигатор данных DBNavigator1 (страничка Data Controls).
6. Сетка для отображения данных DBGrid1 (страничка Data Controls).

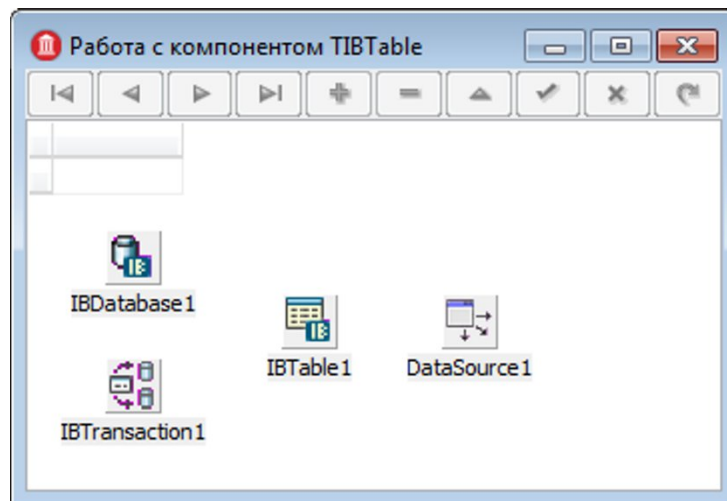


Рис. 26.1. Состав простейшего приложения БД на основе TIBTable

Разработка клиентского приложения всегда начинается с организации соединения с базой данных — эту задачу вы решили в предыдущей лабораторной работе.

Выберите компонент-транзакцию IBTrandaction1 и воспользовавшись свойством DefaultDatabase присоедините его к IBDataBase1.

Переходим к компоненту IBTable1. С помощью свойства Database подключите его к базе данных, затем задействовав свойство TableName и соединяемся с таблицей. Настройка компонента завершена, нам осталось только активировать его, для этого устанавливаем свойство компонента Active в состояние true.

“Программирование” завершено. Нам осталось нажать кнопку <F9>. Наш проект базы данных готов к работе!

Запрос, компонент TIBQuery

Запрос TIBQuery построен на базе рассмотренного ранее TIBCustomDataSet и описан в модуле IBQuery. Основное назначение компонента — организация доступа к данным одной или нескольких таблиц. Как и у всех подобных элементов управления, текст запроса описывается в свойстве

property SQL: TStrings;

Текст запроса может быть задан как во время проектирования (рис. 26.2), так и в коде программы.

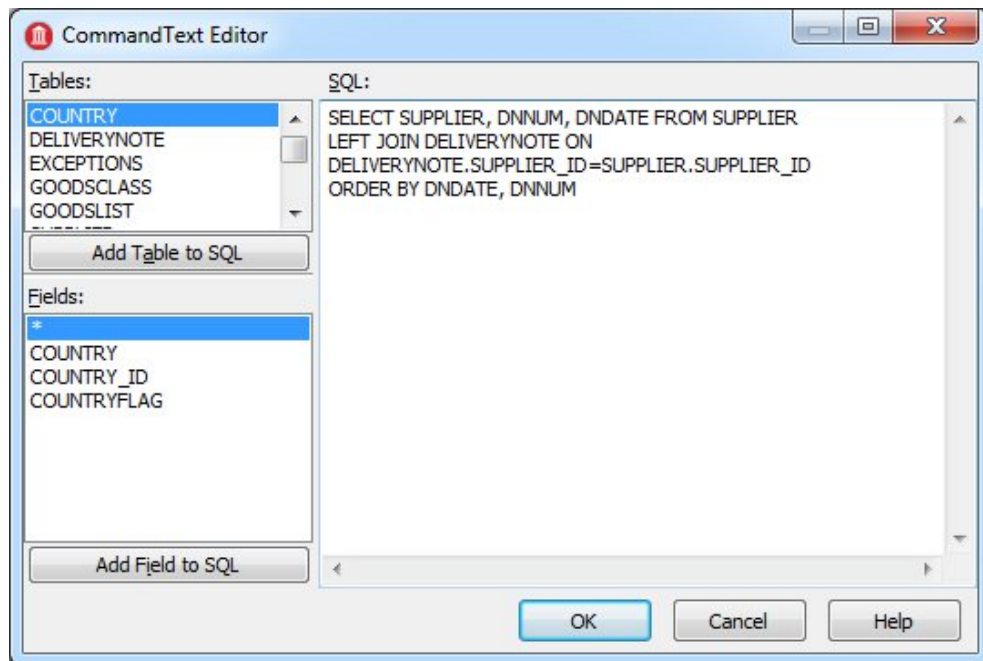


Рис. 26.2. Встроенный редактор SQL с текстом статического запроса

Если запрос основан на инструкции SELECT, то для получения записей следует воспользоваться методом **procedure** Open;

во всех остальных случаях выполнение инструкции ISQL доверяется процедуре

procedure ExecSQL;

Следующий пример (листинг 26.1) подготавливает компонент к получению данных из таблиц поставщиков и заказчиков.

Листинг 26.1. Пример работы с компонентом TIBQuery – статическая инструкция

```
with IBQuery1 do
begin
  Database:=IBDatabase1;
  SQL.Clear;
  SQL.Add('SELECT SUPPLIER, DNNUM, DNDATE FROM SUPPLIER');
  SQL.Add('LEFT JOIN DELIVERYNOTE ON');
  SQL.Add('DELIVERYNOTE.SUPPLIER_ID=SUPPLIER.SUPPLIER_ID');
  SQL.Add('ORDER BY DNDATE, DNNUM');
  Prepare;
  Open;
end;
```

Возвращаемый набор данных доступен только в режиме “для чтения”. Дабы добиться возможности его редактирования нам понадобятся услуги TIBUpdateSQL, подключаемого к свойству

property UpdateObject: TIBDataSetUpdateObject;

Это ссылка на компонент TIBUpdateSQL, способный оказать помощь при работе с недоступными для редактирования данными.

Создатели VCL научили работающие с SQL компоненты-наборы данных передавать в инструкцию-SQL внешние параметры без необходимости модификации текста всего запроса. В проектах, опирающихся на применение параметров, текст SQL-запроса формируется единожды. Это может произойти во время проектирования или, как в очередном примере (листинг 26.2) во время выполнения проекта.

Листинг 26.2. Динамическая инструкция SQL с параметрами

```
with IBQuery1 do
begin
  SQL.Clear; //очистка свойства
  SQL.Add('INSERT INTO SUPPLIER'); //формируем инструкцию
  SQL.Add(' (SUPPLIER) VALUES :SUPPLIER '); //объявляем параметр
```

```
Prepare; //подготовка запроса на сервере
end;
```

Обратите внимание на то, что в строке SQL "(SUPPLIER) VALUES :SUPPLIER" вместо явного указания имени нового поставщика мы передаем начинающееся с символа двоеточия ":" имя параметра.

Если запрос содержит параметры, то для их обслуживания следует воспользоваться коллекцией

```
property Params: TParams;
```

Для того, чтобы компонент автоматически регенерировал список параметров запроса следует установить в true свойство. Теперь, для отправки в БД команды на добавление новой строки, достаточно следующих строк кода (листинг 26.3).

Листинг 26.3. Заполнение параметра новым значением

```
IBQuery1.Params.ParamByName('SUPPLIER'):=Edit1.Text;
IBQuery1.ExecSQL;
```

Таким образом, применение параметров в SQL запросе существенно упрощает процесс разработки приложения и, что очень важно, ускоряет отправку и выполнение запроса на сервере.

Хранимая процедура, компонент *TIBStoredProc*

Компонент *TIBStoredProc* предназначен для обращения к откомпилированным и хранящимся в системном каталоге сервера БД процедурам. Название процедуры, услугами которой мы намерены воспользоваться указываются в свойстве

```
property StoredProcName: string;
```

Параметры процедуры содержатся в коллекции

```
property Params: TParams;
```

О готовности процедуры к выполнению сигнализирует свойство

```
property Prepared: Boolean;
```

Для отправки команды на выполнение обращаемся к процедуре

```
procedure ExecProc;
```

К сожалению, компоненты *TIBStoredProc* не любят работать с хранимыми процедурами возвращающими набор из нескольких записей. Поэтому, несмотря на наличие у *TIBStoredProc* метода *Open()*, основной способ отправки команды на выполнение должен быть *ExecSQL()*. Вместе с тем никто не запрещает нам создавать в InterBase хранимые процедуры, основанные на цикле *FOR SELECT ... DO*, но в этом случае для получения результатов выборки на стороне клиента следует задействовать не компонент *TIBStoredProc*, а компонент *TIBQuery*. В этом случае в текст SQL *TIBQuery* надо просто записать инструкцию *SELECT * FROM <имя_процедуры>* и вызвать метод *Open()*.

Допустим, что в составе БД имеется хранимая процедура *PR_SUPPLIER_INSERT* вставляющая новую запись в таблицу поставщиков (листинг 26.4).

Листинг 26.4. Код хранимой процедуры

```
CREATE PROCEDURE PR_SUPPLIER_INSERT
(ASUPPLIER VARCHAR(100))
RETURNS
(ASUPPLIER_ID INTEGER)
AS
begin
  INSERT INTO SUPPLIER (SUPPLIER) VALUES (:ASUPPLIER);

  SELECT SUPPLIER_ID FROM SUPPLIER
  WHERE SUPPLIER=:ASUPPLIER INTO :ASUPPLIER_ID;
end
```

Процедура обладает единственным входным параметром *ASUPPLIER*, в который следует передать имя нового поставщика и одним выходным параметром *ASUPPLIER_ID* возвращающим значение первичного ключа только-что добавленной записи.

Для подготовки компонента `TIBStoredProc` к работе с процедурой `PR_SUPPLIER_INSERT` следует воспользоваться инспектором объектов или фрагментом кода из листинга 26.5.

Листинг 26.5. Подключение компонента `TIBStoredProc` к хранимой процедуре

```
with IBStoredProc1 do
begin
  Database:=IBDatabase1;
  Transaction:=IBDatabase1.DefaultTransaction;
  StoredProcName:='PR_SUPPLIER_INSERT';
end;
```

Для выполнения процедуры `PR_SUPPLIER_INSERT` достаточно обратиться к листингу 26.6.

Листинг 26.6. Вызов хранимой процедуры

```
var ID:integer;
begin
  with IBStoredProc1 do
  begin
    Params.ParamByName('ASUPPLIER').AsString:=Edit1.Text;
    ExecProc;
    ID:= Params.ParamByName('ASUPPLIER_ID').AsInteger;
  end;
end;
```

В представленном коде мы передаем новое значение в параметр `ASUPPLIER` и выполняем процедуру. После выполнения процедуры в переменную `ID` помещается очередное значение первичного ключа.

Универсальный набор данных `TIBDataSet`

Классический представитель наборов данных класс `TIBDataSet` фактически способен заменить всех остальных своих коллег, базирующихся на фундаменте класса `TIBCustomDataSet`. При желании с помощью компонентов `TIBDataSet` мы сможем разработать полноценное клиентское приложение любой степени сложности без использования таблиц `TIBTable`, запросов `TIBQuery` и хранимых процедур `TIBStoredProc`.

Ключевая особенность компонента заключается в том, что он предоставляет программисту возможность тонкой настройки всех наиболее важных операций (просмотр, вставка, редактирование и удаление данных). Еще одна положительная черта компонента заключается в том, что полученный от сервера набор строк буферизируется на клиентской стороне, что позволяет нам применять все преимущества объектно-ориентированного подхода при работе с реляционными данными.

Формирование запросов

Универсальность компонента `TIBDataSet` достигается за счет инкапсуляции в него целых пяти объектов `TIBSQL`, каждый из которых отвечает за свой фронт работы:

- ~ выборку данных;
- ~ обновление отдельной записи;
- ~ вставку новой записи;
- ~ редактирование записи;
- ~ удаление записи.

Прямого обращения к инкапсулированным классам не требуется — все необходимые рычаги управления предоставлены в форме свойств и методов головного класса `TIBDataSet`. В первую очередь выделим свойства, несущие ответственность за хранение текста запросов.

```
property SelectSQL: TStrings; //запрос на выборку данных
property RefreshSQL: TStrings; //запрос обновления
property InsertSQL: TStrings; //запрос вставки
property ModifySQL: TStrings; //запрос редактирования
property DeleteSQL: TStrings; //запрос удаления
```

Во время визуального проектирования доступ к перечисленным свойствам реализован как традиционно через инспектора объектов, так и через встроенный редактор для вызова которого необходимо обратиться к контекстному меню компонента и выбрать пункт **Dataset Editor**. В простейшем случае разработчик может просто нажать на кнопку **Generate SQL** (рис. 26.3) в результате чего будет автоматически сгенерированы и сохранены в соответствующих свойствах компонента все пять запросов.

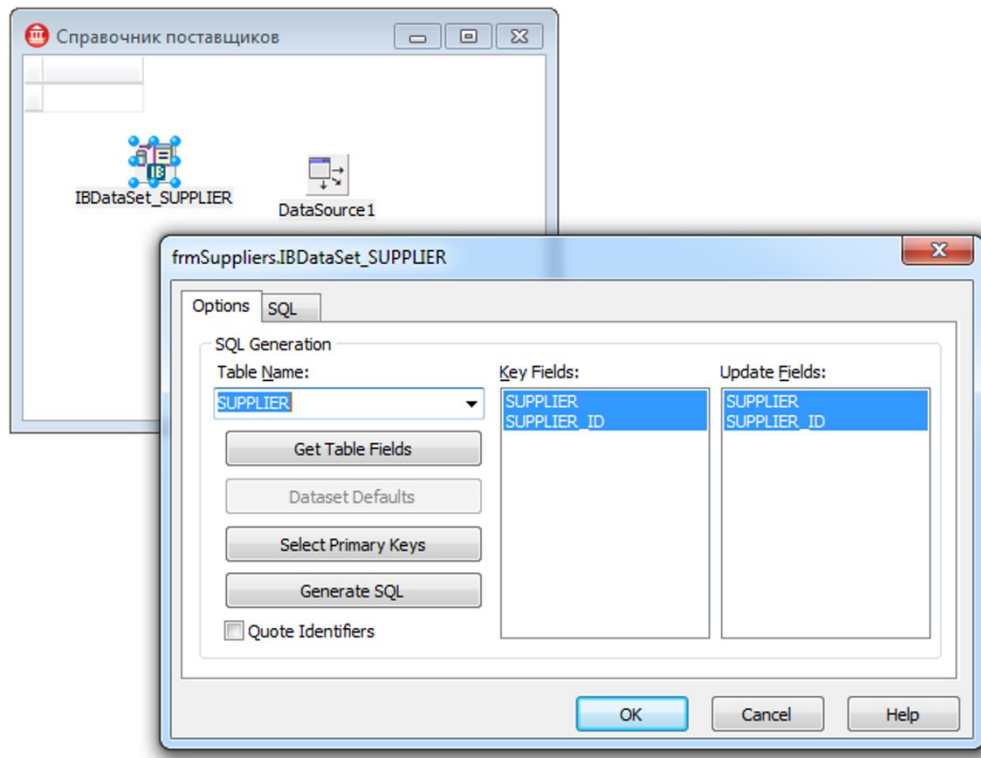


Рис. 26.3. Встроенный редактор текста запросов

Если вы сторонник управления текстом запроса непосредственно из кода программы, то обратите внимание на листинг 26.7 в котором мы готовим компонент IBDataSet1 к работе с таблицей поставщиков SUPPLIER.

Листинг 26.7. Подготовка текста SQL во время работы приложения

```

procedure TForm1.FormShow(Sender: TObject);
begin
With IBDataSet1 do
begin
    SelectSQL.Clear;
    SelectSQL.Add('SELECT * FROM SUPPLIER ORDER BY SUPPLIER');
    RefreshSQL.Clear;
    RefreshSQL.Add('SELECT * FROM SUPPLIER');
    RefreshSQL.Add('WHERE SUPPLIER_ID=: SUPPLIER_ID ORDER BY SUPPLIER');
    InsertSQL.Clear;
    InsertSQL.Add('INSERT INTO SUPPLIER (SUPPLIER,SUPPLIER_ID) ');
    InsertSQL.Add('VALUES (:SUPPLIER, :SUPPLIER_ID) ');

    ModifySQL.Clear;
    ModifySQL.Add('UPDATE SUPPLIER SET SUPPLIER=:SUPPLIER');
    ModifySQL.Add('WHERE SUPPLIER_ID=:OLD_SUPPLIER_ID');
    DeleteSQL.Clear;
    DeleteSQL.Add('DELETE FROM SUPPLIER');
    DeleteSQL.Add('WHERE SUPPLIER_ID=:OLD_SUPPLIER_ID');
    //настроим поле генератора значения первичного ключа
    GeneratorField.ApplyEvent:=gamOnServer;
    Open;
end;
end;

```

Просматривая листинг обязательно обратите внимание на то, что в запросах модификации и удаления строки при объявлении параметра мы воспользовались префиксом “OLD_”. Дело в том, что в OLD_ полях хранится исходное (не модифицированное) значение данного столбца таблицы. В нашем случае OLD_SUPPLIER_ID хранит значение первичного ключа редактируемой записи.

Компонент TIBDataSet автоматически создает дополнительные параметры, названия которых строятся по принципу “OLD_”+“ИМЯ_СТОЛБЦА” и “NEW_”+ “ИМЯ_СТОЛБЦА”. Параметры, имена которых начинаются с префикса

“OLD_” хранят значения столбцов до их изменения пользователем, а параметры “NEW_” — значения после изменения. Эти параметры можно использовать при формировании текста запросов.

Задание

На основе созданного во время выполнения предыдущей лабораторной работы клиентского приложения создайте 2-3 формы, демонстрирующие порядок просмотра и редактирования данных в 2-3 таблицах вашей БД. В лабораторной работе должны быть продемонстрированы варианты использования следующих компонентов:

- ~ таблицы `TIBTable`;
- ~ запроса `TIBQuery`;
- ~ хранимой процедуры `TIBStoredProc`;
- ~ набора данных `TIBDataSet`.

27. Скрипты SQL

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – среда InterBase (FireBird), C++ (Delphi)

Под скриптом (сценарием) SQL понимается одна или несколько инструкций SQL подлежащих последовательному выполнению. С помощью скриптов могут создаваться (модифицироваться или удаляться) как объекты БД, так и сами данные.

За счет скриптов существенно упрощается процесс развертывания БД InterBase на сервере заказчика вашего программного продукта. Экспортировав метаданные БД в виде инструкций SQL мы получаем текстовый файл содержимое которого надо воспроизвести на компьютере с СУБД. Кроме того, за счет скриптов снижается трудоемкость удаленного администрирования БД.

Выполнение скриптов, TIBScript

Работа с компонентом традиционно начинается с подключения к БД и транзакции

```
property Database: TIBDatabase;  
property Transaction: TIBTransaction;
```

Будьте внимательны при настройке параметров транзакции — при создании объектов БД назначение режима “только для чтения” вряд ли логично...

Желательно (но не обязательно) с помощью свойства

```
property Dataset: TIBDataset;
```

подключить к скрипту потомок TIBDataset (например TIBQuery), который возьмет на себя ответственность за выполнение инструкций DML. Если вы это свойство проигнорируете, то перед запуском скрипта на выполнение будет динамически создан внутренний экземпляр TIBSQL.

Следующее действие связано с определением символа (или символов) терминатора, сигнализирующих аналитору о завершении строки с инструкцией SQL

```
property Terminator: string; //по умолчанию ";"
```

По умолчанию в InterBase на роль терминатора назначен символ ";". Это состояние вполне соответствует настройкам консоли **IBConsole** отвечающей за генерацию скриптов.

Готовый к использованию скрипт загружается в свойство

```
property Script: TStrings;
```

Для этого подойдут имеющиеся в распоряжении набора строк TStrings методы работы с файлами LoadFromFile() и потоками LoadFromStream().

Перед запуском скрипта стоит проверить его содержимое на корректность. Для этих целей предназначен метод

```
function ValidateScript: Boolean;
```

Если хранящийся в скрипте SQL-код не содержит ошибок, то метод возвратит true.

Если скрипт велик, и его валидация занимает длительное время, то стоит задуматься о запуске этого процесса в отдельном потоке. В таком случае чтобы убедиться в том, что в данный момент идет проверка загруженного в компонент скрипта воспользуйтесь свойством

```
property Validating : Boolean; //только для чтения
```

Значение true указывает, что процесс проверки еще не закончен, а состояние false подтверждает, что проверка завершена.

Завершающий штрих связан с вызовом ключевого метода

```
procedure ExecuteScript;
```

Сразу после этого скрипт отправляется на сервер.

Разработка помощника установки БД

Разработаем приложение, позволяющее обычному пользователю, знающему пароль для администрирования InterBase разворачивать новую БД на сервере. Для этого нам понадобится помощь следующего перечня компонентов IBX:

- ~ соединение с БД `IBDatabase1: TIBDatabase;`
- ~ транзакция `IBTransaction1: TIBTransaction`, сразу с помощью свойства `DefaultDatabase` подключите транзакцию к `IBDatabase1;`
- ~ скрипт `IBScript1: TIBScript`, не забудьте настроить свойства `Database` и `Transaction`.
- ~ Кроме того, на форме необходимо разместить ряд элементов управления, среди них:
 - текстовые строки `TEdit` для ввода:
 - имени сервера, в простейшем случае 'localhost' (компонент `Edit1`);
 - имени пользователя с правами SYSDBA (`Edit2`);
 - пароля (`Edit3`);
 - имени файла со скриптом (`Edit4`);
 - имени папки в которой следует развернуть новую БД (`Edit5`);
 - название файла с БД (`Edit6`).
 - для упрощения операции выбора файла со скриптом воспользуйтесь диалогом `TOpenDialog;`
- ~ еще нам потребуется несколько кнопок `TButton` (для вызова диалога выбора файла, для запуска скрипта, и т.п.).

Листинг 27.1 демонстрирует порядок доступа к файлу скрипта из приложения.

Листинг 27.1. выбор файла со скриптом SQL

```

procedure TForm1.Button1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
    begin
      IBScript1.Script.Clear;
      IBScript1.Script.LoadFromFile(OpenDialog1.FileName);
      if IBScript1.ValidateScript=true then
        Edit4.Text:=OpenDialog1.FileName;
      else
        begin
          Edit4.Text:='';
          IBScript1.Script.Clear;
          raise Exception.Create('Загруженный скрипт содержит ошибки!');
        end;
      end;
    end;
end;

```

После выбора скрипта мы не только загружаем его в компонент `IBScript1`, но и проверяем корректность инструкций SQL. Если скрипт содержит ошибки — вызываем исключительную ситуацию.

Для выбора целевого каталога для развертывания БД нам понадобится подключить модуль `FileCtrl` (именно в нем спрятана функция `SelectDirectory`) и разместить на форме очередную кнопку (листинг 27.2).

Листинг 27.2. выбор файла со скриптом SQL

```

procedure TForm1.Button2Click(Sender: TObject);
var s:string;
begin
  if SelectDirectory('выбор папки для файла БД','',[sdShowEdit,sdNewUI])
  then Edit5.Text:=s;
end;

```

Наконец, после того как пользователь заполнит все необходимые параметры, самая главная кнопка запустит скрипт на выполнение (листинг 27.3)

Листинг 27.3. выбор файла со скриптом SQL

```

procedure TForm1.Button3Click(Sender: TObject);
begin
  try
    Screen.Cursor:=crHourglass;

  try
    with IBDatabase1 do //сначала создаем БД
    begin
      LoginPrompt:=false; //отключили встроенную регистрацию
      Params.Clear;       //очистка параметров
      SQLDialect:=3;       //диалект ISQL
      //имя хоста:путь к каталогу с БД\имя файла с БД
      DatabaseName:= Edit1.Text+':'+ Edit5.Text+'\'+ Edit6.Text;
      Params.Add('USER "'+ Edit2.Text+'"'); //имя пользователя
      Params.Add('PASSWORD "'+ Edit3.Text+'"'); //пароль
      CreateDatabase; //создаем БД с заданными параметрами

      IBScript1.ExecuteScript; //теперь стартуем скрипт
      ShowMessage('База данных создана!');
    end;
  except
    on E: exception do
      Showmessage(E.ClassName + ': ' + E.Message);
  end;

  finally
    Screen.Cursor:=crDefault;
  end;
end;

```

Обратите внимание на одну важную деталь — для создания новой БД мы задействуем не скрипт IBScript1, а компонент отвечающий за соединение с БД.

Получение метаданных, TIBExtract

Задача компонента TIBExtract (описан в модуле IBX.IBExtract)— выводить у InterBase весь списочный состав имеющихся в БД объектов с подробным их описанием и вернуть полученные сведения в виде скрипта SQL.

Для организации работы с компонентом программист должен подключить TIBDatabase к базе данных (свойство Database) и соединиться с транзакцией (свойство Transaction), а далее наступает черед ключевого метода

```

procedure ExtractObject(ObjectType : TExtractObjectTypes;
                        ObjectName : String = '';
                        ExtractTypes : TExtractTypes = []);

```

Здесь: ObjectType — запрашиваемый у БД тип метаданных (таблица 27.1). Необязательный параметр ObjectName указывает имя конкретного объекта БД. Необязательный параметр ExtractTypes позволяет наложить дополнительный фильтр на тип извлекаемых данных (табл. 27.2).

Результат выполнения метода ExtractObject() помещаются в свойство

property Items: TStrings;

Это список строк с текстом скрипта SQL.

Таблица 27.1. Перечень значений TExtractObjectTypes

Значение	Содержание скрипта
eoDatabase	Список всех определенных в БД объектов. Этот перечень может быть дополнительно отфильтрован при помощи параметров ObjectName и ExtractTypes
eoDomain	Список всех доменов
eoTable	Если программист не воспользовался свойством ObjectName и оставил его пустым, то мы получим список всех таблиц БД. Если же в ObjectName были занесены данные, то процедура интерпретирует их как имя таблицы. В этом случае возвращается информация об этой таблице, причем тип ин-

	формации должен быть определен в параметре <code>ExtractTypes</code>
<code>eoView</code>	Если параметр <code>ObjectName</code> не использовался, возвращается список всех представлений, иначе мы будем проинформированы о представлении с именем <code>ObjectName</code>
<code>eoProcedure</code>	Список хранимых процедур
<code>eoFunction</code>	Список всех внешних функций. Если параметр <code>ObjectName</code> не пуст, то мы получим информацию об отдельной функции.
<code>eoGenerator</code>	Список генераторов
<code>eoException</code>	Список исключений
<code>eoBLOBFilter</code>	Список фильтров BLOB
<code>eoRole</code>	Список ролей
<code>eoTrigger</code>	Перечень триггеров
<code>eoForeign</code>	Перечень внешних ключей
<code>eoIndexes</code>	Список индексов
<code>eoChecks</code>	Ограничения
<code>eoData</code>	Список всех данных, с именем определенном в параметре <code>ObjectName</code>

Таблица 27.2. Описание `TExtractTypes`

Значение	Описание
<code>etDomain</code>	При выборке метаданных о таблице, включить информацию о всех принадлежащих таблице доменах
<code>etTable</code>	Собрать данные только о таблице
<code>etRole</code>	При выборке метаданных о таблице, включить информацию обо всех связанных с таблицей ролях
<code>etTrigger</code>	При выборке метаданных о таблице, собирать данные об ее триггерах
<code>etForeign</code>	При выборке метаданных о таблице, собирать данные об ее внешних ключах
<code>etIndex</code>	При выборке метаданных о таблице, собирать данные об ее индексах
<code>etData</code>	При выборке метаданных о таблице, получить ее данные
<code>etGrant</code>	При выборке метаданных о таблице, включать информацию о правах пользователей
<code>etCheck</code>	При выборке метаданных о таблице, собирать данные об ограничениях
<code>etAlterProc</code>	При сборе данных о хранимых процедурах, исключить информацию о взаимных ссылках между процедурами

Разработка генератора скриптов

Для разработки примера, демонстрирующего процесс получения метаданных из БД нам потребуется следующий перечень компонентов IBX:

- ~ готовый к работе компонент `IBDatabase1` вместе с транзакцией по умолчанию `IBTransaction1`;
- ~ компонент `IBExtract1:TIBExtract`, воспользовавшись свойствами `Database` и `Transaction` подключите `IBExtract1` к БД и транзакции.

Для формирования пользовательского интерфейса нам потребуются следующие элементы управления (рис. 27.1):

- ~ комбинированный список `ComboBox1`, благодаря ему пользователь сможет выбрать какие именно метаданные ему необходимы;
- ~ многострочный редактор `Memo1`, в этот компонент мы станем выгружать текст со скриптом SQL.

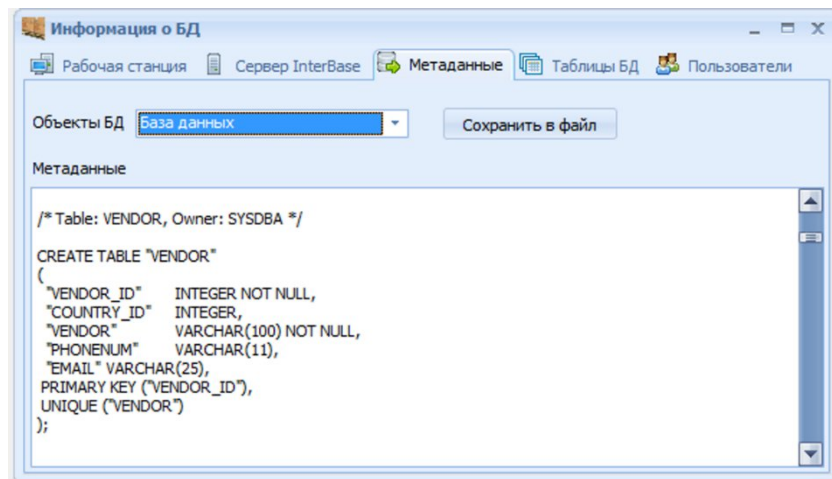


Рис. 27.1. Экранный снимок формы с метаданными БД

В момент создания формы проекта заполним строки комбинированного списка всеми возможными вариантами метаданных (листинг 27.4).

Листинг 27.4. Заполняем комбинированный список перечнем объектов БД

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  with ComboBox1 do
  begin
    Items.Add('База данных');           // TExtractObjectTypes=eoDatabase
    Items.Add('Домены');                 // eoDomain
    Items.Add('Таблицы');                 // eoTable
    Items.Add('Представления');           // eoView
    Items.Add('Хранимые процедуры');      // eoProcedure
    //и т.д.
    Items.Add('Шифрование');              // eoEncryption
    ItemIndex:=0;
  end;
end;
```

Главное условие заполнения строк ComboBox1 заключается в том, чтобы названия элементов списка следовали в точно таком-же порядке, как и объявление элементов типа данных TExtractObjectTypes (для этого можно просмотреть соответствующее объявление в модуле IBX.IBExtract).

В момент выбора пользователем того или иного элемента в комбинированном списке ComboBox1 у компонента генерируется событие OnChange, этим мы воспользуемся в листинге 27.5.

Листинг 27.5. Событие OnChange() комбинированного списка

```
procedure TForm1.ComboBox1Change(Sender: TObject);
begin
  IBExtract1.Items.Clear;
  try
    //каждому ItemIndex соответствует значение TExtractObjectTypes
    IBExtract1.ExtractObject(TExtractObjectTypes(ComboBox1.ItemIndex), '');
    Memo1.Lines.Assign(IBExtract1.Items);
  except
    Memo1.Lines.Clear;
  end;
end;
```

Задание

1. Разработайте приложение, позволяющее пользователю обладающему административным паролем InterBase развернуть на сервере БД из заранее подготовленного файла скрипта.
2. Разработайте приложение, позволяющее создать скрипт любой БД развернутой на доступном пользователю сервере InterBase.

28. Компоненты для администрирования БД

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – среда InterBase (FireBird), C++ (Delphi)

Если вы имеете опыт конфигурирования БД с помощью входящей в установочный пакет InterBase стандартной консоли управления сервером IBConsole, то уже наверняка хорошо знаете функционал администратора. В первую очередь это: ведение учетных записей пользователей; проверка состояния БД и настройка ее параметров; резервное копирование и восстановление БД; управление лицензиями; ведение журнала; анализ статистики и производительности. Доступ к большинству перечисленных операций предоставляет контекстное меню обслуживаемой базы данных.

Система управления базами данных InterBase обладает мощным набором системных процедур и функций API предоставляющих программисту возможность низкоуровневого доступа к серверу. Опираясь на API, разработчик сможет осуществить более тонкую настройку всех параметров системы, начиная от повышения производительности, заканчивая управлением поведением сервера в критических ситуациях.

Общие черты компонентов

В составе палитры компонентов имеется набор компонентов инкапсулирующих базовый набор функций InterBase API отвечающих за конфигурирование и администрирование сервера InterBase и развернутых на нем баз данных. Все компоненты размещены на страничке InterBase Admin и описаны в модуле IBServices, и предназначены для обслуживания сервера InterBase не ниже 6.0 версии.

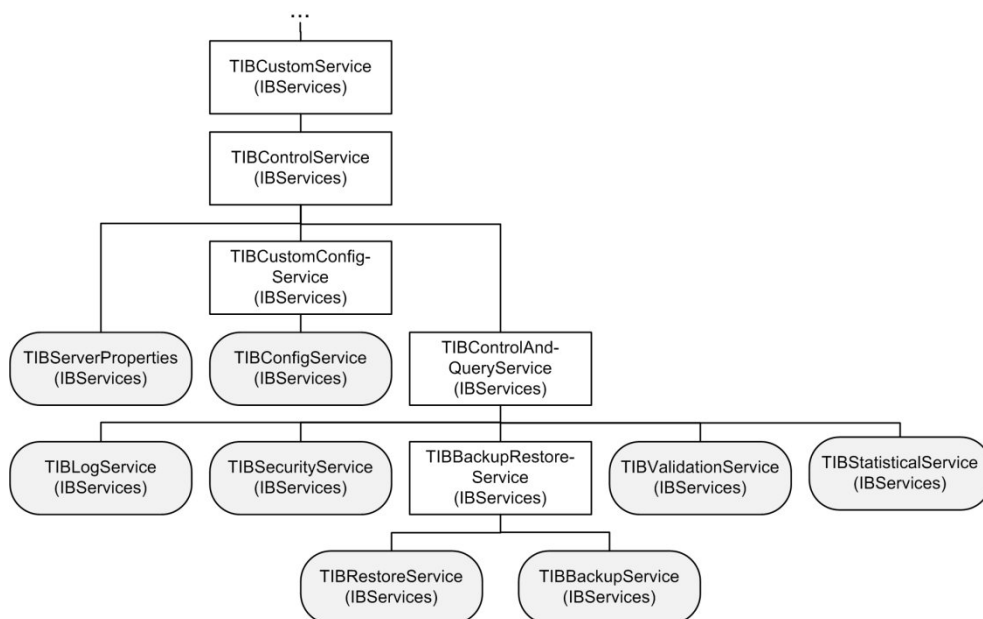


Рис. 28.1. Иерархия компонентов-сервисов InterBase

В цепочке наследования (рис. 28.1) перед компонентами сервисами имеется два промежуточных класса определивших родовые черты всего большого семейства компонентов TIB...Service, это классы TIBCustomService и TIBControlService.

Первый по счету предок отвечает за организацию соединения с хостом, на котором развернут интересующий нас сервер InterBase. Имя сервера следует поместить в свойство

```
property ServerName: string;
```

Вместо символического имени допускается и использование IP-адреса компьютера, для доступа к локальному хосту можно воспользоваться IP-адресом 127.0.0.1 или текстовой аббревиатурой "localhost".

Характерная особенность большинства рассматриваемых в этой главе компонентов является тот факт, что для их работы нет необходимости в поддержке соединения с БД с помощью TIBDatabase, исключение составляет компонент TIBCustomConfigService.

Следующим действием программиста должен стать выбор сетевого протокола доступа к серверу

```
property Protocol: TProtocol; \\ по умолчанию Local
TProtocol = (TCP, SPX, NamedPipe, Local);
```

В перечень доступных вариантов входят: транспортный протокол TCP; неактуальный на сегодняшний день SPX; именованный канал NamedPipe (это надстройка над стеком протоколов TCP/IP); значение Local, указывающее на то, что сервер и клиент размещены на одном хосте.

Порядок регистрации пользователя на сервере зависит от состояния свойства

```
property LoginPrompt: Boolean; \\ по умолчанию true
```

По умолчанию оно установлено в true, это означает, что для аутентификации пользователя должен вызываться специальный диалог. Процесс аутентификации сопровождается событием

```
property OnLogin: TIBLoginEvent;
TIBLoginEvent = procedure (Database: TIBCustomService;
                           LoginParams: TStrings) of object;
```

Параметры компонента, в частности обеспечивающие выполнение процедуры аутентификации пользователя, могут быть занесены в список

```
property Params: TStrings;
```

Подключение к серверу и отключение компонента от сервера производится с помощью свойства

```
property Active: Boolean; //по умолчанию false
```

или пары методов

```
procedure Attach;//подключение
procedure Detach;//отключение
```

В момент подключения генерируется событие

```
property OnAttach: TNotifyEvent;
```

Наличие общих черт у всех сервисных компонентов существенно упрощает процесс управления этими компонентами. В частности, если в проекте задействовано несколько потомков класса TIBControlService, то всех их стоит подготовить к работе в коде единой процедуры, предложенной в листинге 28.1.

Листинг 28.1. Подготовка к работе административных компонентов

```
procedure TDataModule1.PrepareIBService(const Host,
                                         User, Password: string);
var i:integer;
begin
  for i := 0 to TDataModule1.ComponentCount-1 do
    if (TDataModule1.Components[i] is TIBControlService) then
      with (TDataModule1.Components[i] as TIBControlService) do
        begin
          if Active then Active:=false;
          LoginPrompt:=false;
          ServerName:=Host;
          Protocol:=TCP;
          Params.Clear;
          Params.Add('user_name='+User);
          Params.Add('password='+Password);
        end;
  end;
```

На вход функции поступают имя хоста, имя и пароль пользователя. В рамках цикла `for..do` мы перебираем все компоненты принадлежащие модулю данных DataModule1, и если среди них нам встречаются потомки TIBControlService, то заполняем их ключевые свойства, отвечающие за организацию соединения.

У всех потомков TIBControlService имеется обязательное для заполнения свойство Database предназначенное для идентификации, обслуживаемой БД, однако доступ к указанному свойству реализуется по-разному. В простейшем случае Database это строка с полным путем к БД, однако есть и исключения, например, список строк TStrings или соединение с БД TIBDatabase.

Наиболее значимый вклад класса TIBControlService в своих потомков — это метод

```
procedure ServiceStart; virtual;
```

и свойство

```
property IsServiceRunning : Boolean; //только для чтения
```

Метод `ServiceStart` позволяет запустить службу `InterBase`, а доступное только для чтения свойство `IsServiceRunning` проверяет факт выполнения службы. Запрашиваемые параметры передаются с помощью свойства

```
property ServiceParamBySPB(const Idx: integer): string;
```

Ознакомившись с общими родовыми чертами компонентов администрирования СУБД приступим к изучению их индивидуальных характеристик.

Свойства сервера, *TIBServerProperties*

Компонент `TIBServerProperties` позволяет получить сводную информацию о ключевых свойствах сервера. После присоединения к хосту, на котором развернут сервер, компонент приобретает возможность опросить СУБД по шести направлениям.

Во-первых, для выяснения версии СУБД, следует вызвать метод

```
procedure FetchVersionInfo;
```

Результаты выполнения процедуры представляются в виде объекта `TVersionInfo` (листинг 28.2).

Листинг 28.2. Определение `TVersionInfo`

```
type TVersionInfo = class
  ServerVersion: string;           //версия сервера
  ServerImplementation: string;   //реализация сервера
  ServiceVersion: integer;        //версия сервиса
  function IsMinimumVersion(MinVersion : string) : Boolean;
  procedure Clear;
end;
```

И помещаются в свойство

```
property VersionInfo: TVersionInfo; // только для чтения
```

Во-вторых, с помощью метода

```
procedure FetchDatabaseInfo;
```

мы узнаем количество и названия БД развернутых на сервере, и число текущих подключений к СУБД. На этот раз результаты опроса передается в объект `TDatabaseInfo` (листинг 28.3).

Листинг 28.3. Определение `TDatabaseInfo`

```
type TDatabaseInfo = class
public
  NoOfAttachments: integer; //число подключений
  NoOfDatabases: integer;   //число БД
  DbName: array of string;  //имена БД
  //...
end;
```

Надо отметить, что массив `DbName`, предназначенный на хранения имен развернутых на сервере БД заполнится информацией, только если к БД есть хотя бы одно подключение.

Метод `FetchDatabaseInfo()` соберет сведения только об активных в данный момент БД, т.е. БД к которым имеется хотя бы одно подключение.

Доступ к объекту `TDatabaseInfo` предоставляет свойство

```
property DatabaseInfo: TDatabaseInfo; //только для чтения
```

В-третьих с помощью метода

```
procedure FetchConfigParams;
```

мы выясним пути к основным файлам сервера и занесем их в объект TConfigParams (листинг 28.4).

Листинг 28.4. Определение TConfigParams

```
type TConfigParams = class
public
    ConfigFileData: TConfigFileData;
    ConfigFileParams: array of string; //массив с параметрами
    BaseLocation: string;             //расположение сервера
    LockFileLocation: string;         //файл блокировок
    MessageFileLocation: string;      //файл сообщений
    SecurityDatabaseLocation: string; //БД с учетными записями
    //...
end;
```

Результаты выполнения метода FetchConfigParams () помещаются в свойство:

```
property ConfigParams: TConfigParams; //только для чтения
```

В-четвертых, и, если хотите, сразу же в-пятых вопросы связанные с лицензированием решают методы

```
procedure FetchLicenseInfo;
procedure FetchLicenseMaskInfo;
```

Первая процедура спрашивает о поддерживаемых системой лицензиях, а вторая получает подробности о маске лицензии. Результаты представляются в виде объектов TLicenseInfo и TLicenseMaskInfo.

Доступ к объектам можно получить благодаря свойствам

```
property LicenseInfo: TLicenseInfo; // только для чтения
property LicenseMaskInfo: TLicenseMaskInfo; // только для чтения
```

Наконец в-шестых, благодаря методу

```
procedure FetchAliasInfo;
```

мы получим информацию о псевдонимах БД. Результаты передаются в свойство

```
property AliasInfos : TIBAliasInfos; // только для чтения
```

представляющее собой массив записей TIBAliasInfo. В записи хранится информация о псевдониме Alias и файловом пути DBPath.

Число доступных псевдонимов помещается в свойство

```
property AliasCount : integer; // только для чтения
```

При желании, можно обратиться к любому элементу массива. Для этого предназначено свойство

```
property AliasInfo[Index : integer] : TIBAliasInfo; // только для чтения
```

Пара методов позволяет добавлять и удалять псевдонимы

```
procedure AddAlias(Alias, DBPath : string);
procedure DeleteAlias(Alias : string);
```

Если программиста по какой-то причине не устраивает последовательность вопросов, то ни кто не запрещает переставить их в любом подходящем порядке. Более того, все вопросы можно задать одновременно, вызвав метод

```
procedure Fetch;
```

Но перед этим убедитесь, что в свойстве:

```
property Options : TPropertyOptions;
TPropertyOption = (Database, License, LicenseMask, ConfigParameters,
                  Version, DBAlias);
```

подключены все опции, иначе будут заданы только отмеченные в опциях вопросы.

Если у вас сложилось впечатление, что с компонентом TIBServerProperties сложно работать, постараюсь разубедить в этом с помощью небольшого примера.

На главной форме проекта расположите героя этих страниц — компонент `IBServerProperties1`. Кроме того нам потребуются строки ввода `edServerName`, `edUser` и `edPassword` для хранения имени хоста, пользователя и пароля и многострочный редактор `Mem1:ТМемо` (в него мы станем собирать полученную информацию о сервере). Все остальное вы найдете в фрагменте кода из листинга 28.5.

Листинг 28.5. Получение параметров сервера InterBase

```
var i:integer;
begin
with IBServerProperties1 do
begin
  ServerName:=edServerName.Text;
  Protocol:=TCP;
  Params.Clear;
  Params.Add('user_name='+edUser.Text);
  Params.Add('password='+edPassword.Text);
  Active:=true;
  Mem1.Clear;
  FetchVersionInfo; //сведения о версии
  Mem1.Lines.Add('Версия сервера:');
  Mem1.Lines.Add('- версия: '+VersionInfo.ServerVersion);
  Mem1.Lines.Add('- реализация: '+VersionInfo.ServerImplementation);
  Mem1.Lines.Add('- сервис: '+IntToStr(VersionInfo.ServiceVersion));

  FetchDatabaseInfo; //информация об активных БД
  Mem1.Lines.Add('Информация о СУБД:');
  Mem1.Lines.Add('- подключений: '
    +IntToStr(DataBaseInfo.NoOfAttachments));
  Mem1.Lines.Add('- баз данных: '
    +IntToStr(DataBaseInfo.NoOfDatabases));
  for i:=0 to High(DataBaseInfo.DbName) do
    Mem1.Lines.Add(DataBaseInfo.DbName[i]);

  FetchConfigParams; //конфигурационные параметры
  Mem1.Lines.Add('');
  Mem1.Lines.Add('Конфигурация:');
  Mem1.Lines.Add('- размещение: '+ConfigParams.BaseLocation);
  Mem1.Lines.Add('- файл блокировки: '+ConfigParams.LockFileLocation);
  Mem1.Lines.Add('- файл сообщений: '
    +ConfigParams.MessageFileLocation);
  Mem1.Lines.Add('- учетные записи: '
    +ConfigParams.SecurityDatabaseLocation);

  for i:=0 to High(ConfigParams.ConfigFileParams) do
    Mem1.Lines.Add(ConfigParams.ConfigFileParams[i]);

  Active:=false;
end;
end;
```

Вся служебная информация о сервере InterBase хранится в системной базе данных `admin.ib`.

Конфигурирование БД, *TIBConfigService*

Компонент `TIBConfigService` предоставляет программисту рычаги управления основными опциями базы данных, дублируя возможности консоли управления СУБД `IBConsole` по конфигурированию сервера (рис. 28.2).

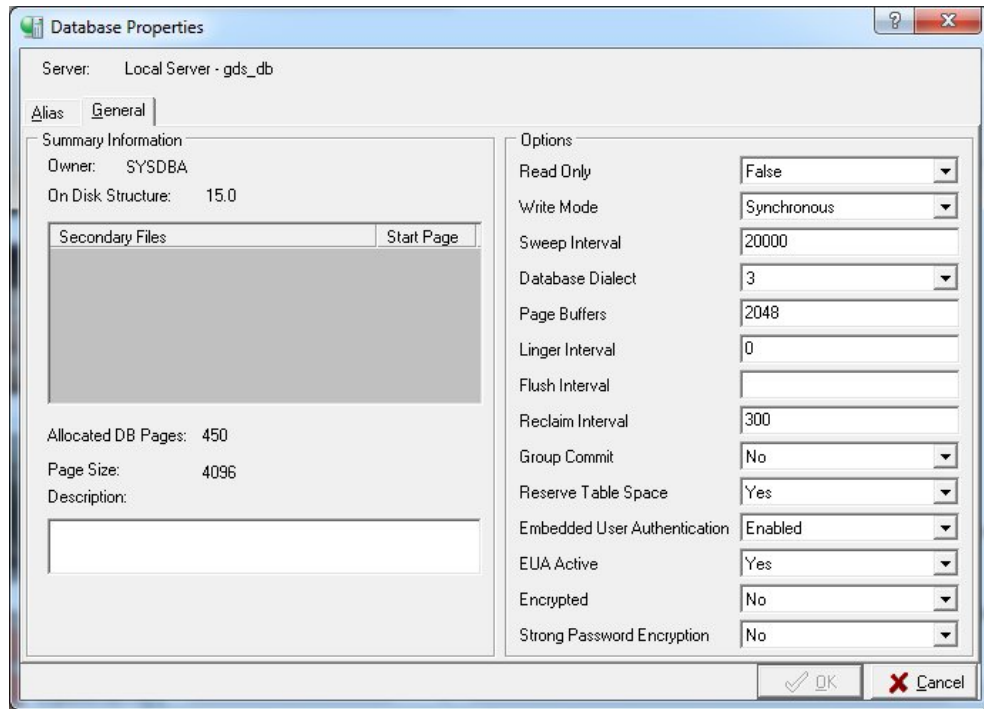


Рис. 28.2. Настройка основных свойств базы данных из консоли IBCConsole

За подключение компонента-сервиса к интересующей его БД отвечает его предок — класс `TIBCustomConfigService`. Для этого в нем объявлено три свойства

```
property Database : TIBDatabase;
property DatabaseName: string;
property Transaction : TIBTransaction;
```

Они предназначены для соединения с предоставляющим доступ к БД компонентом `TIBDatabase`, определением пути и имени файла БД и взаимодействия с транзакцией.

Практика работы с компонентом `TIBConfigService` показывает, что в его интересах следует выделять отдельный компонент-транзакцию `TIBTransaction`.

Отключение и перезапуск БД

В ряде ситуаций (например, для проверки и восстановления данных) системному администратору необходимо получить эксклюзивный доступ к БД. Такой доступ предполагает отключение от БД всех пользователей. Проще всего это сделать при посредничестве метода

```
procedure ShutdownDatabase(Options: TShutdownMode; Wait: integer);
TShutdownMode = (Forced, DenyTransaction, DenyAttachment);
```

Порядок отключения определяется параметром `Options`. Силовое отключение `Forced` отсоединит БД независимо от наличия к ней доступа и активных транзакций через время ожидания в `Wait` секунд. Режим `DenyTransaction` во время ожидания отключения не допустит запуска новых транзакций, а `DenyAttachment` отвергнет попытки подключения к БД новых пользователей.

Есть некоторое расхождение в единицах измерения времени таймаута у компонента `TIBConfigService` и у утилиты администрирования сервера **IBCConsole**. Метод `ShutdownDatabase()` измеряет время в секундах, а консоль **IBCConsole** устанавливает таймаут в минутах.

После отключения БД переходит в режим монопольного доступа для оператора вызвавшего отключение. Завершив все операции следует перезапустить БД, чтобы восстановить нормальную работу системы, для этого предназначен метод

```
procedure BringDatabaseOnline;
```

Пример получения монопольного доступа к БД предложен в листинге 28.6.

Листинг 28.6. Получение монопольного доступа к БД

```
IBConfigService1.Active:=true;
```

```
try  
IBConfigService1.ShutdownDatabase (DenyAttachment, 10);  
//... монопольный доступ к БД  
finally  
IBConfigService1.BringDatabaseOnline;  
end;
```

Задание

Разработайте приложение, позволяющее пользователю обладающему административным паролем InterBase осуществлять следующие операции:

- ~ получить общие сведения о сервере БД;
 - ~ создать резервную копию БД;
 - ~ восстановить БД из резервной копии.
-

29. Создание БД средствами MySQL

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – СУБД MySQL

Уже почти двадцать лет система управления базами данных MySQL пользуется заслуженным авторитетом у многочисленных разработчиков баз данных. Тому много причин:

1. Кроссплатформенность — сервер может быть развернут на базе Windows, Linux и OS X.
2. Поддержка стандарта SQL — разработчики MySQL прилагают все усилия, чтобы приблизить сервер к последнему стандарту SQL:2003.
3. Низкая ресурсоемкость — сервер MySQL абсолютно непривередлив к производительности компьютера, что выгодно его отличает от многих других СУБД.
4. Масштабируемость — сервер не демонстрирует снижения производительности при подключении очередных клиентов.
5. Экономическая — наиболее распространенная редакция MySQL Community Edition является свободным программным обеспечением.

Создание базы данных

Система управления базами данных MySQL допускает одновременную поддержку нескольких пользовательских БД. Поэтому совместно с работой прилагаемым к книге примером демонстрационной БД “Склад” читатель сможет проверить свои силы и на альтернативных решениях. В любом случае перед выбором имени для будущей БД стоит ознакомиться с названиями уже развернутых БД, для этого понадобится помощь представленной в листинге 29.1 инструкции.

Листинг 29.1. Вывод имен всех БД

```
SHOW DATABASES;
```

В MySQL принято в конце каждой из команд ставить точку с запятой.

Если сервер MySQL 5.7 был установлен буквально только что, и с ним еще не успели поэкспериментировать разработчики БД, то вы увидите только три предустановленных БД:

```
~ information_schema — системная БД с информацией о всех объектах сервера;
~ mysql — системная БД с данными о:
    o привилегиях пользователей;
    o настройках временной зоны;
    o управление репликацией;
    o оптимизация системы;
    o протоколом работы сервера;
    o системой помощи (ключевые слова и краткая информация);
    o и т.п.
~ performance_schema — системная БД с информацией о производительности сервера;
```

Кроме того, в поставку могут включаться демонстрационные БД, на которых вы сможете оттачивать свое мастерство.

Для проверки факта существования конкретной БД в инструкцию `SHOW DATABASE` добавляют ключевое слово `LIKE` и имя интересующей нас БД (листинг 29.2).

Листинг 29.2. Проверка существования конкретной БД

```
SHOW DATABASES LIKE 'warehouse';
```

Если ваша БД была создана заранее (допустим она называется 'warehouse' как и демонстрационная БД в примере к этой книге), и вы намерены в дальнейшем работать с ней стоит сказать об этом СУБД (листинг 29.2).

Листинг 29.3. Выбор текущей БД

```
USE 'warehouse';
```

Если же БД пока не существует, то придется уделить несколько минут знакомству с синтаксической конструкцией, используемой для создания новой базы данных

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] имя_базы_данных
[
    [DEFAULT] CHARACTER SET [=] кодовая_страница
| [DEFAULT] COLLATE [=] имя_сравнения
]
```

В минимальной нотации нам достаточно определиться с именем базы данных (листинг 29.4).

Листинг 29.4. Создание БД “СКЛАД”

```
CREATE DATABASE 'warehouse';
```

С таким же успехом вместо ключевого слова DATABASE вы можете использовать SCHEMA, в MySQL это синонимы. Выражение IF NOT EXISTS позволяет избежать конфликтной ситуации, возникающей в случае, если БД с таким именем уже существует.

После создания новая БД помещается в каталог:

C:\ProgramData\MySQL\MySQL Server 5.n\Data\<Имя_БД>

После успешного обращения к CREATE DATABASE (или SCHEMA) в папке \DATA появится единственный файл опций **db.opt** с описанием кодовой страницы БД. Затем, по мере создания остальных объектов в каталоге базы данных расположатся файлы с таблицами (**.frm**), файлы с индексами (**.ibd**), и другие объекты схемы.

По умолчанию вновь создаваемая БД предназначена для работы с языками, соответствующими текущей кодовой странице операционной системы. При необходимости, взяв за основу листинг 29.5 вы сможете получить информацию о кодовой странице любой БД

Листинг 29.5. Выяснение текущей кодовой страницы и порядка сравнения и сортировки

```
SELECT DEFAULT_CHARACTER_SET_NAME, DEFAULT_COLLATION_NAME
FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = 'WAREHOUSE';
```

Если сервер MySQL развернут на компьютере с современной версией Windows 7/8/10, то вероятнее всего запрос возвратит кодовую страницу в кодировке UTF-8 с порядком сравнения символов **utf8_general_ci**.

После всего сказанного предложим листинг 29.6 в котором перед созданием БД конкретизируется как кодовая страница, так и правила сравнения и сортировки символов.

Листинг 29.6. Создание БД с конкретизацией кодовой страницы и порядка сравнения

```
CREATE DATABASE WAREHOUSE CHARACTER SET koi8r COLLATE koi8r_general_ci;
```

Изменение параметров БД

Если вы работаете с БД с помощью утилиты командной строки MySQL, то помните, что факт создания новой БД не означает ваше автоматическое подключение к только что появившейся на свет БД.

Для изменения некоторых параметров БД применяется инструкция ALTER DATABASE (или ALTER SCHEMA). Возможностей у ALTER DATABASE немного, во-первых, с помощью инструкции

```
ALTER {DATABASE | SCHEMA} имя_базы_данных
[DEFAULT] CHARACTER SET [=] кодовая_страница
| [DEFAULT] COLLATE [=] имя_сравнения
```

можно изменить кодовую страницу и порядок сравнения символов. Изменения будут сохранены в файле опций **db.opt**, который вы обнаружите в каталоге базы данных.

Создание таблиц

Одной из визитных карточек сервера MySQL вполне может стать внушительный перечень поддерживаемых форматов таблиц:

- ~ формат ISAM;
 - ~ формат MyISAM;
-

```

~ формат MERGE;
~ формат MEMORY (или HEAP);
~ формат BDB (или Berkeley_DB);
~ формат CSV;
~ формат InnoDB.

```

На сегодняшний день основным форматом таблиц выступает InnoDB (поддерживается, начиная с MySQL 3.23.34). Именно в данном виде с помощью инструкции `CREATE TABLE` создаются таблицы по умолчанию. Наиболее существенная черта InnoDB заключается в полной и безоговорочной поддержке ставших уже хрестоматийными требований к транзакциям описываемых аббревиатурой ACID. Еще одна положительная особенность связана с работой в многопользовательском режиме — теперь блокировки могут накладываться на уровне отдельной строки. Кроме того, InnoDB оснащен развитым механизмом ограничения внешних ключей, что очень важно для поддержки целостности данных.

В ситуации создания новой таблицы применяется следующая синтаксическая конструкция:

```

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] <имя_таблицы>
(определение_столбцов,...)
[опции_таблицы]
[опции_хранения]

```

Необязательный ключ `TEMPORARY` предназначен для указания того, что создается временная таблица доступная только на период выполнения сессии. Сразу после завершения сессии таблица автоматически удаляется. Благодаря условию `IF NOT EXISTS` программист может проверить наличие в БД одноименной таблицы. В таком случае операция создания таблицы будет отменена, иначе уже существующая таблица будет заменена новой.

Рассмотрим простейший пример создания таблицы поставщиков продукции на склад, включающего всего два столбца — первичного ключа и имени поставщика (листинг 29.7).

Листинг 29.7. Создание таблицы поставщиков

```

CREATE TABLE 'suppliers' (
  'SUPPLIER_ID' int(11) NOT NULL AUTO_INCREMENT,
  'SUPPLIER' varchar(100) NOT NULL,
  PRIMARY KEY ('SUPPLIER_ID'),
  UNIQUE KEY 'SUPPLIER_UNIQUE' ('SUPPLIER'),
  KEY 'SUPPLIER_DESC' ('SUPPLIER')
);

```

Обратите внимание на то, что при определении столбцов таблиц помимо указания обязательного названия столбца и типа данных хранящихся в нем мы задействуем ряд дополнительных инструкций, конкретизирующих особенности работы с данными. Так в описание отдельного столбца может включать следующие команды:

```

название_столбца <тип_данных>
[NOT NULL | NULL] [DEFAULT <значение_по_умолчанию>]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT 'текст комментария - до 255 символов']
[COLUMN_FORMAT {FIXED | DYNAMIC | DEFAULT}]
[определение_связи_между_таблицами]

```

Столбец-перечисление

В качестве целевого типа данных для столбца может выступать перечисление — тип данных обозначаемый ключевым словом `ENUM`. Перечисления очень полезны в ситуации когда в столбец должно заноситься одно значение из заранее подготовленного перечня (например день недели (листинг 29.8), название месяца года, и т.п.). Перечень допускает определение 65535 элементов, однако на практике разработчики стараются чтобы он был невелик.

Листинг 29.8. Определение столбца-перечисления

```

CREATE TABLE 'workcalendar' (
...
'taskname' VARCHAR(40),

```

```
'weekday' ENUM('Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница', 'Суббота',
'Воскресение')
);
```

После создания таблицы основанный на перечислении столбец будет контролировать передаваемые в него значения и не допускать попадания в ячейку таблицы некорректного (не входящего в перечень) значения, единственное допустимое исключение отводится неопределенности NULL.

Столбец-множество

Хотя реляционная теория требует, чтобы значение в ячейке таблицы всегда было атомарным, сервер MySQL в данном случае сделал для себя исключение и разрешил размещать в одном столбце несколько текстовых значений, но при условии, если в момент определения таблицы тип данных столбца был определен в виде множества SET. В состав множества может входить не более 64 элементов.

Пример определения столбца-множества предложен в листинге 29.9 в котором создается таблица с расписанием работы магазинов. На роль множества здесь назначен столбец `workday` — в нем хранятся названия рабочих дней недели.

Листинг 29.9. Определение столбца-множества

```
CREATE TABLE `shopcalendar` (
`shopcalendar_id` int NOT NULL AUTO_INCREMENT,
`shopname` VARCHAR(100) NOT NULL,
`workday` SET('Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница',
'Суббота', 'Воскресение'),
`openat` TIME NOT NULL DEFAULT '8:00:00',
`closeat` TIME NOT NULL DEFAULT '22:00:00',
PRIMARY KEY (`shopcalendar_id`)
);
```

Диалект SQL используемый при создании таблиц позволяет управлять процессом создания индексов непосредственно во время проектирования таблицы в теле инструкции `CREATE TABLE`, для этого после определения столбцов таблицы следует воспользоваться ключевым словом `KEY`

```
CREATE TABLE <имя_таблицы> (<имя_столбца>, ...
[UNIQUE] KEY <имя_индекса> (<перечень индексируемых столбцов>));
```

Внешние ключи и связи между таблицами

Механизм обеспечения связи между таблицами реализуемый с помощью внешних ключей имелся у MySQL не всегда — поддержка ограничения внешнего ключа появилась вместе с форматом InnoDB. В более ранних форматах поддержка ссылочной целостности реализовывалась в ручном режиме.

Синтаксическая конструкция определения внешнего ключа для таблиц InnoDB выглядит следующим образом

```
[CONSTRAINT имя_ограничения] FOREIGN KEY [имя_индекса] (имя_столбца,...)
REFERENCES имя_таблицы (имя_столбца, ...)
[ON DELETE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}]
[ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT | SET DEFAULT}]
```

Первое, и самое главное условие для построения связи между таблицами заключается в наличии одностолбцовых первичного ключа в родительской таблице и внешнего ключа в дочерней. Раз в нашем примере в качестве первичного ключа во всех таблицах задействуется целочисленный тип, то столбцы внешнего ключа также должны быть целочисленными. В противном случае связь окажется создать невозможно.

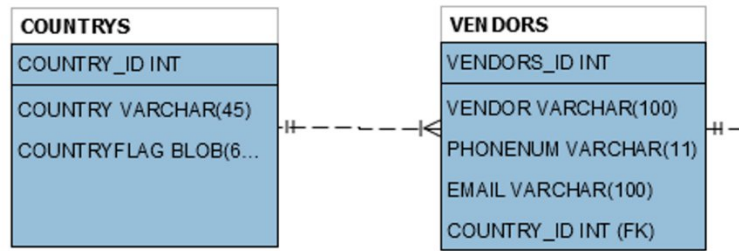


Рис. 29.1. Связь 1:М между таблицами стран COUNTRYS и производителей продукции VENDORS

Схема данных представленная на рисунке 29.2 реализуется средствами листинга 29.10.

Листинг 29.10. Создание и соединение 2-х таблиц

```

CREATE TABLE `countries` (
  `COUNTRY_ID` int(11) NOT NULL AUTO_INCREMENT,
  `COUNTRY` varchar(45) NOT NULL,
  `COUNTRYFLAG` blob,
  PRIMARY KEY (`COUNTRY_ID`),
  UNIQUE KEY `COUNTRY_ID_UNIQUE` (`COUNTRY_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `vendors` (
  `VENDORS_ID` int(11) NOT NULL AUTO_INCREMENT,
  `VENDOR` varchar(100) NOT NULL,
  `PHONENUM` varchar(11) DEFAULT NULL,
  `EMAIL` varchar(100) DEFAULT NULL,
  `COUNTRY_ID` int(11) DEFAULT NULL,
  PRIMARY KEY (`VENDORS_ID`),
  UNIQUE KEY `VENDOR_UNIQUE` (`VENDOR`),
  KEY `FK_COUNTRYS_VENDORS_idx` (`COUNTRY_ID`),
  CONSTRAINT `FK_COUNTRYS_VENDORS` FOREIGN KEY (`COUNTRY_ID`) REFERENCES `countries`
  (`COUNTRY_ID`) ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
  
```

Создание индексов

Все СУБД, в том числе и MySQL, ряд индексов создает автоматически. В частности, без какого-либо дополнительного вмешательства разработчика БД в момент выполнения инструкций CREATE TABLE и ALTER TABLE индексируются:

- ~ столбцы, входящие в состав первичного ключа;
- ~ столбцы внешних ключей;
- ~ столбцы, при определении которых в инструкции CREATE TABLE мы потребовали поддержки уникальности значений.

До MySQL 3.22 полагалось, что индексы должны создаваться одновременно с созданием таблицы, поэтому их определение вносилось непосредственно в тело инструкции CREATE TABLE. Позднее было принято решение о предоставлении разработчику дополнительной возможности управлять индексами с помощью классических инструкций DDL: CREATE, ALTER и DROP.

Синтаксическая конструкция создания индекса для таблицы MySQL выглядит следующим образом:

```

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX <имя_индекса>
[<тип_индекса>]
ON <имя_таблицы> (<столбец1>, ...)
[<опции_индекса>]
[<опции_алгоритма> | <опции_блокировки>] ...
  
```

В минимальной нотации нам будет достаточно всего одной-двух строк кода. Так, для создания индекса, обслуживающего всего один столбец таблицы поставщиков, в котором хранятся названия фирм обратимся к листингу 29.11.

Листинг 29.11. Индекс для одного столбца таблицы

```
CREATE INDEX idx_supplier ON suppliers (supplier) USING BTREE;
```

Для таблиц InnoDB по умолчанию создается индекс на основе В-дерева, поэтому уточнение `USING BTREE` для таблиц такого формата можно считать избыточным.

По умолчанию создаваемый индекс предполагает, что порядок упорядочивания строк в наборе должен осуществляться по возрастанию, от “А” до “Я”. Если нам требуется добиться обратного результата, то сразу после названия столбца следует установить модификатор `DESC`. Немного модифицируем предыдущий пример и построим обратный индекс (листинг 29.12).

Листинг 29.12. Индекс для одного столбца таблицы по убыванию значений

```
CREATE INDEX idx_supplier_desc ON suppliers (supplier DESC);
```

Отметим, что для определения сортировки по возрастанию также существует модификатор `ASC`, но так как он действует по умолчанию им зачастую пренебрегают.

Индекс может охватывать один, два и более столбцов таблицы, при этом разработчик имеет право установить прямой или обратный порядок упорядочивания для каждого из входящих в таблицу столбцов (листинг 29.13).

Листинг 29.13. Индекс для нескольких столбцов таблицы

```
CREATE INDEX idx_vendor ON vendors (vendor, email);
```

Одна из важнейших задач, решаемых при посредничестве индекса заключается в обеспечении уникальности значений, хранящихся в одном или нескольких столбцах таблицы. В таком случае задействуем ключевое слово `UNIQUE` (листинг 29.14).

Листинг 29.14. Индекс для обеспечения уникальности значений в столбце таблицы

```
CREATE UNIQUE INDEX idx_supplier_uniq ON suppliers (supplier);
```

Отметим одну из интересных особенностей построения индекса для текстовых данных. Сервер MySQL с целью экономии системных ресурсов способен индексировать лишь часть строкового значения. Например, в листинге 29.15 предложен пример индексирования лишь первых 20-ти символов из столбца `EMAIL`.

Листинг 29.15. Индекс для первых 20 символов столбца EMAIL

```
CREATE INDEX idx_vendors_email ON vendors (EMAIL(20));
```

Хотя частичный индекс не охватывает содержимое столбца полностью, он все равно существенно ускоряет поиск данных и при этом значительно сокращает размерность В-дерева (или хеш-таблицы).

Удаление индекса

Минимальный вариант синтаксической конструкции SQL позволяющей удалить индекс занимает всего одну строку:

```
DROP INDEX <имя_индекса> ON <имя_таблицы>;
```

Программисту необходимо конкретизировать имя индекса и название таблицы к которой он принадлежит (листинг 29.16).

Листинг 29.16. Удаление индекса

```
DROP INDEX idx_suppliers_supplier ON suppliers;
```

В результате мы удалим индекс “idx_suppliers_supplier” для таблицы поставщиков.

Задание

В соответствии с утвержденной ER–моделью базы данных сделайте следующее:

1. Создайте БД средствами консоли администрирования MySQL.
 2. Создайте все таблицы БД.
 3. Создайте индексы для таблиц.
 4. Самостоятельно разберитесь с инструкцией `CREATE VIEW` и создайте не менее 3 представлений, одно из них должно объединить все таблицы БД.
-

30. Процедурные расширения языка SQL в MySQL

Вид занятия – лабораторное занятие

Время занятия – 4 часа

Программное обеспечение – СУБД MySQL

На этом занятии вам предстоит исследовать процедурные возможности языка SQL применяемого в MySQL.

Переменные

В MySQL можно выделить:

- ~ пользовательские переменные предназначены для решения локальных задач с применением интерактивного SQL, они доступны исключительно в рамках отдельной сессии,
- ~ системные переменные — играют глобальную роль и в своей основной массе связаны с работой сервера.
- ~ локальные переменные используемые в коде хранимых процедур (функций, триггеров).

Переменные, определяемые пользователем

Жизненный путь пользовательской переменной начинается с ее объявления, для этого следует определиться с идентификатором (именем) переменной. Имена переменных, определяемых пользователем должны начинаться с символа “@”.

Несколько примеров объявления переменных вы найдете в листинге 30.1.

Листинг 30.1. Примеры объявления переменных

```
SET @A=1;
SET @B:='Hello, MySQL!';
SET @C=true, @D=false;
SELECT @E:=@A+2;
```

Как видите для определения переменной и присваивания значения потребуются ключевые слова SET или SELECT и операторы присваивания “=” или “:=”.

Пользовательские переменные используются в интерактивном SQL для сохранения промежуточных результатов, передачи/возврата аргументов в процедуры и функции. Такую возможность иллюстрирует рисунок 30.1.

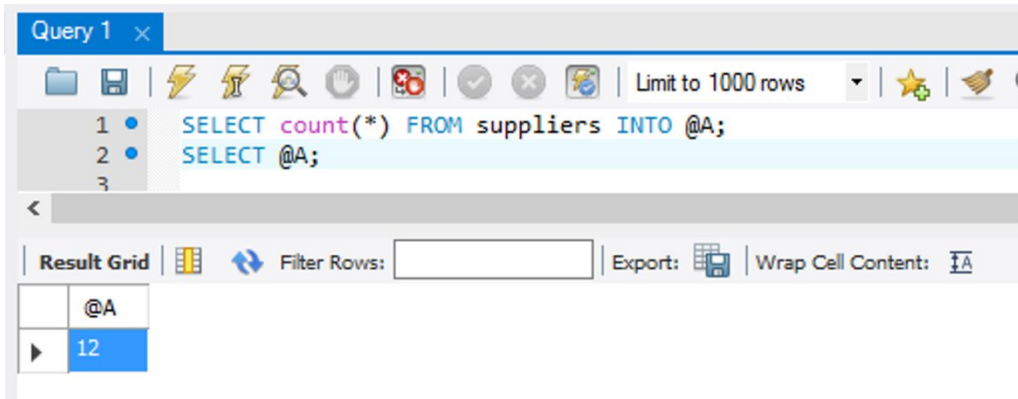


Рис. 30.1. Сохранение результатов запроса в переменной

На рисунке продемонстрировано выполнение запроса в утилите MySQL Workbench с возвратом результата в переменную @A. Позднее полученное значение можно передать в другой запрос, в хранимую процедуру, функцию, и т.п.

Системные переменные

Основное назначение системных переменных заключается в информировании оператора об особенностях конфигурации и работы MySQL и для изменения параметров сервера. При этом различают глобальные и сеансовые системные переменные. Первая категория переменных оказывает влияние на все операции сервера, а сеансовые лишь на операции текущего клиента.

Глобальные переменные в большей степени нужны администратору сервера, чем разработчику БД, поэтому на занятии мы не станем глубоко развивать тему работы с ними. Вместе с тем ряд системных переменных может оказать весьма полезные услуги, например, для организации информирования пользователя о ключевых параметрах сервера (см. табл. 30.1). Они позволяют разработчику создавать более гибкие программные продукты.

Таблица 30.1. Сокращенный перечень глобальных переменных MySQL

Переменная	Назначение
@@connect_timeout	Время таймаута на установку соединения
@@datadir	Путь к файлам БД на сервере
@@datetime_format	Формат даты и времени
@@error_count	Количество ошибок в текущей сессии
@@hostname	Название хоста на котором развернут MySQL
@@last_insert_id	Значение первичного ключа после выполнения INSERT
@@max_connections	Ограничение на число соединений
@@max_user_connections	Максимум соединений для 1 пользователя, значение 0 – ограничений нет
@@port	Номер порта для подключения к серверу
@@secure_file_priv	Путь для сохранения файлов SELECT ... INTO OUTFILE
@@socket	Имя сокета для подключения к серверу
@@tmpdir	Папка для временных файлов
@@version	Версия сервера
@@version_compile_os	Версия операционной системы сервера
@@wait_timeout	Таймаут ожидания транзакции

Один из примеров применения системных переменных с целью информирования пользователя имеется и в демонстрационной базе данных (листинг 30.2). Здесь хранимая процедура pr_DBMSINFO() собирает наиболее востребованную информацию, помещает ее во временную таблицу и затем, после заполнения таблицы возвращает пользователю.

Листинг 30.2. Фрагмент кода хранимой процедуры, использующей системные переменные

```

CREATE PROCEDURE pr_DBMSINFO()
BEGIN

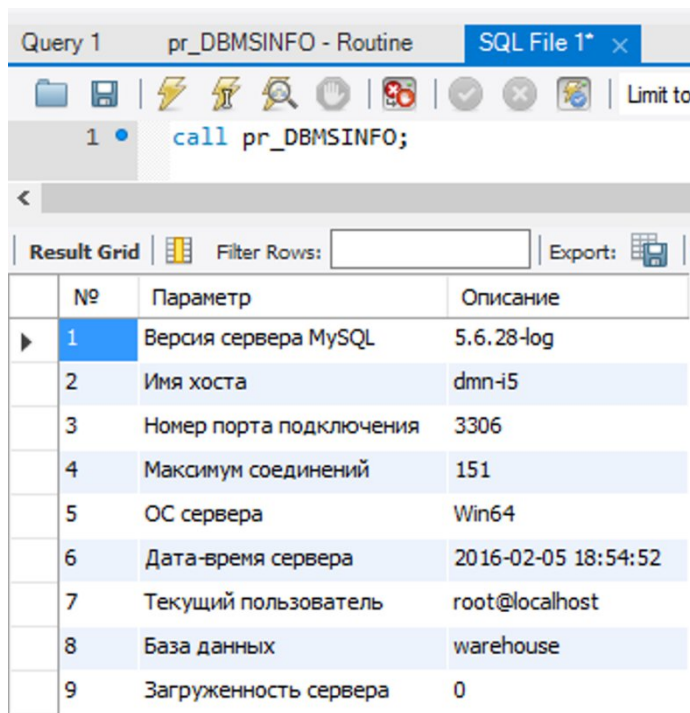
    DROP TEMPORARY TABLE IF EXISTS dbmsinfo_table;
    CREATE TEMPORARY TABLE dbmsinfo_table
        (num int, caption varchar(25), info varchar(100));
    -----
    INSERT INTO dbmsinfo_table VALUES (1, 'Версия сервера MySQL', @@version);
    -----
    INSERT INTO dbmsinfo_table VALUES (2, 'Имя хоста', @@hostname);
    -----
    INSERT INTO dbmsinfo_table VALUES (3, 'Номер порта подключения', @@port);
    -----
    /* и т.д. */

    SELECT num AS '№', caption AS 'Параметр', info AS 'Описание'
    FROM dbmsinfo_table;

    DROP TEMPORARY TABLE IF EXISTS dbmsinfo_table;
END

```

Результаты выполнения процедуры, использующей системные переменные предложен читателю на рисунке 30.2.



	№	Параметр	Описание
▶	1	Версия сервера MySQL	5.6.28-log
	2	Имя хоста	dmn-i5
	3	Номер порта подключения	3306
	4	Максимум соединений	151
	5	ОС сервера	Win64
	6	Дата-время сервера	2016-02-05 18:54:52
	7	Текущий пользователь	root@localhost
	8	База данных	warehouse
	9	Загруженность сервера	0

Рис. 30.2. Выполнение процедуры `pr_DBMSINFO()`

Переменные в хранимой процедуре

Для объявления локальной переменной в теле хранимой процедуры потребуется помощь ключевого слова `DECLARE`, затем указывается имя и тип переменной.

DECLARE имя_переменной [, имя_переменной, ...] тип_данных
 [DEFAULT значение_по_умолчанию];

Переменные объявляются в первых строках кода хранимой процедуры, сразу за ключевым словом `BEGIN` и будут доступными до `END` закрывающего тело процедуры.

Составной оператор `BEGIN...END`

Составной оператор `BEGIN...END` предназначен для логического объединения двух и более операторов, которые должны выполняться как единое целое. Задача оператора — указание начала и конца программного блока. Характерным примером подобного блока является создаваемый по умолчанию каркас кода хранимой процедуры, здесь весь исполняемый код заносится вовнутрь `BEGIN...END`. Кроме того, составные операторы активно используются в коде функций и триггеров в условных операторах и при построении циклов,

Условные операторы

Организуя вычислительный процесс, программист должен обладать возможностью направлять ход вычислений в то или иное русло в зависимости от полученных в ходе выполнения программы промежуточных результатов или введенных пользователем исходных данных. Для этих целей в MySQL предусмотрен набор привычных нам по классическим языкам программирования операторов.

Оператор `IF..THEN..ELSE`

Для построения едва ли не самой важной для всех процедурных языков программирования конструкции ветвления (если...то...иначе) в MySQL предусмотрен классический оператор условия `IF`. В простейшем случае, когда выполнение выражения осуществляется если проверяемое условие возвращает истину (`true`) синтаксис оператора таков:

```
IF (проверяемое_условие) THEN
    выполняемое_выражение;
END IF;
```

Если же нас интересует и действие, осуществляемое при невыполнении условия, то конструкция дополняется `ELSE`:

```
IF (проверяемое_условие) THEN
    выполняемое_выражение_1;
```

```
ELSE
    выполняемое_выражение_2;
END IF;
```

Пример работы с операторами IF...THEN...ELSE предложен в листинге 30.3.

Листинг 30.3. Функция возвращающая максимальное из 2 целых чисел

```
DELIMITER <$$>
CREATE FUNCTION fn_getmax(x INT,y INT) RETURNS INT
BEGIN
    IF x>y THEN RETURN x;
    ELSE RETURN y;
    END IF;
END <$$>
DELIMITER ;
```

Оператор–селектор CASE

Практически во многих высокоуровневых языках программирования оператор–селектор CASE введен для повышения удобства работы программиста и предоставления ему возможности создавать читабельные листинги программ.

В MySQL решаемая селектором задача схожая — осуществить множественный выбор, но особенностью CASE в процедурном SQL является то, что здесь оператор обладает двумя синтаксическими формами.

Первая форма CASE практически ничем не отличается от аналогичных конструкций в C++, C# или Delphi.

```
CASE контрольное_значение
    WHEN значение_1 THEN выражение_1
    [WHEN значение_2 THEN выражение_2] ...
    [ELSE выражение_3]
END CASE
```

Селектор последовательно сравнивает контрольное_значение со значениями значение_1, значение_2, и т.д. и, в случае совпадения выполняет соответствующее выражение. Если совпадений было не обнаружено выполнению подлежит код ELSE. Пример использования подобного селектора представлен в листинге 30.4.

Листинг 30.4. Функция преобразования номера дня недели в текст

```
CREATE FUNCTION fn_weekday(weekdaynum tinyint) RETURNS varchar(12)
BEGIN
    CASE weekdaynum
        WHEN 1 THEN RETURN 'Понедельник';
        WHEN 2 THEN RETURN 'Вторник';
        WHEN 3 THEN RETURN 'Среда';
        WHEN 4 THEN RETURN 'Четверг';
        WHEN 5 THEN RETURN 'Пятница';
        WHEN 6 THEN RETURN 'Суббота';
        WHEN 7 THEN RETURN 'Воскресение';
        ELSE RETURN NULL;
    END CASE;
END
```

Представленная функция преобразует поступающие на вход номера дней недели в их текстовое название, а если число оказывается в недопустимом диапазоне функция возвратит неопределенность NULL.

Вторая форма CASE в MySQL более гибкая чем первая.

```
CASE
    WHEN выражение_сравнения THEN выражение_1
    [WHEN выражение_сравнения THEN выражение_2] ...
    [ELSE выражение_3]
END CASE
```

Здесь вместо простого указания единственного контрольного значения сравнение описывается непосредственно для каждой из конструкции WHEN. В качестве выражения сравнения может приниматься любой набор операторов при условии, что он возвратит истину или ложь. Как только выражение сравнения возвращает истину выполняется соответствующее выражение.


```

BEGIN
  declare x float;
  set x=minX;
  while x<=maxX do -- цикл выполняется пока x меньше и равно maxX
    select x,k/x;
    set x=x+0.1;
  end while;
END <$$>
DELIMITER ;

```

На рисунке 30.4 представлен экранный снимок работы процедуры. Обратите внимание на особенность реакции сервера MySQL на попытку деления k на 0. В данном случае СУБД хитрит и вместо того, чтобы сгенерировать исключительную ситуацию как реакцию на недопустимую математическую операцию сервер возвращает неопределенность NULL.

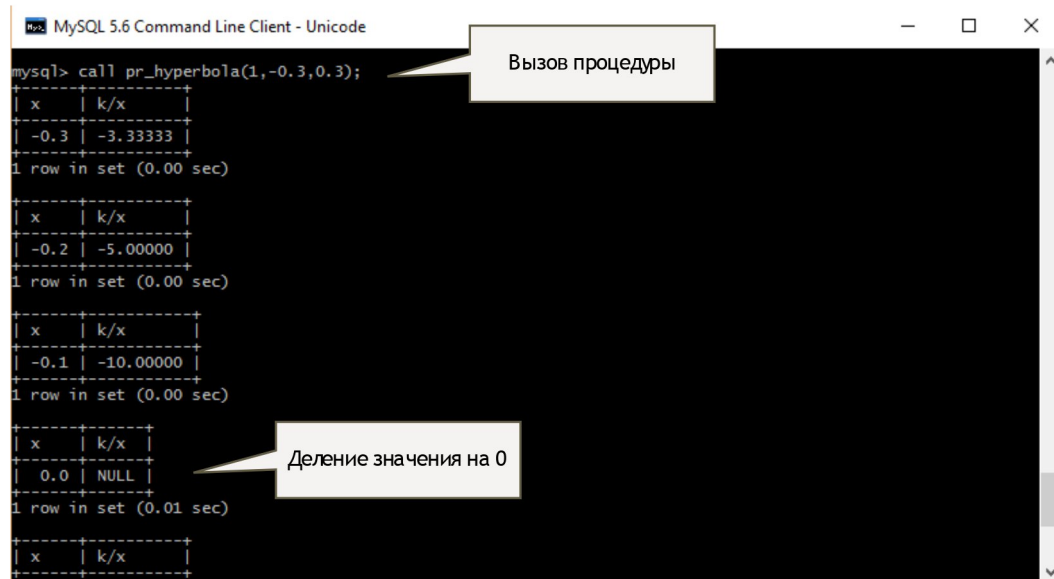


Рис. 30.4. Отклик MySQL на попытку деления на 0

Цикл с постусловием REPEAT

Цикл с постусловием основан на операторе REPEAT, его синтаксис выглядит следующим образом:

```

[метка:] REPEAT
операторы_цикла
UNTIL условие_цикла
END REPEAT [метка]

```

Как и у всех высокоуровневых языков программирования цикл REPEAT в MySQL имеет две особенности:

1. Цикл выполняется пока его условие ложно.
2. Вне зависимости от состояния условия цикл выполнит по крайней мере одну итерацию, ведь проверяемое условие находится в последней строке его кода.

Для демонстрации работы цикла REPEAT вновь решим математическую, точнее тригонометрическую задачу построения таблицы синусов. Но на этот раз объединим тригонометрию с задачами характерными для БД — поместим результаты расчетов в временную таблицу.

Листинг 30.7. Хранимая процедура, возвращающая таблицу синусов и цикл REPEAT

```

DELIMITER <$$>
CREATE PROCEDURE pr_sin_table()
BEGIN
  DECLARE d int DEFAULT 0;  # переменная для хранения градусов
  DECLARE r float DEFAULT 0; # переменная для хранения радиан
  # ---- создаем временную таблицу с 3-мя столбцами ----
  DROP TEMPORARY TABLE IF EXISTS sin_table;
  CREATE TEMPORARY TABLE sin_table (deg int, rad float, SinValue float);
  # ---- цикл заполнения таблицы данными ----

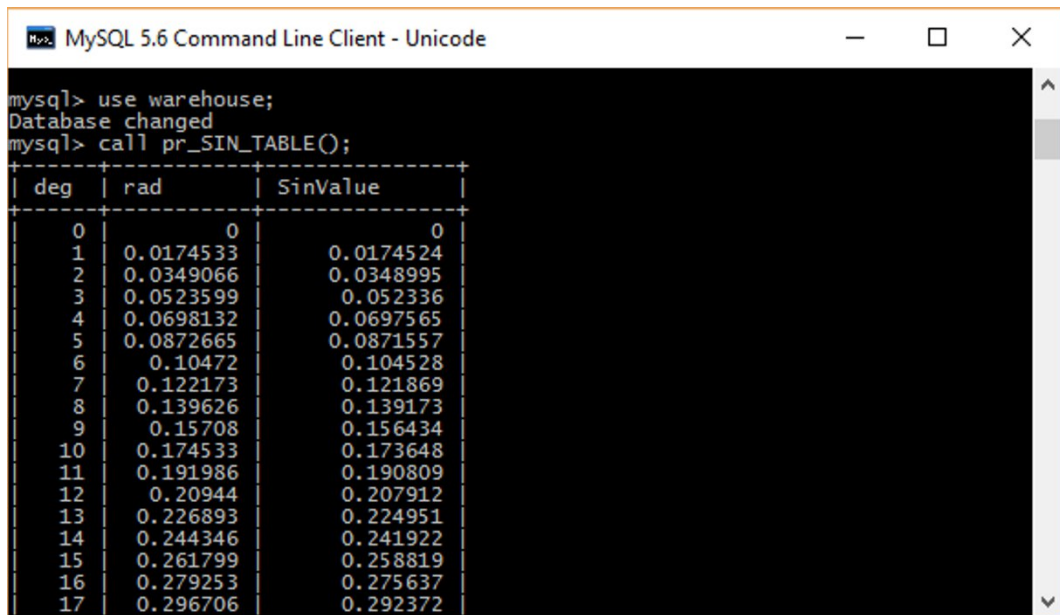
```

```

REPEAT
  INSERT INTO sin_table VALUES (d, r, sin(r));
  SET r=r+pi()/180;
  SET d=d+1;
  UNTIL r>=pi()
  END REPEAT;
# -----
SELECT * FROM sin_table; # результат работы - выборка из таблицы синусов
DROP TEMPORARY TABLE IF EXISTS sin_table; # временная таблица больше не нужна
END <$$>
DELIMITER ;

```

В примере цикл продолжает работу до тех пор, пока не будет просмотрен диапазон значений от 0 до π . В теле цикла осуществляется расчет значения синуса (для этого задействуется библиотечная функция `sin()`) и результат заносится в соответствующий столбец временной таблицы. Затем, применив запрос `SELECT` мы делаем полную выборку из таблицы и удаляем ее. Результат выполнения процедуры отражен на рисунке 30.5.



The screenshot shows a MySQL 5.6 Command Line Client window. The user has entered the following commands:

```

mysql> use warehouse;
Database changed
mysql> call pr_SIN_TABLE();

```

The output is a table with three columns: `deg`, `rad`, and `SinValue`. The table contains 18 rows of data, starting from 0 degrees and ending at 17 degrees.

deg	rad	SinValue
0	0	0
1	0.0174533	0.0174524
2	0.0349066	0.0348995
3	0.0523599	0.052336
4	0.0698132	0.0697565
5	0.0872665	0.0871557
6	0.10472	0.104528
7	0.122173	0.121869
8	0.139626	0.139173
9	0.15708	0.156434
10	0.174533	0.173648
11	0.191986	0.190809
12	0.20944	0.207912
13	0.226893	0.224951
14	0.244346	0.241922
15	0.261799	0.258819
16	0.279253	0.275637
17	0.296706	0.292372

Рис. 30.5. Вызов хранимой процедуры построения таблицы синусов угла

Петлевой цикл LOOP

Основная особенность петлевого цикла заключается в том, что в нем, в отличие от циклов `WHILE` и `REPEAT` отсутствует условие выхода. Поэтому программист обязан предусмотреть способ прерывания петли иначе мы рискуем получить бесконечный процесс. Синтаксис цикла выглядит следующим образом;

```

[метка:] LOOP
операторы_цикла
END LOOP [метка]

```

Обратите внимание, что на этот раз при работе с циклом целесообразно использовать метки, они помогут организовать выход из цикла. Кроме меток для выхода из цикла еще понадобятся услуги выражения `LEAVE`.

Рассмотрим практический пример использования в хранимой процедуре петлевого цикла. В листинге 30.8 мы научим MySQL рассчитывать показательную функцию вида $y=a^x$. На вход хранимой процедуры поступают три аргумента: значение a , начало и конец диапазона значений x .

Листинг 30.8. Хранимая процедура расчета показательной функции и цикл LOOP

```

DELIMITER <$$>
CREATE PROCEDURE pr_exponential(IN a float, IN minX dec(4,1), IN maxX dec(4,1))
BEGIN
  DECLARE x dec(4,1);
  DECLARE y float;
  SET x=minX;
  # ----- начало цикла -----

```

```

label1: LOOP
    SET y=pow(a,x); -- расчет
    SELECT x, y;    -- вывод результатов расчета
    SET x=x+0.1;    -- приращение x
    IF x>maxX THEN LEAVE label1; -- условие выхода из цикла
    END IF;
END LOOP label1;
# -----конец цикла -----
END <$$>
DELIMITER ;

```

Прерывание цикла — LEAVE

Зачастую программная логика выполнения той или иной задачи предполагает необходимость досрочного выхода из цикла. Для этой цели в MySQL предусмотрен оператор

LEAVE название_метки

Оператор LEAVE позволит нам покинуть цикл, организованный любым способом (будь то LOOP, WHILE или REPEAT) и перейти к указанной после ключевого слова метке (листинг 30.8).

Пропуск итерации — ITERATE

Все три имеющиеся в распоряжении MySQL типа циклов предоставляют программисту услугу по пропуску итерации, для этой цели в процедурный SQL введен оператор

ITERATE название_метки

Встретив оператор цикл досрочно прекращает выполнение текущей итерации, но цикл не прерывается, а лишь переходит к началу следующей итерации.

Листинг 30.9 демонстрирует работу операторов ITERATE и LEAVE. Задача хранимой процедуры заключается в генерации N целых 2-значных случайных нечетных чисел.

Листинг 30.9. Генерируем N случайных нечетных 2-значных чисел

```

DELIMITER <$$>
CREATE PROCEDURE pr_random(IN N INT)
BEGIN
    DECLARE x int DEFAULT 0;
    DECLARE r int;
    # ----- начало цикла -----
label1: LOOP
    SET r=CEILING(100*Rand()); -- генерируем случайное 2-значное целое
    # проверка на четность, если да -> пропуск итерации
    IF (r % 2 = 0) THEN ITERATE label1; END IF;
    # если выполняется этот блок, то число нечетное
    SET x=x+1;    -- приращение счетчика
    SELECT x, r;  -- вывод результатов расчета
    # проверяем число успешно сгенерированных случайных чисел
    IF (x>N-1) THEN LEAVE label1; END IF; -- если да -> завершаем цикл
END LOOP label1;
# -----конец цикла -----
END <$$>
DELIMITER ;

```

В нашем примере оператор ITERATE пропускает итерацию в случае если в переменной r в результате выполнения выражения CEILING(100*Rand()) оказалось неподходящее нам четное число. Оператор LEAVE стоит на страже границ цикла и прекращает его работу, когда счетчик удачных генераций x достигнет поставленного предела N.

Задание

Разработайте следующий перечень хранимых процедур:

1. Процедуры вставки, редактирования и удаления записей в одну из таблиц БД.
 2. Процедура выборки всех строк из 2-х объединенных таблиц.
 3. Произвольная процедура демонстрирующая работу условного оператора.
-

4. Произвольная процедура демонстрирующая работу с циклом.
5. Самостоятельно изучите особенности работы с триггерами и предложите пример триггера.

В отчет о выполненной лабораторной работе необходимо включить исходный код разработанных процедур.

Приложение 1. Настройка строки соединения ADO

Щелчок по кнопке с многоточием напротив свойства `ConnectionString` компонента ADO приведет к выводу на экран компьютера редактора строки соединения. Для подключения к БД предстоит выполнить определенную последовательность действий:

1. выбираем элемент Use Connection String и нажимаем кнопку Build. Это действие вызывает редактор “Свойства канала передачи данных”;
2. На странице “Поставщик данных” редактора “Свойства канала передачи данных” выделяем строку Microsoft Office 14 Access Database Engine... и нажимаем кнопку “Далее >>”;
3. На странице “Соединение” в строке ввода “Источник данных” указываем путь и имя файла базы данных. Щелкаем по кнопке “Проверить соединение”. Если вы ни в чем не ошиблись, то на экране будет выведено сообщение “Проверка соединения выполнена”;
4. Нажав на кнопку ОК, запоминаем настройки строки соединения.

В итоге наших действий в свойстве `ConnectionString` компонента `ADOTable1` появится примерно такая строка, содержащая параметры соединения:

```
Provider=Microsoft.ACE.OLEDB.14.0;Data Source=D:\DATA\books.acdb;Persist Security Info=False
```

Если вы внимательно изучите параметры соединительной строки и их значения, то обязательно обнаружите путь к папке с вашей БД.

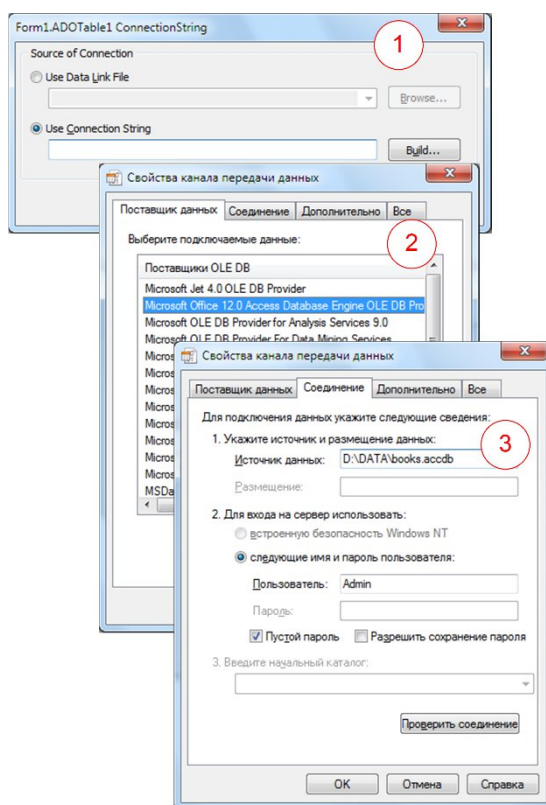


Рисунок Настройка строки соединения ADO

Приложение 2. Встроенные функции InterBase

Таблица ПП.1. Встроенные функции

Функция	Описание
AVG ()	Функция агрегирования, возвращающая среднее арифметическое из множества значений
CAST ()	Функция конвертирования, позволяет осуществлять преобразование типов данных
COUNT ()	Функция агрегирования, возвращающая число строк в отношении
EXTRACT ()	Функция конвертирования, позволяет получить отдельные значения из типов данных DATE, TIME и TIMESTAMP
GEN_ID ()	Получение значения генератора
MAX ()	Функция агрегирования, возвращающая максимальный элемент из множества
MIN ()	Функция агрегирования, возвращающая минимальный элемент из множества
SUM ()	Функция агрегирования, возвращающая сумму значений множества
UPPER ()	Преобразование текстовых символов к верхнему регистру

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Осипов Д. Базы данных и Delphi. Теория и практика. — СПб.: Издательство БХВ, 2011. — 720 с.
2. Осипов Д. Delphi. Профессиональное программирование. — СПб.: Символ-Плюс, 2006. — 1056 с.
3. Осипов Д.Л. InterBase и Delphi. Клиент-серверные БД. — М.: ДМК-Пресс – 2015. 536 с.:ил.
4. Осипов Д.Л. MySQL и Delphi. Базы данных. Технология DireDAC. — [б.м.] Издательские решения — 2017. 624 с.
5. Аткинсон Леон. MySQL. Библиотека профессионала.: Пер. с англ. — М.: Издательский дом “Вильямс”, 2002. — 624 с.
6. Браст Эндрю Дж., Форте Стивен. Разработка приложений на основе Microsoft SQL Server 2005. Мастер-класс./Пер. с англ. — М.: Издательство “Русская редакция”, 2007. — 880 с.
7. Вигерс Карл. Разработка требований к программному обеспечению. Пер. с англ. — М.: Издательско-торговый дом “Русская редакция”, 2004 — 576 с.
8. Гарольд Э., Минс С. XML. Справочник. — Пер. с англ. — СПб: Символ-Плюс, 2002. — 576 с.
9. Грофф Дж., Вайнберг П., SQL: Полное руководство: Пер. с англ. — 2-е изд., перераб. и доп. — К. Издательская группа BHV, 2001. — 816 с.
10. Дейв Энсор, Йен Стивенсон. Oracle. Проектирование баз данных.: Пер. с англ. — К.: Издательская группа BHV, 1999. — 560 с.
11. Дейт, К. Дж. Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом “Вильямс”, 2006. — 1328 с.
12. Елманова Н., Трепалин С., Тенцер А. Delphi и технология COM. — СПб.: Питер, 2004. — 704 с.
13. Кайт Том. Oracle для профессионалов. Пер. с англ./Том Кайт — СПб.: ООО «ДиаСофтЮП», 2003. — 672 с.
14. Клайн К. SQL. Справочник. 2-е издание.: Пер. с англ. — М.: КУДИЦ-ОБРАЗ, 2006. — 832 с.
15. Коннолли Томас, Бегг Каролин, Страчан Анна. Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2001. — 1120 с.
16. Молинаро Э. SQL. Сборник рецептов. — Пер. с англ. — СПб: Символ-Плюс, 2009. — 672 с.
17. Нильсен Пол. Microsoft SQL Server 2005. Библия пользователя.: Пер. с англ. — М.: ООО “И.Д. Вильямс”, 2008. — 1232 с.ил.
18. Райордан Р. Основы реляционных баз данных.: Пер. с англ. — М.: Издательско-торговый дом “Русская редакция”, 2001. — 384 с.: ил.
19. Роб П., Коронел К. Системы баз данных: проектирование, реализация и управление. — 5-е изд., перераб. и доп.: Пер с англ. — СПб.: БХВ-Петербург, 2004. — 1040 с.: ил.

СПИСОК ИСПОЛЬЗУЕМОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1. Microsoft SQL Server 2012
 2. Microsoft Access
 3. Microsoft Visio
 4. Microsoft Visual Studio (C++, C#)
 5. Interbase XE3
 6. Firbird 2.53
 7. MySQL 5.7
 8. Embarcadero RAD Studio XE10 (Delphi, C++ Builder)
-

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ
Методические рекомендации к самостоятельной работе студентов
Специальность 10.05.01 «Компьютерная безопасность»
Специализация «Информационно-аналитическая и техническая экспертиза
компьютерных систем»
Квалификация выпускника – специалист
Год начала обучения – 2026 г.
Форма обучения – очная

Ставрополь
2026

Оглавление

Содержание.....	Ошибка! Закладка не определена.
Пояснительная записка.....	173
План-график выполнения СРС по дисциплине.....	173
Методические рекомендации к СРС.....	173
Форма контроля СРС.....	174
Задания для СРС.....	174
Примерные темы рефератов.....	174
Примерные темы докладов.....	176
Примерные темы индивидуальных творческих заданий.....	177
Требования к представлению и оформлению результатов СРС.....	178
Рекомендуемая литература и интернет-ресурсы.....	180

Пояснительная записка

В ходе изучения дисциплины студенты должны выполнить домашнее задание, способствующее приобретению навыков проектирования баз данных и работы с реляционными СУБД. Студенту предлагается смоделировать предметную область и спроектировать базу данных для СУБД. По итогам выполнения работы преподавателю предоставляется набор скриптов, разработанная студентом программа и оформленный отчёт с полученными результатами.

Текущий контроль усвоения знаний осуществляется путем прохождения 6 контрольных точек, подготовки и сдачи отчетов по итогам выполнения лабораторных работ, проверки выполнения домашнего задания, опросов на практических занятиях.

План-график выполнения СРС по дисциплине

№ контрольной точки	Вид самостоятельной работы	Форма контроля	Зачетные единицы (часы)
	7 семестр		
1, 2	Подготовка к лабораторным занятиям	Защита лаб.работ	21
1	Самостоятельное изучение темы	Тестирование	10
2	Написание реферата	Проверка	10
3	Курсовая работа	Проверка	30
	Итого за 7 семестр		81
	8 семестр		
3, 4	Подготовка к лабораторным занятиям	Защита лаб.работ	7
3	Самостоятельное изучение темы	Тестирование	1
4	Написание доклада	Проверка	1
	Итого за 8 семестр		9
	Итого		90

Методические рекомендации к СРС

Раздел 1. Основы реляционных БД: Формирование SQL запросов, обеспечивающих выполнение второго раздела задания, связанного с формированием отчетов.

Демонстрация преподавателю отработанных запросов.

Задание: Написание разделов пояснительной записки, связанных с формированием посредством SQL запросов отчетов по предметной области.

Литература: О[1, 2].

Раздел 2. Проектирование БД

Основные понятия: Построение модели сущность-связь. Написание скриптов создания таблиц базы данных. Создание таблиц и наполнение их небольшим (5-10) количеством записей.

Задание: Написание (в Word) разделов пояснительной записки, связанных с проектированием модели сущность-связь, созданием таблиц БД и наполнением их информацией.

Литература: О[1-3].

Раздел 3. Современные технологии проектирования ПО Формирование SQL запросов.

Задание Написание скриптов создания таблиц базы данных.

Литература: Д[1-6].

Форма контроля СРС

Качество усвоения знаний студентами осуществляется в форме тестирования, индивидуального собеседования, проверки доклада, реферата и проверки индивидуального творческого задания.

Задания для СРС

Примерные темы рефератов

1. История развития, назначение и роль баз данных

Содержание реферата: Этапы развития информационных систем.

Понятие базы данных. Файловые системы и базы данных. Классификация задач, решаемых с использованием СУБД.

2. Физическая организация баз данных

Содержание реферата: Структуры данных и базы данных. Способы хранения информации в базах данных. Способы повышения эффективности обработки данных за счет их организации.

Инвертированные файлы.

3. Общие принципы построения СУБД

Содержание реферата: Общая характеристика, назначение, возможности, состав и архитектура СУБД. Классификация СУБД.

Информационное, лингвистическое, математическое, аппаратное, организационное, правовое обеспечения СУБД.

4. Средства поддержания целостности базы данных

Содержание реферата: Метаданные. Словарь-справочник данных.

Ссылочная целостность. Механизм транзакций. Управление доступом.

Средства дублирования и восстановления. Особенности реализации баз данных с высокими требованиями на надежность хранения и обработки.

5. Эксплуатация баз данных

Содержание реферата: Состав, порядок планирования и проведения регламентных работ. Сервисные средства СУБД. Задачи администратора базы данных. Организация труда обслуживающего персонала.

6. Технология и модели архитектуры клиент/сервер

Содержание реферата: Достоинства и недостатки моделей архитектуры клиент/сервер и их влияние на функционирование сетевых СУБД.

7. Серверы баз данных

Содержание реферата: Использование средств прямого ввода-вывода, управления памятью, поддержания целостности, защиты от сбоев.

Возможности по обработке неструктурированных данных большого объема (Oracle Multimedia Server).

8. Серверы баз данных

Содержание реферата: Поддержка Internet (Oracle Web Server). Оценка эффективности и адаптации функционирования сервера баз данных (тесты производительности). Проблемы оптимизации доступа к базе данных.

9. Клиентская часть архитектуры клиент/сервер

Содержание реферата: Средства поддержания интерфейса с различными категориями пользователей. Языки запросов. Языки описания данных.

Языки манипулирования данными. Стандарты Xbase, SQL. Языки четвертого поколения 4GL.

10. Клиентская часть архитектуры клиент/сервер

Содержание реферата: Интерфейс языков СУБД с языками программирования. Средства реализации диалогового интерфейса и подготовки отчетов в языках СУБД. Стандарты на графический пользовательский интерфейс GUI.

11. Интерфейс между клиентом и сервером.

Содержание реферата: Протоколы согласованной работы.

Распределенные базы данных в сетях ЭВМ. Средства интеграции и взаимодействия разнородных распределенных баз данных.

12. Централизация логики приложения на сервере базы данных

Содержание реферата: Создание и использование процедур, функций, триггеров, пакетов. Программные утилиты СУБД Oracle11.

13. Автоматизированное проектирование

Содержание реферата: Средства автоматизации проектирования баз данных: общая характеристика, назначение и возможности, классификация, универсальные и специализированные генераторы программ для СУБД.

14. Объектно-ориентированное программирование в СУБД

Содержание реферата: Принципы объектно-ориентированного программирования. Недостатки реляционных СУБД. Объектные расширения реляционных СУБД. Объектно-реляционные адаптеры.

Объектно-реляционные СУБД. Объектные СУБД. Стандарты на объектные СУБД.

15. Многоплатформные СУБД

Содержание реферата: СУБД Oracle, Informix, Sybase, DB2. Область применения. Особенности их реализации. Сетевые компоненты многоплатформных СУБД. Требования по их эксплуатации.

16. СУБД, ориентированные на конкретные платформы

Содержание реферата: СУБД DBManager в OS/2. SQL/400 в AS/400.

СУБД Access в Microsoft Windows. Связь компонентов СУБД с особенностями операционной среды. Аппаратная поддержка управления данными. Использование возможностей пакетов прикладных программ конкретных платформ совместно с СУБД. Средства распределенной обработки данных.

17. СУБД семейства InterBase

Содержание реферата: InterBase, Yaffi, Firebird. История развития и причины популярности СУБД данного семейства. Языки программирования и языки манипулирования данными в СУБД семейства. Проектирование и эксплуатация информационных систем. Механизмы блокирования и управления доступом в многопользовательской среде.

Примерные темы докладов

1. Управление данными во внешней памяти.
 2. Управление буферами оперативной памяти.
 3. Управление транзакциями.
 4. Журнализация.
 5. Поддержка языков БД.
 6. Типовая организация современной СУБД.
 7. Основные особенности систем, основанных на инвертированных списках.
 8. Структуры данных.
 9. Файловая система.
 10. Ограничения целостности БД.
 11. Иерархические системы.
 12. Иерархические структуры данных.
 13. Нормальные формы.
 14. Сериализация транзакций.
 15. Сетевые системы.
 16. Сетевые структуры данных.
 17. Манипулирование данными.
 18. Модели клиент\сервер.
 19. Достоинства и недостатки реляционных СУБД.
 20. Перспективы развития баз данных.
-

Примерные темы индивидуальных творческих заданий

Разработайте базу данных:

1. “База данных поликлиники”. Учѣту подлежат: больные (ФИО, домашний адрес), лечащий персонал (ФИО, специальность), дата посещения больным врача и поставленный диагноз.
 2. “База данных футбольных матчей”. Учѣту подлежат: спортивный клуб, игроки команд (ФИО), дата матча, игравшие команды, забитые мячи, авторы голов.
 3. “База данных “Бюро по туризму”. Учѣту подлежат: клиенты фирмы (ФИО, домашний адрес), список курортов с ценами и продолжительностью путѣвок, даты поездок клиентов на курорт.
 4. “База данных “Магазин видеофильмов”. Учѣту подлежат: название фильма, жанр, год выхода, киностудия, режиссѣр, цена, носитель (DVD/Видеокассета), число экземпляров.
 5. “База данных “Домашняя фонотека”. Учѣту подлежат: исполнитель, страна к которой принадлежит исполнитель, название альбома, год выхода альбома, музыкальный жанр альбома, наименование и продолжительность музыкальных композиций входящих в альбом, формат хранения (CD, mp3, ...).
 6. База данных “Склад автомобильных запчастей”. Учѣту подлежат: название детали, производитель, цена, в какой марке автомобиля применяется, поставщик детали.
 7. База данных “Метеостанция”. Учѣту подлежат: названия станций, их географические координаты, персонал станций (ФИО, должность), каждая метеостанция ежечасно сохраняет данные о температуре окружающей среды.
 8. База данных “Библиотека”. Учѣту подлежат: жанр книги, авторы книги, название книги, количество экземпляров, место хранения книги, читатель (ФИО и адрес), дата получения читателем, дата возврата книги.
 9. База данных “Склад телевизоров”. Учѣту подлежат: поставщик, производитель телевизора, модель телевизора, дата поставки телевизора, дата списания телевизора со склада, не менее 5 характеристик телевизора (производитель, диагональ, вес, и т.п.), цена телевизора от поставщика, отпускная цена телевизора, количество телевизоров.
 10. База данных “Автовокзал.” Учѣту подлежат: автобусы (марка, дата выпуска и государственный регистрационный номер), водители (ФИО, дата рождения, водительская категория, какой автобус закреплѣн), расписание движения автобусов (наименование населѣнного пункта, день и время отправления, день и время прибытия, стоимость билета, автобус выполняющий рейс).
 11. База данных “Магазин одежды.” Учѣту подлежат: товар (название, размер, цена, материал, производитель), дата поступления, дата продажи.
 12. База данных “Почтовое отделение” Учѣту подлежат: поступающая на почту заказная корреспонденция (письма, ценные бандероли и посылки). Дата поступления, дата выдачи, паспортные данные получателя корреспонденции.
-

13. База данных “Аптечный киоск” Учёту подлежат: лекарства (наименование, дозировка, производитель, отпускная цена, место хранения, выдача только по рецепту или нет, показания и противопоказания).

14. База данных “Расписание занятий в институте” Учёту подлежат: специальность, учебная группа, учебные дисциплины (название и вид занятия), день недели, номера часов проведения занятий, преподаватель, номер аудитории

15. База данных “Гостиница” Учёту подлежат: номера гостиницы (номер помещения, число спальных мест, наличие душа, туалета, холодильника, телевизора, сплит-системы, стоимость проживания), занятость номера (дата заселения, паспортные данные жильцов, дата освобождения номера)

16. База данных “Кинотеатр”. Учёту подлежат: название кинофильма, жанр кинофильма, возрастные ограничения, дата и время киносеанса, название кинозала, номер посадочного места, цена билета.

17. База данных “Торговый зал”. Учёту подлежат: название товара, отпускная цена товара, количество товара в зале, дата выпуска, срок годности. База данных должна предусматривать отпуск товара покупателю со списанием товара из зала.

18. База данных “Учёт электроэнергии”. База учитывает количество потреблённой электроэнергии в небольшом посёлке и ежемесячно подготавливает индивидуальные квитанции об оплате. Учёту подлежат: адрес дома, владелец дома, количество потреблённой электроэнергии за месяц, стоимость за кВт/ч. Стоимость электроэнергии может изменяться!

19. База данных “Отдел кадров завода”. Учёту подлежат: ФИО, рабочая специальность, квалификация (разряд), образование работника, дата трудоустройства, стаж работы, должностной оклад, надбавка (зависит от квалификации и стажа).

20. База данных “Художественный музей” Учёту подлежат: автор картины, название картины, краткое описание сюжета, дата написания картины, художественное направление (реализм, абстракция, и т.д.), зал в котором выставлена картина.

Требования к представлению и оформлению результатов СРС

База данных разрабатывается на основе Microsoft Access (или на другой БД по согласованию с преподавателем). БД должна быть нормализована до 4 нормальной формы.

Клиентское приложение разрабатывается на языках Delphi, C++ Builder, C# или на другом языке по согласованию с преподавателем.

Приложение должно включать формы ввода данных, формы просмотра и редактирования данных. В рабочих формах приложения должна быть предусмотрена возможность:

1. Упорядочивания данных;
2. Фильтрации данных;
3. Поиску данных.

Программа должна предоставлять возможность вывода отчётов. В отчётах должны быть предусмотрены возможности:

1. Вывода всех данных из всех таблиц БД;
2. Использования агрегирующих функций SQL (Avg, Count, Min, Max, и т.д.).

Бумажный отчёт курсовой работы оформляется по требованиям, установленным в ВУЗе. Отчёт должен содержать:

1. ER-модель БД;
2. Полную спецификацию таблиц.
3. Примеры используемых в работе SQL запросов.
4. Описание клиентского приложения.
5. Инструкцию по работе с приложением.

Объём отчёта 25-30 стр. К отчёту прилагается CD-диск с БД и исходным кодом проекта. Допускается один CD- на несколько человек или всю учебную группу.

Рекомендуемая литература и интернет-ресурсы

Основная литература:

1. Хомоненко А.Д. и др. Базы данных. Учебник для высших учебных заведений – СПб.: КОРОНА принт, 2010. – 736 с.
2. Б.Я. Советов, В.В. Цехановский. Базы данных: теория и практика. 2-е издание, учебник для бакалавров - М.: ЮРАЙТ-ИЗДАТ, 2011. – 463 с.
3. В.Ю. Пирогов Информационные системы и базы данных: организация и проектирование – СПб.: Издательский дом БХВ, 2009. – 528 с.: ил.

Дополнительная литература:

1. Конноли, Бегг. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание. М.: Вильямс, 2003, - 1436 с.
2. Дейт К. Дж. Введение в системы баз данных, 8-е издание.: Пер.с англ. - М. Издательский дом “Вильямс”, 2005,- 1328с., ил.
3. Марков А.С., Лисовский К.Ю. Базы данных. Введение в теорию и методологию: Учебник. – М.: Финансы и статистика, 2006. – 512 с., ил.
4. Осипов Д. Delphi. Профессиональное программирование. Профессиональное программирование. – СПб.: Символ-Плюс, 2006. – 1056 с.: ил.
5. Осипов Д. Базы данных и Delphi. Теория и практика. – СПб.: Издательский дом БХВ, 2011. – 720 с.: ил.
6. Основы проектирования и разработки реляционных баз данных: Учебное пособие. – Ставрополь: Изд-во СГУ, 2007. – 203 с.

Интернет-ресурсы:

- www.microsoft.com – сайт компании Microsoft
 - www.embarcadero.com – сайт компании Embarcadero
 - <http://algolist.manual.ru/> - сайт посвящённый разработке алгоритмов
-

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Методические рекомендации по выполнению курсовых работ

Специальность 10.05.01 «Компьютерная безопасность»

Специализация «Информационно-аналитическая и техническая экспертиза компьютерных систем»

Квалификация выпускника – специалист

Год начала обучения – 2026 г.

Форма обучения – очная

Ставрополь
2026

1 Общие положения

Курсовая работа по дисциплине “СУБД” представляет собой теоретическое и исследование, и практическую (программную) реализацию базы данных по определённой для студента тематике. Курсовая работа оформляется в соответствии с общими требованиями и правилами, установленными ГОСТ 7.32-2001 Система стандартов по информатизации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления.

Курсовая работа выполняется студентом самостоятельно и является основанием для допуска к экзамену по дисциплине.

В соответствии с Государственными образовательными стандартами высшего профессионального образования по специальности 080801 «Прикладная информатика в экономике» курсовая работа имеет целью:

- систематизацию и углубление знаний студентов по дисциплине Базы данных (БД);
- совершенствование навыков и умений выпускника в решении сложных задач проектирования программного обеспечения (ПО).

Результаты работы оформляются в виде текстуальной части с приложением графиков, таблиц, чертежей, схем. Текстовые документы работы должны быть составлены и сброшюрованы в папки формата А4 по ГОСТ 2.105-95.

В процессе подготовки работы разрабатываются и последовательно реализуются следующие этапы:

- изучение литературных источников;
- планирование и выполнение исследований;
- работа над текстовой частью, чертежами, схемами и т.п.;
- оформление курсовой работы;
- защита курсовой работы.

За принятые решения и правильность всех выводов и фактических данных отвечает студент - автор работы.

2 Структура и содержание курсовой работы

Текстовый материал оформляется в следующей последовательности:

- титульный лист;
 - задание;
 - содержание;
 - перечень условных обозначений, символов, единиц и терминов;
 - введение;
 - основная часть работы;
 - заключение;
 - список использованных источников;
 - приложения.
-

3.1 Бланк титульного листа работы заполняется студентом самостоятельно.

3.2 Содержание включает наименование всех структурных элементов, разделов, подразделов, пунктов и подпунктов с указанием номера страницы, на которых размещается начало материала работы. В содержании указываются приложения с обозначением и наименованием. Слово «СОДЕРЖАНИЕ» записывают в виде заголовка (симметрично тексту) прописными буквами.

3.3 Перечень условных обозначений, символов, единиц и терминов далеко не всегда может присутствовать в работе. Как правило, он приводится в том случае, если в работе использованы малоизвестные сокращения и аббревиатуры. Причем если они встречаются в тексте довольно редко, перечень не составляется, а расшифровка приводится прямо в тексте, непосредственно после первого упоминания. Перечень печатается после содержания, перед введением двумя колонками: в левой (узкой) перечислены сокращения, справа (в более широкой) - их расшифровка.

3.4 Введение содержит в сжатой форме все фундаментальные положения, обоснованию которых посвящена курсовая работа. Писать введение методологически лучше не в начале, а по окончании работы над курсовой работой, когда основная часть работы уже полностью готова. Введение начинается с небольшой преамбулы, где упоминается сущность и состояние исследуемой проблемы, ее значимость в современных условиях, указывается на необходимость научного исследования. Далее введение составляется в следующем порядке:

- актуальность темы;
- цель и задачи работы;
- объект и предмет исследования;
- методы исследования;
- новизна полученных результатов;

3.5 Основная часть работы должна содержать данные, отражающие сущность, методику и основные результаты. Она включает:

– выбор направления работы, включающий обоснование выбора, методы решения задач, их сравнительную оценку. Это теоретический раздел работы дает теоретическое обоснование выбранной тематики и выполняется на основании литературных источников. Обучаемый должен представить анализ использованной литературы и дать свою точку зрения по проблеме исследования;

– второй раздел является аналитическим и охватывает основные технико-экономические показатели деятельности объекта исследования. Результаты анализа представляются в виде таблиц, диаграмм, графиков. Программное обеспечение может быть представлено в приложении. По аналитическому разделу работы представляются выводы в конце раздела;

– третий раздел рассматривает вопросы, связанные с темой работы. Обучаемый анализирует объект исследования и предлагает свои решения, исходя из результатов первых двух разделов.

3.6 В заключении к работе приводятся выводы, сделанные по результатам всей работы. В заключении целесообразно соотнести полученные выводы с целями и задачами, поставленными во введении. Здесь должны быть суммированы основные результаты исследования, сформулирована сущность решенной проблемы, даны грамотные и четкие предложения, касающиеся научного и практического использования полученных результатов.

3.7 В приложения рекомендуется включать материалы, связанные с выполненной работой, которые по каким-либо причинам не могут быть включены в основную часть. В приложения могут быть включены:

- материалы, дополняющие работу;
- промежуточные математические доказательства, формулы и расчеты;
- таблицы вспомогательных цифровых данных;
- методики, описания алгоритмов и программ задач, решаемых на ЭВМ, разработанных в процессе выполнения исследований;
- иллюстрации вспомогательного характера и др.

3.8 Сведения об источниках приводятся в соответствии с требованиями:

- ГОСТ 7.1-84 Система стандартов по информации, библиотечному и издательскому делу. Библиографическое описание документа. Общие требования и правила составления;
- ГОСТ 7.32-2001 Система стандартов по информатизации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления, п.6.1.

Порядок построения сведений об источниках – по мере упоминания в тексте. При ссылке в тексте номер источника из библиографического списка заключается в квадратные скобки, например, [1].

3.9 Приложения оформляют как продолжение курсовой работы на последующих листах. В тексте на все приложения должны быть даны ссылки. Приложения могут быть обязательными и информационными. Информационные приложения имеют рекомендуемый или справочный характер.

Каждое приложение следует начинать с новой страницы с указанием наверху посередине страницы слова «ПРИЛОЖЕНИЕ» и его обозначения, а под ним в скобках для обязательного приложения пишут слово «обязательное», а для информационного – «рекомендуемое» или «справочное».

Приложение должно иметь заголовок, который записывают симметрично относительно текста с прописной буквы отдельной строкой.

Приложения обозначают заглавными буквами русского алфавита начиная с «А», за исключением букв Е, З, И, О, Ч, Ъ, Ы, Ъ. После слова «ПРИЛОЖЕНИЕ» следует буква, обозначающая его последовательность, например, «ПРИЛОЖЕНИЕ А».

Если в документе одно приложение, оно обозначается «ПРИЛОЖЕНИЕ А».

Приложения выполняют на листах формата А4, допускается оформление на листах формата А3 (297х420) мм по ГОСТ 2.301.

Приложения должны иметь общую с остальной частью работы сквозную нумерацию страниц.

3 Оформление курсовой работы

Курсовую работу оформляют с использованием компьютерной техники на одной стороне стандартного листа белой бумаги (А4, 210 x 297 мм). Чертежи, большие таблицы и иллюстрации могут быть поданы в приложениях на формате А3.

Объем работы 20-30 листов, без учета приложений.

При наборе, печати работы делают следующие поля: сверху и снизу - 20 мм, справа - 10 мм, левое поле - 30 мм.

Предпочтительная гарнитура шрифта работы - Times New Roman, кегль - 14, цвет печати - черный, межстрочный интервал (интерлиньяж) - полуторный при печати на компьютере и двойной при печати на пишущей машинке. Плотность текста должна быть равномерной (без разрежений и уплотнений).

3.1 Изложение текста

3.1.1 Изложение текста работы лучше вести от третьего лица. Излагать мысли используя местоимение "я" не принято. В то же время, нельзя признать удачным излишний перегруз текста местоимением "мы", фразами "по нашему мнению", "считаем", "думаем" - это абсолютно нормально, но, в то же время, создает неосознанное впечатление, что работу писал коллектив авторов. Наиболее удачные в этом плане более неопределенные словесные обороты, например: "представляется, что...", "автор работы считает, что...", "разработанный подход позволяет...", "сначала производился опрос...", "данный вывод сделан на основании..." и т.д.

3.1.2 Текст работы должен быть кратким, четким и не допускать различных толкований. При изложении обязательных требований должны применяться слова «должен», «следует», «необходимо», «требуется», «не допускается», «запрещается», «не следует».

3.1.3 Изложение текста делается в безличной форме, например: «...значение коэффициента принято ...», или «принимается».

3.1.4 В тексте должны применяться научно-технические термины, обозначения и определения, установленные соответствующими стандартами, а при их отсутствии – общепринятые в научно-технической литературе.

3.1.5 В тексте не допускается:

- применять обороты разговорной речи;
 - применять для одного и того же понятия различные научно-технические термины, близкие по смыслу (синонимы), а также иностранные слова при наличии равнозначных слов и терминов на русском языке;
 - сокращать обозначения единиц физических величин, если они употребляются без цифр, за исключением единиц физических величин в головках и боковиках таблиц, в расшифровках буквенных обозначений, входящих в формулы и рисунки; применять математический знак минус (-) перед отрицательными значениями величин (следует писать слово «минус»);
-

– применять без числовых значений математические знаки, например, > (больше), < (меньше), = (равно), ≠ (не равно), а также знака № (номер), % (процент); применять индексы стандартов без регистрирующего номера; перечень допускаемых сокращений слов установлен в ГОСТ 2.316, ГОСТ 7.12; в тексте следует применять стандартизованные единицы физических величин, их наименование и обозначение, установленные в ГОСТ 8.417;

– если в тексте документа приводится диапазон числовых значений физической величины, то обозначение единицы физической величины указывается, после последнего числового значения диапазона.

Примеры: 1. От 1 до 5 мм.

2. От плюс 10 до минус 40° С.

Недопустимо отделять единицу физической величины от числового значения (переносить их на разные строки или страницы), кроме единиц физических величин, помещенных в таблицах, выполненных машинописным способом.

3.2 Оформление графической части

3.2.1 Материал, выносимый на листы большого формата, должен быть обязательно представлен в пояснительной записке. Размеры листов чертежей и плакатов должны быть стандартные. Надписи, технические требования и таблица на чертежах должны быть выполнены чертежным шрифтом. Блок – схемы алгоритмов и программ должны соответствовать требованиям ЕСКД.

3.2.2 Допускается представление графической части в виде презентации состоящей из 5-6 слайдов в Microsoft PowerPoint.

3.3 Вопросы нумерации

3.3.1 Страницы работы следует нумеровать арабскими цифрами, соблюдая сквозную нумерацию по всему тексту. Номер страницы проставляют внизу по центру страницы.

3.3.2 Титульный лист, задание включают в общую нумерацию страниц курсовой работы, но номера страниц на них не проставляются.

3.3.3 Структурные элементы работы не нумеруются, записывают в виде заголовка (симметрично тексту) прописными буквами жирным шрифтом. Расстояние между заголовками структурных элементов и текстом должно быть 3 интервала.

3.3.4 Иллюстрации и таблицы, расположенные на отдельных листах, и распечатки с ЭВМ включают в общую нумерацию страниц.

3.3.5 Разделы, подразделы, пункты, подпункты нумеруют арабскими цифрами. Расстояние между заголовком раздела и подраздела – 3 интервала, а между подразделом и текстом – 1,5 интервала.

3.3.6 Разделы работы должны иметь порядковую нумерацию в пределах основной части, например, 1, 2, 3 и т.д.

3.3.7 Пункты должны иметь порядковую нумерацию в пределах каждого раздела или подраздела. Номер пункта включает номер раздела и порядковый номер подраздела или пункта, разделенные точкой, например, 1.1, 1.2 или 1.1.1, 1.1.2 и т.д.

3.3.8 Номер подпункта включает номер раздела, подраздела, пункта и порядковый номер подпункта, разделенные точкой, например, 1.1.1.1, 1.1.1.2 и т.д. Если раздел или подраздел имеет только один пункт или пункт имеет один подпункт, то нумеровать пункт (подпункт) не следует. Заголовки подразделов следует начинать с абзацного отступа и печатать с прописной буквы в разрядку, не подчеркивая, без точки в конце.

3.3.9 Заголовки разделов следует располагать с абзацного отступа без точки в конце и писать прописными буквами, не подчеркивая. Если заголовок включает несколько предложений, их разделяют точками. Переносы слов в заголовках не допускаются.

3.3.10 Пункты и подпункты основной части следует начинать печатать с абзацного отступа.

3.3.11 Внутри пунктов или подпунктов могут быть приведены перечисления. Перед каждой позицией перечисления следует ставить дефис или при необходимости ссылки в тексте работы на одно из перечислений, арабскую цифру, после которой ставится скобка. Для дальнейшей детализации перечислений необходимо использовать строчные буквы, после которых ставится скобка, а запись производится с абзацного отступа, как показано в примере.

Пример:

- _____
- _____
- 1) _____
- 2) _____
- а) _____
- б) _____
- в) _____

3.3.12 Примечания приводят в том случае, если необходимы пояснения или справочные данные к содержанию текста, таблиц или графического материала. Примечания не должны содержать требований. Примечания следует помещать непосредственно после текстового, графического материала или в таблице, к которым относятся эти примечания, и печатать с прописной буквы в разрядку, с абзаца. Если примечание одно, то после слова «Примечание» ставится тире и текст примечания печатается тоже с прописной буквы. Одно примечание не нумеруют. Несколько примечаний нумеруют по порядку арабскими цифрами. Примечание к таблице помещают в конце таблицы под линией, обозначающей окончание таблицы.

Примеры:

Пр и м е ч а н и е – _____

П р и м е ч а н и я

- 1 _____
- 2 _____

3.4 Формулы

3.4.1 Формулы пишутся отдельной строкой. Выше и ниже каждой формулы должно быть оставлено не менее одной свободной строки.

Знаки препинания ставят на продолжении основной строки формулы, непосредственно за ней, например:

Математическая модель данной экспертной системы выглядит следующим образом. Продукционная система

$$PS = \langle F, Z, I \rangle, \quad (4.1)$$

где F — рабочая память системы (называемая также базой данных), содержащая текущие данные (элементы рабочей памяти);

Z — база знаний, содержащая множество продукций (правил вида: "условие $\rightarrow$ действие");

I — интерпретатор (решатель), реализующий процедуры вывода.

3.4.2 После формулы помещают перечень всех принятых в формуле символов с расшифровкой их значений и указанием размерности (если в этом есть необходимость). Буквенные обозначения дают в той же последовательности, в которой они приведены в формуле. Первая строка пояснения должна начинаться со слова «где» без двоеточия после него.

3.4.3 Переносы формул на другую строку допускаются на знаках неравенства, умножения, сложения, вычитания, деления, соотношений больше - меньше и других математических знаков.

3.4.4 При ссылке на формулу указывается ее полный номер в круглых скобках, например, ... в формуле (1.1).

3.4.5 Формулы нумеруются порядковой нумерацией в пределах всей работы арабскими цифрами в круглых скобках в крайнем правом положении на строке. Одну формулу обозначают – (1).

3.4.6 Формулы, помещенные в приложениях, должны нумероваться отдельной нумерацией арабскими цифрами в пределах каждого приложения с добавлением перед каждой цифрой обозначения приложения, например, формула (В. 1).

3.5 Иллюстрации

3.5.1 Все иллюстрации (графики, чертежи, схемы, фотографии) размещают сразу после первой ссылки на них. Например:



Ограничения (людские, материальные, финансовые и др.)

Рисунок 1 – Основные характеристики системы защиты

3.5.2 Иллюстрации сопровождаются:

- номером рисунка и наименованием (под рисунком посередине строки), например: Рисунок 1 – Основные характеристики системы защиты;
- поясняющими надписями, раскрывающими суть иллюстрации и детали содержания (они размещаются над словом рисунок и наименованием иллюстрации).

3.5.3 Иллюстрации следует нумеровать арабскими цифрами порядковой нумерацией в пределах работы, если иллюстрация одна, ее не нумеруют. При ссылках на иллюстрации следует писать «... в соответствии с рисунком 1». Иллюстрации должны выполняться на белой непрозрачной бумаге, не допускается калька и миллиметровая бумага.

3.6 Таблицы

3.6.1 Каждая таблица должна иметь порядковый номер в пределах всей работы. Название таблицы следует помещать над таблицей слева, без абзацного отступа в одну строку с ее номером через тире, например: «Таблица 1 – Наименование». При переносе части таблицы на другую страницу пишут «Продолжение», указывают номер таблицы, например: «Продолжение таблицы 1». В данном случае, нижняя черта проставляется на последней части таблицы. На все таблицы должна быть ссылка, при ссылке следует писать слово «таблица» с указанием ее номера. Таблица помещается после первого упоминания о ней в тексте так, чтобы ее можно было читать без поворота листа или с поворотом по часовой стрелке. Графа № п/п в таблицы не включается. Для облегчения ссылок по тексту в таблицах допускается нумерация граф.

3.6.2 В заголовках таблиц недопустимы нестандартные сокращения. В названиях граф надписи пишут в именительном падеже, единственном числе.

Таблица 1 –.....

Таблица 2 – Результаты структурирования информации

№ элемента информации	Наименование элемента информации	Гриф конфиденциальности информации	Цена информации	Наименование источника информации	Местонахождение источника информации
1	2	3	4	5	6

Продолжение таблицы 2

1	2	3	4	5	6

4 Примерный перечень тем курсовых работ

1. “База данных поликлиники”. Учёту подлежат: больные (ФИО, домашний адрес), лечащий персонал (ФИО, специальность), дата посещения больным врача и поставленный диагноз.
2. “База данных футбольных матчей”. Учёту подлежат: спортивный клуб, игроки команд (ФИО), дата матча, игравшие команды, забитые мячи, авторы голов.
3. “База данных “Бюро по туризму”. Учёту подлежат: клиенты фирмы (ФИО, домашний адрес), список курортов с ценами и продолжительностью путёвок, даты поездок клиентов на курорт.
4. “База данных “Магазин видеофильмов”. Учёту подлежат: название фильма, жанр, год выхода, киностудия, режиссёр, цена, носитель (DVD/Видеокассета), число экземпляров.
5. “База данных “Домашняя фонотека”. Учёту подлежат: исполнитель, страна к которой принадлежит исполнитель, название альбома, год выхода альбома, музыкальный жанр альбома, наименование и продолжительность музыкальных композиций входящих в альбом, формат хранения (CD, mp3, ...).
6. База данных “Склад автомобильных запчастей”. Учёту подлежат: название детали, производитель, цена, в какой марке автомобиля применяется, поставщик детали.
7. База данных “Метеостанция”. Учёту подлежат: названия станций, их географические координаты, персонал станций (ФИО, должность), каждая метеостанция ежечасно сохраняет данные о температуре окружающей среды.
8. База данных “Библиотека”. Учёту подлежат: жанр книги, авторы книги, название книги, количество экземпляров, место хранения книги, читатель (ФИО и адрес), дата получения читателем, дата возврата книги.
9. База данных “Склад телевизоров”. Учёту подлежат: поставщик, производитель телевизора, модель телевизора, дата поставки телевизора, дата списания телевизора со склада, не менее 5 характеристик телевизора (производитель, диагональ, вес, и т.п.), цена телевизора от поставщика, отпускная цена телевизора, количество телевизоров.
10. База данных “Автовокзал.” Учёту подлежат: автобусы (марка, дата выпуска и государственный регистрационный номер), водители (ФИО, дата рождения, водительская категория, какой автобус закреплён), расписание движения автобусов (наименование населённого пункта, день и время отправления, день и время прибытия, стоимость билета, автобус выполняющий рейс).

- 11.База данных “Магазин одежды.” Учёту подлежат: товар (название, размер, цена, материал, производитель), дата поступления, дата продажи.
- 12.База данных “Почтовое отделение” Учёту подлежат: поступающая на почту заказная корреспонденция (письма, ценные бандероли и посылки). Дата поступления, дата выдачи, паспортные данные получателя корреспонденции.
13. База данных “Аптечный киоск” Учёту подлежат: лекарства (наименование, дозировка, производитель, отпускная цена, место хранения, выдача только по рецепту или нет, показания и противопоказания).
14. База базы данных “Расписание занятий в институте” Учёту подлежат: специальность, учебная группа, учебные дисциплины (название и вид занятия), день недели, номера часов проведения занятий, преподаватель, номер аудитории
- 15.База данных “Гостиница” Учёту подлежат: номера гостиницы (номер помещения, число спальных мест, наличие душа, туалета, холодильника, телевизора, сплит-системы, стоимость проживания), занятость номера (дата заселения, паспортные данные жильцов, дата освобождения номера)
- 16.База данных “Кинотеатр”. Учёту подлежат: название кинофильма, жанр кинофильма, возрастные ограничения, дата и время киносеанса, название кинозала, номер посадочного места, цена билета.
- 17.База данных “Торговый зал”. Учёту подлежат: название товара, отпускная цена товара, количество товара в зале, дата выпуска, срок годности. База данных должна предусматривать отпуск товара покупателю со списанием товара из зала.
18. База данных “Учёт электроэнергии”. База учитывает количество потреблённой электроэнергии в небольшом посёлке и ежемесячно подготавливает индивидуальные квитанции об оплате. Учёту подлежат: адрес дома, владелец дома, количество потреблённой электроэнергии за месяц, стоимость за кВт/ч. Стоимость электроэнергии может изменяться!
19. База данных “Отдел кадров завода”. Учёту подлежат: ФИО, рабочая специальность, квалификация (разряд), образование работника, дата трудоустройства, стаж работы, должностной оклад, надбавка (зависит от квалификации и стажа).
20. База данных “Художественный музей” Учёту подлежат: автор картины, название картины, краткое описание сюжета, дата написания картины, художественное направление (реализм, абстракция, и т.д.), зал в котором выставлена картина.

Условия выполнения

База данных разрабатывается на основе Microsoft Access (или на другой БД по согласованию с преподавателем). БД должна быть нормализована до 4 нормальной формы.

Клиентское приложение разрабатывается на языках Delphi, C++ Builder, C# или на другом языке по согласованию с преподавателем.

Приложение должно включать формы ввода данных, формы просмотра и редактирования данных. В рабочих формах приложения должна быть предусмотрена возможность:

1. Упорядочивания данных;
2. Фильтрации данных;
3. Поиску данных.

Программа должна предоставлять возможность вывода отчётов. В отчётах должны быть предусмотрены возможности:

1. Вывода всех данных из всех таблиц БД;
2. Использования агрегирующих функций SQL (Avg, Count, Min, Max, и т.д.).

Отчёт должен содержать:

1. ER-модель БД;
2. Полную спецификацию таблиц.
3. Примеры используемых в работе SQL запросов.
4. Описание клиентского приложения.
5. Инструкцию по работе с приложением.

К отчёту прилагается CD-диск с БД и исходным кодом проекта. Допускается один CD- на несколько человек или всю учебную группу.

ПРИЛОЖЕНИЕ А Образцы библиографической записи.

Книги.

Домашев А.В. Программирование алгоритмов защиты информации: учебное пособие. / Домашев А.В., Грунтович М.М. – М.: Нолидж, 2005.-416с.

Перроун П. Д. Создание корпоративных систем на базе Java2 Enterprise Edition: рук. разработчика : [пер. с англ.] / Поль Дж. Перроун, Венката С. Р., Р. Чаганти. – М. : Вильямс, 2005. – 1179 с.

Управление персоналом: учебное пособие / В.А.Якушев [и др.]; под ред. В.А.Якушева. – М.: Высшая школа, 2006. – 511с.

Стандарты и кодексы

ГОСТ Р 517721–2001. Аппаратура радиоэлектронная бытовая. Входные и выходные параметры и типы соединений. Технические требования [Текст]. – Введ. 2002–01–01. – М. : Изд-во стандартов, 2001. – IV, 27 с. : ил. ; 29 см.

ГОСТ 7. 53–2001. Издания. Международная стандартная нумерация книг [Текст]. – Взамен ГОСТ 7.53–86 ; введ. 2002–07–01. – Минск : Межгос. совет по стандартизации, метрологии и сертификации ; М. : Изд-во стандартов, сор. 2002. – 3 с. – (Система стандартов по информации, библиотечному и издательскому делу).

Гражданский кодекс РФ. Часть четвертая от 18 декабря 2006 г. № 230-ФЗ: принят Гос. Думой Федер. Собр. Рос. Федерации 24.11.2006 г. // Российская газета – 2006. – 22 дек., ч.1, ст 5444.

Статья из газет и журналов

Боголюбов А. Н. О вещественных резонансах в волноводе с неоднородным заполнением [Текст] / А. Н. Боголюбов, А. Л. Делицын, М. Д. Малых // Вестн. Моск. ун-та. Сер. 3, Физика. Астрономия. – 2001. – № 5. – С. 23–25.

Громов В.А. Информационная безопасность России / В.А.Громов // Известия. – 2008. – 2 марта. – С.2.

Словари и энциклопедии

Социальная философия: словарь / под общ.ред. В.А.Иванова. – М.: Академия, 2005.-353 с.

Экономическая энциклопедия / В.И.Иванов [и др.]. – М.: Экономика, 2007. – 233 с.

Статьи из сборников

Иванов В.Р. Конфликты в системе управления персоналом / В.Р.Иванов // Материалы конференции – М.: Юрист, 2007. – С. 344-355.

Электронные ресурсы

Художественная энциклопедия зарубежного классического искусства [Электронный ресурс]. – Электрон. текстовые, граф., зв. дан. и прикладная прогр. (546 Мб). – М. : Большая Рос. энцикл. [и др.], 2005. – 1 электрон. опт. диск (CD-ROM).

Логинов Л.Г. Информационная безопасность объектов / Безопасность техно-

логий: Интернет-журнал. URL: <http://www.secur.ru> (дата обращения: 17.05.2005).

Иванов В.Е. Справочник по полупроводниковым приборам // Сервер радиолюбителей России. URL: <http://www.radio.ru> (дата обращения: 17.05.2005).

Информационная безопасность России: [сайт].[2004]. URL: <http://www.info.ru> (дата обращения: 17.05.2005).
